TCG Software Stack (TSS) Specification Version 1.2 Level 1

Errata A

Part1: Commands and Structures

March 7, 2007

Change History

Version	Date	Description
Errata 1	9/9/05	Based on Golden Candidate1. Fixed numerous typos, per HP's and NTRU review. Set port to 30003. Fixed parameters in TickStampBlob. Removed 3 duplicate functions, moved one function to a different section, added TSS_TCSCAP_RETURNVALUE_INFO, put in correction for allowing both hashing with/without terminating zero (backwards compatibility-previously agreed upon) and some clarifications in the text of the document.
Errata 1a	9/14/05	Added missing attribs under details of SetAttrib Command for Policy object and missing details on the hashing in Errata 1.
Errata 2	9/26/05	Added changes to Transport, audit, commas, misnamed variables
Errata 2a	10/05/05	Added in parameter for key size in GetAttrib for Tspi_Key Class definitions
Errata 2b	10/12/05	Remove ', out' from hPolicy parameter in delegation. Clarification in NV_Define/ReleaseSpace from Infineon
Errata 3	10/20/05	TSS_TPMCAP_MIN_COUNTER added into section 4.3.4.10.1 and 2.3.2.19 Deleted TSS_ES_RSAESOIAP_SHA1_MGF1 and TSS_ES_RSAESOAP_SHA1_MGF2 from approved schemes, section 2.3.2.26 as they didn't make sense. Removed “automatic registration of keys” as it made no sense. Added descriptive text in TSS_TPM_PcrExtend and in 2.6.2 on how the data to extend is calculated.
Errata 3b	11/04/05	Formatting changes.
Errata 3c	11/28/05	Synchronized 2.3.2.17 and the Get/Set commands that use those flags
Errata 3d	11/30/05	Remove Return values from DAA Tcsi commands. Remove blank sections. Change format of title of a number of sections (which renumbers them)
Errata 4	12/14/05	Change two pictures of DAA (Zimmerman). Update fomulae for DAA (per Zimmerman note, December 05). This includes adding some parameters to one function, VerifyInit, and changing DAA_VerfiySignature parameters from out to in (which also changes the type). Add public key information to Tcsi_KeyControlOwner function. Update formatting of algorithms to better reflect comparison paper.
Errata 5	12/21/05	Added in Tcsip_OwnerReadInternalPub, contents at beginning of second section.
Errata 5a	1/3/06	Fixed some typos in DAA section pointed out by Roger, added “p” to pointers variables in NV_ReadValue and CreateRevokableEndorsementKey
Errata 5b	1/6/06	Fixed name CONTEXT_VERSION ->CONTEXT_VERSION_MODE and formatting of Tcsip_ReleaseTransportSigned. Fixed Tables of Content that

		correspond to them as well. Added in GetAttribUint32 section for additional Context regarding transport
Changes	1/23/06	Added last two lines of table 4.3.3.2.4 to 4.3.3.2.3
Started errata	2/23/06	Included DAA changes noted by Lin Yang, and also the two Tddl power management functions (PowerManagement and PowerControl) that needed a pointer as the last parameter.
Version A.12	3/10/06	Added in new command / flag for resetting the lock when multiple incorrect authorization attempts set a lock
Version A.14	4/10/06	Standardized on TSS_TSPATTRIB_ENCDATASEAL_PROTECT Added TSS_KEY_TYPE_MIGRATE to 2.3.2.2 Added TSS_E_NO_ACTIVE_COUNTER Removed Tspi_Data_SealX, Removed Tspi_TPM_SetOrdinalAuditStatus Changed TPM_CAP_PROP_TPM_ORDINATL_AUDIT_VALUE to TPM_COMMAND_CODE Changed Default for POPUP from add terminating NULL to removed terminating NULL
Version A.15	5/24/06	Clarification put into 4.3.4.4.5
Version A.16	5/31/06	Removed “by the user” in informative comment in 4.3.4.4.5 (“user” is an undefined term)
Version A.17	6/12/06	Removed last two lines from 4.3.4.1.3 and 4.3.4.1.4 Added two commands: 5.6.2.7.2 and 5.6.2.7.3 so that credentials can be obtained via a transport session
Version A.18	6/15/06	Added in all the NTRU errata from 6/14
Version A.19	7/26/06	Added in numerous corrections from IBM (DAA), Infineon (Cert management), NTRU (various) and Wave (various).
Version A.22	8/31/06	Added in changes per Kent Yoder's note of 8/29/06 with discussion at 8/30 meeting.
Version A.23	9/1/06	Added in changes re the DAA cert callbacks
Version A.24	9/22/06	Added informative comments in CollateIdenity and ActivateIdentity callbacks
GC2	10/11/06	Included latest changes to DAA
Version A.25	10/27/06	Fixed numerous typos in DAA formulae, reorganized DAA data into 3 objects, DAACredential, DAAIssuerKey, and DAAARAKey, and defined attributes and sub-attributes for each.
Mod1	1/10/07	Update Table of Contents
Mod 2	1/11/07	Added changes to remarks of Tspi_NV_DefinesSpace
Mod 3	2/9/07	Added Tspi_TPM_GetSRKPubKey
Mod 4	3/6/07	Added informative remark to Tspi_Key_CreateKey and Tspi_Key_WrapKey and Tspi_TPM_Delegate_CreateDelegation

Acknowledgement

TCG wishes to thank all those who contributed to this specification. This version builds on the work published in version 1.1b and those who helped on that version have helped on this version.

A special thank you goes to the members of the TSS workgroup who had early access to this version and made invaluable contributions, corrections and support.

David Challener

TSS Workgroup Chair

Introduction

Start of informative comment:

The TCG 1.2 Main specification defines a subsystem with protected storage and protected capabilities. This subsystem is the Trusted Platform Module (TPM). Since the TPM is both a subsystem intended to provide trust and to be an inexpensive component, resources within it are restricted. This narrowing of the resources, while making the security properties easier and cheaper to build and verify, causes the interfaces and capabilities to be cumbersome. TCG has solved this by separating the functions requiring the protected storage and capabilities from the functions that do not; putting those that do not into the platform's main processor and memory space where processing power and storage exceed that of the TPM. The modules and components that provide this supporting functionality comprise the TSS.

The TSS 1.2 specification contains additions to the TSS 1.1b specification that correspond to enhancement that were made in the main TPM specification. Code written using the TSS 1.1b specification should continue to work when executed against a TSS 1.2 stack. However, the reverse is not always the case, as there is new functionality included in the 1.2 main specification that is reflected in the TSS.

That enhancement comes in several areas:

New Functionality:

Auditing – Auditing was broken in the 1.1b main specification, so it was not implemented in the 1.1b TSS specification. With changes made in the 1.2 specification, it is now ready to be used by application vendors.

Transport sessions – This new functionality allows an application to talk to a TPM securely.

Non-volatile monotonic counters – This new functionality is just what it sounds like – the TPM will have several non-volatile monotonic counters, and making use of them requires new APIs.

Delegation: In the 1.1b specification, the only way to let someone or something to use a key was to give them the use_authentication data. Needless to say, this removed the possibility of retracting that permission. Delegation allows a finer control, so that use of a function may be delegated to another person / piece of software, without losing the ability to retract that permission.

Context Saving – Since loading keys into the TPM turned out to be time consuming, context saving gives the TPM the ability to cache its internal memory outside the chip without causing a security vulnerability. In 1.1b, this functionality was optional. In 1.2 it is now mandatory.

NonVolatile Storage – Early in the boot sequence, some systems don't have access to persistent storage. Additionally, some customers typically wipe a hard drive and install a corporate image before using a system, leaving no place to store a certificate on the platform. NonVolatile storage provides a small amount of persistent storage that can be used in these cases.

Secure Timing: It was too expensive to require a TPM to have a real time clock, with battery, that could be used to do time stamping. However, it is possible to do something similar by correlating a tick counter with an external time stamping source, and then using the TPM to do secure time stamping that piggybacks off the external time source. 1.2 provides a means of implementing this.

Direct Proof: Some privacy organizations were worried that no privacy CA would exist by the time that the need for them arose. As a result, a new means of providing anonymous (and pseudonymous) proof that a key came from a genuine TPM was developed that does not require a third party anonymizer.

Because the TPM has limited resources, a requirement for direct anonymous attestation was that the operations carried out on the TPM be minimal and, if possible, be outsourced to TSS. Of course, security must be maintained, i.e., a (corrupted) Platform TSS should not be able to authenticate without interacting with the TPM. However, privacy/anonymity needs only be guaranteed if the Platform TSS is not corrupted: as the host controls all the communication of the TPM to the outside, a corrupted Platform TSS can always break privacy/anonymity by just adding some identifier to each message sent by the TPM. In fact, our scheme satisfies an even stronger requirement: when the corrupted software is removed, the privacy/anonymity properties are restored.

Compared to other TSS functions, the TSS DAA functions will do a great amount of computations for reasons explained above.

Besides the TPM and its Platform TSS, DAA interacts with a DAA Issuer, DAA Verifier, DAA Mediator and DAA Anonymity Revocation Authority which do not need a TPM themselves. Their behavior is specified in optional TSS functions.

***Extended functionality*:** Some of the functions that already existed in the 1.1b specification have been extended to provide more flexibility in their use. These include:

Identity generation – Identity certificates can now be locked to PCRs and locality, and "soft" identities, whose private key does not reside in the TPM can now be created.

PCR use – It used to be the case that the same PCRs were recorded at creation and unseal – now each can be specified separately. In addition, PCR 15 has been reserved for software testing, and can be reset without problem. In addition, some PCRs can be set to be resettable only when the TPM in in a specific locality state.

Authentication – Authentication necessary to use a key used to be either through an HMAC or PCR values (or both). Now locality can also be used as well

New signing key types – Some new varieties of keys have been generated, which will have restricted usage.

New types of migratable keys – These CMKs (Certified Migratable Keys) are tied at creation to a migration pub key or migration authority.

New flexibility in EKs. In the 1.1b specification, endorsement keys were fixed in the chip at manufacture. This allowed a certificate to be provided by the manufacturer for the key. However, some privacy advocates are worried about the EK becoming a non-changeable identifier (in spite of all the privacy controls around it, which would make doing this very difficult). As a result, the specification allows a manufacturer to allow the key to be removed by the end user and regenerated. Of course the certificate at that point would become worthless, and it could be very expensive for the end user to get a new certificate.

New Attributes:As mentioned in the above, some structures in the specification are getting new attributes. Specifically, locality, which is a state of the TPM controlled by special signals from the bus, and PCR attributes, which allow the PCR to become resettable in certain locality states. Additionally, PCR[15] will be a debug PCR and will be resettable in every locality.

As new TPM specifications come out and more platform specific specifications come out, the programmer has a more and more difficult time determining when he is using a attribute that is guaranteed to exist on the platform (s) he is targeting. This is not just a problem of functions, which might be solvable through a table, but also a problem of key types. For example, the signing function does not change when going from a 1.1 platform to a 1.2 platform, but if PCR_long is used instead of PCR_short in that function, it will only work in a 1.1 platform and not a 1.2 platform. It is necessary for a program designer to know what functionality is guaranteed available in the platforms he is targeting to run his code. Failing this, a programmer will have to provide code to determine if the platform the application is running on actually supports all the features he needs.

In spite of the fact that there are multiple localities that can use the TPM, the TSS is written assuming it has exclusive access to the TPM and to the sessions it has access to according to the values recorded in NVRAM. If virtualization of the TPM is necessary to provide this service, it is assumed such will be provided.

End of informative comment.

Table of Contents

List of Figures

Preface

This document is an industry specification that enables trust in computing platforms in general.

This specification defines a TCG Software Stack (TSS) that is an integral part of each platform, and provides functions that can be used by enhanced operating systems and applications. The software stack employs cryptographic methods when establishing trust, and while this does not in itself convert a platform into a secure computing environment, it is a significant step in that direction.

Standardization is necessary so that the security and cryptographic community can assess the mechanisms involved, and so that customers can understand and trust the effectiveness of new features. Manufacturers will compete in the marketplace by installing software stacks with varying capabilities and cost points. The software stack itself will have basic functions that maintain privacy, yet support the identity and authentication of entities such as the platform, the user, and other entities. The software stack will have other capabilities to protect data and verify certain operational aspects of the platform. It can be a separate device or devices, or it can be integrated into some existing component or components provided the implementation meets the requirements of this specification. This is necessary to achieve the fundamental goal of ubiquity.

Please note a very important distinction between different sections of text throughout this document. Beginning after this section, you will encounter two distinctive kinds of text: informative comment and normative statements. Because most of the text in this specification will be of the kind normative statements, the authors have informally defined it as the default and, as such, have specifically called out text of the kind informative comment. They have done this by flagging the beginning and end of each informative comment and highlighting its text in gray. This means that unless text is specifically marked as an informative comment, you can consider it as a normative statements.

The key words “MUST,” “MUST NOT,” “REQUIRED,” “SHALL,” “SHALL NOT,” “SHOULD,” “SHOULD NOT,” “RECOMMENDED,” “MAY,” and “OPTIONAL” in the chapters 2-10 normative statements are to be interpreted as described in [RFC-2119].

For example:

Informative Statements:

Start of informative comment:

This is the first paragraph of 1–n paragraphs containing text of the kind informative comment,

This is the second paragraph of text of the kind informative comment,

This is the nth paragraph of text of the kind informative comment,

End of informative comment.

Normative Statements:

To understand the TCPA specification the user must read the specification. (This use of MUST does not require any action).

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

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

1. The TCG Software Stack (TSS)

1.1 General Introduction

The TCG Software Stack (the TSS) is the supporting software on the platform supporting the platform's TPM.

This document is written with the assumption that the reader or TSS implementer has an understanding of the TCG TPM 1.2 Main Specification (Published under this name by the Trusted Computing Group, Inc.) and cryptographic infrastructures in general.

A note on specification, organization and architectural naming: The Trusted Computing Group has adopted the specification titled "TCG TPM 1.2 Main Specification". The name of the specification and definition labels (e.g., structure names, etc.) from the TSS version 1.1b were retained to preserve existing references and implementations and are therefore retained in this specification.

Start of informative comment:

The TCG TPM 1.2 Main specification defines a subsystem with protected storage and protected capabilities. This subsystem is the Trusted Platform Module (TPM). Since the TPM is both a subsystem intended to provide trust and to be an inexpensive component, resources within it are restricted. This narrowing of the resources, while making the security properties easier and cheaper to build and verify, causes the interfaces and capabilities to be cumbersome. The TCG architecture has solved this by separating the functions requiring the protected storage and capabilities from the functions that do not; putting those that do not into the platform's main processor and memory space where processing power and storage exceed that of the TPM. The modules and components that provide this supporting functionality comprise the TSS.

End of informative comment.

1.2 Introduction to the TSS

Start of informative comment:

The block diagram in Figure 1-1 TSS Block Diagram and Description of Sections illustrates the TSS modules, components and their relationships. TSS modules are the major parts of the TSS providing fundamental resources to support the TPM. Each module is comprised of components, which provide the more specialized functions of the modules.

The primary design goals are:

- Supply one entry point for applications to the TPM functionality
- Provide synchronized access to the TPM
- Hide building command streams with appropriate byte ordering and alignment from the applications
- Manage TPM resources (Including creation and release)

End of informative comment.

The TSS is comprised of discreet modules which are designed to be discreet. Therefore, interfaces between them are defined in this specification to provide interoperability. The horizontal dashed lines represent these interfaces. While architecturally there is no constraint on the nature of the interfaces, 'C' and IDL are specifically defined in this revision of the TSS Specification. Modules are comprised of one or more internal components. Interfaces between components within each modules is a design consideration for each module and does not affect interoperability, therefore, this specification does not address these interfaces.

Figure 1-1 TSS Block Diagram and Description of Sections depicts the components of the TCG software stack and their related sections.

1.3 TSS functions defined are not exclusive

By providing all the functions listed within the normative text the TSS implementation SHALL be considered a complete and valid implementation (compliance testing, if any that may be defined, notwithstanding.) The list of functions defined within this specification (at any level) may be augmented by a TSS component provider to add extra functionality. The addition of any functions does not exclude a TSS implementation from being considered a valid implementation.

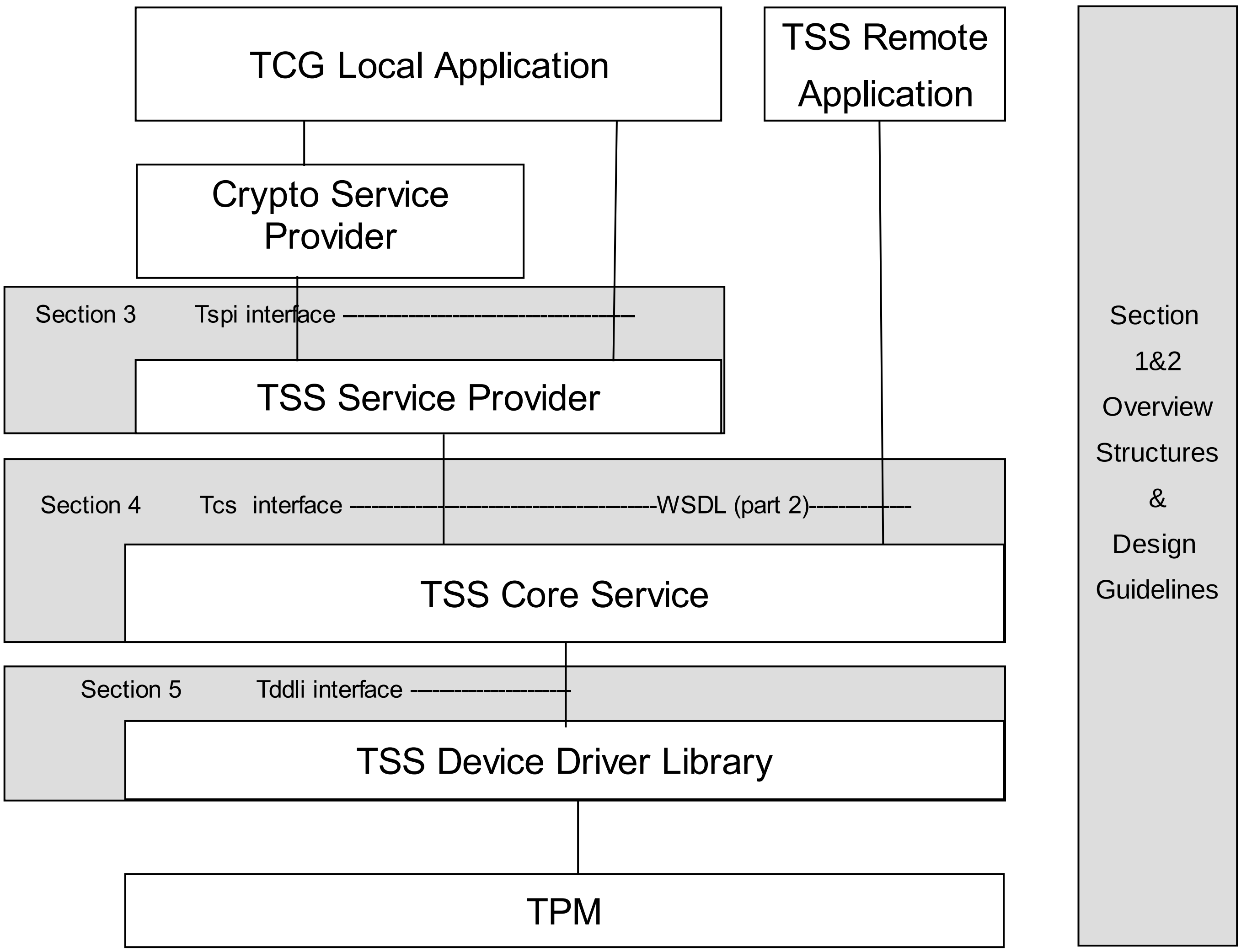


Figure 1-1 TSS Block Diagram and Description of Sections

1.4 Platform Architecture

Start of informative comment:

The architecture of the TSS defined in this specification is intended to be neither platform nor operating system dependent. While the nature and extent of the modules and components will vary for each platform and operating system, the modules, their interactions, and relationship will all be the same whether this platform is a large server or a personal digital assistant. While the nature of the platforms and their operating systems will vary, the TSS architecture assumes all will have the attributes described to the left of the diagram in Figure 1-2 TSS Architecture Diagram.

End of informative comment.

1.4.1 Platform Modes

Start of informative comment:

Kernel Mode: This is where the device drivers and the core components of the operating system reside. Code in this area services and protects the applications running in User Mode. Code in this area usually requires some type of administration authorization to modify or update. Kernel mode typically doesn't have separate processes separating the execution of the executable within it.

User Mode: This is where user applications and services execute. These applications and services are usually loaded and executed at the request of the user but may be loaded as part of a startup sequence. The operating system provides one or more protected areas called processes for different application or services to execute providing protection from each other. Within User Mode there are typically two classes of applications:

User Applications: These are executable code run at the request or behalf of the user. Due to the fact that this code is initiated and may be provided by the user, it may not be as trusted as other code executing on the platform.

System Services: These applications are generally started during initialization of the operating system as part of a startup script or may be initiated as a result of a request of the server. These typically provide common and usually trusted services for the User Applications. Since this code executes in its own process, separate from the User Application and is started by the operating system, it is considered more trusted than User Application and may be as trusted as Kernel Mode executables. In a Windows operating system this is a "System Service" while in a Unix operating system this would be a daemon.

End of informative comment.

1.4.2 Procedure Calls

Start of informative comment:

There are two methods a routine calls another routine: Local Procedure Call (LPC) or Remote Procedure Call (RPC).

End of informative comment.

1.4.2.1 Local Procedure Call (LPC)

Start of informative comment:

An LPC is a direct call from one application, module, or component to another within the same process or address space allocated by the OS. In an LPC the calling routine can resolve directly (compile time) or indirectly (run time) the local address of the routine to call or pass control to.

End of informative comment.

1.4.2.2 Remote Procedure Call (RPC)

Start of informative comment:

An RPC provides the interaction between processes. It is:

- A set of rules for marshalling and unmarshalling parameters and results;
- A set of rules for encoding and decoding information transmitted between two processes;
- A few primitive operations to invoke an individual call, to return its results, and to cancel it;
- Provision in the operating system and process structure to maintain and reference state that is shared by the participating processes.

RPC requires a communications infrastructure to set up the path between the processes and provide a framework for naming and addressing. Anything that provides these services qualifies as an RPC.

End of informative comment.

1.5 Trust Boundaries

Start of informative comment:

Security architectures are best done using a layered and simple approach separating the components that require trust from the components that don't. An architecture where lines can be drawn around and between the trusted and untrusted modules allows for easier maintenance and validation of the security properties of the system. This line is historically called the Trusted Computing Base or TCB. The components below or within the TCB are small, simple and trusted while the components outside the TCB can be more complex and larger. In the TCG architecture, the boundary around the TPM is the TCB. All components outside the TPM (i.e., the TCB) are untrusted such as the TSS, OS, and applications.

End of informative comment.

1.6 Privacy Boundaries

Start of informative comment:

Privacy is different from security, and is much harder to maintain. While it is possible to keep keys secret in a TPM, it is clearly the case that if an adversary is able to run arbitrary code in most OSs, the privacy of the end user has been compromised. Thus while information the TSS stores and uses may not be security sensitive, it very well may be privacy sensitive. An example of this would be the certificate for the system and the public portion of the SRK. These both could be used to identify the system. They may be “spoofable”, but if a remote user can get access to this data for the most part it would be a privacy break. The TSS has no way to protect this data securely, as it is software. However there is no reason that the TSS can’t obey a policy set out by the owner of the system so that absent of arbitrary code running on the system the TSS does not expose private information (defined by policy) to the outside world. The TSS design is meant to enable such functionality.

End of informative comment.

1.7 OS Dependency

Implementation of a TSS may not be restricted to any operating systems, independent of the operating system's resource limitations and user restrictions. However it is not the case that every TSS stack will provide the same functionality to a process. This is for several reasons. First, different platform specific specifications may provide different mandatory and optional commands for a TPM. Beyond this, some commands (such as symmetric encryption) are optional in the TSS itself. Additionally, for privacy reasons, an OS may restrict access to certain functionality of a TSS based on the rights of the currently logged in user/process. As a result, it will be necessary for the TSS to provide a means for the end user/process to determine exactly what functionality is available in the TSS.

1.8 Roles

Start of informative comment:

Entities (human or machine) perform functions on the TSS and the TPM. These functions are performed while acting in one of the following roles:

End of informative comment.

1.8.1 TPM Owner

Start of informative comment:

This is the entity that owns and has “title” to the platform. There is only one TPM owner of the platform. Since proof of ownership is made by the presentation of authentication data, if the owner chooses, the owner’s authentication data can be shared among other entities allowing a form of delegation.

Examples:

In a corporate environment, the TPM owner would be the corporation or IT department.

In a home environment, the TPM owner would be the person who owns the platform.

End of informative comment.

1.8.2 TPM User

Start of informative comment:

These are entities that can load or use TPM objects such as TPM keys. It is important to understand that the TPM itself does not maintain a list of TPM users. A TPM user is any entity that can present the authentication data for an object. The first TPM user is created by the TPM owner. Thereafter more TPM users can be created by either the TPM owner or other TPM users.

A TPM user is not necessarily a human. A TPM user may be service provider such as a mail or file server.

Examples:

In a corporate environment, a TPM user would be an employee of the company or the servers of the corporation.

In a home environment, a TPM user could be a member of the household (or guest) that the TPM owner allows access to TPM objects or external service providers such as a bank or stock broker.

End of informative comment.

1.8.3 Platform Administrator

Start of informative comment:

This is the entity that controls the platform’s OS, filesystem, or data. The Platform administrator may or may not be the TPM owner.

End of informative comment.

1.8.4 Platform User

Start of informative comment:

These are entities that the Platform administrator allows use of the data or resources of the platform. The Platform users may or may not be TPM users.

End of informative comment.

1.8.5 Operator

Start of informative comment:

The human physically at the platform able to directly operate it and observe physical indications. The operator may or may not be a TPM owner or TPM user but will likely be either a Platform administrator or Platform user.

End of informative comment.

1.8.6 Public

Start of informative comment:

Performs any function on either the platform's OS, filesystem, or data allowed without an identity or authentication. Performs any function on the TPM that does not require authentication.

End of informative comment.

1.9 TSS Architecture

Start of informative comment:

Describing the modules starting from the bottom up:

- The TPM Device driver is typically provided by the TPM manufacture and incorporates code that has understanding of the specific behavior of the TPM. This code is expected to be loaded and function in Kernel Mode. Since user mode executables can not gain access to hardware directly in modern operating systems, and the TCG imposes the restriction that the Kernel Mode Executable device driver exports its functionality only through the TDDL and disallows other components from bypassing the TDDL, the manufacturer also provides the TCG Device Driver Library. The TSS exclusively opens the TPM device driver; the driver does not allow any applications to have an additional connection to the TPM device besides the TSS.
- The TCG Device Driver Library (TDDL) provides two functions:
  - A standard interface defined in this specification for the TPM so all TPMs look and behave the same at this interface (Tddli).
  - Provides the transition between the User Mode and Kernel Mode. There will typically be one executable image of each of these per TPM on the platform.
- The TSS Core Services (TCS) resides in User Mode and communicates to the TPM via the TPM Device Drivers Library Interface (Tddli) provided by the TDDL. There will typically be one image of this component per platform and it executes as a system service. This module provides all the primitives and more sophisticated functions such as key management required to efficiently manage the TPM's limited resources. The interface to the TCS is the TSS Core Service Interface (Tcsi). This interface is designed to provide a straight forward, simple method for controlling and requesting services from the TPM. The functions are designed to be atomic in nature requiring little setup and overhead.
- TSS Service Providers (TSP) are the top-most modules and provide a rich, object-oriented interface for applications to incorporate the full capabilities of a TCG-enabled platform. The interface used by the applications to access the TSP is the TSS Service Provider Interface (Tspi). While not an architectural requirement, it is intended that the TSP obtain many TCG services such as TPM byte stream generation, key management, etc from the TCS.

Another type of module that may make use of the TCS is an RPC server. This module marshals the TCS functions and data from one TCG platform to another platform or device.

It's important to emphasize that since the security properties of the TPM's protocols have been specifically designed to not rely on the security properties of the data transport, none of the TSS modules, components, the RPC communications or RPC events affect the trusted properties of the TPM. All modules, components and interfaces outside the TPM are considered untrusted in relation to the TPM.

End of informative comment.

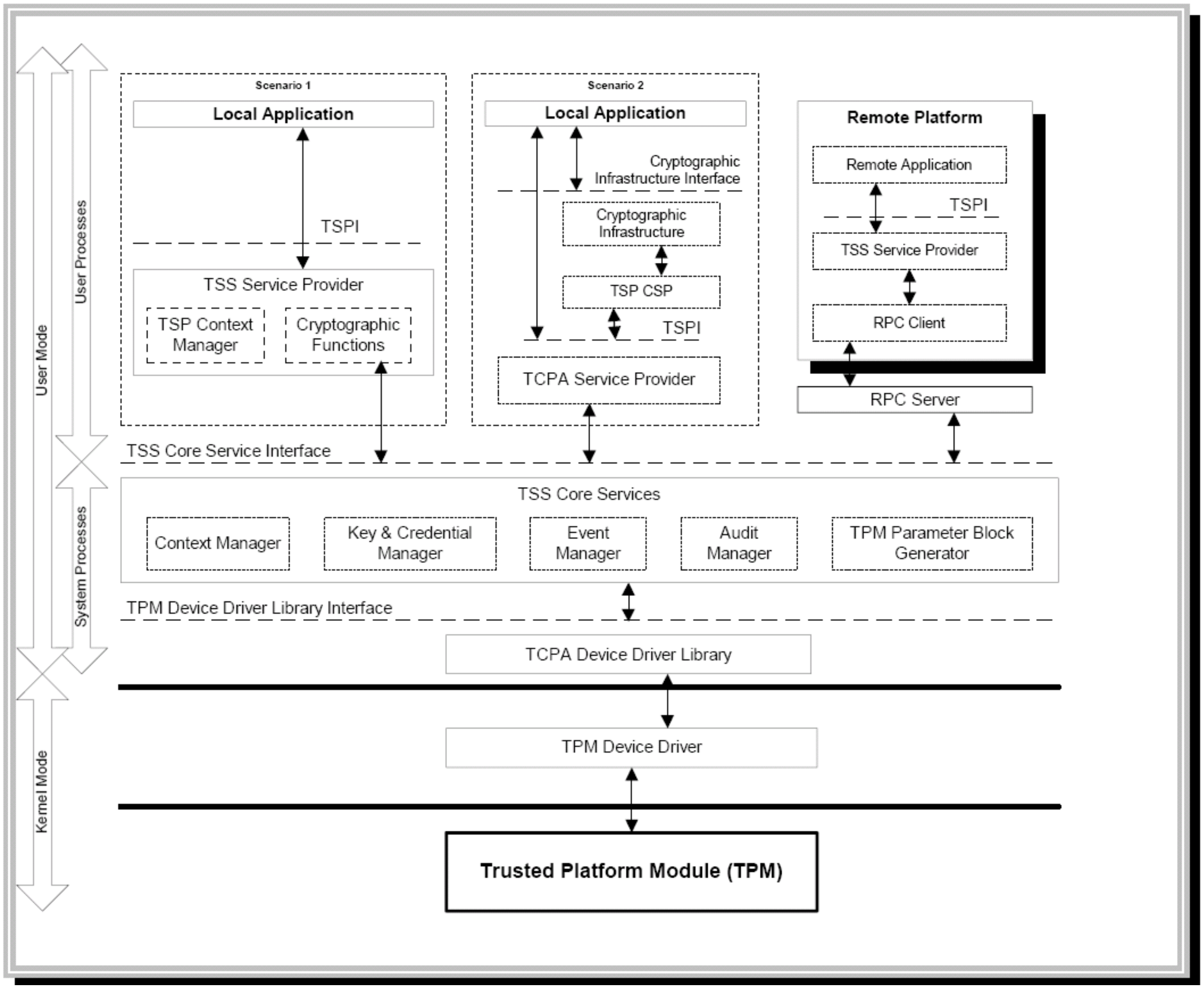


Figure 1-2 TSS Architecture Diagram

1.9.1 TCG Service Provider (TSP)

Start of informative comment:

This module provides TCG services for applications. It provides the high-level TCG functions allowing applications to focus on their specialty while relying on the TSP to perform most of the trusted functions provided by the TPM. This module also provides a small number of auxiliary functions for convenience not provided by the TPM such as signature verification.

In environments that provide layers of protections (i.e., rings) or separation of applications into processes, this module is intended to reside within the same ring and process as the application. There will likely be one TSP per application. On operating systems that provide multiple processes, there may be multiple instances of TSPs running on the platform.

End of informative comment.

A TSS implementation MUST provide the functionality of the TCG Service Provider (TSP). Depending on the implementation, the TSP MAY be a discreet module or MAY be integrated into other platform modules. This module MAY provide protected transfer of information or data between the application by residing within the same process as the application.

The TSP MUST provide the 'C' interface as defined in section 4 TCG Service Provider (TSP) of this specification. For the 'C' interface the TSP MUST provide dynamic linking to the application and MAY provide static linking to the application. If the TSP is implemented in a Microsoft® Windows® application, the TSS MAY, in addition, provide a COM interface.

1.9.1.1 TSP Interface (TSPI)

Start of informative comment:

The interface to the TSP is the TSP Interface (TSPI). This is an object oriented interface. It resides within the same process as the application. In some implementations it may be trusted with sensitive data such as authorization data to the same extent as the application itself. This may be required for functions where the application gathers the object's authentication data from the user and passes it to the TSP for processing into the command's authentication data format.

End of informative comment.

If a TSS provides a TSP, it MUST provide the required interfaces defined in this document.

1.9.1.2 TSP Context Manager (TSPCM)

Start of informative comment:

The TSP Context Manager (TSPCM) provides dynamic handles that allow for efficient usage of both the application's and TSP's resources. Each handle provides context for a set of interrelated TCG operations. Different threads within the application may share the same context or may acquire a separate context per thread.

End of informative comment.

If a TSS provides a TSP, it MUST provide the functions required to establish and maintain context for a set of interrelated functions.

1.9.1.3 TSP Cryptographic Functions (TSPCF)

Start of informative comment:

To make full use the TPM's protected functions, supporting cryptographic functions must be provided. It is not necessary for these supporting functions to be protected. Example functions are: Hashing algorithm and Byte-stream generator.

End of informative comment.

If a TSS provides a TSP, it MUST provide the cryptographic functions required as defined in section 4 Functional Overview.

1.9.2 TCG Core Services (TCS)

Start of informative comment:

A service provider is any component used by the application that allows that application access to the TCS (and thus the TPM) from within the application's process. Service providers, of which the TSP is but one possible instantiation, cannot communicate directly with the TPM. Additionally, there are multiple common services that either are either required to be shared or should be shared among the set of the platform's service providers.

The TCG Core Services (TCS) provides a common set of services per platform for all service providers. Since the TPM is not required to be multithreaded, it provides threaded access to the TPM.

End of informative comment.

Any service provider MUST be allowed to connect to and obtain services from the TCS. The TCS MUST not restrict access to only a TSP.

The TCS MUST provide single threaded access to the TPM.

There MUST be only one TCS per platform operating system.

1.9.2.1 TCS Interface (Tcsi)

Start of informative comment:

The interface to the TCS is the TCS Interface (Tcsi). This is a simple 'C' style interface. While it may allow multithreaded access to the TCS, each operation is intended to be atomic. In most environments it resides as a system process, separate from the application and service provider processes. If the environment provides for the TCS to reside in a system process, communication between the service providers and the TCS would be via an RPC. If a transport session is not used, the channel between the service provider and the TCS will be outside the protection of the process, so no unprotected data (e.g., raw authentication data) should be transferred directly to the TCS in this case.

End of informative comment.

If a TSS provides a TCS, it MUST provide the required interfaces defined in this document.

If the platform's environment provides, the TCS MUST function as a system service.

If the TCS functions as a system service, the TCS MUST NOT be implemented to allow the transfer of raw authentication data between the service provider and the TCS.

1.9.2.2 TCS Context Manager (TCSCM)

Start of informative comment:

The TCS Context Manager (TCSCM) provides dynamic handles that allow for efficient usage of both the service provider's and TCS's resources. Each handle provides context for a set of interrelated TCG operations. Different threads within the service provider may share the same context or may acquire a separate context per service provider.

End of informative comment.

The TCS MUST provide the functions required to establish and maintain context for a set of interrelated functions.

1.9.2.3 TCS Key & Credential Manager (TCSKCM)

Start of informative comment:

Keys and credentials may be associated with the platform, the user, or individual applications. In all cases it may be more convenient for an application to use a common resource to store and manage the keys and credentials. Keys and credentials associated with the platform (e.g., The Endorsement, Platform, and Conformance Credentials) should be stored and managed by the Key and Credential Manager to allow multiple applications access to them.

The Endorsement and Platform credentials contain information that identifies the specific platform thus these are considered privacy sensitive. Other credentials and keys may also contain privacy sensitive information. The TCSKM must, therefore, provide a mechanism to protect this information from unauthorized access.

End of informative comment.

The TCS MUST provide the functions required to perform Key and Credential Management as described in this document.

The TCSKM MUST provide a mechanism to protect privacy sensitive keys and credentials from unauthorized access.

1.9.2.4 TCS Event Manager (TCSEM)

Start of informative comment:

This component manages the TSS_PCR_EVENT structures and their associations with their respective PCR. Since these are associated with the platform and not the application, applications and service providers should retain only copies of these structures.

End of informative comment.

The TCS MUST provide the functions required to store, manage, and report the TSS_PCR_EVENT structures and their associated PCR indexes.

1.9.2.5 TCS TPM Parameter Block Generator (TcsipBG)

Start of informative comment:

Calls into the TCS are ‘C’-style functions. Communication to the TPM is via a byte-stream parameter block. This component converts the parameters passed into TCS into the byte-stream expected by the TPM.

End of informative comment.

The TCS MUST provide the functions required to convert input parameters into a TPM byte-stream.

1.9.3 TCG Device Driver Library (TDDL)

Start of informative comment:

The TCG Device Driver Library (TDDL) is an intermediate module that exists between the TCS and the kernel mode TPM Device Driver (TDD). The TDDL provides a user mode interface. Such an interface has several advantages over a kernel mode driver interface:

- It ensures different implementations of the TSS properly communicate with any TPM.
- It provides an OS-independent interface for TPM applications.
- It allows the TPM vendor to provide a software TPM simulator as a user mode component.

Because the TPM is not required to be multithreaded, the TDDL is to be a single-instance, single threaded module. The TDDL expects the TPM command serialization to be performed by the TCS. The exception to the single threaded nature of the TDDL is the Tddli_Cancel operation. The Tddli_Cancel allows the TCS to send an abort operation to the TPM.

The TPM vendor is responsible for defining the interface between the TDDL and the TDD. The TPM vendor can choose the communication and resource allocation mechanisms between this library and any kernel mode TPM device driver or software TPM simulator.

End of informative comment.

If the platform environments provides kernel and user mode separation, the TDDL MUST reside in the user mode.

The TDDL MUST only connect to the TCS.

TPM commands sent to the TDDL MUST be serialized.

The TDDL MUST provide the interface: Tddli_Cancel. This interface MUST function and attempt to send an abort command to the TPM even if a thread is currently busy sending a TPM command.

1.9.3.1 TDDL Interface (Tddli)

Start of informative comment:

The interface to the TDDL is the TDDL Interface (Tddli).

The Tddli is consists of three types of functions:

- Maintenance functions (Open, Close, GetStatus): maintains the communication with the TDD.
- Indirect Functions (GetCapability, SetCapability): gets/sets attributes of the TPM/TDD/TDDL.
- Direct functions (Transmit, Cancel): transmits/cancels transmission of commands to the TPM.

End of informative comment.

1.9.4 TPM Device Driver (TDD)

Start of informative comment:

The TCG Device Driver (TDD) is the kernel mode component that receives byte-streams from the TDDL and sends them to the TPM returning the responses back to the TDDL.

The TDD is TPM vendor and OS specific. It may provide additional functionality such as power management as required by the platform, OS, or operating environment.

End of informative comment.

The TDD MUST be the only platform component to interface with the TPM.

The TDD MUST provide power management functions if the platform's environment provides or requires it.

1.9.4.1 TDD Interface (TDDI)

Start of informative comment:

The interface between the TDD and TDDL is called the TDDI.

End of informative comment.

The TPM vendor is responsible for defining the TDDI. The vendor is also responsible for defining the interface between the TDD and the actual TPM device.

1.9.5 Remote Procedure Calls

The goal of the Remote Procedure Calling method, or RPC, defined in this specification is to provide a common way for various TSPs from various platforms to communicate to a TCS of a given system. As a platform and language agnostic technology, Web Services over SOAP provide such a goal. Some platforms, like handhelds or mobile phones, may not have the space or the desire to expose a TCS interface to other systems or even other local TSPs and that is a choice of the platform designer. For those systems that do wish to support a functional TCS interface running in a service accessible by compliant TSPs, the following guidelines should be adhered.

- The port number used shall be 30003
- SOAP1.1 format shall be used to support a greater number of toolkits.
- HTTP is the transport type
- By default, the TCS should not be open to remote system requests.

Provided with this specification is a Web Services Description Language file called tcs.wsdl. This file is a common way to describe the 1.2 functions such that a TSS programmer can use this file as a common way to generate TCS interface code. A 1.1 TSS programmer may choose to use this file to create a 1.1 TCS that uses the same protocol as a means to achieve a remote procdure call that will work with 1.2 software and with platforms that do no support DCOM if that is desired.

As memory allocation across this type of technology is undefined, the function Tcsi_FreeMemory shall be omitted.

Privacy and security by untrusted software, like the TSS, is not the goal of this specification. As such, the TSS will use open communication between the TSP and TCS and rely on the inherent security provided by the TPM in shared secret encryptions and transport sessions to keep the data safe. However, that doesn't mean the TSS should open up more privacy holes to do so. To address this concern, the TSS will attempt to do a simple check of allowed connections and allowed commands for the given context based on policy provided by the system administrator.

First, it should be assumed that all available commands are exposed locally.

Complete administration of the TCS is undefined by this specification and is left to the TSS vendor. Using the TCS API to administrate the TCS is not the proper way to perform administration, as the TCS interface is inherently untrusted. This specification will define a format in which a system administrator can describe policy in a file, but will not define how to apply the file to the software. That shall be left to the TSS vendor to determine. By defining a common way to describe policy in a file, an administrator of many systems, which also may have multiple TCS types on them, can have a common way to describe his policy to these systems.

Filtering that can be done for remote connections: The best way is to allow the owner to determine which commands (or groups of commands) can be run. Examples of all of these can be found in tcsadmin.xml.

1.9.5.1 Command Filtering

One level of administration involves what commands are allowed by a connected context. A TCS can choose to be stateful or stateless in this filtering.

1.9.5.2 Stateless

A stateless TCS is much easier to implement, but is more open to denial of service attacks or other improper usage of the TPM, however may suit the needs of the administrator if only a few commands are exposed. In a stateless TCS implementation, the administrator determines the full set of allowed functions by any remote connection that is allowed access to the machine. This is only recommended in the case where few basic functions are exposed, like Tcsi_GetCapability, or when the system is in a private LAN with 'trusted' clients.

1.9.5.3 Stateful

A TCS that monitors state is much more complicated, but provides the best prevention against DOS attacks and privacy holes. The assumption here is that the TCS was initially designed with remote capabilities to allow certain types of functions to be exposed from the TPM. One such function is the ability to attest another system. An administrator may choose to only allow certain types of 'services' to be exposed remotely and as such, the TCS shall monitor the activity of the context to make sure the caller is doing one of these service types.

To achieve this type of monitoring, the caller will declare which functions are required when connecting to the TCS. When the intent is declared, the TCS can keep a state of the connected context and determine for each request if the request is reasonable for the given state. For example, requesting an OSAP after a Quote doesn't sound reasonable. A remote caller in this case, should understand that the context will only be good for the desired service and when that is completed, the context is no longer usable.

Several examples can be found below. Many of these are more appropriate for a server implementation than a client implementation. This list is not expected in any sense to be complete. Control of these services will be done via an xml file. The example xml file for defining a stateful service is given in the document: tcsadmin.xml, which gives examples of how an administrator can define what services a TCS will respond to remotely. Example services (not all of which are in the xml file) that a user/server might want to expose are:

- Attestation
- Time Stamping
- GetCapabilities
- UnLock Remote File (UnBind or UnSeal)
- Backup
- Recover
- Migrate
- DAA Credential Issuance

The xml file contains two types of policies. The first is used to group command types. The second is to define a service. Both are definable by the administrator. The TSS implementation MUST have a means of applying the xml file, so that the policies defined in the xml file are followed.

1.9.6 Cryptographic Infrastructures

Start of informative comment:

There are several existing general purpose Cryptographic Infrastructure available to the industry. Examples include: CDSA, MS-CAPI, and PKCS #11. These interfaces usually provide standard underlying interfaces allowing modules that provide specific features or functions to be incorporated and used by the Cryptographic Infrastructure's applications. These custom modules are called service providers.

End of informative comment.

The TSS MAY be one or more of the following:

- A service provider for an existing Cryptographic Infrastructure. This can provide for either all or a subset of the functions required by the Cryptographic Infrastructure. The remaining functions provided by the TSS that do not fit within the scope of the Cryptographic Infrastructure can be provided between the application and the TSP.
- A stand-alone service providing no functions to an existing Cryptographic Infrastructure.

Note: While the TSS does not provide bulk, general purpose symmetric encryption/decryption functions symmetric encryption/decryption functions are required by some TPM and TSS operations. Encryption/decryption functions MAY be provided either internally or externally to the TSS.

2. Common Environment

2.1 Naming Conventions

Label	Any Value Appropriate to the Platform Addressing a Handle
TCS	When used as a prefix, this is a structure or definition that applies only the TCS layer.
Tcsi	TCS Interface Prefix where the function does **not** result in a call to the TPM. These few commands include such things as administrating a context, freeing memory, and asking the TCS layer what its capabilities are.
Tcsip	TCS Interface Prefix where the function **does** result in a call to the TPM
Tddli	TSS Device Driver Library Interface Prefix.
Tddi	TSS Device Driver Interface Prefix (To be defined in a future version of the TSS Specification).
Tspi	TSP Interface Prefix.
Tspicb	Application supplied callback functions for the TSP prefix
TSS	When used as a prefix, this is a structure or definition used in the TSP or other TSS layers.

2.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
HMAC	Hashed Message Authentication Code
OIAP	Object Independent Authorization Protocol
OSAP	Object Specific Authorization Protocol
PCR	Platform Configuration Register
TCG	Trusted Computing Group
TCS	TSS Core Services Module.
TDD	TSS Device Driver Module.
TDDL	TSS Device Driver Library Module.
TPM	Trusted Platform Module
TSP	TSS Service Provider
TSS	TCG Software Stack
VE	Validation Entity

2.3 Definitions

2.3.1 Data Types

TCG-enabled platforms are expected to be 32-bit or 64-bit systems. This section defines the basic data types on these systems. The section after the next section then contains data type declarations especially required at the TSPI. Data type declarations for the below definitions are not included in the TSS headers file. These are expected to be defined by the platform-specific SDK and header files for the target environment and platform. They MUST, however, conform to the specific definitions below since these are TPM specific definitions.

Pointer Size:

Pointer size becomes 32 bits on 32-bit systems and 64 bits with 64-bit system.

Basic Types:

There are some new types for 64-bit systems that were derived from the basic C-language integer and long types, so they work in existing code. These are the expected values and definitions.

Type	Definition
UINT16	Unsigned INT16
UINT32	Unsigned INT32.
UINT64	Unsigned INT64
BYTE	Unsigned character
TSS_BOOL	Signed character
TSS_UNICODE	TSS_UNICODE character. TSS_UNICODE characters are to be treated as an array of 16 bits.
PVOID	void Pointer (32 or 64 bit depending on architecture)

Boolean types

Name	Value	Description
TRUE	0x01	Assertion
FALSE	0x00	Negation

Derived Types

Type	Definition	Usage
TSS_FLAG	UINT32	Object attributes.
TSS_HOBJECT	UINT32	Basic object handle.(can be UINT32 or UINT64)
TSS_ALGORITHM_ID	UINT32	Type of TSS Algorithm IDs
TSS_MIGRATE_SCHEME	UINT16	Type of TSS Migration Scheme IDs
TSS_KEY_USAGE_ID	UINT32	Type of TSS Key Usage IDs
TSS_KEY_ENC_SCHEME	UINT16	Type of TSS Encryption Scheme IDs
TSS_KEY_SIG_SCHEME	UINT16	Type of TSS Signature Scheme IDs

TSS_EVENTTYPE	UINT32	Type of TSS event
TSS_COUNTER_ID	UINT32	Counter Identifier
TSS_RESULT	UINT32	Result of a TSP interface command

Object Types

Type	Definition	Usage
TSS_HCONTEXT	TSS_HOBJECT	Context object handle.
TSS_HPOLICY	TSS_HOBJECT	Policy object handle.
TSS_HTPM	TSS_HOBJECT	TPM object handle.
TSS_HKEY	TSS_HOBJECT	Key object handle.
TSS_HENCDATA	TSS_HOBJECT	Encrypted data object handle.
TSS_HPCRS	TSS_HOBJECT	PCR composite object handle.
TSS_HHASH	TSS_HOBJECT	Hash object handle.
TSS_HNVSTORE	TSS_HOBJECT	NVRAM data object handle.
TSS_HMIGDATA	TSS_HOBJECT	Migration data handling object handle.
TSS_HDELFAMILY	TSS_HOBJECT	Delegation family object handle.
TSS_HDAA_CREDENTIAL	TSS_HOBJECT	DAA credential object handle.
TSS_HDAA_ISSUER_KEY	TSS_HOBJECT	DAA Issuer Key object handle.
TSS_HDAA_ARA_KEY	TSS_HOBJECT	DAA Anonymous Revocation Authority Key object handle.

2.3.2 Defined Constants

2.3.2.1 Object Type Definitions

Start of informative comment:

Definition of object types that can be used with the method Tspi_Context_CreateObject( ).

End of informative comment.

The defined object types are based on the data type TSS_FLAG.

Object Type	Description
TSS_OBJECT_TYPE_POLICY	Policy object.
TSS_OBJECT_TYPE_RSAKEY	RSAKey object.
TSS_OBJECT_TYPE_ENCDATA	Encrypted data object; sealed data or bound data.
TSS_OBJECT_TYPE_PCRS	PCR composite object.
TSS_OBJECT_TYPE_HASH	Hash object.
TSS_OBJECT_TYPE_NV	Non Volatile RAM object.
TSS_OBJECT_TYPE_MIGDATA	CMK-Migration data object.
TSS_OBJECT_TYPE_DAA_CERTIFICATE	DAA certificate object.
TSS_OBJECT_TYPE_DAA_ISSUER_KEY	DAA Issuer Key object.
TSS_OBJECT_TYPE_DAA_ARA_K	DAA Anonymous Revocation Authority Key

EY	object.

2.3.2.2 Object Initialization Definitions

Start of informative comment:

Definition of object initialization flags that can be used with the method Tspi_Context_CreateObject( ).

End of informative comment.

The defined initialization flags are based on the data type TSS_FLAG.

InitFlag	Description
TSS_KEY_SIZE_DEFAULT	Default size (see remarks below)
TSS_KEY_SIZE_512	Key size 512.
TSS_KEY_SIZE_1024	Key size 1024.
TSS_KEY_SIZE_2048	Key size 2048.
TSS_KEY_SIZE_4096	Key size 4096.
TSS_KEY_SIZE_8192	Key size 8192.
TSS_KEY_SIZE_16384	Key size 16384.
TSS_KEY_TYPE_STORAGE	Key for wrapping keys.
TSS_KEY_TYPE_SIGNING	Key for signing operations.
TSS_KEY_TYPE_BIND	Binding Key.
TSS_KEY_TYPE_AUTHCHANGE	Ephemeral key used during the ChangeAuthAsym process only
TSS_KEY_TYPE_IDENTITY	Key for an identity.
TSS_KEY_TYPE_LEGACY	Key that can perform signing and binding.
TSS_KEY_TYPE_AUTHCHANGE	An ephemeral key used to change authorization value
TSS_KEY_NON_VOLATILE	Key is non-volatile. MAY be unloaded at startup
TSS_KEY_VOLATILE	Key is volatile. MUST be unloaded at startup
TSS_KEY_NOT_MIGRATABLE	Key is not migratable (DEFAULT).
TSS_KEY_MIGRATABLE	Key is migratable.
TSS_KEY_CERTIFIED_MIGRATABLE	Key is certified migratable.
TSS_KEY_TYPE_MIGRATE	Key to be used as a migration authority (MA) in Tspi_Key_MigrateKey
TSS_KEY_NOT_CERTIFIED MIGRATABLE	Key is not certified migratable.
TSS_KEY_NO_AUTHORIZATION	Key needs no authorization (DEFAULT).
TSS_KEY_AUTHORIZATION	Key needs authorization.
TSS_KEY_AUTHORIZATION_PRIV_USE_ONLY	Key needs authorization for use of private portion of key.
TSS_KEY_STRUCT_DEFAULT	Key object uses a 1.1 TCPA_KEY or 1.2 TCPA_KEY12 structure based on the Context's TSS_TSPATTRIB_CONTEXT_VERSION_MODE attribute (DEFAULT).
TSS_KEY_STRUCT_KEY	Key object uses a 1.1 TCPA_KEY structure

TSS_KEY_STRUCT_KEY12	Key object uses a 1.2 TCPA_KEY12 structure
TSS_KEY_EMPTY_KEY	no TCG key template (empty TSP key object)
TSS_KEY_TSP_SRK	use a TCG SRK template (TSP key object for SRK)
TSS_ENCDATA_SEAL	Data object is used for seal operation.
TSS_ENCDATA_BIND	Data object is used for bind operation.
TSS_ENCDATA_LEGACY	Data for legacy bind operation.
TSS_HASH_DEFAULT	Default hash algorithm
TSS_HASH_SHA1	Hash object with algorithm SHA1
TSS_HASH_OTHER	Hash object with other algorithm
TSS_POLICY_USAGE	Policy object used for authorization
TSS_POLICY_MIGRATION	Policy object used for migration
TSS_POLICY_OPERATOR	Policy object used for operator authorization
TSS_PCRS_STRUCT_DEFAULT	PcrComposite object uses a 1.1 TCPA_PCR_INFO structure or a 1.2 TCPA_PCR_INFO_LONG structure based on the Context's TSS_TSPATTRIB_CONTEXT_VERSION_MODE attribute (DEFAULT).
TSS_PCRS_STRUCT_INFO	PcrComposite object uses a 1.1 TCPA_PCR_INFO structure (DEFAULT)
TSS_PCRS_STRUCT_INFO_LONG	PcrComposite object uses a 1.2 TCPA_PCR_INFO_LONG structure
TSS_PCRS_STRUCT_INFO_SHORT	PcrComposite object uses a 1.2 TCPA_PCR_INFO_SHORT structure

Remarks:

TSS_KEY_SIZE_DEFAULT: This is not a fix defined key size. The key size is internally queried from the TSS Core Service running on the TCG system.

2.3.2.3 Attribute Definitions for a Context Object

Attribute flags.

TSS_TSPATTRIB_CONTEXT_SILENT_MODE	Get/set the silent mode of a context object
TSS_TSPATTRIB_CONTEXT_MACHINE_NAME	Get the machine name of the TSS given as a zero terminated TSS_UNICODE string the context object is connected with.
TSS_TSPATTRIB_CONTEXT_VERSION_MODE	Get/Set the version handling mode of a context object.
TSS_TSPATTRIB_CONTEXT_CONNECTION_VERSION	Get the highest supported version of the

	connection (the highest common version of the TCS and TPM)
TSS_TSPATTRIB_CONTEXT_TRANSPORT	Get/Set attributes related to the transport session associated with this context object.
TSS_TSPATTRIB_SECRET_HASH_MODE	Get/Set the string hash operation handling selection of a context or policy object

Attribute subflags

TSS_TSPATTRIB_CONTEXTTRANS_CONTROL	Enable/disable the transport session.
TSS_TSPATTRIB_CONTEXTTRANS_MODE	Control properties of the transport session.
TSS_TSPATTRIB_SECRET_HASH_MODE_POPUP	Get/Set the behavior for hashing in popup mode.

Attribute values

TSS_TSPATTRIB_CONTEXT_NOT_SILENT	TSP dialogs are shown asking a user for a secret (Default).
TSS_TSPATTRIB_CONTEXT_SILENT	TSP dialogs are not shown.
TSS_TSPATTRIB_CONTEXT_VERSION_V1_1	TSP sets the default version of the context to be 1.1 for the connection in this context object (Default). Since this is the default, it only needs to be used if the application has previously used either TSS_TSPATTRIB_CONTEXT_VERSION_V1_2 or TSS_TSPATTRIB_CONTEXT_VERSION_AUTO to change the default of the context to something else
TSS_TSPATTRIB_CONTEXT_VERSION_V1_2	TSP switches the default version to 1.2 for the connection in this context object. Objects created after this will default to being 1.2 objects. (PCR objects will default to being LONG, and if anything different is wanted, it will have to be specified at object creation as to which type of 1.2 object they are.)
TSS_TSPATTRIB_CONTEXT_VERSION_AUTO	The TSP detects the underlying main version and sets the default for the context object

	to be the highest level consistent with both the TCS and the TPM. This avoids the necessity of software figuring this out on its own. If set to 1.2, check TSS_TSPATTRIB_CONTEXT_VERSION_V1_2 for the behavior of PCR objects.
TSS CONNECTION_VERSION_1_1	Indicates that the connection supports only the functionality defined in the 1.1 TSS specification due to the version of the TCS, TPM, or both.
TSS CONNECTION_VERSION_1_2	Indicates that the connection supports only the functionality defined in the 1.2 TSS specification due to the version of the TCS, TPM, or both.
TSS_TSPATTRIB_TRANSPORT_NO_DEFAULT_ENCRYPTION	Disable the encryption of data within the transport session.
TSS_TSPATTRIB_TRANSPORT_DEFAULT_ENCRYPTION	Enable the encryption of data within the transport session.
TSS_TSPATTRIB_TRANSPORT_AUTHENTIC_CHANNEL	Enable logging of the transport session.
TSS_TSPATTRIB_TRANSPORT_EXCLUSIVE	

Remarks:

TSS_TSPATTRIB_CONTEXT_SILENT is intended for use with applications for which the UI should not be displayed by the TSP. The application can request that the TSS Service Provider not display any user interface (UI) for this context by setting this attribute to TSS_TSPATTRIB_CONTEXT_SILENT. If this is done and the TSP must display a UI to operate, the call fails and the TSS_E_SILENT_CONTEXT error code is returned.

Getting the TSS_TSPATTRIB_CONTEXT_CONNECTION_VERSION attribute is only available for an open connection (remote or local). If there is no connection, TSS_E_NO_CONNECTION is returned.

The TSS_TSPATTRIB_CONTEXT_VERSION_MODE attribute controls the selection of the default structure type for objects within this context whose TPM structures have changed between versions 1.1b and 1.2 (e.g. the TPM_KEY vs. TPM_KEY12). The default VERSION_MODE is v1.1. At the time Tspi_Context_CreateObject is called, the TSP checks the object initialization flags to see if the caller has requested a specific TPM structure for the object. If the caller has not requested a specific structure a default structure type is selected based on the TSS_TSPATTRIB_CONTEXT_VERSION_MODE. For keys, v1.1 results in a TPM_KEY structure and v1.2 results in a TPM_KEY12 structure. For PCR objects, v1.1 results in a TPM_PCR_INFO structure and v1.2 results in a TPM_PCR_INFO_LONG structure. No other object types use structure versioning. The structure version is determined at the time of the call to Tspi_Context_CreateObject, so changing the TSS_TSPATTRIB_CONTEXT_VERSION_MODE after a Key object is created will not affect the Key object. Changing the structure version of an existing object requires modification of one of the object's attribute, and must be done on a per-object basis.

The attribute values TSS_TSPATTRIB_TRANSPORT_NO_DEFAULT_ENCRYPTION, TSS_TSPATTRIB_TRANSPORT_DEFAULT_ENCRYPTION, TSS_TSPATTRIB_TRANSPORT_AUTHENTIC_CHANNEL, and TSS_TSPATTRIB_TRANSPORT_EXCLUSIVE are bitmasks which may be logically ORed together to form the attribute value for the TSS_TSPATTRIB_CONTEXTTRANS_MODE subflag.

2.3.2.4 Attribute Definitions for a TPM Object

Attribute flags.

TSS_TSPATTRIB_TPM_CALLBACK_COLLATEIDENTITY	Get/Set the the address of the callback function to be used
TSS_TSPATTRIB_TPM_CALLBACK_ACTIVATEIDENTITY	Get/Set the the address of the callback function to be used
TSS_TSPATTRIB_TPMCAP_SET_VENDOR	This area allows the vendor to set specific areas in the TPM according to the normal shielded location requirements
TSS_TSPATTRIB_TPM_CREDENTIAL	Get/Set a TPM Credential
TSS_TSPATTRIB_TPM_ORDINAL_AUDIT_STATUS	Add/Clear an ordinal to/from the audit list.

Attribute subflags

Flag	***SubFlag***	***Attribute***	***Description***
TSS_TSPATTRIB_TPM_CREDENTIAL	TSS_TPMATTRIB_EKCERT	The Endorsement credential data blob	The endorsement credential information set
	TSS_TPMATTRIB_TPM_CC	The TPM conformance credential data blob	The TPM conformance credential data set
	TSS_TPMATTRIB_PLATFORMCERT	Platform credential data blob	The platform credential data set

Flag	SubFlag	Attribute	Description
	TSS_TPMATTRIB_PLATFORM_CC	Platform conformance credential data blob	The platform credential dat set
TSS_TSPATTRIB_TPM_ORDINAL_AUDIT_STATUS	TPM_CAP_PROP_TPM_SET_ORDINAL_AUDIT	TPM_COMMAND_CODE: The ordinal to add/clear to/from this audit list	Adds an ordinal to the audit list.
	TPM_CAP_PROP_TPM_CLEAR_ORDINAL_AUDIT	TPM_COMMAND_CODE: The ordinal to add/clear to/from this audit list	Clears an ordinal from the audit list.

Remarks: For the identity processing (i.e. CollateIdentityRequest) the TPM_Object should have access to the credential data set containing the endorsement, platform and conformance credential. This information can have different enrollment, lifetime and storage requirements. The TPM v1.2 introduces the capability that these data MAY be stored in the NV-Area but it is also possible that only some or none of the credentials will be available there, and other will be available from the system environment. The TSP-TPM-Object offers the ability to use externally provided data as well as information form the TSS_Stack (e.g. TPM-Device) for the identity processing.

2.3.2.5 Attribute Definitions for a Policy Object

Attribute flags.

TSS_TSPATTRIB_POLICY_CALLBACK_HMAC	Get/Set the address of the callback function to be used.
TSS_TSPATTRIB_POLICY_CALLBACK_XOR_ENC	Get/Set the address of the callback function to be used.
TSS_TSPATTRIB_POLICY_CALLBACK_TAKEOWNERSHIP	Get/Set the the address of the callback function to be used.
TSS_TSPATTRIB_POLICY_CALLBACK_CHANGEAUHTASYM	Get/Set the the address of the callback function to be used.
TSS_TSPATTRIB_POLICY_SECRET_LIFETIME	Get/Set the lifetime of a secret.
TSS_TSPATTRIB_POLICY_POPUPSTRING	Set a NULL terminated TSS_UNICODE string which is displayed in the TSP policy popup dialog.
TSS_TSPATTRIB_POLICY_DELEGATION_INFO	Get/set information

	about a delegation blob.
TSS_TSPATTRIB_POLICY_DELEGATION_PCR	Get/set PCR fields from a delegation blob.
TSS_TSPATTRIB_SECRET_HASH_MODE	Get/Set the string hash operation handling selection of a context or policy object
TSS_TSPATTRIB_POLICY_CALLBACK_SEALX_MASK	Get/Set the address of a callback function to be used to mask the data input to a Sealx command

Attribute subflags

TSS_TSPATTRIB_POLSECRET_LIFETIME_ALWAYS	Secret will not be invalidated.
TSS_TSPATTRIB_POLSECRET_LIFETIME_COUNTER	Secret may be used n-times.
TSS_TSPATTRIB_POLSECRET_LIFETIME_TIMER	Secret will be valid for n seconds.
TSS_TSPATTRIB_POLDEL_TYPE	The delegation type (key or owner)
TSS_TSPATTRIB_POLDEL_INDEX	The index into the TPM delegation table
TSS_TSPATTRIB_POLDEL_PER1	The per1 field of the delegation blob
TSS_TSPATTRIB_POLDEL_PER2	The per2 field of the delegation blob.
TSS_TSPATTRIB_POLDEL_LABEL	The label of the delegation.
TSS_TSPATTRIB_POLDEL_FAMILYID	The family id of the delegation.
TSS_TSPATTRIB_POLDEL_VERCOUNT	The version count of the delegation.
TSS_TSPATTRIB_POLDEL_OWNERBLOB	The entire delegation blob for owner delegations.
TSS_TSPATTRIB_POLDEL_KEYBLOB	The entire delegation blob for key delegations.
TSS_TSPATTRIB_POLDELPCR_LOCALITY	The locality restrictions of the delegation.
TSS_TSPATTRIB_POLDELPCR_DIGESTATRELEASE	The digestAtRelease from the PCR restrictions of the delegation.
TSS_TSPATTRIB_POLDELPCR_SELECTION	The PCRSelection from the PCR restrictions of the delegation.
TSS_TSPATTRIB_SECRET_HASH_MODE_POPUP	Get/Set the behavior for hashing in popup mode.

Attribute values:

TSS_DELEGATIONTYPE_NONE	Indicates the policy is not configured for delegation.
TSS_DELEGATIONTYPE_OWNER	Indicates the policy is configured for owner delegation.
TSS_DELEGATIONTYPE_KEY	Indicates the policy is configured for key delegation.

Remarks:

The application can request that the TSS Service Provider implements the handling for a particular mode by selecting the mode at the Context-Object. Policy objects generated at the TSP inherits the info from the context object.

The selection is dynamically this means the application is able to change the attribute at the same context/policy on the fly or can open a different context/policy with separate settings.

2.3.2.6 Attribute Definitions for a Key Object

Attribute flags.

TSS_TSPATTRIB_KEY_REGISTER	Get the persistent storage the key is registered in
TSS_TSPATTRIB_KEY_BLOB	Get/Set a key blob
TSS_TSPATTRIB_KEY_INFO	Get key information
TSS_TSPATTRIB_KEY_UUID	Get TSS_UUID structure containing the UUID the key is assigned to.
TSS_TSPATTRIB_KEY_PCR	Get PCR information the key is sealed to(for keys using TSS_KEY_STRUCT_KEY)
TSS_TSPATTRIB_KEY_PCR_LONG	Get PCR information the key is sealed to (for keys using TSS_KEY_STRUCT_KEY12)
TSS_TSPATTRIB_KEY_CONTROLBIT	Get loaded key attribute
TSS_TSPATTRIB_RSAKEY_INFO	Get exponent/modulus info from a RSA key
TSS_TSPATTRIB_CMK_INFO	Get/Set MA data for CMK keys

Attribute subflags

TSS_TSPATTRIB_KEYREGISTER_USER	Key is registered in the persistent storage of TSP.
TSS_TSPATTRIB_KEYREGISTER_SYSTEM	Key is registered in persistent storage of TCS.

TSS_TSPATTRIB_KEYREGISTER_NO	Key is not registered in persistent storage.
TSS_TSPATTRIB_KEYBLOB_BLOB	Key information as a key blob.
TSS_TSPATTRIB_KEYBLOB_PUBLIC_KEY	Public key information as public key blob.
TSS_TSPATTRIB_KEYBLOB_PRIVATE_KEY	Encrypted private key information as private key blob.
TSS_TSPATTRIB_KEYINFO_SIZE	Key size in bits
TSS_TSPATTRIB_KEYINFO_USAGE	Key usage info
TSS_TSPATTRIB_KEYINFO_KEYFLAGS	Key flags
TSS_TSPATTRIB_KEYINFO_AUTHUSAGE	Key auth usage info
TSS_TSPATTRIB_KEYINFO_ALGORITHM	Key algorithm ID
TSS_TSPATTRIB_KEYINFO_SIGSCHEME	Key sig scheme
TSS_TSPATTRIB_KEYINFO_ENCSCHEME	key enc scheme
TSS_TSPATTRIB_KEYINFO_MIGRATABLE	if true then key is migratable
TSS_TSPATTRIB_KEYINFO_CMK	If true, then key is certified migratable
TSS_TSPATTRIB_KEYINFO_REDIRECTED	key is redirected
TSS_TSPATTRIB_KEYINFO_VOLATILE	if true key is volatile
TSS_TSPATTRIB_KEYINFO_AUTHDATAUSAGE	if true authorization is required
TSS_TSPATTRIB_KEYINFO_VERSION	version info as TSS version struct
TSS_TSPATTRIB_KEYINFO_RSA_EXPONENT	Exponent of the key
TSS_TSPATTRIB_KEYINFO_RSA_MODULUS	Modulus of the key
TSS_TSPATTRIB_KEYINFO_RSA_PRIMES	Primes of the key
TSS_TSPATTRIB_KEYINFO_RSA_KEYSIZE	Size of the key
TSS_TSPATTRIB_KEYINFO_KEYSTRUCT	Type of key structure used
TSS_TSPATTRIB_KEYPCR_DIGEST_ATCREATION	Get composite digest value of the PCR values, at the time when the sealing was performed.
TSS_TSPATTRIB_KEYPCR_DIGEST_ATRELEASE	This is the digest of the PCR value to verify when revealing sealed data.
TSS_TSPATTRIB_KEYPCR_SELECTION	This is the selection of PCRs to which the key is bound.
TSS_TSPATTRIB_KEYPCRLONG_LOCALITY_ATCREATION	The locality modifier when the blob was created.
TSS_TSPATTRIB_KEYPCRLONG_LOCALITY_ATRELEASE	The locality modifier required for using the key.
TSS_TSPATTRIB_KEYPCRLONG_CREATION_SELECTION	The selection of PCRs active when the blob was created.
TSS_TSPATTRIB_KEYPCRLONG_RELEASE_SELECTION	The selection of PCRs required for using the key.
TSS_TSPATTRIB_KEYPCRLONG_DIGEST_ATCREATION	The digest of the PCRs corresponding to the creation PCR selection
TSS_TSPATTRIB_KEYCONTROL_OWNEREVICT	Get current status of owner evict flag
TSS_TSPATTRIB_KEYPCRLONG_DIGEST_ATR	The digest of the PCRs

ELEASE	corresponding ot the release PCR selection necessary for use of the key
TSS_TSPATTRIB_CMK_INFO_MA_APPROVAL	HMAC of the migration authority approval
TSS_TSPATTRIB_CMK_INFO_MA_DIGEST	Migration authority digest data

2.3.2.7 Attribute Definitions for a Data Object

Attribute flags.

TSS_TSPATTRIB_ENCDATA_BLOB	Get/Set a data blob
TSS_TSPATTRIB_ENCDATA_PCR	Get PCR information the data is sealed to (for encdata objects using TSS_PCRS_STRUCT_INFO)
TSS_TSPATTRIB_ENCDATA_PCR_LONG	Get PCR information the data is sealed to (for encdata objects using TSS_PCRS_STRUCT_INFO_LONG)
TSS_TSPATTRIB_ENCDATA_SEAL	Get/Set parameters for the Seal operation.

Attribute subflags

TSS_TSPATTRIB_ENCDATABLOB_BLOB	Data blob that represents the encrypted data depending on its type (seal or bind).
TSS_TSPATTRIB_ENCDATAPCR_DIGEST_ATCREATION	Get composite digest value of the PCR values, at the time when the sealing was performed.
TSS_TSPATTRIB_ENCDATAPCR_DIGEST_ATRELEASE	Get the composite digest of the PCRs selected for the time the unsealing is to be performed.
TSS_TSPATTRIB_ENCDATAPCR_SELECTION	Get the bit map indicating the active PCRs.
TSS_TSPATTRIB_ENCDATAPCRLONG_LOCALITY_ATCREATION	Get the locality value at the time the sealing was performed.
TSS_TSPATTRIB_ENCDATAPCRLONG_LOCALITY_ATRELEASE	Get the locality value for the time the unsealing is to be performed.
TSS_TSPATTRIB_ENCDATAPCRLONG_CREATION_SELECTION	Get the bit map indicating the active PCRs at the time the sealing was performed.
TSS_TSPATTRIB_ENCDATAPCRLONG_RELEASE_SELECTION	Get the bit map indicating the active PCRs for the time the unsealing is to be performed.
TSS_TSPATTRIB_ENCDATAPCRLONG_DIGEST_ATCREATION	Get the composite digest of the PCRs selected at the time the sealing was performed.
TSS_TSPATTRIB_ENCDATAPCRLONG_DIGEST_ATRELEASE	Get the composite digest of the PCRs selected for the time the unsealing is to be performed.
TSS_TSPATTRIB_ENCDATASEAL_PROT	Configure whether the TPM Seal or

ECT_MODE	Sealx command will be used. The Sealx command protects the plaintext while it is in transit to the TPM, but is not available on 1.1 TPMs.

2.3.2.8 Attribute definitions for NV objects

Attribute Flags

TSS_TSPATTRIB_NV_INDEX	The index value of the requested area of NV Storage.
TSS_TSPATTRIB_NV _PERMISSIONS	The permissions area
TSS_TSPATTRIB_NV_STATE	Gets the various states of the NV Storage area. Only available if the NV space has been defined.
TSS_TSPATTRIB_NV_DATASIZE	Size of the defined NV storage area.
TSS_TSPATTRIB_NV_PCR	PCR restrictions of the NV area

Sub-Flags

TSS_TSPATTRIB_NVSTATE_READSTCLEAR	Set to FALSE on each TPM_Startup(ST_Clear) and set to TRUE after a ReadValuexxx with datasize of 0
TSS_TSPATTRIB_NVSTATE_WRITESTCLEAR	Set to FALSE on each TPM_Startup(ST_CLEAR) and set to TRUE after a WriteValuexxx with a datasize of 0
TSS_TSPATTRIB_NVSTATE_WRITEDEFINE	Set to FALSE after TPM_NV_DefineSpace and set to TRUE after a successful WriteValue with a datasize of 0
TSS_TSPATTRIB_NVPCR_READPCRSELECTION	The PCR selection mask for the PCR read restrictions of the NV space
TSS_TSPATTRIB_NVPCR_READDIGESTATRELEASE	The digestAtRelease for the PCR read restrictions of the NV space
TSS_TSPATTRIB_NVPCR_READLOCALITYATRELEASE	The locality mask for the PCR read restrictions of the NV space
TSS_TSPATTRIB_NVPCR_WRITEPCRSELECTION	The PCR selection mask for the PCR write restrictions of the NV space
TSS_TSPATTRIB_NVPCR_WRITEDIGESTATRELEASE	The digestAtRelease for the PCR write restrictions of the NV space
TSS_TSPATTRIB_NVPCR_WRITELOCALITYATRELEASE	The locality mask for the PCR write restrictions of the NV space

NV Constants

NV Index Domain Bits	Description
TSS_NV_TPM	Value “T”. This index is the TPM Manufacturer reserved bit. 0 indicates TCG defined value 1 indicates a TPM manufacturer specific value.
TSS_NV_PLATFORM	Value “P”. This index is the platform manufacturer reserved bit. 1 indicates

	that the index controlled by the platform manufacturer.
TSS_NV_USER	Value "U". This index is the platform user. 1 indicates that the index controlled by the platform user.
TSS_NV_DEFINED	Value "D" This index is defined. 1 indicates that the index is permanently defined and that any defineSpace operation will fail.

NV Index Masks	Description
TSS_NV_MASK_DOMAIN_BITS	Bits having the value 1 are used for Index Domain
TSS_NV_MASK_RESERVED	Reserved bit.
TSS_NV_MASK_PURVIEW	These bits are used as the purview.
TSS_NV_MASK_INDEX	These bits are used for the index.

NV Required Indexes	Description
TSS_NV_INDEX_LOCK	Size for this MUST be 0. This value turns on the NV authorization protections. Once executed all NV areas us the protections as defined. This value never resets
TSS_NV_INDEX0	Size for this MUST be 0. This value allows for the setting of the persistent lock bit which is only reset on TPM_Startup(ST_Clear)
TSS_NV_INDEX_DIR	This index points to the deprecated DIR command area from 1.1.

NV Reserved Indexes	Description
TPM_NV_INDEX_EKCert	The Endorsement credential
TPM_NV_INDEX_TPM_CC	The TPM Conformance credential
TPM_NV_INDEX_PlatformCert	The platform credential
TPM_NV_INDEX_Platform_CC	The Platform conformance credential
TPM_NV_INDEX_Sessions	Array containing the number of sessions allocated for each locality
TPM_NV_INDEX_TSS	Reserved for TSS
TPM_NV_INDEX_PC	Reserved for PC
TPM_NV_INDEX_SERVER	Reserved for Server
TPM_NV_INDEX_MOBILE	Reserved for Mobile
TPM_NV_INDEX_PERIPHERAL	Reserved for Peripheral
TPM_NV_INDEX_GPIO_xx	Reserved for GPIOs
TPM_NV_INDEX_GROUP_RESV	Reserved

NV Permissions	Description
TPM_NV_PER_READ_STCLEAR	The value can only be read once per TPM_Startup(ST_Clear) cycle. Lock set by a read with a datasize of 0

TPM_NV_PER_AUTHREAD	The value requires authorization to read
TPM_NV_PER_OWNERREAD	The value requires TPM Owner authorization to read.
TPM_NV_PER_PPREAD	The value requires physical presence to read
TPM_NV_PER_GLOBALLOCK	The value is writeable until a write to index 0 is successful. The lock of this attribute is reset by TPM_Startup(ST_CLEAR). Lock held by SV -> bGlobalLock
TPM_NV_PER_WRITE_STCLEAR	The value is writeable until a write to the specified index with a datasize of 0 is successful. The lock of this attribute is reset by TPM_Startup(ST_CLEAR). Lock held for each area in bWriteSTClear
TPM_NV_PER_WRITEDEFINE	The value can only be written once after performing the TPM_NV_DefineSpace command. Lock held for each area as bWriteDefine. Lock set by writing to the index with a datasize of 0
TPM_NV_PER_WRITEALL	The value must be written in a single operation
TPM_NV_PER_AUTHWRITE	The value requires authorization to write
TPM_NV_PER_OWNERWRITE	The value requires TPM Owner authorization to write
TPM_NV_PER_PPWRITE	The value requires physical presence to write

2.3.2.9 Attribute definitions for MigData objects

Attribute Flags for Set- and GetAttribData

Flag	SubFlag	Data Description
TSS_MIGATTRIB_MIGRATION BLOB	TSS_MIGATTRIB_MIGRATION_XOR_BLOB	Output data packet from CreateBlob operation.
	TSS_MIGATTRIB_MIGRATION_REWRAPPED_BLOB	Data format after MigrateKey operation.
	TSS_MIGATRIB_MIG_MSALIST_PUBKEY_BLOB	Public key information from a migration authrity as a key blob
	TSS_MIGATTRIB_MIG_AUTHORITY_PUBKEY_BLOB	Public key belonging to migration authority
	TSS_MIGATTRIB_MIG_DESTINATION_PUBKEY_BLOB	Approved destination public key
	TSS_MIGATTRIB_MIG_SOURCE_PUBKEY_BLOB	Public key to be migrated
TSS_MIGATTRIB_MIGRATION TICKET	0	Accesses the migration ticket data from the authorize migration key process.
TSS_MIGATTRIB_AUTHORITY_DATA	TSS_MIGATTRIB_AUTHORITY_DIGEST	Digest of the selected migration selection authorities.
	TSS_MIGATTRIB_AUTHORITY_APPROVAL_HMAC	Approved migration authority ticket data.
	TSS_MIGATTRIB_AUTHORITY_MSALIST_DIGEST	Digest of the public key belonging to a migration authority.
TSS_MIGATTRIB_MIG_AUTH_DATA	TSS_MIGATTRIB_MIG_AUTH_AUTHORITY_DIGEST	Digest (public key) of the selected migration selection authorities.
	TSS_MIGATTRIB_MIG_AUTH_DESTINATION_DIGEST	Digest of a public key for the approved destination.
	TSS_MIGATTRIB_MIG_AUTH_SOURCE_DIGEST	Digest of a public key for the key to be migrated.
TSS_MIGATTRIB_TICKET_DATA	TSS_MIGATTRIB_TICKET_SIG_DIGEST	Data portion do be verfied that signature is valid.
	TSS_MIGATTRIB_TICKET_SIG	Signature value do be

	_VALUE	verified.
	TSS_MIGATTRIB_TICKET_SIG_TICKET	Signature ticket to prove the creation on a specific TPM.
	TSS_MIGATTRIB_TICKET_RESTRICT_TICKET	Containing the digests of public keys belonging to the migration authority
TSS_MIGATTRIB_PAYLOAD_TYPE	TSS_MIGATTRIB_PT_MIGRATE_RESTRICTED	Key was created locally
	TSS_MIGATTRIB_PT_MIGRATE_EXTERNAL	Key was migrated here

2.3.2.10 Attribute Definitions for Hash Objects

Attribute flags.

TSS_TSPATTRIB_HASH_IDENTIFIER	Deprecated. Use TSS_TSPATTRIB_ALG_IDENTIFIER instead.
TSS_TSPATTRIB_ALG_IDENTIFIER	Get/Set the AlgorithmIdentifier as defined in the PKCS#1v2.1 standard.

2.3.2.11 Attribute Definitions for a PcrComposite Object

Attribute flags.

TSS_TSPATTRIB_PCRS_INFO	Get/Set PcrComposite object information

Attribute subflags

TSS_TSPATTRIB_PCRSINFO_PCRSTRUCT	Type of PCR structure used

2.3.2.12 Attribute Definitions for DelFamily Objects

Attribute flags.

TSS_TSPATTRIB_DELFAMILY_STATE	Get/Set dynamic state information for a DelFamily object
TSS_TSPATTRIB_DELFAMILY_INFO	Get/Set static information about a DelFamily object

Attribute subflags

TSS_TSPATTRIB_DELFAMILYSTATE_LOCKED	Get/Set boolean indicating whether the family table entry is locked
TSS_TSPATTRIB_DELFAMILYSTATE_ENABLED	Get/Set boolean indicating whether the delegation family is enabled.
TSS_TSPATTRIB_DELFAMILYINFO_LABEL	Get/Set the delegation family label.
TSS_TSPATTRIB_DELFAMILYINFO_VERCOUNT	Get the delegation family version count.
TSS_TSPATTRIB_DELFAMILY_FAMILYID	Get/Set the TPM_FAMILY_ID for the delegation family.

2.3.2.13 Attribute Definitions for DAA Credential Objects

Attribute flags

TSS_TSPATTRIB_DAACRED_COMMIT	Get/Set attributes that apply to attribute commitments

TSS_TSPATTRIB_DAACRED_ATTRIB_GAMMAS	Get/Set gammas for credential attributes
TSS_TSPATTRIB_DAACRED_CREDENTIAL_BLOB	Get/Set the DAA credential.
TSS_TSPATTRIB_DAACRED_CALLBACK_SIGN	Get/Set the address of the callback function to be used.
TSS_TSPATTRIB_DAACRED_CALLBACK_VERIFYSIGNATURE	Get/Set the address of the callback function to be used.

Attribute subflags

TSS_TSPATTRIB_DAACOMMIT_NUMBER	Get the number of commitments to selective attributes of or for the DAA Credential that will be used during the DAA Join or DAA Sign protocol. If zero (default) no commitments will be used.
TSS_TSPATTRIB_DAACOMMIT_SELECTION	Get/Set a selection of attributes that will have commitments for a DAA Credential. The data type is TSS_DAA_SELECTED_ATTRIB.
TSS_TSPATTRIB_DAACOMMIT_COMMITMENTS	Get/Set an array of commitments on selected attributes of the DAA Credential. The length of the array is equal to TSS_TSPATTRIB_DAACOMMIT_NUMBER. The data type of a single commitment is TSS_DAA_ATTRIB_COMMIT. Note, that this one includes the randomness to open the commitment.
TSS_TSPATTRIB_DAAATTRIBGAMMAS_BLOB	Get/Set the gammas for the attributes for a credential.

2.3.2.14 Attribute Definitions for DAA Issuer Key Objects

Attribute flags

TSS_TSPATTRIB_DAAISSUERKEY_BLOB	Get/Set attributes for a DAA Issuer key.
TSS_TSPATTRIB_DAAISSUERKEY_PUBKEY	Set/Get attributes for a DAA Issuer public key.

Attribute subflags

TSS_TSPATTRIB_DAAISSUERKEYBLOB_PUBLIC_KEY	Get/Set an Issuer's public key as a TSS_DAA_PK structure.
TSS_TSPATTRIB_DAAISSUERKEYBLOB_SECRET_KEY	Get/Set an Issuer's secret key as a TSS_DAA_SK structure.
TSS_TSPATTRIB_DAAISSUERKEYBLOB_KEYBLOB	Get/Set an Issuer's key pair as a TSS_DAA_KEY_PAIR structure.
TSS_TSPATTRIB_DAAISSUERKEYBLOB_PROOF	Get/Set an Issuer's public key proof as a TSS_DAA_PK_PROOF structure.
TSS_TSPATTRIB_DAAISSUERPUBKEY_NUM_ATTRIBS	Get the total number of credential attributes: $l_h + l_i$
TSS_TSPATTRIB_DAAISSUERPUBKEY_NUM_PLATFORM_ATTRIBS	Get/Set the number of platform attributes for a credential: l_h
TSS_TSPATTRIB_DAAISSUERPUBKEY_NUM_ISSUER_ATTRIBS	Get/Set the number of issuer attributes for a credential: l_i

2.3.2.15 Attribute Definitions for DAA ARA Key Objects

Attribute flags

TSS_TSPATTRIB_DAAARAKEY_BLOB	Sets/Gets attributes for a DAA ARA key.

Attribute subflags

TSS_TSPATTRIB_DAAARAKEYBLOB_PUBLIC_KEY	Get/Set an ARA's public key as a TSS_DAA_AR_PK structure.
TSS_TSPATTRIB_DAADARAKEYBLOB_SECRET_KEY	Get/Set an ARA's secret key as a TSS_DAA_AR_SK structure.
TSS_TSPATTRIB_DAAARAKEYBLOB_KEYBLOB	Get/Set an ARA's key pair as a TSS_DAA_KEY_PAIR structure.

2.3.2.16 Policy Definitions for Secret Mode

Start of informative comment:

Definition of policy mode flags that can be used with the method Tspi_Policy_SetSecret( ).

End of informative comment.

The defined secret modes are based on the data type TSS_FLAG.

Secret Mode	Description
TSS_SECRET_MODE_NONE	No authorization will be processed; different from secret of 20 bytes of 0x00.
TSS_SECRET_MODE_SHA1	Secret string will not be touched by TSS SP and MUST be size of 20 bytes.
TSS_SECRET_MODE_PLAIN	Secret string will be hashed using SHA1.
TSS_SECRET_MODE_POPUP	TSS SP will ask for a secret. The provided pass phrase MUST be represented as a TSS_UNICODE string and MUST be hashed using SHA1 to get the authorization secret.
TSS_SECRET_MODE_CALLBACK	Application has to provide a call back function.

2.3.2.17 Policy Definition for Secret Lifetime

Start of informative comment:

Definition of secret lifetime flags that can be used with the method Tspi_SetAttribUint32( ) and Tspi_GetAttribUint32( ) addressing a policy object.

End of informative comment.

The defined secret modes are based on the data type TSS_FLAG.

TSS_SECRET_LIFETIME_ALWAYS	Secret will not be invalidated.
TSS_SECRET_LIFETIME_COUNTER	Secret may be used n-times.
TSS_SECRET_LIFETIME_TIMER	Secret will be valid for n seconds.

2.3.2.18 TPM Status Flags Definitions

Start of informative comment:

These flags are used to set and get the TPM status by calling the methods Tspi_TPM_SetStatus( ) and Tspi_TPM_GetStatus( ).

End of informative comment.

Flag	Description	Usage
TSS_TPMSTATUS_DISABLEOWNERCLEAR	Permanently disable the TPM owner authorized clearing of TPM ownership. The method Tspi_TPM_ClearOwner( ) with fForcedClear = FALSE is not available any longer.	SetStatus GetStatus
TSS_TPMSTATUS_DISABLEFORCE	Prevent temporarily (until next power on) a forced clear of the TPM	SetStatus GetStatus

CLEAR	ownership. The method Tspi_TPM_ClearOwner( ) with fForcedClear = TRUE is temporarily not available.	
TSS_TPMSTATUS _OWNERSETDISABLE	fTpmState = TRUE: Disable the TPM. Owner authorization is required. Supported for backward compatibilty. Recommend using TSS_TPMSTATUS_DISABLED flag	SetStatus GetStatus
TSS_TPMSTATUS _PHYSICALDISABLE	fTpmState = TRUE: Disable the TPM. Proof of physical access is required. Supported for backward compatibilty. Recommend using TSS_TPMSTATUS_DISABLED flag	SetStatus GetStatus
TSS_TPMSTATUS _PHYSICALSETDEACTIVATED	fTpmState = TRUE: Deactivate the TPM. Proof of physical access is required. Supported for backward compatibilty. Recommend using TSS_TPMSTATUS_DISABLED flag	SetStatus GetStatus
TSS_TPMSTATUS _SETTEMPDEACTIVATED	Temporarily deactivate (until next power on) the TPM. Operator authorization or physical presence is required on 1.2 TPMs.	SetStatus GetStatus
TSS_TPMSTATUS _SETOWNERINSTALL	fTpmState = TRUE: Set the ability to take TPM ownwership utilizing the method Tspi_TPM_TakeOwnership( ).	SetStatus GetStatus
TSS_TPMSTATUS _DISABLEPUBEKREAD	Permanently disable (1.1 TPMs) Disable or enable (not 1.1 TPMs) the ability to read the endorsement public key without required TPM owner authorizition. The method Tspi_TPM_GetPubEndorsementKey ( ) with fOwnerAuthorized = FALSE is not available any longer.	SetStatus GetStatus
TSS_TPMSTATUS _DISABLED	Query and set whether TPM is disabled or enabled. (TSS 1.1b will not allow setting this flag)	SetStatus GetStatus
TSS_TPMSTATUS _DEACTIVATED	Query whether the TPM is deactivated or activated.	SetStatus GetStatus
TSS_TPMSTATUS _ALLOWMAINTENANCE	Query whether the TPM owner may create a maintenance archive utilizing the method Tspi_TPM_CreateMaintenanceArchive( ) or not.	SetStatus GetStatus
TSS_TPMSTATUS _MAINTENANCEUSED	Query whether the TPM owner has already created a maintenance archive for the current SRK	GetStatus
TSS_TPMSTATUS _PHYSPRES_LIFTIMELOCK	Query whether both physicalPresenceHWEnable and physicalPresenceCMDEnable flags are locked and cannot be changed for the	GetStatus

	life of the TPM.	
TSS_TPMSTATUS_PHYSPRES_HWENABLE	Query whether the TPM hardware signal <physical presence> is enabled to provide proof of physical presence.	GetStatus
TSS_TPMSTATUS_PHYSPRES_CMDENABLE	Query whether the TPM command TSC_PhysicalPresence is enabled to provide proof of physical presence.	GetStatus
TSS_TPMSTATUS_CEKP_USED	Query whether the endorsement key pair was created using the methode Tspi_TPM_CreateEndorsementKey( ) or it was created using a manufacturers process.	GetStatus
TSS_TPMSTATUS_PHYSPRESENCE	Query whether a TPM owner is present indicated by the TPM command TSC_PhysicalPresence.	GetStatus
TSS_TPMSTATUS_PHYSPRES_LOCK	Query whether changes to the physicalPresence flag are permitted.	GetStatus
TSS_TPMSTATUS_POSTINITIALISE	Indicates that the TPM is between the TPM_Init state and the execution of the TPM_Startup command.	GetStatus
TSS_TPMSTATUS_TPMPOST	Sets the TPM to force a full sefttest before allowing commands to be performed.	GetStatus
TSS_TPMSTATUS_TPMPOSTLOCK	Locks the state of the TSS_TPMSTATUS_TPMPOST flag for the lifetime of the TPM	GetStatus
TSS_TPMSTATUS_DISABLEPUBSRKREAD Not valid for 1.1 TPMs	Indicates/sets the ability to read the public portion of the SRK using Tspi_Key_GetPubKey( ) with hKey = the handle of the SRK.	GetStatus SetStatus
TSS_TPMSTATUS_OPERATOR_INSTALLED Not valid for 1.1 TPMs	Indicates whether or not the operator authorization has been set.	GetStatus
TSS_TPMSTATUS_FIPS	Indicates whether or not the TPM operates in FIPS mode	GetStatus
TSS_TPMSTATUS_ENABLE_REVOKEEK	Indicates whether or not the ability to revoke EK is enabled	GetStatus
TSS_TPMSTATUS_NV_LOCK	Indicates whether or not the authorization is active to access NV area. TRUE - authorization active FALSE - no authorization is active, (except for maxNVWrites)	GetStatus
TSS_TPMSTATUS_TPM_ESTABLISHED	Indicates whether or not the dynamic root of trust of measurement has been executed.	GetStatus

TSS_TPMSTATUS _RESETLOCK	Resets the effect of a number of authorization failures	SetStatus

Remarks:

Please see the manual of your TCG system, to set the physical access state.

If the TPM status is set to *DISABLED* only the following TSPI methods will execute. All other methods will return the TPM error TCPA_DISABLED.

- Tspi_TPM_GetCapability
- Tspi_TPM_PcrExtend
- Tspi_TPM_SetStatus using the flag TSS_TPMSTATUS_OWNERSETDISABLE
- Tspi_TPM_SetStatus using the flag TSS_TPMSTATUS_PHYSICALDISABLE
- Tspi_TPM_SelfTestFull
- Tspi_TPM_GetTestResult

If the TPM status is set to *DEACTIVATED* only the following TSPI methods will execute. All other methods will return the TPM error TCPA_DEACTIVATED.

- Tspi_TPM_GetCapability
- Tspi_TPM_TakeOwnership
- Tspi_TPM_SetStatus using the flag TSS_TPMSTATUS_OWNERSETDISABLE
- Tspi_TPM_SetStatus using the flag TSS_TPMSTATUS_PHYSICALDISABLE
- Tspi_TPM_SetStatus using the flag TSS_TPMSTATUS_PHYSICALSETDEACTIVATED
- Tspi_TPM_SelfTestFull
- Tspi_TPM_GetTestResult

fTpmState must be TRUE to execute the TPM operation.

2.3.2.19 Algorithm ID Definitions

Start of informative comment:

This table defines the types of algorithms which may be supported.

End of informative comment.

The defined algorithm IDs are based on the data type TSS_ALGORITHM_ID.

Algorithm ID	Description
TSS_ALG_RSA	The RSA algorithm.
TSS_ALG_DES	The DES algorithm.
TSS_ALG_3DES	The 3DES algorithm.
TSS_ALG_SHA	The SHA1 algorithm.
TSS_ALG_SHA256	The SHA256 algorithm
TSS_ALG_HMAC	The RFC 2104 HMAC algorithm.
TSS_ALG_AES128	The AES algorithm, key size 128.
TSS_ALG_MGF1	The XOR algorithm using MGF1 to create a

	string the size of the encrypted block
TSS_ALG_AES192	The AES algorithm, key size 192
TSS_ALG_AES256	The AES algorithm, key size 256
TSS_ALG_XOR	XOR using rolling nonces
TSS_ALG_AES	The AES algorithm. (legacy)

Remarks:

The TPM must support the algorithms TSS_ALG_RSA, TSS_ALG_SHA, TSS_ALG_HMAC.

2.3.2.20 Capability Flag Definitions

Start of informative comment:

Flags indicating a capability to be queried

End of informative comment.

The defined capability flags are based on the data type TSS_FLAG.

TPM Capabilities:

Capability Area	Description
TSS_TPMCAP_ORD	Queries whether an ordinal is supported
TSS_TPMCAP_ALG	Queries whether an algorithm is supported.
TSS_TPMCAP_FLAG	Returns the bitmap of all persistent and volatile flags (see discussion following this table)
TSS_TPMCAP_PROPERTY	Determines a physical property of the TPM.
TSS_TPMCAP_VERSION	Queries the current TPM version.
TSS_TPMCAP_VERSION_VAL	Queries the TPM_VERSION_VAL for a 1.2 or later TPM. TPM_CAP_VERSION_INFO?
TSS_TPMCAP_NV_LIST	Retrieves the list of indices for defined NV storage areas.
TSS_TPMCAP_NV_INDEX	Retrieves a TPM_NV_DATA_PUBLIC structure that indicates the values for the specified NV area.
TSS_TPMCAP_MFR	Retrieves manufacturer specific TPM and TPM state information.
TSS_TPMCAP_SYM_MODE	Queries whether or not the TPM supports a particular type of a symmetric encryption
TSS_TPMCAP_HANDLE	Returns list of handles of objects currenctly loaded in the TPM
TSS_TPMCAP_TRANS_ES TSS_TPMCAP_AUTH_ENCRYPT	Queries whether the TPM supports a particular encryption scheme in the transport session. Queries whether the TPM supports a particular encryption scheme in the OSAP encryption of the AuthData values.

Capability Area	Description
TSS_TCSCAP_TRANSPORT	Queries the support of transport features
TSS_TCSCAP_PLATFORM_INFO	Queries information about the host platform

Tspi_TPM_GetCapability with the Capability Area set to TSS_TPMCAP_FLAG returns a byte string in big-endian byte order of the TPM persistent and volatile flags. This is a 1.1b capability and only 1.1b flags are reported.

The first four bytes returned represent the UINT32 bit map of persistent flags, and the second four bytes returned represent the UINT32 bit map of volatile flags. Therefore, bits 31-28 of the persistent flags are returned in the first byte; bits 3-0 of the volatile flags are returned in the last byte. Bit 0 is defined to be the lsb of a UINT32.

The persistent flag bit map is as follows (in 1.2 terminology):

TPM_PF_DISABLE	0x00000001
TPM_PF_OWNERSHIP	0x00000002
TPM_PF_DEACTIVATED	0x00000004
TPM_PF_READPUBEK	0x00000008
TPM_PF_DISABLEOWNERCLEAR	0x00000010
TPM_PF_ALLOWMAINTENANCE	0x00000020
TPM_PF_PHYSICALPRESENCELIFETIMELOCK	0x00000040
TPM_PF_PHYSICALPRESENCEHWENABLE	0x00000080
TPM_PF_PHYSICALPRESENCECMDENABLE	0x00000100
TPM_PF_CEKPUSED	0x00000200
TPM_PF_TPMPOST	0x00000400
TPM_PF_TPMPOSTLOCK	0x00000800

The volatile flag bit map is as follows (in 1.2 terminology):

TPM_SF_DEACTIVATED	0x00000001
TPM_SF_DISABLEFORCECLEAR	0x00000002
TPM_SF_PHYSICALPRESENCE	0x00000004
TPM_SF_PHYSICALPRESENCELOCK	0x00000008
TPM_AF_POSTINITIALIZE	0x00000010

All bits positions not enumerated above are not used.

TSS Core Service Capabilities:

Capability Area	Description
TSS_TCSCAP_ALG	Queries whether an algorithm is supported.
TSS_TCSCAP_VERSION	Queries the current TCS version.
TSS_TCSCAP_MANUFACTURER	Queries TCS manufacturer information.
TSS_TCSCAP_CACHING	Queries the support of key and authorization caching.
TSS_TCSCAP_PERSSTORAGE	Queries the support of a persistent storage
TSS_TCSCAP_PLATFORM_CLASS	Queries the class of the host platform
TSS_TCSCAP_PLATFORM_INFO	Queries information about the TCS host platform
TSS_TCSCAP_TRANSPORT	Queries properties of the TCS transport session support

TSS Service Provider Capabilities:

Capability Area	Description
TSS_TSPCAP_ALG	Queries whether an algorithm is supported.
TSS_TSPCAP_VERSION	Queries the current TSP version.
TSS_TSPCAP_RETURNVALUE_INFO	Queries if return value is ASN.1 ecoded or not
TSS_TSPCAP_PERSSTORAGE	Queries the support of a persistent storage
TSS_TSPCAP_MANUFACTURER	Queries TSP manufacturer information.
TSS_TSPCAP_PLATFORM_INFO	Queries information about the TSP host platform

2.3.2.21 Sub-Capability Flag Definitions

Start of informative comment:

Sub-Flags indicating a capability to be queried dependent on the capability flag

End of informative comment.

The defined sub-capability flags for capability TSS_TPMCAP_PROPERTY.

TPM Sub-Capabilities:

SubCap Area	Response
TSS_TPMCAP_PROP_PCR	UINT32 value. Returns the number of PCR registers supported by the TPM
TSS_TPMCAP_PROP_PCRMAP	Returns the bitmap TPM_PCR_ATTRIBUTES. The array of TPM_PCR_ATTRIBUTES for the PCRs

SubCap Area	Response
	refered to in TSS_TPMCAPPROP_PCR
TSS_TPMCAP_PROP_DIR	UINT32 value. Returns the number of DIR registers supported by the TPM.
TSS_TPMCAP_PROP_MANUFACTURER	UINT32 value. Returns the Identifier of the TPM manufacturer.
TSS_TPMCAP_PROP_SLOTS or TSS_TPMCAP_PROP_KEYS	UINT32 value. Returns the maximum number of 2048 bit RSA keys that the TPM is capable of loading. This MAY vary with time and circumstances.
TSS_TPMCAP_PROP_MIN_COUNTER	UINT32. Returns the minimum amount of time in 10ths of a second that must pass between invocations of incrementing the monotonic counter
TSS_TPMCAP_PROP_FAMILYROWS	UINT32. Returns the number of rows in the family table
TSS_TPMCAP_PROP_DELEGATEROWS	UINT32. Returns the number of rows in the delegate table.
TSS_TPMCAP_PROP_OWNER	TSS_BOOL. Returning a value of TRUE indicates that the TPM has successfully installed an owner.
TSS_TPMCAP_PROP_MAXKEYS	UINT32. Returns the maximum number of 2048-bit RSA keys that the TPM can support. The number does not include the EK or SRK.
TSS_TPMCAP_PROP_AUTHSESSIONS	UINT32. Returns the number of available authorization sessions. This MAY vary with time and circumstances.
TSS_TPMCAP_PROP_MAXAUTHSESSIONS	UINT32. Returns the maximum number of loaded authorization sessions the TPM supports.
TSS_TPMCAP_PROP_TRANSESSIONS	UINT32. Returns the number of available transport sessions. This MAY vary with time and circumstances.
TSS_TPMCAP_PROP_MAXTRANSESSIONS	UINT32. Returns the maximum number of loaded transport sessions the TPM supports.
TSS_TPMCAP_PROP_SESSIONS	UNIT32. Returns the number of available sessions from the pool. Pool sessions include authorization and transport sessions. This MAY vary with time and circumstances.
TSS_TPMCAP_PROP_MAXSESSIONS	UINT32. Returns the maximum number of sessions (authorization and transport) the TPM supports.
TSS_TPMCAP_PROP_CONTEXTS	UINT32. Returns the number of available saved session slots. This MAY vary with time and circumstances.
TSS_TPMCAP_PROP_MAXCONTEXTS	UINT32. Returns the maximum number of saved session slots.

SubCap Area	Response
TSS_TPMCAP_PROP_DAASESSIONS	UINT32. Returns the number of available DAA sessions. This MAY vary with time and circumstances.
TSS_TPMCAP_PROP_MAXDAASESSIONS	UINT32. Returns the maximum number of DAA sessions (join or sign) that the TPM supports.
TSS_TPMCAP_PROP_DAA_INTERRUPT	TSS_BOOL. Returning a value of TRUE indicates that the TPM will accept ANY command while executing a DAA Join or Sign. Returning a value of FALSE indicates that the TPM will invalidate the DAA Join or Sign upon the receipt of any command other than the next join/sign in the session or a TPM_SaveContext.
TSS_TPMCAP_PROP_COUNTERS	UINT32. Returns the number of available monotonic counters. This MAY vary with time and circumstances.
TSS_TPMCAP_PROP_MAXCOUNTERS	UINT32. Returns the maximum number of monotonic counters under control of TPM_CreateCounter.
TSS_TPMCAP_PROP_ACTIVECOUNTER	TPM_COUNT_ID. Returns the ID of the current counter. 0xff..ff is returned if no counter is active.
TSS_TPMCAP_PROP_TISTIMEOUTS	Returns a 4-element array of UINT32 values each denoting the timeout value in microseconds for the following in this order: TIMEOUT_A, TIMEOUT_B, TIMEOUT_C, TIMEOUT_D Where these timeouts are to be used is determined by the platform-specific TPM Interface Specification.
TSS_TPMCAP_PROP_STARTUPEFFECTS	Returns the TPM_STARTUP_EFFECTS structure.
TSS_TPMCAP_PROP_MAXCONTEXTCOUNTDIST	UINT32. Returns the maximum distance between context count values. This MUST be at least 2^16-1.
TSS_TPMCAP_PROP_CMKRESTRICTION	UINT32 Returns TPM_Permanent_Data -> restrictDelegate
TSS_TPMCAP_PROP_DURATION	Returns a 3-element array of UINT32 values each denoting the value in microseconds of the duration of the three classes of commands in the following order: SMALL_DURATION, MEDIUM_DURATION, LONG_DURATION
TSS_TPMCAP_PROP_MAXNVAVAILABLE	UINT32. Returns the maximum number of NV space that can be allocated. This MAY vary with time and circumstances.
TSS_TPMCAP_PROP_MAXNVWRITE	The count of NV writes that have

SubCap Area	Response
	occurred when there is no TPM Owner
TSS_TPMCAP_PROP_REVISION	2 BYTEs: This is the TPM major and minor revision indicator in the standard structure Major first then minor
TSS_TPMCAP_PROP_LOCALITIES_AVAIL	The number of localities available in the TPM
TSS_TPMCAP_PROP_INPUTBUFFERSIZE	UINT32. Returns the size of the TPM input buffer in bytes.

TSS Core Service Sub Capabilities:

The defined sub-capability flags for capability TSS_TCSCAP_ALG

TSS_ALG_XX	Returns a boolean indicating whether the algorithm is supported by the TCS
TSS_ALG_DEFAULT	Returns the default public key algorithm
TSS_ALG_DEFAULT_SIZE	Returns the default public key length

The defined sub-capability flags for capability TSS_TCSCAP_MANUFACTURER

SubCap Area	Description
TSS_TCSCAP_PROP_MANUFACTURER_STR	Returns a TSS_UNICODE string of the TCS manufacturer. The contents of this string are determined by the manufacturer and are subject to change in subsequent releases of the TCS.
TSS_TCSCAP_PROP_MANUFACTURER_ID	Returns the UINT32 value which identifies the TCS manufacturer as specified in the main specification by: Capability: TPM_CAP_PROPERTY; Sub-Capability: TPM_CAP_PROP_MANUFACTURER

The defined sub-capability flags for capability TSS_TCSCAP_CACHING

SubCap Area	Description
TSS_TCSCAP_PROP_KEYCACHE	TSS_BOOL value. Indicates support of key caching
TSS_TCSCAP_PROP_AUTHCACHE	TSS_BOOL value. Indicates support of authorization session caching

The defined sub-capability flags for capability TSS_TCSCAP_PLATFORM_CLASS

TSS_TCSCAP_PROP_HOST_PLATFORM	Returns a single TSS_PLATFORM_CLASS structure containing the definition of only the host platform's class
TSS_TCSCAP_PROP	Returns an array of TSS_PLATFORM_CLASS

_ALL_PLATFORMS	strucutures which enumerates all the TCG_defined platforms associated with the Host Platform. The Host Platform MUST NOT be returned as one of these platform classes. There is no relationship required in the order of the platforms listed.

The defined sub-capability flags for capability TSS_TCSCAP_TRANSPORT

0	Queries the support of transport features. (bool)
TSS_TCSCAP_PROP_TRANS_EXCLUSIVE	Queries whether the exclusive transport mode is supported. (bool)

The defined sub-capability flags for TSS_TCSCAP_PLATFORM_INFO

TSS_TCSCAP_PLATFORM_TYPE	Queries the type of platform
TSS_TCSCAP_PLATFORM_VERSION	Queries the version of the platform type

TSS Service Provider Sub Capabilities:

The defined sub-capability flags for capability TSS_TSPCAP_RETURNVALUE_INFO

SubCap Area	Description
TSS_TSPCAP_PROP_RETURNVALUE_INFO	0 indicates ASN.1 encoding.

The defined sub-capability flags for capability TSS_TSPCAP_ALG

TSS_ALG_XX	Returns a boolean indicating whether the algorithm is supported by the TSP
TSS_ALG_DEFAULT	Returns the default public key algorithm
TSS_ALG_DEFAULT_SIZE	Returns the default public key length

The defined sub-capability flags for TSS_TSPCAP_PLATFORM_INFO

TSS_TSPCAP_PLATFORM_TYPE	Queries the type of platform
TSS_TSPCAP_PLATFORM_VERSION	Queries the version of the platform type

The defined sub-capability flags for capability TSS_TSPCAP_MANUFACTURER

SubCap Area	Description
TSS_TSPCAP_PROP_MANUFACTURER_STR	Returns a TSS_UNICODE string of the TSP manufacturer. The contents of this string is determined by the manufacturer and is subject to change in subsequent releases of the TSP.

SubCap Area	Description
TSS_TSPCAP_PROP_MANUFACTURER_ID	Returns the UINT32 value which identifies the TSP manufacturer as specified in the main specification by: Capability: TPM_CAP_PROPERTY; Sub-Capability: TPM_CAP_PROP_MANUFACTURER

2.3.2.22 Persistent Storage Flag Definitions

Start of informative comment:

Definition of flags indicating the persistent storage to be used within the method Tspi_Context_RegisterKey( ).

End of informative comment.

The defined persistent storage flags are based on the data type TSS_FLAG.

Persistent Storage Type	Description
TSS_PS_TYPE_USER	Key is registered persistently in the user storage database.
TSS_PS_TYPE_SYSTEM	Key is registered persistently in the system storage database.

2.3.2.23 Migration Scheme Definitions

Start of informative comment:

The scheme indicates how the migration of a key should be done

End of informative comment.

The defined migration scheme flags are based on the data type TSS_MIGRATE_SCHEME.

Migration Scheme	Description
TSS_MS_RESTRICT_MIGRATE	The key is to be migrated to a migration authority
TSS_MS_MIGRATE	A public key that can be used for migrating a key utilizing Tspi_Key_CreateMigrationBlob followed by Tspi_Key_ConvertMigrationBlob.
TSS_MS_REWRAP	A public key that can be used for migrating a key by just rewrapping this key utilizing Tspi_Key_CreateMigrationBlob.
TSS_MS_MAINT	A public key that can be used for the maintenance commands.

2.3.2.24 Key Usage Definitions

Start of informative comment:

This table defines the types of keys that are possible.

Each key has a setting defining the encryption and signature scheme to use. The selection of a key usage value limits the choices of encryption and signature schemes.

End of informative comment.

The defined key usage types are based on the data type TSS_KEY_USAGE_ID.

Key Usage	Description
TSS_KEYUSAGE_BIND	The key can be used for binding and unbinding operations only.
TSS_KEYUSAGE_IDENTITY	The key is used for operations that require a TPM identity, only.
TSS_KEYUSAGE_LEGACY	The key can perform signing and binding operations.
TSS_KEYUSAGE_SIGN	The [private] key is used for signing operations, only. This means that it MUST be a leaf of the Protected Storage key hierarchy
TSS_KEYUSAGE_MIGRATE	This key is used for the TPM_MigrateKey operation
TSS_KEYUSAGE_STORAGE	The key is used to wrap and unwrap other keys in the Protected Storage hierarchy, only.
TSS_KEYUSAGE_AUTHCHANGE	The key is used to change authorization

Remarks:

2.3.2.25 Key Size Definitions

Start of informative comment:

This table defines the key sizes returned by Tspi_GetAttribUint32(TSS_TSPATTRIB_KEY_INFO, TSS_TSPATTRIB_KEYINFO_SIZE).

End of informative comment.

The defined key sizes as returned by Tspi_GetAttribUint32(TSS_TSPATTRIB_KEY_INFO, TSS_TSPATTRIB_KEYINFO_SIZE).

Key Size	Description
TSS_KEY_SIZEVAL_512BIT	key size is 512 bit
TSS_KEY_SIZEVAL_1024BIT	key size is 1024 bit
TSS_KEY_SIZEVAL_2048BIT	key size is 2048 bit
TSS_KEY_SIZEVAL_4096IT	key size is 4096 bit
TSS_KEY_SIZEVAL_8192BIT	key size is 8192 bit
TSS_KEY_SIZEVAL_16384BIT	key size is 16384 bit

Remarks:

2.3.2.26 Key Type Flags

Start of informative comment:

This table defines the key flag types.

End of informative comment.

Key Type Flag	Description
TSS_KEYFLAG_REDIRECTION	set to 1 if redirection key
TSS_KEYFLAG_MIGRATABLE	set to 1 if migratable key
TSS_KEYFLAG_CERTIFIED_MIGRATABLE	Set to 1 if certified migratable key
TSS_KEYFLAG_VOLATILEKEY	Set to 1 if volatile key

Remarks:

2.3.2.27 Key Structure Types

Start of informative comment:

This table defines the key structure types returned by Tspi_GetAttribUint32(TSS_TSPATTRIB_KEY_INFO, TSS_TSPATTRIB_KEYINFO_KEYSTRUCT) or set by Tspi_SetAttribUint32(TSS_TSPATTRIB_KEY_INFO, TSS_TSPATTRIB_KEYINFO_KEYSTRUCT).

End of informative comment.

Key Structure Type	Description
TSS_KEY_STRUCT_KEY	Key object uses 1.1 TCPA_KEY structure
TSS_KEY_STRUCT_KEY12	Key object uses 1.2 TCPA_KEY12 structure

Remarks:

2.3.2.28 Key Authorization

Start of informative comment:

This table defines the key authorization types.

End of informative comment.

Key Authorization	Description
TSS_KEYAUTH_AUTH_NEVER	Key never requires authorization
TSS_KEYAUTH_AUTH_ALWAYS	Key always requires authorization
TSS_KEYAUTH_AUTH_PRIV_USE_ONLY	This indicates that on commands that require the TPM to use the private portion of the key, the authorization MUST be performed. For commands that cause the TPM to read the public portion of the key, but not to use the private portion (e.g. TPM_GetPubKey), the authorization may be omitted.

Remarks:

2.3.2.29 Key Encryption Scheme Definitions

Start of informative comment:

The TPM performs the encryption or decryption in accordance with the specification of the encryption scheme to be used for a key.

End of informative comment.

The defined encryption scheme IDs are based on the data type TSS_KEY_ENC_SCHEME.

Algorithm ID	Description
TSS_ES_NONE	No encryption scheme is set.
TSS_ES_RSAESPKCSV15	The encryption is performed using the scheme RSA_ES_PKCSV15 defined in

	[PKCS #1v2.0: 8.1].
TSS_ES_RSAESOAEP_SHA1_MGF1	The encryption and decryption is performed using the scheme RSA_ES_OAEP defined in [PKCS #1v2.0: 8.1] using SHA1 as the hash algorithm for the encoding operation.
TSS_ES_RSAESOIAP_SHA1_MGF1	The encryption is performed using the Object Independent Authorization Protocol (see TPM Main Part 1 Design Principles Specification Version 1.2)
TSS_ES_RSAESOSAP_SHA1_MGF1	The encryption and decryption is performed using the scheme Object Specific Authorization Protocol (see TPM Main Part 1 Design Principles Specification Version 1.2)

Remarks:

The TPM checks that the encryption scheme defined for use with the key is a valid scheme for the key type, as follows:

Key algorithm	Approved schemes
TSS_ALG_RSA	TSS_ES_NONE
	TSS_ES_RSAESPKCSv15
	TSS_ES_RSAESOAEP_SHA1_MGF1

2.3.2.30 Key Signature Scheme Definitions

Start of informative comment:

The TPM performs the digital signatures in accordance with the specification of the signature scheme to be used for a key.

End of informative comment.

The defined signature scheme IDs are based on the data type TSS_KEY_SIG_SCHEME.

Algorithm ID	Description
TSS_SS_NONE	
TSS_SS_RSASSAPKCS1V15_SHA1	The signature is be performed using the scheme RSASSA-PKCS1-v1.5 defined in [PKCS #1v2.0: 8.1] using SHA1 as the hash algorithm for the encoding operation.
TSS_SS_RSASSAPKCS1V15_DER	The signature is performed using the scheme RSASSA-PKCS1-v1.5 defined in [PKCS #1v2.0: 8.1]. The caller must properly format the area to sign using the DER rules. The provided area maximum size is k-11 octets

Remarks:

The TPM checks that the signature scheme defined for use with the key is a valid scheme for the key type, as follows:

Key algorithm	Approved schemes
TSS_ALG_RSA	TSS_ES_NONE
	TSS_SS_RSASSAPKCS1V15_SHA1
	TSS_SS_RSASSAPKCS1V15_DER

2.3.2.31 PCR Structure Types

Start of informative comment:

This table defines the PCR structure types returned by Tspi_GetAttribUint32(TSS_TSPATTRIB_PCRS_INFO, TSS_TSPATTRIB_PCRSINFO_PCRSTRUCT) or set by Tspi_SetAttribUint32(TSS_TSPATTRIB_PCRS_INFO, TSS_TSPATTRIB_PCRSINFO_PCRSTRUCT).

End of informative comment.

PCR Structure Type	Description
TSS_PCRS_STRUCT_INFO	PcrComposite object uses the 1.1 TCPA_PCR_INFO structure
TSS_PCRS_STRUCT_INFO_LONG	PcrComposite object uses the 1.2 TCPA_PCR_INFO_LONG structure
TSS_PCRS_STRUCT_INFO_SHORT	PcrComposite object uses the 1.2 TCPA_PCR_INFO_SHORT structure

Remarks:

2.3.2.32 Event Type Definitions

Start of informative comment:

These are platform specific, and hence this has been removed from the TSS spec. The header files will keep the numbers reserved for the platform specific specs.

Flags indicating the type of event/supporting information are the purview of the preBoot and post boot environments, and the TSS has no capability of modifying them. The TSS is resposible for reporting the event table for a specific PCR, but not to interpret that information.

End of informative comment.

2.3.2.33 Well Known Secret

Start of informative comment:

This is simply a "helper" definition for those applications where the well know secret is defined as all zeros. Note, there is no required value for this field.

End of informative comment.

This value is only used for convenience for a "well known secret" for authentication data. The actual value of its definition is not mandated.

Well Known Secret	Description
TSS_WELL_KNOWN_SECRET	A value used for convenience as a "well known value"

Remarks:

2.3.2.34 TSS_DAA_PSEUDONYM Structure Payload Flag

Start of informative comment:

This is simply the structure payload flag for the TSS_DAA_PSEUDONYM

End of informative comment.

Flag	***Description***
TSS_FLAG_DAA_PSEUDONYM_PLAIN	The payload is a TSS_DAA_PSEUDONYM_PLAIN structure
TSS_FLAG_DAA_PSEUDONYM_ENCRYPTED	The payload is a TSS_DAA_PSEUDONYM_ENCRYPTED structure

2.4 Return Codes

Start of informative comment:

Return codes contain information divided into components: OS error code, TSS layer field and TSS error code. The OS error code is specific to the OS and the services that the TSS may rely upon. The content of this field is OS-specific and details outside the scope of this specification and applications should only assume OS-specific error codes are valid if they are non-zero.

Each layer within the TSS should adopt a “best effort” approach the correction of errors from lower layers. If the “best effort” approach fails, the layer must return the lower layer error that is relevant to the requested function and operation. For example, if a key is not loaded it is expected the TCS key manager will attempt to make room for a key and load it, all transparently to the layers above it. However, if the TCS key manager cannot load the needed key it must return the TPM error based on the last error received that is relevant to the currently requested function and object. To further refine the example, if the TPM does not have sufficient space to load a key needed for a TPM_Sign function, the TCS could just attempt to load the key and receive a TCPA_NOSPACE error but that would violate the “best effort” requirement unless the TPM did not implement the TPM_SaveKeyContext in which case the TCS would in fact return TCPA_NOSPACE error. If the TPM allowed saving the key context the TCS would be expected to perform this function to make room for the key. If an error occurred while attempting to load the key because the unwrapping key was not loaded the TCS would be expected to attempt to load the unwrapping key. If the unwrapping key could not be loaded the TCS would return TCPA_NOSPACE error code because that error, not the failure to load the parent’s key, is the most relevant to the requested function and the object.

The coding system should specify room for platform specific requirements and extensions. It should also provide an opportunity to integrate vendor specific return codes in each layer and allow the possibility of specific common return codes over all layers.

End of informative comment.

There are three components of each return code: OS code, TSS layer, TSS code.

- The OS code contains information regarding the OS-specific error. The details of this code are outside the scope of this specification and this field is not required. This field may not be used for anything other than the platform defined fields.
- The TSS layer field identifies the layer originating the error.
- The TSS code identifies the specific cause or condition of the error.

Each layer within the TSS SHOULD make a “best effort” to correct errors from lower layers. If the “best effort” fails, the layer must return the lower layer error that is relevant to the requested function and operation.

2.4.1 Return Codes Scheme

A TSS_RESULT is defined as a 32-bit value as follows:

```
 3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
+---+-+-+-----------------------+-------------------------------+
|      OS Specific              |        TSS Specific          |
+---+-+-+-----------------------+-------------------------------+
|      OS Specific              | Layer |         Code          |
+---+-+-+-----------------------+-------------------------------+
```

OS specific error information:

The bits #16 to #31 are specific to the operating system. More information can be found below.

TSS specific error information:

The bits #0 up to #15 are TSS specific and subdivided in two parts:

- Layer information and
- Error code information

Platform Specific errors:

Any platform specific error return codes must not be returned by any TSS SW stack component; this means that a TSS SW stack component only returns error codes applying to the above described rules and must hide any platform specific error return codes and therefore must map these error codes to TSS specific error codes. (e.g. internal TSP error TSS_E_INTERNAL_ERROR)

Layer Information:

Bit #12 to Bit #15 are specifying the layer of the TSS SW stack returning the error.

Each TSS SW stack component (TDDL, TCS and TSP) must return an error code either with the layer nibble set to its own layer information or with the TPM layer information. The latter will be returned if an error was encountered by the TPM.

The layer code MUST be chosen from the following list.

Layer	Value	Description
TPM	0x0	Error returned by TPM
TDDL	0x1	Error returned by TDDL
TCS	0x2	Error returned by TCS
TSP	0x3	Error returned by TSP

Error Code Information:

Bit #0 to Bit #11 are reporting the appropriate TSS specific error code.

2.4.2 Common Return Code Defines

The following table lists the error codes (Bit #0 to Bit #11 reporting the appropriate TSS specific error code) common to all TSS Layers.

Type	Definition
TSS_SUCCESS	Success
TSS_E_FAIL	Non-specific failure
TSS_E_BAD_PARAMETER	One or more parameter is bad.
TSS_E_INTERNAL_ERROR	An internal SW error has been detected.
TSS_E_NOTIMPL	Not implemented.
TSS_E_PS_KEY_NOTFOUND	The key cannot be found in the persistent storage database.
TSS_E_KEY_ALREADY_REGISTERED	Key could not be registered because UUID has already registered.
TSS_E_KEY_NO_MIGRATION_POLICY	No migration policy available
TSS_E_CANCELED	The action was canceled.
TSS_E_TIMEOUT	The operation has timed out.
TSS_E_OUTOFMEMORY	Ran out of memory.
TSS_E_TPM_UNEXPECTED	TPM returns with success but TSP/TCS notice that something is wrong.
TSS_E_COMM_FAILURE	A communications error with the TPM has been detected.
TSS_E_TPM_UNSUPPORTED_FEATURE	The TPM does not support the requested feature.

2.4.3 Common Return Code Rules

The above return codes may be returned by any function or method from any TSS layer. Other error codes MUST be explicitly stated within the respective function or specified in the layer-specific return code rules sections below.

2.5 OS Specific Considerations

2.5.1 OS Specific Error Information:

Start of informative comment

Unless specified below, the OS specific error bits for systems should be set to all 0's

End of informative comment

2.5.1.1 Windows Operating System:

Note: the Sev, S, C, N, R, and Facility fields are defined by the Windows environment.

CAPI Error Codes:

```
 3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
+---+-+-+-----------------------+-------------------------------+
|Sev|C|R|    Facility           | Layer    |       Code         |
+---+-+-+-----------------------+-------------------------------+
```

Type	Description
Sev	Severity Code. 00 – Success 01 – Informational 10 – Warning 11 = Error
C	Customer Code Flag. (1)
R	Reserved Bit. (0)
Facility	Facility Code. (0x028)
Layer	TSS layer information
Code	TSS error code value

2.5.1.1.1 COM Error Codes:

HRESULTs are 32 bit values layed out as follows:

```
 3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
+-+-+-+-+-+---------------------+-------------------------------+
|S|R|C|N|r|    Facility         | Layer    |     Code           |
+-+-+-+-+-+---------------------+-------------------------------+
```

Type	Description
S (Severity)	Indicates Success or Fail. 0 - Success. 1 - Fail (COERROR).
R	Reserved portion of the facility code, corresponds to NT's second severity

	bit. (1)
C	Reserved portion of the facility code, corresponds to NT's C field. (1)
N	Reserved portion of the facility code. Used to indicate a mapped NT status value. (0)
r	Reserved portion of the facility code. Reserved for internal use. Used to indicate HRESULT values that are not status values, but are instead message IDs to display strings. (0)
Facility	Facility Code. (0x028)
Layer	TSS layer information
Code	TSS error code value

2.5.1.1.2 ErrorSample

Start of Example:

Sample usage of error codes.

```
int main(int argc, char* argv[])
{
    TSS_RESULT dwTssRetCode = TSS_E_UNKNOWN_ERROR;
    /* pseudo code section to get the TPM object at the TSP global context handle */
    do
    {
        TSS_HTPMhTPM = NULL;
        BYTE      *pRandomData = NULL;
        // hContext is the global TSP-Context handle
        dwTssRetCode = Tspi_Context_GetTPMObject(hContext, &hTPM);

        // test the return code of the GetTPMObject function (TSS-ReturnCode)
        if (dwTssRetCode == TSS_E_BAD_PARAMETER)
        {
            // bad parameter for the GetTPMObject call
            // at the TSP interface
            // e.g. hContext not set
            break;
        }
        else
            if (dwTssRetCode != TSS_SUCCESS);
                break;
        // call the GetRandom method at the TPM object of the current context
        dwTssRetCode = Tspi_TPM_GetRandom(hTPM, 64, &pRandomData);

        if (dwTssRetCode == TPM_E_DISABLED_CMD)         (TPM-ReturnCode)
        {
            // GetRandom command not enabled at the TPM device
            // start enable sequence for the device
        }
        else
```

```
            if (dwTssRetCode != TSS_SUCCESS);
                break;
    } while(FALSE);
    // free local resources and...
    return dwTssRetCode;
}
```

End of example:

2.5.2 Unicode considerations

Start of Informative Comment

While Microsoft Windows uses Unicode UTF-16, other OSs may use UTF-8. In this specification UTF-16LE is specified. For compatibility of keys between OSs as well as platforms, OSs that use UTF-8 must convert strings to UTF-16LE before hashing them for use as authorization data. Note that UTF-16LE is an array of 16-bit values without a byte order mark, which may be found in strings encoded using UTF-16

End of Informative Comment

2.6 Structures

2.6.1 TSS_VERSION

Start of informative comment:

This structure allows the TSS Service Provider to communicate to outside entities the version of the TPM, TSS Core Service or TSS ServiceProvider.

End of informative comment.

Definition:

```
typedef struct tdTSS_VERSION
{
   BYTE  bMajor;
   BYTE  bMinor;
   BYTE  bRevMajor;
   BYTE  bRevMinor;
} TSS_VERSION;
```

Parameters:

bMajor

This SHALL be the major version indicator for this implementation of the TSS specification. For version 1 this must be 0x01

bMinor

This SHALL be the minor version indicator for this implementation of the TSS specification. For version 1.1b this must be 0x01, for version 1.2, this must be 0x02.

bRevMajor

This SHALL be the major value of the TSS vendor's implementation. The value of this is left to the TSS vendor to determine.

bRevMinor

This SHALL be the minor value of the TSS vendor's implementation. The value of this is left to the TSS vendor to determine.

Remarks:

The version points to the version of the specification that defines the structure or a service.
If the validity of a structure depends on conformity to a version of the specification and/or to a version of the TSS, that structure will include the current instance of TSS_VERSION

2.6.2 TSS_PCR_EVENT

Start of informative comment:

This structure provides information about an individual PCR extend event.

End of informative comment.

Definition:

```
typedef struct tdTSS_PCR_EVENT
{
   TSS_VERSION      versionInfo;
   UINT32           ulPcrIndex;
   TSS_EVENTTYPE    eventType;
   UINT32           ulPcrValueLength;
   BYTE*            rgbPcrValue;
   UINT32           ulEventLength;
   BYTE*            rgbEvent;
} TSS_PCR_EVENT;
```

Parameters:

versionInfo

Version data set by the TSP.

ulPcrIndex

Index of the PCR this event belongs to set by the TSP.

eventType

Flag indicating the type of the event (see section 2.3.2.32 for definition).

ulPcrValueLength

The length (in bytes) of the *rgbPcrValue* parameter set by the TSP

rgbPcrValue

Pointer to memory containing the value extended into the TPM by Tspi_TPM_PcrExtend. This SHALL be the result of the calculation of

SHA-1(ulPcrIndex || pbPcrData || eventType || rgbEvent), where ulPcrIndex and pbPcrData are passed as parameters to the Tspi_TPM_PcrExtend command. Note that 1.1 TSPs may calculate this parameter as SHA-1(ulEventLength || ulPcrIndex || rgbEvent || eventType), where ulPcrIndex is passed as a parameter to the Tspi_TPM_PcrExtend command.

ulEventLength

The length (in bytes) of the *rgbEvent* parameter

rgbEvent

Pointer to the event information data.

2.6.3 TSS_EVENT_CERT

Start of informative comment:

Certificate structure to use for events of type *TSS_EV_CODE_CERT*.

End of informative comment.

Definition

```
typedef struct tdTSS_EVENT_CERT
{
   TSS_VERSION  versionInfo;
   UINT32       ulCertificateHashLength
   BYTE*        rgbCertificateHash;
   UINT32       ulEntityDigestLength
   BYTE*        rgbEntityDigest;
   TSS_BOOL     fDigestChecked;
   TSS_BOOL     fDigestVerified;
   UINT32       ulIssuerLength;
   BYTE*        rgbIssuer;
) TSS_EVENT_CERT;
```

Parameters

versionInfo

Version data.

ulCertificateHashLength

The length (in bytes) of the *rgbCertificatHash* parameter

rgbCertificateHash

Pointer to memory containing the hash value of the entire VE certificate

ulEntityDigestLength

The length (in bytes) of the *rgbEntityDigest* parameter

rgbEntityDigest

Pointer to memory containing the actual digest value of the entity

fDigestChecked

TRUE if the entity logging this event checked the measured value against the digest value in the certificate.

FALSE if no checking was attempted.

fDigestVerified

Only valid when *fDigestChecked* is TRUE.

TRUE if measured value matches digest value in certificate, FALSE otherwise.

ulIssuerLength

The length (in bytes) of the *rgbIssuer* parameter *rgbIssuer*

Pointer to actual issuer certificate.

2.6.4 TSS_UUID

Start of informative comment:

This structure provides information about a UUID identifier that is unique within a particular key hierarchy for a given platform. Several UUIDs are reserved for particular keys, such as the SRK. These UUIDs are used to register keys in the persistent storage of the TSS Key Manager. This is specified in accordance to IEEE 802.

End of informative comment.

Definition:

```
typedef struct tdTSS_UUID
{
   UINT32  ulTimeLow;
   UINT16  usTimeMid;
   UINT16  usTimeHigh;
   BYTE    bClockSeqHigh;
   BYTE    bClockSeqLow;
   BYTE    rgbNode[6];
} TSS_UUID;
```

Parameters

ulTimeLow

The low field of the timestamp.

usTimeMid

The middle field of the timestamp.

usTimeHigh

The high field of the timestamp multiplexed with the version number.

bClockSeqHigh

The high field of the clock sequence multiplexed with the variant.

bClockSeqLow

The low field of the clock sequence.

rgbNode

The spatially unique node identifier.

Remarks

2.6.5 TSS_KM_KEYINFO

Start of informative comment:

The TSS_KM_KEYINFO structure provides information about a key registered in the TSS Persistent Storage.

End of informative comment.

Definition

```
typedef struct tdTSS_KM_KEYINFO
{
   TSS_VERSION   versionInfo;
   TSS_UUID      keyUUID;
   TSS_UUID      parentKeyUUID;
   BYTE          bAuthDataUsage;
   TSS_BOOL      fIsLoaded;          // TRUE: actually loaded in TPM
   UINT32        ulVendorDataLength;// may be 0
   BYTE*         rgbVendorData;      // may be NULL
} TSS_KM_KEYINFO;
```

Parameters

versionInfo

Version data.

keyUUID

The UUID the key is registered in the persistent storage of the TSS Key Manager.

parentKeyUUID

The UUID the parent key which wraps the key addressed by *keyUUID* is registered in the persistent storage of the TSS Key Manager.

bAuthDataUsage

Flag indicating whether key usage requires authorization or not. Currently the values 0x00 and 0x01 are defined. The value 0x00 means usage of the key without authorization is permitted. The value 0x01 means that on each usage of the key the authorization must be performed. All other values are reserved for future use.

fIsLoaded

Flag indicating the key is loaded into the TPM.

TRUE: Key is loaded into the TPM.

FALSE: Key is not loaded into the TPM.

ulVendorDataLength

Supplies the length (in bytes) of the *rgbVendorData* parameter. Set to 0 if this data is not of interest.

rgbVendorData

Pointer to vendor specific data.
Set to NULL if data is not of interest.

Remarks

When calling GetRegisteredKeysByUUID, the TSS_KM_KEYINFO structure returned for the SRK MUST either designate the parent UUID as the SRK's UUID.

2.6.6 TSS_KM_KEYINFO2

Start of informative comment:

The TSS_KM_KEYINFO2 structure provides information about a key registered in the TSS Persistent Storage. This structure is identical to TSS_KM_KEYINFO except that it additionally includes the key's storage type.

End of informative comment.

Definition

```
typedef struct tdTSS_KM_KEYINFO2
{
   TSS_VERSION   versionInfo;
   TSS_UUID      keyUUID;
   TSS_UUID      parentKeyUUID;
   BYTE          bAuthDataUsage;
   TSS_FLAG      persistentStorageType;
   TSS_FLAG      persistentStorageTypeParent;
   TSS_BOOL      fIsLoaded;         // TRUE: actually loaded in TPM
   UINT32        ulVendorDataLength;// may be 0
   BYTE*         rgbVendorData;     // may be NULL
} TSS_KM_KEYINFO2;
```

Parameters

versionInfo

Version data.

keyUUID

The UUID the key is registered in the persistent storage of the TSS Key Manager.

parentKeyUUID

The UUID the parent key which wraps the key addressed by *keyUUID* is registered in the persistent storage of the TSS Key Manger.

bAuthDataUsage

Flag indicating whether key usage requires authorization or not. Currently the values 0x00 and 0x01 are defined. The value 0x00 means usage of the key without authorization is permitted. The value 0x01 means that on each usage of the key the authorization must be performed. All other values are reserved for future use.

persistentStorageType

Flag indicating the persistent storage (see section 2.3.2.22) the key is registered in.

persistentStorageTypeParent

Flag indicating the persistent storage (see section 2.3.2.22) of the parent key.

fIsLoaded

Flag indicating the key is loaded into the TPM.

TRUE: Key is loaded into the TPM.

FALSE: Key is not loaded into the TPM.

ulVendorDataLength

Supplies the length (in bytes) of the *rgbVendorData* parameter.

Set to 0 if this data is not of interest.

rgbVendorData

Pointer to vendor specific data.

Set to NULL if data is not of interest.

Remarks

When calling GetRegisteredKeysByUUID2, the TSS_KM_KEYINFO2 structure returned for the SRK MUST designate the parent UUID as the SRK's UUID.

2.6.7 TSS_VALIDATION

Start of informative comment:

The TSS_VALIDATION structure provides the ability to verify signatures and validation digests built over certain TPM command parameters. These parameters are returned as a byte stream and are defined within the TCPA 1.2 Main Specification. The caller must provide some random data (external Data value) as input, which is included in the signature/digest calculation.

The following functions use this structure:

Tspi_TPM_CertifySelfTest,
Tspi_TPM_GetCapabilitySigned,
Tspi_TPM_LoadMaintenancePubKey,
Tspi_TPM_CheckMaintenancePubKey,
Tspi_Key_CertifyKey,
Tspi_TPM_CreateEndorsementKey,

Tspi_TPM_GetPubEndorsementKey

Tspi_TPM_CreateRevocableEndorsementKey
Tspi_TPM_Quote

Tspi_TPM_Quote2

Tspi_Context_CloseSignTransport

If the validation of the signature/digest should be done by the TSS Service Provider itself, a NULL pointer must be passed to these methods. In this case the TSS Service Provider generates its own random data to be included in the signature/digest (external Data value).

End of informative comment.

Definition

```
typedef struct tdTSS_VALIDATION
{
   TSS_VERSION   versionInfo;
   UINT32        ulExternalDataLength;
   BYTE*         rgbExternalData;
   UINT32        ulDataLength;
   BYTE*         rgbData;
   UINT32        ulValidationLength;
   BYTE*         rgbValdationData;
} TSS_VALIDATION;
```

Parameters

versionInfo

Version data.

ulExternalData

The length (in bytes) of the *rgbExternalData* parameter

rgbExternalData

Pointer to memory containing the random data used to avoid replay attacks.

ulDataLength

Supplies the length (in bytes) of the *rgbData* parameter.

rgbData

Pointer to the data which was used to calculate the validation.

ulValidationLength

Supplies the length (in bytes) of the *rgbValidationData* parameter.

rgbValidationData

Pointer to the validation data.

2.6.8 TPM_COUNTER_VALUE

Start of informative comment:

This structure returns the counter value. For interoperability, the value size should be 4 bytes.

End of informative comment.

Definition

```
typedef struct tdTPM_COUNTER_VALUE
{
   TPM_STRUCTURE_TAG  tag;
   BYTE               label[4];
   TPM_ACTUAL_COUNT   counter;
} TPBM_COUNTER_VALUE;
```

Parameters

tag

For a counter this has value 0x000E, per TPM mainP2Structure Spec 1.2, Section 3.1. Per standards, this is assigned to TPM_COUNTER_VALUE

label

The is the label for the counter (4 bytes)

counter

The is the 32 bit counter value.

2.6.9 TSS_CALLBACK

Start of informative comment:

This structure holds the address of a callback function as well as what algorithm will be used as the mask function if one is used..

End of informative comment.

Definition

```
typedef struct tdTSS_CALLBACK
{
   PVOID             callback;
   PVOID             appData;
   TSS_ALGORITHM_ID  alg;
} TSS_CALLBACK;
```

Parameters

callback

The address of a callback function.

appData

A pointer to applicaiton provide data. This pointer will be passed to the "lpAppData" parameter when the clalback is invoked. This pointer will not be interpreted, dereferenced or freed by the TSS.

alg

The symmetic algorithm to be used for masking data if that is chosen.

Remarks

For applications that wish to be compatible with version 1.1 of the TSS specification, Tspi_SetAttribUint32 should be used to set all callbacks. For all others, Tspi_SetAttribData should be used with a TSS_CALLBACK structure. See section TSPI_POLICY CLASS Definition section 4.3.4.3 and section TSPI_TPM CLASS Definition section for more information on setting callbacks.

In order to set a 64bit callback, an application should:

1. Set the address of the callback in the callback variable of a TSS_CALLBACK structure.
2. Call Tspi_SetAttribData, passing the appropriate Flag parameter and the address of the TSS_CALLBACK structure as rgbAttribData.
3. The TSS will then set the callback by pulling the callback variable out of the TSS_CALLBACK structure.

2.6.10 TSS_PLATFORM_CLASS

Start of informative comment:

This structure identifies a class of a platform. These classes are defined by the TCG administration. There are two values returned.

The first is a simple integer value (actually unsigned integer) which is an index or reference number identifying the platform class. This value is maintained by the TCG administrator and provides a reference to the TCG Platform Specific Specification defining this class.

The second value is the URI. This is a reference to either the platform manufacturer or is a link to the platform specific specification maintained on the TCG web site. It may also be NULL indicating no specific platform is indicated beyond information provided by the value returned in platformClassSimpleIdentifier.

The source of this information is not defined in this specification. It is likely the TCS is at least compiled if not specifically written for this a particular host platform. In this case, the identification of the host platform could be compiled into the TCS itself. Other cases may require the TCS to “discover” that Host platform’s class. Further, the TCS may be provided by the Host Platform manufacturer. In this situation, the TCS may provide a link to the Host Platform’s manufacturer in the platformCallURI. If not, this value may be a link to the TCG site’s location of the platform specific specification or this value may be NULL.

The Host Platform may contain other platforms within it. An example would be a peripheral with a TPM. In this case the TCG-enabled peripheral would be considered a platform that would be listed using the sub-cap TSS_TCSCAP_PROP_ALL_PLATFORMS.

The values for platformClassSimpleIdentifier are not maintained as part of this specification, rather, they are maintained by the TCG administrator.

End of informative comment.

Definition:

```
typedef struct tdTSS_PLATFORM_CLASS
{
   UINT32             platformClassSimpleIdentifier;
   UINT32             ulPlatformClassURISize;
   BYTE[]             pPlatformClassURI;
} TSS_PLATFORM_CLASS;
```

Parameters

platformClassSimpleIdentifier

The value defining the platform as defined by and registered with TCG Administration.

ulPlatformURISize

The size in bytes of the platformURI parameter

pPlatformURI

The reference to either the platform manufacturer or if the TCS provider cannot or doesn't want to disclsoe the specific platform manufacturer, this is either NULL or contains a reference to the platform specific specification on the TCG web site.

3. DAA Structures

Overview

Structure name	External	Internal	Exportable	TSS_HDAA_CREDENTIAL	TSS_HDAA_ISSUER_KEY	TSS_HDAA_ARA_KEY
TSS_DAA_KEY_PAIR	X	X	X		X	
TSS_DAA_PK	X		X		X	
TSS_DAA_PK_PROOF	X		X		X	
TSS_DAA_SK		X	X		X	
TSS_DAA_AR_PK	X		X			X
TSS_DAA_AR_SK		X	X			X
TSS_DAA_AR_KEY_PAIR	X	X	X			X
TSS_DAA_CRED_ISSUER	X		X			
TSS_DAA_CREDENTIAL		X	X	X		
TSS_DAA_SELECTED_ATTRIB	X		X	X		
TSS_DAA_SIGNATURE	X		X			
TSS_DAA_SIGN_CALLBACK		X	X			
TSS_DAA_ATTRIB_COMMIT	X	X	X	X		
TSS_DAA_ATTRIB_COMMIT_PARAM	X	X	X	X		
TSS_DAA_PSEUDONYM	X	X	X			
TSS_DAA_PSEUDONYM_PLAIN	X		X			
TSS_DAA_PSEUDONYM_ENCRYPTED	X		X			
TSS_DAA_IDENTITY_PROOF	X		X			

External means that the structure is also used outside of the entity where it is created, e.g., DAA Issuer public key (TSS_DAA_PK) is used by TCG Platform. External includes naturally internal use.

External and Internal means that the structure is used at multiple places explicitly either internal or external.

Internal and Exportable means that the structure needs to be exportable to other TSS implementations.

Internal applies also to remote TSS calls, however the secrecy must be preserved.

For the following structures and where possible, the fixed byte length of each basic data type is annotated.

3.1 TSS_DAA_PK

Start of informative comment:

DAA Public key of DAA Issuer.

End of informative comment.

Definition

```
typedef struct tdTSS_DAA_PK
{
  TSS_VERSION            versionInfo;
  UINT32                 modulusLength;            // l_n/8
  BYTE*                  modulus;                  // n
  UINT32                 capitalSLength;           // l_n/8
  BYTE*                  capitalS;                 // S
  UINT32                 capitalZLength;           // l_n/8
  BYTE*                  capitalZ;                 // Z
  UINT32                 capitalR0Length;          // l_n/8
  BYTE*                  capitalR0;                // R_0
  UINT32                 capitalR1Length;          // l_n/8
  BYTE*                  capitalR1;                // R_1
  UINT32                 gammaLength;              // l_Γ/8
  BYTE*                  gamma;                    // γ
  UINT32                 capitalGammaLength;       // l_Γ/8
  BYTE*                  capitalGamma;             // Γ
  UINT32                 rhoLength;                // l_P/8
  BYTE*                  rho;                      // ρ
  UINT32                 capitalYLength;           // l_h+l_i
  UINT32                 capitalYLength2;          // l_n/8
  BYTE**                 capitalY;              //  Y_0,..,Y_{l_n+l_i-1}
  UINT32                 issuerBaseNameLength;  // (dynamic)
  BYTE*                  issuerBaseName;        //  bsn
  UINT32                 numPlatformAttributes     // l_h
  UINT32                 numIssuerAttributes       // l_i
} TSS_DAA_PK;
```

Parameters

versionInfo

Version data set by the TSP, only including the TSS level, not the manufacturer's level of TSS implementation due to privacy concerns.

modulusLength

Length of n

modulus

n

capitalSLength

Length of capitalS

capitalS

S

capitalZLength

Length of capitalZ

capitalZ

Z

capitalR0Length

Length of capitalR0

capitalR0

R0

capitalR1Length

Length of capitalR1

capitalR1

R1

gammaLength

Length of gamma

gamma

gamma

capitalGammaLength

Length of capitalGamma

capitalGamma

Gamma

rhoLength

Length of rho

rho

rho

capitalSprimeLength

Length of capitalSprime

capitalSprime

S'

capitalYLength

Number of elements in the capitalY array. This should equal *numPlatformAttributes + numIssuerAttributes.*

capitalYLength2

Length of an element in the capitalY array.

capitalY

An array of cryptographic values which allow encoding of attributes into the DAA Credential.

issuerBaseNameLength

Length of the issuerBaseName.

issuerBaseName

Label (base name) of DAA Issuer

numPlatformAttributes

Number of attributes that can be chosen by the platform

numIssuerAttributes

Number of attributes that can be chosen by the issuer

3.2 TSS_DAA_PK_PROOF

Start of informative comment:

Proofs the correctness of the DAA public key (TSS_DAA_PK) of the DAA Issuer.

End of informative comment.

Definition

```
typedef struct tdTSS_DAA_PK_PROOF
{
   TSS_VERSION              versionInfo;
   UINT32                   challengeLength; //  l_H/8
   BYTE*                    challenge;       //  c
   UINT32                   responseLength;  //  (l_n/8) l_H (3+l_h+l_i)
                                                 (dynamic)
   BYTE**                   response;        //  x̂(i, j)
} TSS_DAA_PK_PROOF;
```

Parameters

versionInfo

Version data set by the TSP, only including the TSS level, not the manufacturer's level of TSS implementation due to privacy concerns.

challengeLength

Length of challenge (20 bytes - - I $l_H/8$).

challenge

“Challenge” for proof.

responeLength

Length of response (dynamic)

response

An array of cryptographic values representing the response to the challenge which proofs the correctness of the public key variables $Z, R_{0,} R_{1,} Y_i$. The byte length of one element of the array is $l_n/8$.

3.3 TSS_DAA_SK

Start of informative comment:

DAA private key of DAA Issuer

End of informative comment.

Definition

```
typedef struct tdTSS_DAA_SK
{
   TSS_VERSION             versionInfo;
   UINT32                  productPQprimeLength; // l_n/8
   BYTE*                   productPQprime;       // p'q'
} TSS_DAA_SK;
```

Parameters

versionInfo

Version data set by the TSP, only including the TSS level, not the manufacturer's level of TSS implementation due to privacy concerns.

productPQprimeLength

Length of productPQprime

productPQprime

The private key data of the DAA Issuer key which is the product of p’ and q’.

3.4 TSS_DAA_KEY_PAIR

Start of informative comment:

DAA Issuer key pair containing private/secret and public key.

End of informative comment.

Definition

```
typedef struct tdTSS_DAA_KEY_PAIR
{
   TSS_VERSION              versionInfo;
   TSS_DAA_SK               secretKey;
   TSS_DAA_PK               publicKey;
} TSS_DAA_KEY_PAIR;
```

Parameters

versionInfo

Version data set by the TSP, only including the TSS level, not the manufacturer's level of TSS implementation due to privacy concerns.

secretKey

Secret/private DAA Issuer key

publicKey

Public DAA Issuer key

3.5 TSS_DAA_AR_PK

Start of informative comment:

The public key of Anonymity Revocation (AR) Authority.

End of informative comment.

Definition

```
typedef struct tdTSS_DAA_AR_PK
{
    TSS_VERSION         versionInfo;
    UINT32              etaLength;        // l_Γ /8
    BYTE*               eta;              // η
    UINT32              lambda1Length;    // l_Γ /8
    BYTE*               lambda1;          // λ1
    UINT32              lambda2Length;    // l_Γ /8
    BYTE*               lambda2;          // λ2
    UINT32              lambda3Length;    // l_Γ /8
    BYTE*               lambda3;          // λ3
} TSS_DAA_AR_PK;
```

Parameters

versionInfo

Version data set by the TSP, only including the TSS level, not the manufacturer's level of TSS implementation due to privacy concerns.

etaLength

Length of eta

eta

Eta

lambda1Length

Length of lambda1

lambda1

Lambda1

lambda2Length

Length of lambda2

lambda2

Lambda2

lambda3Length

Length of lambda3

lambda3

Lambda3

3.6 TSS_DAA_AR_SK

Start of informative comment:

The private key of Anonymity Revocation (AR) Authority.

End of informative comment.

Definition

```
typedef struct tdTSS_DAA_AR_SK
{
    TSS_VERSION             versionInfo;
    UINT32                  x0Length;  // l_ρ/8
    BYTE*                   x0;        // x_0
    UINT32                  x1Length;  // l_ρ/8
    BYTE*                   x1;        // x_1
    UINT32                  x2Length;  // l_ρ/8
    BYTE*                   x2;        // x_2
    UINT32                  x3Length;  // l_ρ/8
    BYTE*                   x3;        // x_3
    UINT32                  x4Length;  // l_ρ/8
    BYTE*                   x4;        // x_4
    UINT32                  x5Length;  // l_ρ/8
    BYTE*                   x5;        // x_5
} TSS_DAA_AR_SK;
```

Parameters

versionInfo

Version data set by the TSP, only including the TSS level, not the manufacturer's level of TSS implementation due to privacy concerns.

X0Length

Length of x0

X0

x0

x1Length

Length of x1

x1

x1

x2Length

Length of x2

x2

x2

x3Length

Length of x3

x3

x3

x4Length

Length of x4

x4

x4

x5Length

Length of x5

x5

x5

3.7 TSS_DAA_AR_KEY_PAIR

Start of informative comment:

DAA Anonymous Revocation Authority key pair containing private/secret and public key.

End of informative comment.

Definition

```
typedef struct tdTSS_DAA_AR_KEY_PAIR
{
  TSS_VERSION              versionInfo;
  TSS_DAA_AR_SK            secretKey;
  TSS_DAA_AR_PK            publicKey;
} TSS_DAA_AR_KEY_PAIR;
```

Parameters

versionInfo

Version data set by the TSP, only including the TSS level, not the manufacturer's level of TSS implementation due to privacy concerns.

secretKey

Secret/private DAA ARA key

publicKey

Public DAA ARA key

3.8 TSS_DAA_CRED_ISSUER

Start of informative comment:

DAA credential message of the DAA Issuer to the TCG Platform, including proof of correctness of the credential.

End of informative comment.

Definition

```
typedef struct tdTSS_DAA_CRED_ISSUER
{
   TSS_VERSION          versionInfo;
   UINT32               capitalALength;        // l_n/8
   BYTE*                capitalA;              // A
   UINT32               eLength;               // l_e/8
   BYTE*                e;                     // e
   UINT32               vPrimePrimeLength;     // l_v/8
   BYTE*                vPrimePrime;           // v''
   UINT32               attributesIssuerLength;  //            l_i
                              (dynamic)
   BYTE**               attributesIssuer;      // a_{l_h}, ..., a_{l_h+l_i-1}
   UINT32               cPrimeLength;          // l_H/8
   BYTE*                cPrime;                // c'
   UINT32               sELength;              // l_ρ/8
   BYTE*                sE;                    // s_e
} TSS_DAA_CRED_ISSUER;
```

Parameters

versionInfo

Version data set by the TSP, only including the TSS level, not the manufacturer's level of TSS implementation due to privacy concerns.

capitalALength

Length of capitalA.

capitalA

A cryptographic value forming the credential.

eLength

Length of e.

e

A cryptographic value forming the credential.

vPrimePrimeLength

Length of vPrimePrime

vPrimePrime

A cryptographic value forming the credential.

attributesIssuerLength

Length of attributesIssuer array. The length of the byte arrays representing a single attribute is determined by the isserPk.

attributesIssuer

An array of attributes encoded into the DAA Credential which are visible to the DAA Issuer.

cPrimeLength

Length of cPrime (20 bytes - - $l_H/8$).

cPrime

A cryptographic value forming the proof of correctness of the credential.

sELength

Length of sE.

sE

A cryptographic value forming the proof of correctness of the credential.

3.9 TSS_DAA_CREDENTIAL

Start of informative comment:

Final DAA Credential issued by the DAA Issuer to the TCG Platform. It includes all data to compute a DAA Signature in the DAA Sign protocol.

End of informative comment.

Definition

```
typedef struct tdTSS_DAA_CREDENTIAL
{
    TSS_VERSION          versionInfo;
    UINT32               capitalALength;          // l_n/8
    BYTE*                capitalA;                // A
    UINT32               exponentLength;          // l_i/8
    BYTE*                exponent;                // e
    UINT32               vBar0Length;             // (dynamic)
    BYTE*                vBar0;                   // v̄_0
    UINT32               vBar1Length;             // (dynamic)
    BYTE*                vBar1;                   // v̄_1
    UINT32               attributesLength;        //  (dynamic)
    BYTE**               attributes;              // l l_i
    TSS_DAA_PK           issuerPK;                // a_0,..,a_{l_h+l_i-1}
    UINT32               tpmSpecificEncLength;    // (dynamic)
    BYTE*                tpmSpecificEnc;
    UINT32               daaCounter;

} TSS_DAA_CREDENTIAL;
```

Parameters

versionInfo

Version data set by the TSP, only including the TSS level, not the manufacturer's level of TSS implementation due to privacy concerns.

capitalALength

Length of capitalA.

capitalA

A cryptographic value forming the credential.

exponentLength

Length of exponent.

exponent

A cryptographic value forming the credential.

vBar0Length

Length of vBar0 (Actual length is TPM vendor specific)

vBar0

Encrypted internal data of the TPM, which represent secret data of the credential.

vBar1Length

Length of vBar1 (Actual length is TPM vendor specific)

vBar1

Encrypted internal data of the TPM, which represent secret data of the credential.

attributesLength

Length of the attributes array. The length of the byte arrays representing a single attribute is l_f.

attributes

Array of all the attributes encoded into the DAA Credential.

issuerPK

DAA Issuer public key.

tpmSpecificEncLength

Length of tpmSpecificEnc (Actual length is TPM vendor specific)

tpmSpecificEnc

Encrypted internal data of the TPM, which represent secret data of the credential (key).

daaCounter

The daaCounter that was used to issue this DAA Credential. This value might be reused in case a different credential from the same DAA Issuer is requested. However, this value is not used by the TSS during the DAA Sign protocol.

3.10 TSS_DAA_CREDENTIAL_REQUEST

Start of informative comment:

DAA public key of the platform and other protocol relevant data to be received by the DAA Issuer

End of informative comment.

Definition

```
typedef struct tdTSS_DAA_CREDENTIAL_REQUEST
{
    TSS_VERSION          versionInfo;
    UINT32               capitalULength;              // $l_n/8$
    BYTE*                capitalU;                    // $U$
    UINT32               capitalNiLength;             // $l_\Gamma/8$
    BYTE*                capitalNi;                   // $N_I$
    UINT32               authenticationProofLength;   // $l_H/8$
    BYTE*                authenticationProof;         // $a_{U'}$
    UINT32               challengeLength;             // $l_H/8$
    BYTE*                challenge;                   // $c$
    UINT32               nonceTpmLength;              // $l_H/8$
    BYTE*                nonceTpm;                    // $n_t$
    UINT32               noncePlatformLength;         // $l_H/8$
    BYTE*                noncePlatform;               // $n_h$
    UINT32               sF0Length;                   // $(l_f+l_\Phi+l_H)/8+1$
    BYTE*                sF0;                         // $s_{f_0}$
    UINT32               sF1Length;                   // $(l_f+l_\Phi+l_H)/8+1$
    BYTE*                sF1;                         // $s_{f_1}$
    UINT32               sVprimeLength;               // $(l_f+l_\Phi+l_H)/8+1$
    BYTE*                sVprime;                     // $s_{v'}$
    UINT32               sVtildePrimeLength;          // $(l_f+l_\Phi+l_H)/8+1$
    BYTE*                sVtildePrime;                // $s_{\tilde{v}'}$
    UINT32               sALength;                    // $l_h$ (dynamic)
    BYTE**               sA;                          // $s_{a_0},\dots,s_{a_{l_h-1}}$
    UINT32               attributeCommitmentsLength;  // $l_c$
    TSS_DAA_ATTRIB_COMMIT* attributeCommitments       // $s_\mu$
} TSS_DAA_CREDENTIAL_REQUEST;
```

Parameters

versionInfo

Version data set by the TSP, only including the TSS level, not the manufacturer's level of TSS implementation due to privacy concerns.

capitalU

Public key of platform.

capitalNi

Pseudonym with DAA Issuer

authenticationProof

Decryption of the encrypted nonce

challenge

Challenge for DAA Issuer

nonceTpm

Nonce of TPM

noncePlatform

Nonce of Platform TSS

sFo

Part of correctness proof of f0

sF1

Part of correctness proof of f1

sVprime

Part of correctness proof

sVtildePrime

Part of correctness proof

sALength

Length of array of sA.

sA

Array of sA. The byte length of an array element is $(l_f + l_\Phi + l_H)/8+1$.

attributeCommitmentsLength

Length of attributeCommitments

attributeCommitments

Array of correctness proofs of commitments

3.11 TSS_DAA_SELECTED_ATTRIB

Start of informative comment:

This structure specifies a list of indices used to selected attributes of the DAA Credential.

End of informative comment.

Definition

```
typedef struct tdTSS_DAA_SELECTED_ATTRIB
{
   TSS_VERSION          versionInfo;
   UINT32               indicesListLength;    // l_h+l_i (dynamic)
   TSS_BOOL*            indicesList;          // X or C
} TSS_DAA_SELECTED_ATTRIB;
```

Parameters

versionInfo

Version data set by the TSP, only including the TSS level, not the manufacturer's level of TSS implementation due to privacy concerns.

indicesListLenth

Length of indicesList which is equal to the number of possible attributes of a DAA Credential.

indicesList

A list of Boolean indices. An index with the value 'TRUE' denotes a selected attribute.

3.12 TSS_DAA_SIGNATURE

Start of informative comment:

DAA signature of the TCG Platform to be verified by the DAA Verifier.

End of informative comment.

Definition

```
typedef struct tdTSS_DAA_SIGNATURE
{
  TSS_VERSION           versionInfo;
  UINT32                zetaLength;             // l_Γ/8
  BYTE*                 zeta;                   // ζ
  UINT32                capitalTLength;         // l_n/8
  BYTE*                 capitalT;               // T
  UINT32                challengeLength;        // l_H/8
  BYTE*                 challenge;              // c
  UINT32                nonceTpmLength;         // l_H/8
  BYTE*                 nonceTpm;               // n_t
  UINT32                sVLength;               // (l_ν + l_Φ + l_H)/8+1
  BYTE*                 sV;                     // s_ν
  UINT32                sF0Length;              // (l_ν + l_Φ + l_H)/8+1
  BYTE*                 sF0;                    // s_f0
  UINT32                sF1Length;              // (l_ν + l_Φ + l_H)/8+1
  BYTE*                 sF1;                    // s_f1
  UINT32                sELength;               // (l_ν + l_Φ + l_H)/8+1
  BYTE*                 sE;                     // s_i
  UINT32                sALength;               //  (dynamic)
  BYTE**                sA;                     // (s_ai)_i∉X
  UINT32                attributeCommitmentsLength; //           l_c
                                   (dynamic)
  TSS_DAA_ATTRIB_COMMIT*   attributeCommitments;
  TSS_DAA_PSEUDONYM     signedPseudonym;
  TSS_DAA_SIGN_CALLBACK callbackResult
} TSS_DAA_SIGNATURE;
```

Parameters

versionInfo

Version data set by the TSP, only including the TSS level, not the manufacturer's level of TSS implementation due to privacy concerns.

zeta Length

Length of zeta

zeta

zeta

capitalTLength

Length of T

capitalT

T

challengeLength

Length of challenge (20 bytes - - $l_H/8$)

challenge

Challenge

nonceTpmLength

Length of nonceTpm (20 bytes - - $l_H/8$)

nonceTpm

Nonce of TPM

sVLength

Length of sV

sV

sV

sF0Length

Length of sF0

sF0

sF0

sF1Length

Length of sF1

sF1

sF1

sELength

Length of sE

sE

sE

sALength

Length of array of sA.

sA

Array of sA. The byte length of an array element is $(l_f + l_\Phi + l_H)/8 + 1$.

attributeCommitmentsLength

Length of the array of attributeCommitments.

attributeCommitments

Array of structure representing the commitments to selected attributes.

signedPsuedonym

Structure representing the pseudonym with respect to which the DAA Signature is produced.

3.13 TSS_DAA_SIGN_CALLBACK

Start of informative comment:

This structure is returned by the callback function Tspicb_DAA_Sign that allows proving additional properties of the DAA Credential.

End of informative comment.

Definition

```
typedef struct tdTSS_DAA_SIGN_CALLBACK
{
   TSS_VERSION          versionInfo;
   TSS_HHASH            challenge;
   TSS_FLAG             payloadFlag;
   UINT32               payloadLength;
   BYTE*                payload;
} TSS_DAA_SIGN_CALLBACK;
```

Parameters

versionInfo

Version data set by the TSP, only including the TSS level, not the manufacturer's level of TSS implementation due to privacy concerns.

challenge

Challenge of the additional proof

payloadFlag

Determines the payload type

payloadLength

Length of payload

payload

Payload that contains additional proof data.

3.14 TSS_DAA_ATTRIB_COMMIT

Start of informative comment:

Commitment to attributes that can be used, e.g., to verifiably encrypt the attributes or to prove the relation between an attribute and a constant value.

End of informative comment.

Definition

```
typedef struct tdTSS_DAA_ATTRIB_COMMIT
{
   TSS_VERSION          versionInfo;
   UINT32               betaLength;      // l_Γ/8
   BYTE*                beta;            β
   UINT32               sMuLength;    // l_ρ/8
   BYTE*                sMu;          // s_μ
} TSS_DAA_ATTRIB_COMMIT;
```

Parameters

versionInfo

Version data set by the TSP, only including the TSS level, not the manufacturer's level of TSS implementation due to privacy concerns.

betaLength

Length of beta

beta

Beta

sMuLength

Length of sMu. This value is zero when this structure is input as a TSS attribute to the Tspi_DAA_IssueCredential function.

sMu

Proofs correctness of commitment. This value is null when this structure is input as a TSS attribute to the Tspi_DAA_IssueCredential function.

3.15 TSS_DAA_ATTRIB_COMMIT_PARAM

Start of informative comment:

Parameter for a commitment on a attributes.

End of informative comment.

Definition

```
typedef struct tdTSS_DAA_ATTRIB_COMMIT_PARAM
{
    TSS_VERSION          versionInfo;
    UINT32               baseGammaLength;        // l_Γ/8
    BYTE*                baseGamma;              // γ
    UINT32               gammasLength;           //        l_h+l_i
                                (dynamic)
    BYTE*                gammas;                 // γ_0,..,γ_{l_h+l_i-1}
    UINT32               capitalGammaLength;     // l_Γ/8
    BYTE*                capitalGamma;           // Γ
    UINT32               rhoLength;              // l_ρ/8
    BYTE*                rho;                    // ρ
    UINT32               sizeMu;                 // l_μ
} TSS_DAA_ATTRIB_COMMIT_PARAM;
```

Parameters

versionInfo

Version data set by the TSP, only including the TSS level, not the manufacturer's level of TSS implementation due to privacy concerns.

baseGammaLength

Length of baseGamma

baseGamma

baseGamma

gammasLength

Length of the array of gammas

gammas

Array of gammas. A gamma for each attribute.

capitalGammaLength

Length of capitalGamma

capitalGamma

Modulus capitalGamma

rhoLength

Length of rho

rho

In case of an RSA based commitment parameter, rho will not be defined.

sizeMu

In case of an RSA based commitment parameter, sizeMu is defined.

3.16 TSS_DAA_PSEUDONYM

Start of informative comment:

Contains information about the pseudonym with respect to which the DAA Signature is produced for the Verifier. It allows containing the pseudonym itself or a verifiable encryption of the pseudonym. It is part of the DAA Signature structure TSS_DAA_SIGNATURE.

End of informative comment.

Definition

```
typedef struct tdTSS_DAA_PSEUDONYM
{
   TSS_VERSION                 versionInfo;
   TSS_FLAG                    payloadFlag;
   UINT32                      payloadLength;
   BYTE*                       payload;
} TSS_DAA_PSEUDONYM;
```

Parameters

versionInfo

Version data set by the TSP, only including the TSS level, not the manufacturer's level of TSS implementation due to privacy concerns.

payloadFlag

If payloadFlag == TSS_FLAG_DAA_PSEUDONYM_PLAIN, then payload is a structure of type TSS_DAA_PSEUDONYM_PLAIN
else if payloadFlag ==TSS_FLAG_DAA_PSEUDONYM_ENCRYPTED, then payload is a structure of type TSS_DAA_PSEUDONYM_ENCRYPTED.

payloadLength

Length of structure

payload

The structure representing the pseudonym, which is either the pseudonym itself (TSS_DAA_PSEUDONYM_PLAIN) or a verifiable encryption of it (TSS_DAA_PSEUDONYM_ENCRYPTED).

3.17 TSS_DAA_PSEUDONYM_PLAIN

Start of informative comment:

The pseudonym with respect to which the DAA Signature is produced.

End of informative comment.

Definition

```
typedef struct tdTSS_DAA_PSEUDONYM_PLAIN
{
   TSS_VERSION              versionInfo;
   UINT32                   capitalNvLength; // l_Γ/8
   BYTE*                    capitalNv;       // N_V
} TSS_DAA_PSEUDONYM_PLAIN;
```

Parameters

versionInfo

Version data set by the TSP, only including the TSS level, not the manufacturer's level of TSS implementation due to privacy concerns.

capitalNvLength

Length of capitalNv

capitalNv

N_v

3.18 TSS_DAA_PSEUDONYM_ENCRYPTED

Start of informative comment:

The verifiable encryption of the pseudonym with respect to which the DAA Signature is produced.

End of informative comment.

Definition

```
typedef struct tdTSS_DAA_PSEUDONYM_ENCRYPTED
{
   TSS_VERSION              versionInfo;
   UINT32                   delta1Length; // l_Γ/8
   BYTE*                    delta1;       // δ_1
   UINT32                   delta2Length; // l_Γ/8
   BYTE*                    delta2;       // δ_2
   UINT32                   delta3Length; // l_Γ/8
   BYTE*                    delta3;       // δ_3
   UINT32                   delta4Length; // l_Γ/8
   BYTE*                    delta4;       // δ_4
   UINT32                   sTauLength;      // l_ρ/8
   BYTE*                    sTau;         // s_τ
} TSS_DAA_PSEUDONYM_ENCRYPTED;
```

Parameters

versionInfo

Version data set by the TSP, only including the TSS level, not the manufacturer's level of TSS implementation due to privacy concerns.

delta1Length

Length of delta1

delta1

Delta1

delta2Length

Length of delta2

delta2

Delta2

delta3Length

Length of delta3

delta3

Delta3

delta4Length

Length of delta4

delta4

Delta4

sTauLength

Length of sTau

sTau

sTau

3.19 TSS_DAA_IDENTITY_PROOF

Start of informative comment:

This structure is similar to the TPM_IDENTITY_PROOF as it contains also the Endorsement, Platform and Conformance credentials. It fulfills a similar purpose - to convince the DAA Issuer to issue the DAA Credential.

End of informative comment.

Definition

```
typedef struct tdTSS_DAA_IDENTITY_PROOF
{
   TSS_VERSION             versionInfo;
   UINT32                  endorsementLength;
   BYTE*                   endorsementCredential;
   UINT32                  platformLength;
   BYTE*                   platform;
   UINT32                  conformanceLength;
   BYTE*                   conformance;
} TSS_DAA_IDENTITY_PROOF;
```

Parameters

versionInfo

Version data set by the TSP, only including the TSS level, not the manufacturer's level of TSS implementation due to privacy concerns.

endorsementLengthl

Length of endorsementCredential

endorsementCredential

The TPM endorsement credential

platformLengthl

Length of platformCredential

platformCredential

The TPM platform credential

conformanceLengthl

Length of conformance credential

conformanceCredential

The TPM conformanceCredential

3.20 DAA Error codes

DAA Error codes

TSS_E_DAA_ISSUER_KEY_ERROR	DAA Issuer's authentication key chain could not be verified or is not correct.
TSS_E_DAA_CREDENTIAL_PROOF_ERROR	Verification of the credential TSS_DAA _CRED_ISSUER issued by the DAA Issuer has failed.
TSS_E_DAA_AUTHENTICATION_ERROR	The TPM could not be authenticated by the DAA Issuer.

TSS_E_DAA_PSEUDONYM_ERROR	While verifying the pseudonym of the TPM, the private key of the TPM was found on the rogue list.
TSS_E_DAA_CREDENTIAL_REQUEST_PROOF_ERROR	Verification of the platform's credential request TSS_DAA_CREDENTIAL_REQUEST has failed.
TSS_E_DAA_AR_DECRYPTION_ERROR	Decryption of the encrypted pseudonym has failed, due to either a wrong secret key or a wrong decryption condition.

3.21 NonVolatile Memory Functions Definitions-Object Type Definitions

Attribute Definitions for a TPM Object

Persistent Storage

Start of informative comment:

Any data can be rendered confidential through encryption and protection of the key used for encryption.

Similarly, any signing authority can be protected if the signing key is protected. A service that protects keys is therefore useful, and sometimes essential. Similarly, it is useful, and sometimes essential; to provide a service that protects authorization data.

The TCG Software Stack enables such a service because a TPM can act as a portal to keep arbitrary amounts of data and keys confidential. “Protected Storage” is a set of commands provided by the TPM to enable virtual secure storage space.

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

For a user application the persistent storage looks like a data archive, therefore the main function set is adapted to data archiving function sets.

End of informative comment.

3.22 Key Management

Start of informative comment:

The Key Management Services of TSS allow definition of a persistent key hierarchy.

The Key Management Services interface was designed to allow a flexible key structure such that an instance like an IT department of an enterprise may define a deep key hierarchy, a shallow hierarchy, roaming keys, migration base keys, etc.

All keys, which should be internally managed by the Key Management Services of TSS must be registered in the persistent storage database of TCS (*System Persistent Storage*) or TSP (*User Persistent Storage*). Each key registered in one of these databases will be referenced by its UUID and called a persistent key from this specification's point of view.

TSS understands the identity of the calling process and will restrict access to keys registered in the User Persistent Storage to the allowed user/process. Of course some system processes can impersonate the user and then use that user's persistent storage.

Some registered keys have a defined fixed UUID by which they can be referenced on all systems providing the same registered key hierarchy. These UUIDs do not provide any information to identify the system the key is registered on.

An application can also load keys not registered in the TCS database. These keys are loaded utilizing the Tcsi by providing a key blob as defined by TCPA_KEY. These keys are called temporary keys from this specification's point of view.

Using the Key Management Services provided by TSS will simplify the whole mechanism of loading a key into the TPM from a calling context's point of view. The application must only address a key to be loaded by its well known UUID and the Key Management Services will do all the required loading of the underlying parent keys depending on the registered key hierarchy, which may be totally hidden from the application's scope.

The key hierarchy can be defined by some entity. For example the IT department of an enterprise and the TCG-aware applications may not need to know this key hierarchy at all.

Keys once registered in Persistent Storage (PS) will stay registered in PS until they are unregistered. The PS will stay valid across boots.

NOTE: This specification uses the UUID structure to define fixed values for predefined key identifiers.

End of informative comment.

Keys once registered in Persistent Storage (PS) MUST stay registered in PS until they are unregistered. The PS MUST stay valid across platform resets.

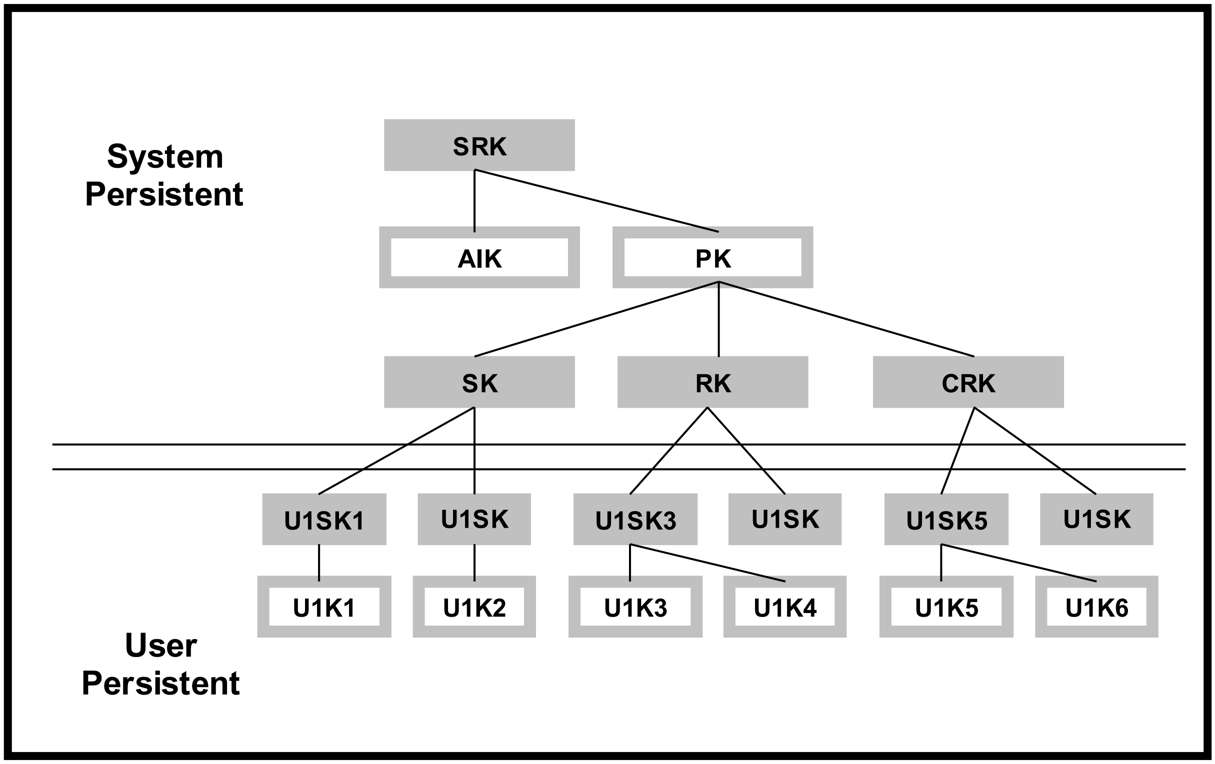


Figure 2-3 Load Key Flow Diagram

The grayed keys in the key hierarchy diagram above are mandatory storage keys and are addressed by fixed UUIDs, they have the same attributes (e.g. migratable, auth) and are stored either in the persistent storage of TCS or the persistent storage of TSP on all platforms. Keys stored in the user specific persistent storage of TSP can be addressed by the same UUID for each user but of course the UUID will still reference a different user storage key.

The following table lists the definition of the keys shown in figure 3.1:

Key	**UUID**	**PS Type**	**Migratable**	**Authorization**	**Description**
SRK	Fixed by TCG	System	No	No	Storage Root Key.
PK		System	No	No	Platform specific key.
RK	Fixed by TCG	System	Yes	No	Roaming Key.
SK	Fixed by TCG	System	No	No	System specific storage key.
CRK	Fixed by TCG	System	Yes-CMK	No	Certified Roaming key

ID1K		System	No	Yes	Identity key #1.
U1SK1	Fixed by TCG	User	No	No	Storage key #1 of User #1.
U1SK2	Fixed by TCG	User	No	Yes	Storage key #2 of User #1.
U1SK3	Fixed by TCG	User	Yes	No	Storage key #3 of User #1.
U1SK4	Fixed by TCG	User	Yes	Yes	Storage key #4 of User #1.
U1SK5	Fixed by TCG	User	Yes-CMK	No	Storage key #5 of User #1.
U1SK6	Fixed by TCG	User	Yes-CMK	Yes	Storage key #6 of User #1.
U1K1		User	No	Yes/No	Leaf Key #1 of user #1.
U1K2		User	No	Yes/No	Leaf Key #2 of user #1.
U1K3		User	Yes	Yes/No	Leaf Key #3 of user #1.
U1K4		User	Yes	Yes/No	Leaf Key #4 of user #1.
U1K5		User	Yes-CMK	Yes/No	Leaf Key #5 of user #1.
U1K6		User	Yes-CMK	Yes/No	Leaf Key #6 of user #1.

Additionally, keys that are marked by the owner as being non-volatilely resident in the TPM need to have fixed UUID as well. 256 UUIDs have been reserved for this purpose in the header file.

3.22.1 TSS Load Key Command Flow

Start of informative comment:

This chapter describes the flow of the Tspi Load Key commands in different scenarios.

End of informative comment.

TSP Definitions:

Type	Description
KS (TSP Key Storage)	TSP Storage of Keys. These keys are typically associated with an application.
HMG (HMAC Generator)	HMAC and SHA1 generator. Takes the relevant parameters and generates the authorization data using HMAC and SHA1 operations. The BSG (Byte Stream Generator) uses "TCG Specific Knowledge" to build the authorization data, for example, it must add the command ordinal to the HMAC calculation.

TCS Definitions:

Type	Description
KCS (TCS Key and Credential Storage)	TCS Storage of Keys and Credentials. These keys and credentials are typically related to the platform. Therefore the keys cannot be roaming keys.
KCM (Key Cache Manager)	Handles key-caching whenever required. The Key cache manager typically used TPM_SaveKeyContext and TPM_LoadKeyContext for the key caching. Similarly TPM_SaveAuthContext and TPM_LoadAuthContext were used to save loaded authorization sessions outside the TPM. All these commands are deprecated from the 1.1 and replaced with LoadContext and SaveContext. (Resource type is selected by a parameter).
KCMS (Key Cache Manager Storage)	The storage of the KCM.
PBG (Parameter Block Generator)	The Parameter Block Generator uses "TCG Specific Knowledge" to concatenate its input parameters and other parameters (ordinal, tag, etc.) to a TPM Parameter Block command.

General Definitions:

Type	Description
BS Key	"Byte Stream" format structure of a TCG Key.

3.22.2 TSS Load Key Flow Diagram

Start of informative comment:

Load Key Flow Description as depicted in the following diagram.

Case 1: Tspi_LoadKeyByBlob

Case 2: Tspi_LoadKeyByUUID, Key registered in KS,
2.1 Parent Key Authorization is Not Required.
2.2 Parent Key Authorization is Required.

Case 3: Tspi_LoadKeyByUUID, Key registered in KCS
3.1 Parent Key Authorization is Not Required.
3.2 Parent Key Authorization is Required.

End of informative comment.

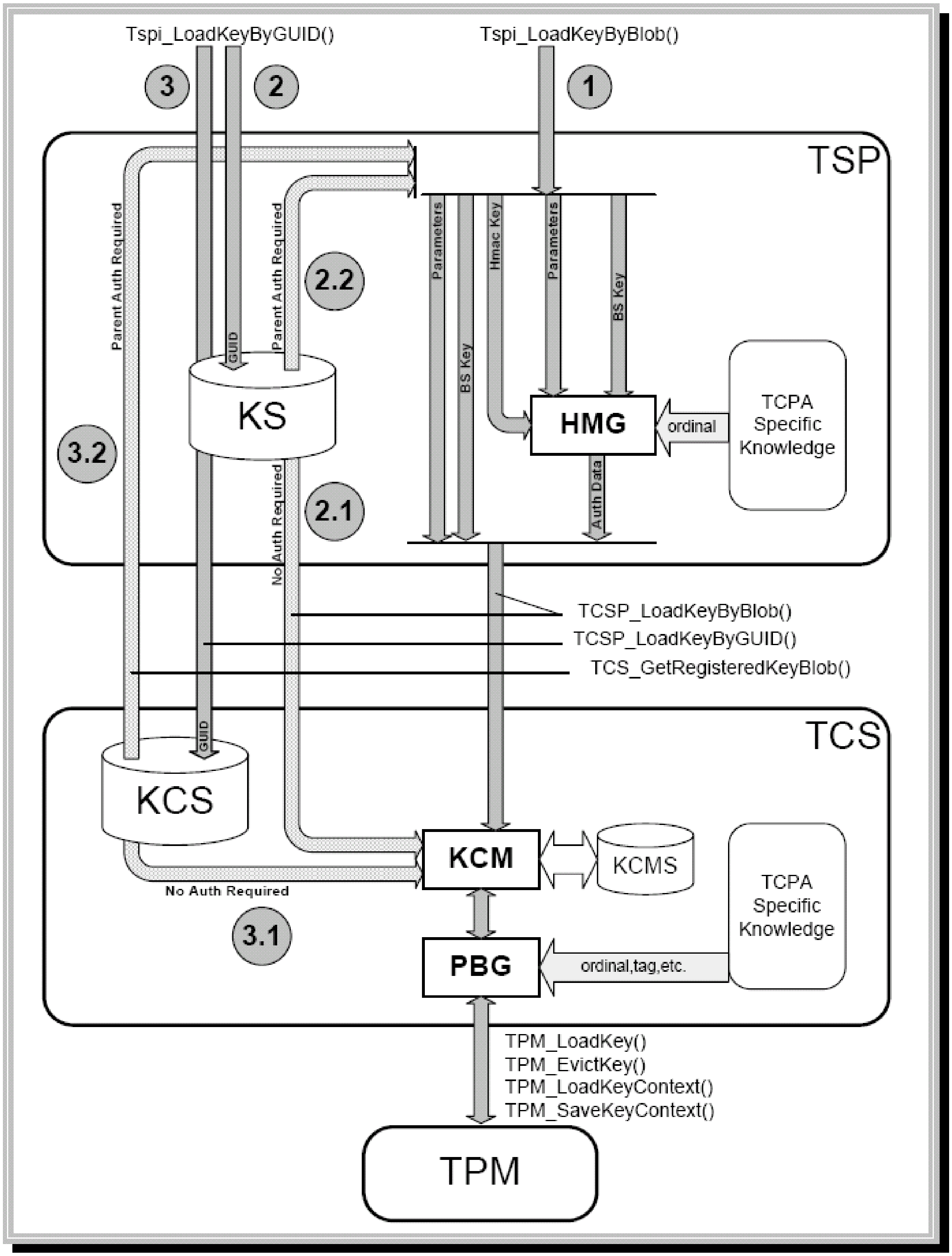


Figure 2-4 Load Key Flow Diagram

3.22.3 Key Handles

Start of informative comment:

To further explain the relationship between the various key handles the following illustration is presented:

First defining a common labeling convention:

- TPM - KeyHandle

  UINT32 to address a key loaded in the TPM. Created and maintained by TPM

- TCS - KeyHandle

  TCS_KEY_HANDLE to address a key object created and maintained by the TCS.

  Handle will internally be mapped to the appropriate TPM KeyHandle.

- TSP - KeyHandle

  TSS_HKEY to address a key object created and maintained by the TSP.

  Handle will internally be mapped to the appropriate TCS KeyHandle.

Using CreateWrapKey: as a rough sequence from the application (app) through the TSP, the TCS and down to the TPM

- (App -> TSP) Load parent wrapping key: Tspi_Context_LoadKeyByBlob. TSP returns key handle (Psp).
- (App -> TSP) Create a key object and initialize object parameters according to the key to be created. TSP returns a key handle (Nsp).
- (App -> TSP) Create a policy object and assign the policy to key object addressed by (Nsp).
- (App -> TSP) Call Tspi_Key_CreateKey
- (TSP internal) Take KeyHandle(Psp) and get appropriate TCS Keyhandle(Pcs).
- (TSP internal) Establish an OSAP session using the policy assigned to the wrapping parent key (secret) and the key object representing the wrapping parent key (TCS KeyHandle (Pcs) of parent key).
- (TSP internal) Encrypt the secrets of the key to be created and compute the authorization. The new secrets are provided by the policy object assigned to the key object addressed by (Nsp).
- (TSP -> TCS) Call Tcsip_CreateWrapKey using the appropriate TCS KeyHandle (Pcs) of the already loaded wrapping parent key
- (TCS internal) Take TCS KeyHandle(Pcs) and get appropriate TPM Keyhandle(Ptpm).
- (TCS internal) Create TPM byte stream for TPM_CreateWrapKey
- (TCS internal) Call TPM

Note:

The KCM may need to reload the parent wrapping key before calling TPM_CreateWrapKey. Using the Load/Save Key Context and Auth Context commands of the TPM, this should be no problem.

End of informative comment.

3.23 Portable Data

Start of informative comment:

The Key Management Services and the Data Management Services of TSS allow exporting/importing information from/into the TSS utilizing functionalities provided by TSPI.

The format of the data blobs is designed to allow transporting the information independent of a platform using ASN.1 BER encoding.

End of informative comment.

Definition:

```
TssBlobType ::= ENUMERATED
{
  Key-Blob (1),        -- TCPA_KEY as returned from TPM
  PubKey-Blob (2),     -- TCPA_PUBKEY as returned from TPM
  MigKey-Blob (3),     -- TCPA_KEY as returned from the TSP
                          Operation Tspi_Key_CreateMigrationBlob
                          In dedicated mode (see the command for details)

  SealedData-Blob (4),  -- TCPA_STORED_DATA as returned from TPM
  BoundData-Blob (5),-- TCPA_BOUND_DATA as returned from TPM
  Migticket-Blob (6) -- TCPA migration data as returned from TPM
  PrivateKey-Blob(7) -- Encrypted private TCG key blob returned from TPM
  PrivateKey-MOD1-Blob (8) --  Encrypted private key to be
                               wrapped by the TSS
  RandomXOR-Blob (9) -- String used for xor encryption of the
                        migration blobs
  CertifyInfo-Blob (10) -- TCPA_CERTIFY_INFO as returned from
                           the TPM
  Key12_Blob (11)       -- TPM_KEY12 as returned from TPM
  CertifyInfo2-Blob (12)  -- TPM_CERTIFY_INFO2 structure as
                             returned from TPM
  CMKMigKey-Blob (13)  -- TCPA-KEY as returned from the TSP
                          Operation Tspi_CMKCreateBlob
  CMK_Byte-Stream (14) --  Used by CMK commands for
                           transmitting HMACs and digest
}


TssBlobType ::= INTEGER

TssBlob ::= SEQUENCE
{
   StructVersion INTEGER, -- Version of this structure; at the
moment 1
   BlobType      TssBlobType,   -- Type of Blob; see enum
   BlobLength    INTEGER,       -- Length of Blob
   Blob          OCTET STRING-- Blob as returned from TPM (no
ASN1
                                encoding)
```

```
}
```

Blob Type	Content definition from TPM spec	TSS Function Usage
Key-Blob	TPM_KEY with encrypted TPM_STORE_ASYMKEY	• Tspi_SetAttribData or Tspi_GetAttribData with TSS_TSPATTRIB_KEYBLOB_BLOB
	TPM_KEY with encrypted TPM_MIGRATE_ASYMKEY	• Tspi_Key_CreateMigrationBlob • Tspi_Key_ConvertMigrationBlob
PubKey-Blob	TPM_PUBKEY	• Tspi_SetAttribData or Tspi_GetAttribData with TSS_TSPATTRIB_KEYBLOB_PUBLIC_KEY attrib • Tspi_Key_GetPubKey • Tspi_Key_GetSRKPubKey
PublicKey-Modulus-Blob		• Tspi_Context_GetKeyByPublicInfo
MigKey-Blob	Not used	Not used
SealedData-Blob	TPM_STORED_DATA	• Tspi_SetAttribData or Tspi_GetAttribData with TSS_TSPATTRIB_ENCDATABLOB_BLOB attrib
BoundData-Blob	TPM_BOUND_DATA	• Tspi_SetAttribData or Tspi_GetAttribData with TSS_TSPATTRIB_ENCDATABLOB_BLOB attrib
Migticket-Blob	TPM_MIGRATIONKEYAUTH	• Tspi_Key_CreateMigrationBlob • Tspi_Key_CreateMigrationBlob
PrivateKey-Blob	Byte stream	• Tspi_SetAttribData or Tspi_GetAttribData with TSS_TSPATTRIB_KEYBLOB_PRIVATE_KEY attrib
RandomXOR-Blob	Byte stream	• Tspi_Key_CreateMigrationBlob • Tspi_Key_ConvertMigrationBlob
Key12-Blob	TPM_KEY with encrypted TPM_STORE_ASYMKEY	• Tspi_SetAttribData or Tspi_GetAttribData with TSS_TSPATTRIB_KEYBLOB_BLOB attrib • Tspi_Context_LoadKeyByBlob
	TPM_KEY with encrypted TPM_MIGRATE_ASYMKEY	• Tspi_Key_CreateMigrationBlob • Tspi_Key_ConvertMigrationBlob
CertifyInfo2-blob	TPM_CERTIFY_INFO2	Tspi_Key_CertifyKey
CertifyInfo-Blob	TPM_CERTIFY_INFO	Tspi_Key_CertifyKey

For DAA structures that are represented using TSS_HDAA_DATA handles, an ASN.1 encoding is defined. These structures are TSS_DAA_PK, TSS_DAA_PK_PROOF, TSS_DAA_SK, TSS_DAA_AR_PK, TSS_DAA_AR_SK, TSS_DAA_CREDENTIAL.

Note, that the ASN.1 encoding of these structures is not used in hash functions defined in the DAA algorithm specifications, instead the byte array representation of the structure is relevant there. See also section 4.3.4.29.10 for platform independent definition of the structures.

Definition:

```
TssDaaPk ::= SEQUENCE
{
   modulus                 INTEGER,-- n
   capitalS                INTEGER,-- S
   capitalZ                INTEGER,-- Z
   capitalR0               INTEGER,-- R0
   capitalR1               INTEGER,-- R1
   capitalY                IntegerSequence, -- sequence of Yi
   capitalYPlatformLenght   INTEGER,-- number of Yi relevant for
                                   Platform
   gamma                   INTEGER,-- gamma
   captialGamma            INTEGER,-- Gamma
   rho                     INTEGER,-- rho
   issuerBaseName          INTEGER -- bsnI
}

IntegerSequence ::= SEQUENCE SIZE(1..MAX) OF INTEGER

TssDaaPkProof ::= SEQUENCE
{
   challenge     OCTET STRING, -- c (20 bytes long)
   response      IntegerSequence -- xHead(i,j)
}

TssDaaSk ::= SEQUENCE
{
   productPQ       INTEGER -- p*q
}

TssDaaArPk ::= SEQUENCE
{
   eta           INTEGER,-- eta
   lambda1       INTEGER,-- lambda1
   lambda2       INTEGER,-- lambda2
   lambda3       INTEGER -- lambda3
}

TssDaaArSk ::= SEQUENCE
```

```
{
   x0       INTEGER,-- x0
   x1       INTEGER,-- x1
   x2       INTEGER,-- x2
   x3       INTEGER,-- x3
   x4       INTEGER,-- x4
   x5       INTEGER -- x5
}

TssDaaCredential ::= SEQUENCE
{
   capitalA       INTEGER,-- A
   exponent       INTEGER,-- e
   vBar0          INTEGER,-- vBar0
   vBar1          INTEGER,-- vBar1
   attributes     IntegerSequence,-- a
   issuerPk       TssDaaPk,  -- DAA Issuer public key
   tpmSpecificEnc   OCTET STRING  --      Encrypted      TpmSpecific
                                    Structure
}
```

Portable Data Format Conversion Functions

While different TCS's may internally store data about keys in different ways, there are times when it is useful to be able to transfer blobs between different TCSs. ASN.1 BER encoding is the method defined for doing this. In order to facilitate the conversion of data to this format, the following two utility functions are defined. They are different from other functions in this specification as they are not locked to a particular object.

3.23.1.1 Tspi_EncodeDER_TssBlob

Start of informative comment:

This function generates a DER-encoded blob in accordance with the ASN.1 data definitions in the Portable Data section of this document. The input is an unwrapped blob (e.g. a TPM_KEY encoded as a byte array as generated by the TPM, or a 20 byte hash value) and a tag indicating what type of blob is being presented. The output is a DER encoding of the data.

This function is defined to facilitate data interchange among TSS implementations that do and do not use the ASN.1 encoding for data blobs.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_EncodeDER_TssBlob
(
  UINT32        rawBlobSize,  // in
  BYTE*         rawBlob,      // in
  UINT32        blobType,     // in
  UINT32*       derBlobSize,  // in,out
  BYTE*         derBlob       // out
);
```

Parameters:

rawBlobSize

Size of the unwrapped blob.

rawBlob

pointer to the unwrapped blob.

blobType

Integer indicating what type of blob is being wrapped. The value should come from the enumerated TssBlobType list in the Portable Data section.

derBlobSize

Inputs the max size of the derBlob buffer. Returns the length of the DER-encoded blob.

derBlob

Returns a pointer to the DER-encoded blob.

Return Values:

TSS_SUCCESS
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks:

This function will perform a DER-encoding of the supplied data and blobType tag to produce a TssBlob as defined in the section on PortableData.

The caller must provide the output buffer. To determine the necessary buffer length the caller may supply a 0-length output buffer, in which case the function will simply return the size of the buffer required without writing any data to the output buffer. Alternately the caller may note that if the raw data blob length is less than 2^16 bytes then the DER-encoding may add no more than 17 bytes.

3.23.1.2 Tspi_DecodeBER_TssBlob

Start of informative comment:

This function unwraps a BER-encoded blob in accordance with the ASN.1 data definitions in the Portable Data section of this document. The input is an BER-encoded TssBlob blob. The output is a raw data blob (e.g. a TPM_KEY encoded as a byte array as generated by the TPM, or a 20 byte hash value) and a tag indicating what type of blob was presented.

This function is defined to facilitate data interchange among TSS implementations that do and do not use the ASN.1 encoding for data blobs.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_DecodeBER_TssBlob
(
    UINT32        berBlobSize,  // in
    BYTE*         berBlob,      // in
    UINT32*       blobType,     // out
    UINT32*       rawBlobSize,  // in,out
    BYTE*         rawBlob       // out
);
```

Parameters:

berBlobSize

Size of the BER-encoded blob.

berBlob

pointer to the BER-encoded blob.

blobType

Returns an integer indicating what type of blob is being wrapped. The value will come from the enumerated TssBlobType list in the Portable Data section.

rawBlobSize

Inputs the max size of the rawBlob buffer. Returns the length of the unwrapped blob.

rawBlob

A buffer to hold the unwrapped blob.

Return Values:

TSS_SUCCESS
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks:

This function will parse a BER-encoded TssBlob supplied by the caller, returning a tag indicating what type of data was wrapped and the raw wrapped blob.

The caller must provide the output buffer. To determine the necessary buffer length the caller may supply a 0-length output buffer, in which case the function will simply return the size of the buffer required without writing any data to the output buffer. Alternatively the caller may note that the output data must be shorter than the ber-encoding, so *berBlobSize* is a useful upper limit on the output buffer size.

4. TCG Service Provider (TSP)

4.1 Theory of Operation

4.1.1 Functional Overview

The TSS Service Provider module provides an API making a set of TCG functionalities accessible for TCG-aware application software. It is through this TSP that an application can access data or services on a specific TPM.

The following classes of services are implemented within existing TPMs:

- Integrity Collection and Reporting Services
- Protected Storage Services
- Cryptographic Services
- Credential Services

The standardization of that API to these services enables development and maintenance of TCG-aware application software with minimal knowledge of TPM internals.

Authorization Session Handling

The TSS Service Provider (TSP) hides the management of TCG related authorization sessions from the calling application. There is no requirement for the application to initialize any OIAP or OSAP authorization session. The TSP initializes a required authorization session and handles all internal data of that session.

4.1.2 Interface Design

Although the TSPI is defined as a C interface, this API uses an object-oriented approach. All TSPI functions deal with one or more object handle parameters addressing certain instances of a class. Callers perform actions on objects utilizing public methods. Attributes are accessed by calling the set or get object attribute methods.

4.1.2.1 Classes

The TSPI defines the following classes:

- Context class
- Policy class
- TPM class
- Key class
- Encrypted Data class (sealed or bound data)
- PCR Composite class
- NV RAM class
- Hash class

Context class

The context contains information about the TSP-Object's execution environment, such as the identity of the object and the transaction/communication with other TSS-Software modules (e.g. TSS-Core-Service). A context object in the TSP environment is similar in concept to the process context that an operating system maintains for an executing program.

Policy class

The policy class infrastructure of the TSP can be used to configure policy settings and behaviors for different user applications. The application can use the TSP-Policy infrastructure to provide specialized secret handling (e.g. CallBack, Lifetime, …) for the authorization.

TPM class

One purpose of the TPM class is to represent the owner for a TCG subsystem (TPM). The owner of a TPM is comparable with an administrator in the PC environment. For that reason there exists only one instance of the TPM class per context. This object is automatically associated with one policy object; which must be used to handle the owner authentication data. On the other hand it provides some basic control and reporting functionality.

Key class

The key class type defined by the TSS service provider represents an entry into the TCG key handling and functionality. Each instance of a key object represents a specific key node that is part of the TSS key hierarchy. A key object, which needs authentication, can be assigned to a policy object that controls the secret management.

Encrypted Data class (Seal and Bind)

This class can be used to join externally (e. g. user, application) generated data to a TCG-aware system (bound to PCR or Platform). For the authentication process this class can be assigned to a policy object.

PCR-Composite class

The contents of the platform configuration register (PCR) of a TCG system can be used to establish a confidence level for this system. This class provides a comfortable way to deal with PCR values (e.g. select, read, write). An object handle of such a class is used from all TSP functions that need PCR information in their parameter list.

Hash class

A hash value represents a unique value corresponding to a particular set of bytes. This class provides a cryptographically secure way to use these functions for digital signature operations.

4.1.2.2 Object Relationship

Working objects are subdivided into authorized and non-authorized working objects. Non-authorized working objects include the PCR composite objects and hash objects, MigData, DelFamily, and DAA objects. Authorized working objects include the TPM object, key objects, NVStore objects and encrypted data objects.

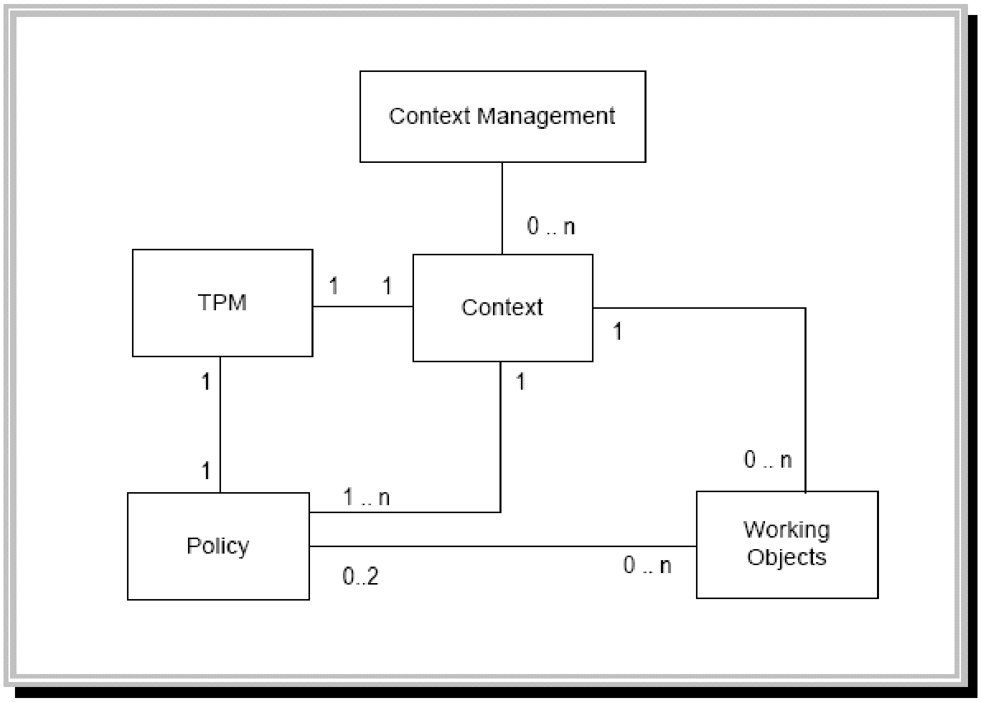


Figure 3-5 Object Relationship

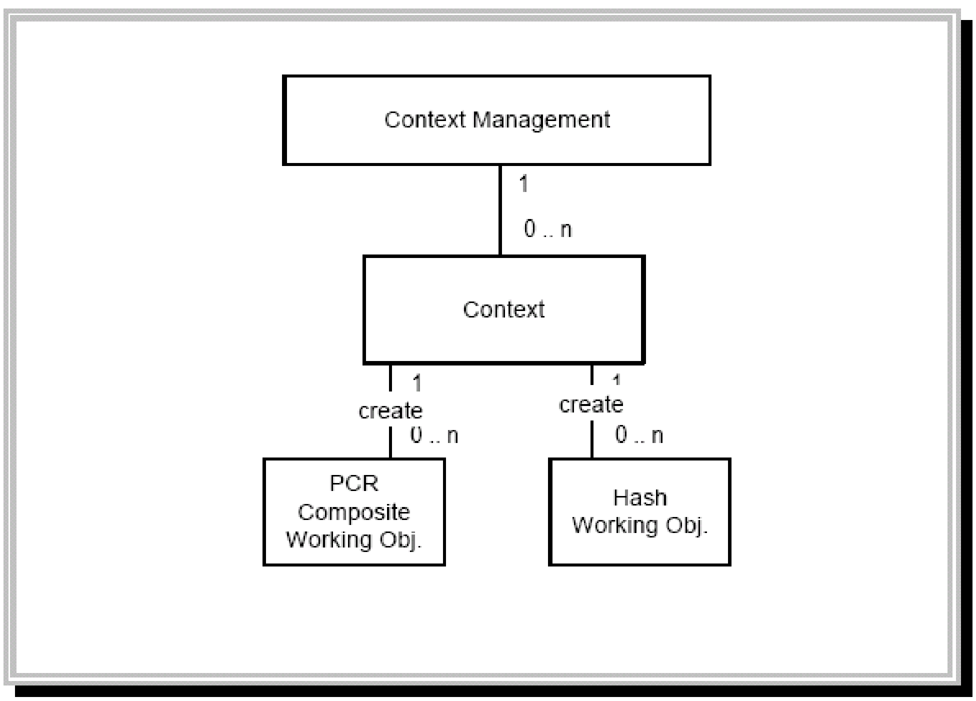


Figure 3-6 Non authorized working object relationship

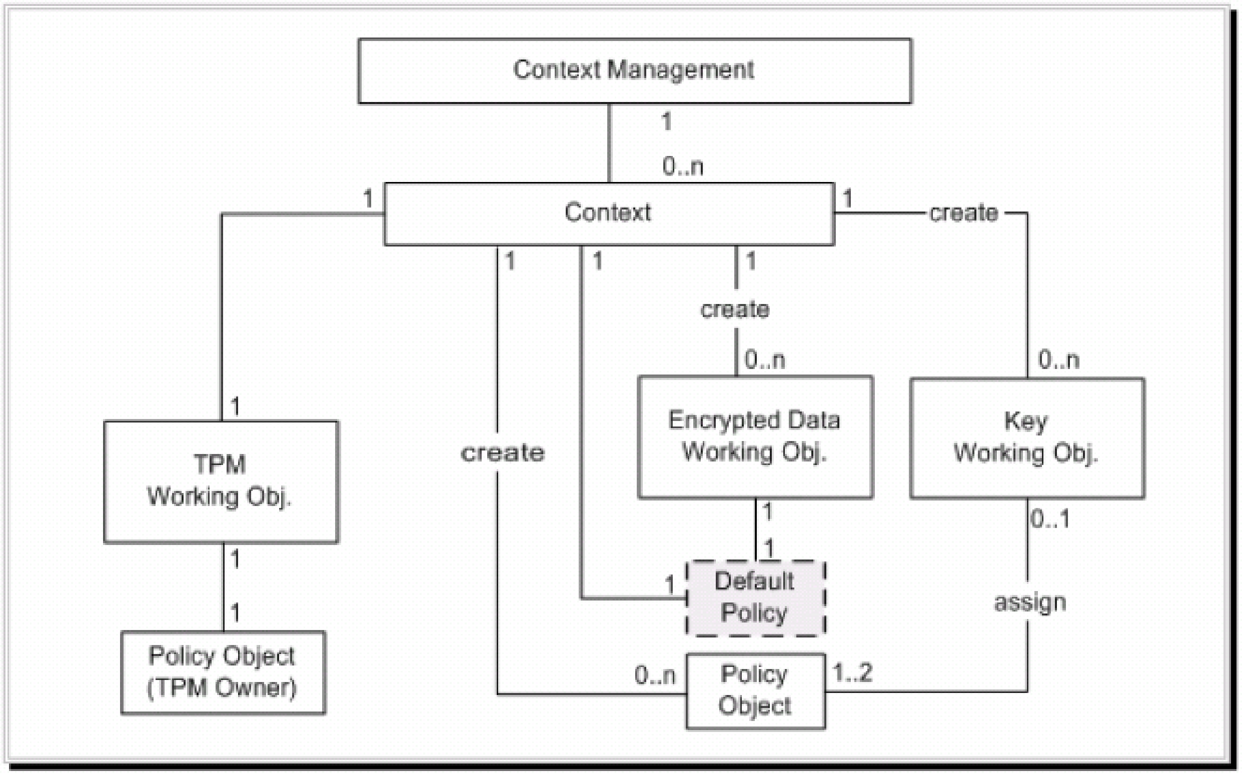


Figure 3-7 Authorized working object relationship

The calling application (the user) may have to supply authorization data only once for each policy it wants to utilize. A policy may be assigned to several objects like key objects, encrypted data objects or a TPM object utilizing the Tspi_Policy_AssignToObject( ) method. Each of these objects will utilize its assigned policy object to process authorized TPM commands using internal functions of the policy object.

On creation of a context a default policy is created and each new created object is automatically assigned to this default policy. The default policy for each working object exists as long as no assign command sets a new policy object to the working object. The TPM object has a separate policy object that represent the owner of the TPM. Assigning one or more working objects to a policy object is done by internal policy and working object functions.

4.1.3 Authorization Data Handling

The TSP provides policy objects helping the calling application handling and caching secrets for authorized objects. The following objects are authorized objects: TPM, Key and encrypted data. The TSP also knows when to use the secret of the object (for OIAP) or a session secret derived from the object secret (for OSAP). An authorized object is assigned to exactly one policy object but a policy object may be assigned to 0..n authorized objects to facilitate the same secret for various authorized objects. If an authorized object is not explicitly assigned to a certain policy object, the TSP automatically assigns this authorized object to a default policy object.

There is no requirement for the application to provide a secret when calling a function where authorization is necessary. The TSP asks the user to enter a secret in a dialog box for this purpose. This service is not restrictive and can be modified by an application on a policy object base. Depending on the platform the service provider should use non-paged memory for secrets provided to it. Before freeing this memory it should also zero the memory area used for this.

The default mode for a policy object is TSS_SECRET_MODE_POPUP, but it can also be set to TSS_SECRET_MODE_PLAIN, TSS_SECRET_MODE_SHA1 or TSS_SECRET_MODE_CALLBACK. If the mode is set to TSS_SECRET_MODE_CALLBACK, the application is responsible to register a callback function during initialization of a policy object. If the callback is not registered the command will fail.

In 1.2 new kinds of authorization became available. Operator_Auth is an authorization only available to the operator, which can be used to temporarily turn off the TPM (until the next power cycle). Delegation allows the owner of the TPM or a key to delegate the ability to use certain commands or keys based on the authentication provided either by locality or another password. Applications have to keep track of whether a password has been delegated – the TPM will accept either the actual password or the delegated authority as is required.

Mode:

TSS_SECRET_MODE_POPUP

The TSP displays a dialog to enter pass phrase. The pass phrase provided by the user is handled as a null terminated TSS_UNICODE string and will be hashed using SHA1 to get the authorization secret. Once a pass phrase was provided the TSP may cache the resulting authorization secret in the appropriate policy object (depending on the policy object settings) and the dialog will not pop up again.

Examples:

- If an identity key is to be used for quoting a system the dialog is asking for the secret required to use that identity key.
- If a secret has to be changed the dialog will ask for the old and new secret.

TSS_SECRET_MODE_PLAIN or TSS_SECRET_MODE_SHA1

The application can set a secret per policy base. If this is done the TSP caches and uses the secret for processing authorized commands on assigned authorized objects.

The secret can be set as plaintext or as digest (SHA1) by the application.

Type	Definition
TSS_SECRET_MODE_SHA1	The TSP will only accept an array of 20 bytes and will not touch this data at all. The data will be handled as the authorization secret.
TSS_SECRET_MODE_PLAIN	The TSP will accept any byte array and will calculate a hash using SHA1 to get the authorization secret.

TSS_SECRET_MODE_SHA1:

- Secret string will not be touched by TSP and MUST be size of 20 bytes.

- (ulSecretLength == 20 and rgbSecret points to the hashed secret byte stream

```
TSS_SECRET_MODE_PLAIN:

- Secret string will be hashed by the TSP using SHA1.
- ulSecretLength contains the length of the secret string
(e.g. ulSecretLength = StringLen(strSecret);
- - rgbSecret points to the first byte of the secret string
  stream
```

TSS_SECRET_MODE_CALLBACK

This mode can be used by an application in the following situations:

- An application doesn't want to reveal the secret.
- The secret is collected by another mechanism like a biometric device.
- The secret is protected by another security token like a smart card.

The TSP will call one of the application provided callback functions. The first one is used to calculate the HMAC for the authorization data required for TPM command authorization. The second one is used for XOR encryption of a new or changed secret. The third one is used to encrypt a secret with a public key.

All necessary parameters are included in the definition of the callback function to perform the above actions.

Secret-Mode

TSS_SECRET_MODE_NONE:

- No authorization will be processed of all assigned working object (e.g. Key-Object);
- Different from a secret of 20 bytes of 0x00.
- (ulSecretLength == 0 and rgbSecret == NULL)

4.1.3.1 Secrets Handled by Service Provider

Secret Caching

The concept of TCG defines that keys may require authorization for their use. The secret for a key requiring authorization has to be presented for each usage of such a key. For usability the TSS Service Provider supports a mechanism allowing that the secret has to be presented only once to the TSP for a specified duration. For all subsequent calls requiring this secret, the Service Provider takes the secret from its internal secret cache.

Caching of secrets increases security concerns.

Generally an application which uses the secret caching facility of the TSP must trust that the loaded TSP is the authentic TSP it wants to load. For this purpose it must utilize any appropriate means e.g. the TCG means to rely on a software stack.

If the application cannot rely on the authenticity of the loaded TSP it must not use it. Instead it has to call the TSS Core Service interface directly.

Lifetime of a cached secret

The cache along with its secret MUST be destroyed when:

- the secret is changed,
- the secret is flushed,
- the secret use counter runs down,
- the secret gets timed out or,
- the policy object is closed.

4.1.3.2 Secrets Handled by Application

If an application does not trust the TSP to be authentic and trusted, an application will not want to reveal the secret to the TSS Service Provider. Usage of a smart card containing the secret could be one solution for such a scenario.

For this purpose the TSPI provides a callback function mechanism.

4.1.4 Implementation Considerations

The main focus of the TSS Service Provider is to abstract implementation details at TPM level and expose TPM functionality in a way that TCG-aware application software can access it easily. In particular, this will enable application developers to write TCG-aware applications without requiring much TPM intimate knowledge.

The interfaces exposed by the TSP are developed within the context of a given platform being consistent with API standards of that platform. It is possible to implement the interfaces defined in this specification in a way that is suitable for use with procedural programming languages such as C as well as with object-oriented languages such as C++. This may entail some modification of the naming conventions, parameter types, and so on. As long as the implementation is functionally equivalent, it is consistent with the intent of this specification.

A TSP is built upon the services exposed by the TSS Core Service. This provides TSP developers with a set of functionality they can use to simplify the development and maintenance of their software.

4.1.5 User Interface Elements

The user interface (UI) provided by the TSS Service Provider should be implemented according to the published guidelines of the target environment.

In particular, the TSP implements a UI for authorization data. The TSP is the best place to implement authorization data management, because it encapsulates knowledge about authorization protocols, authorization session, authorization data length and so on.

4.1.6 Runtime Considerations

The TSS Service provider is implemented as a user-mode application module running in the context of and with the same privileges as the calling TCG-aware application. In a Windows system the TSP will typically be an in-process COM sever or a DLL. In a server, it is possible that multiple TSS service providers will be all running simultaneously talking to virtual TPMs. This can be done in a variety of ways.

4.2 TSPI-specific Return Code Defines

The TSPI common return codes are returned as TSS error code and are defined in section 2.4.2 Common Return Code Defines. Below are TSP-specific return codes.

Type	Definition
TSS_E_INVALID_OBJECT_TYPE	Object type not valid for this operation.
TSS_E_INVALID_OBJECT_INIT_FLAG	Invalid object initialization flag
TSS_E_INVALID_HANDLE	Invalid object handle
TSS_E_NO_CONNECTION	TCS connection has not been established, but is required.
TSS_E_CONNECTION_FAILED	Establishing a connection to Core Service failed
TSS_E_CONNECTION_BROKEN	Communication with Core Service has been established but has since failed.
TSS_E_HASH_INVALID_ALG	Invalid hash algorithm.
TSS_E_HASH_INVALID_LENGTH	Hash length is inconsistent with hash algorithm.
TSS_E_HASH_NO_DATA	Hash object has no internal hash value.
TSS_E_SILENT_CONTEXT	Context is silent, but requires user input.
TSS_E_INVALID_ATTRIB_FLAG	Flag value for attrib-functions invalid.
TSS_E_INVALID_ATTRIB_SUBFLAG	Subflag value for attrib-functions invalid.
TSS_E_INVALID_ATTRIB_DATA	Data for attrib-functions invalid.
TSS_E_NO_PCRS_SET	No PCR register is selected or set.
TSS_E_KEY_NOT_LOADED	The addressed key is currently not loaded.
TSS_E_KEY_NOT_SET	No key information is currently available.
TSS_E_VALIDATION_FAILED	Internal validation of data failed.
TSS_E_TSP_AUTHREQUIRED	Authorization is required.
TSS_E_TSP_AUTH2REQUIRED	Multiple authorization is required.
TSS_E_TSP_AUTHFAIL	Authorization failed.
TSS_E_TSP_AUTH2FAIL	Multiple authorization failed.
TSS_E_KEY_NO_MIGRATION_POLICY	There's no migration policy object set for the addressed key.
TSS_E_POLICY_NO_SECRET	No secret information is currently available for the

	addressed policy object, but secret information is required.
TSS_E_INVALID_OBJ_ACCESS	The operation failed due to an invalid object status.
TSS_E_INVALID_ENCSCHEME	Invalid encryption scheme.
TSS_E_INVALID_SIGSCHEME	Invalid signature scheme.
TSS_E_ENC_INVALID_LENGTH	Invalid length of data to be encrypted.
TSS_E_ENC_NO_DATA	No data to encrypt.
TSS_E_ENC_INVALID_TYPE	Invalid encryption type.
TSS_E_INVALID_KEYUSAGE	Invalid key usage.
TSS_E_VERIFICATION_FAILED	Verification of signature failed.
TSS_E_HASH_NO_IDENTIFIER	The hash algorithm identifier is not set.
TSS_E_BAD_PARAMETER	One of the parameters was not as expected
TSS_E_INTERNAL_ERROR	TPM internal error
TSS_E_INVALID_RESOURCE	Pointer to memory wrong
TSS_E_PS_KEY_NOTFOUND	Key not in persistent storage
TSS_E_NOTIMPL	Function not implemented
TSS_TPM_NOT_RESETABLE	This PCR is not resetable
TSS_E_WRONG_LOCALITY	This command cannot be executed from this locality
TSS_E_KEY_NO_MIGRATION_POLICY	Need a migration authorization set
TSS_E_NV_AREA_NOT_EXIST	The non-volatile area referenced doesn't exist
TSS_E_DAA_ISSUER_KEY_ERROR	DAA Issuer’s authentication key chain could not be verified or is not correct.
TSS_E_DAA_CREDENTIAL_PROOF_ERROR	Verification of the credential TSS_DAA _CRED_ISSUER issued by the DAA Issuer has failed.
TSS_E_DAA_AUTHENTICATION_ERROR	The TPM could not be authenticated by the DAA Issuer.
TSS_E_DAA_PSEUDONYM_ERROR	While verifying the pseudonym of the TPM, the private key of the TPM was found on the rogue list.
TSS_E_DAA_AR_DECRYPTION_ERROR	Decryption of the encrypted pseudonym has failed, due ot either a wrong secret key or a wrong decryption condition
TSS_E_DAA_CREDENTIAL_REQUEST_PROOF _ERROR	Verification of the credential TSS_DAA_CREDL_ISSUER issued by the DAA issuer failed

TSS_E_NV_AREA_EXIST	Attempt to define an area that already exists
TPM_SELFTEST_FAILED	TPM error
TPM_DELEGATE_LOCK	TPM error
TPM_DELEGATE_FAMILY	TPM error
TPM_WRONGPCRVALUE	TPM error: The named PCR value does not match the current PCR value.
TPM_NOT_FULLWRITE	TPM error: The write is not a complete write of the area
TPM_NOSPACE	TPM error
TPM_DISABLED_CMD	TPM error
TPM_BAD_PRESENCE	TPM error: Either the physicalPresence or physicalPresenceLock bits have the wrong value.
TPM_BAD_LOCALITY	TPM error: The locality is incorrect for the attempted operation.
TPM_BAD_INDEX	TPM error: The index to a PCR, DIR or other register is incorrect.
TPM_AUTH_CONFLICT	TPM error
TPM_AUTHFAIL	TPM error
TPM_OWNERSET	TPM error
TPM_BAD_DATASIZE	TPM error: The size of the data (or blob) parameter is bad or inconsistent with the referenced key.
TPM_MAXNVWRITE	TPM error: The maximum number of NV writes without an owner has been exceeded.
TPM_INVALID_STRUCTURE	TPM error
TPM_NOWRITE	TPM error
TPM_AREA_LOCKED	TPM error: The NV area is locked and not writeable
TPM_KEY_OWNER_CONTROL	TPM error
TSS_E_NO_ACTIVE_COUNTER	The TPM does not have an active counter yet
TSS_SUCCESS	Command completed

4.3 Interface Description

4.3.1 Syntax

The syntax used in describing the TSS Service Provider is based on the common procedural language constructs. Data types are described in terms of ANSI C.

4.3.2 Calling Conventions regarding Memory Management

The TSP allocates memory for out parameters and provides a function to free the memory previously allocated by the TSP on a context object base. The calling application MUST free memory allocated by the TSS Service Provider. The caller of the TSP functions is responsible for calling Tspi_Context_FreeMemory for each call that produced allocation of memory.

Example

```
// prototyping
TSS_RESULT Func(
   UINT32* pulLength, // out
   BYTE** prgbData    // out
);

// C++ sample
TSS_RESULT Result = TSS_SUCCESS;
UINT32     ulLength = 0L;
BYTE*      prgbData = NULL;

// Init TSS_HCONTEXT hContext via a TSPI function

Result = Func(&ulLength, &prgbData);

// if (TSS_SUCCEEDED(Result)) …
// work with prgbData

// afterwards cause the TSP to free pData
Tspi_Context_FreeMemory(hContext, prgbData);
```


4.3.3 Classes and Methods

4.3.3.1 Common Methods Definition

4.3.3.1.1 Tspi_SetAttribUint32

Start of informative comment:

This method sets a 32-bit attribute of the object. If the data being set is smaller than a UINT32, casting must be used to get the data to the right size.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_SetAttribUint32
(
  TSS_HOBJECT   hObject,    // in
  TSS_FLAG      attribFlag,// in
  TSS_FLAG      subFlag,    // in
  UINT32        ulAttrib    // in
);
```

Parameters

hObject

Handle of the object where the attribute is to be set.

attribFlag

Flag indicating the attribute to set (see table Defined Attributes).

subFlag

Sub flag indicating the attribute to set (see table Defined Attributes).

ulAttrib

Value which is to be set for the specified attribute (see table Defined Attributes).

Defined Attributes

See table Defined Attributes of appropriate Class Definition section

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_INVALID_ATTRIB_FLAG
TSS_E_INVALID_ATTRIB_SUBFLAG
TSS_E_INVALID_ATTRIB_DATA
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

4.3.3.1.2 Tspi_GetAttribUint32

Start of informative comment:

This method gets a 32-bit attribute of the object.f the data being requested is smaller than a UINT32, casting must be used to get the data to the right size.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_GetAttribUint32
(
    TSS_HOBJECT   hObject,      // in
    TSS_FLAG      attribFlag,   // in
    TSS_FLAG      subFlag,      // in
    UINT32*       pulAttrib     // out
);
```

Parameters

hObject

Handle of the object to retrieve the attribute.

attribFlag

Flag indicating the attribute to query (see table Defined Attributes).

subFlag

Sub flag indicating the attribute to query (see table Defined Attributes).

pulAttrib

Receives the value of the specified attribute (see table Defined Attributes).

Defined Attributes

See table Defined Attributes of the appropriate class definition section

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_INVALID_ATTRIB_FLAG
TSS_E_INVALID_ATTRIB_SUBFLAG
TSS_E_INVALID_ATTRIB_DATA
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

4.3.3.1.3 Tspi_SetAttribData

Start of informative comment:

This method sets a non 32-bit attribute of the object. The structure and size of the attribute data depends on the attribute.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_SetAttribData
(
    TSS_HOBJECT   hObject,          // in
    TSS_FLAG      attribFlag,       // in
    TSS_FLAG      subFlag,          // in
    UINT32        ulAttribDataSize, // in
    BYTE*         rgbAttribData     // in
);
```

Parameters

hObject

Handle of the object where the attribute is to be set.

attribFlag

Flag indicating the attribute to set (see table Defined Attributes).

SubFlag

Sub flag indicating the attribute to set (see table Defined Attributes).

ulAttribDataSize

Supplies the length (in bytes) of the *rgbAttribData* parameter. If the *rgbAttribData* parameter is a TSS_UNICODE string, the size includes the terminating null character.

rgbAttribData

Pointer to the actual data which is to be set for the specified attribute (see table Defined Attributes).

Defined Attributes

See table Defined Attributes of appropriate Class Definition section

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_INVALID_ATTRIB_FLAG
TSS_E_INVALID_ATTRIB_SUBFLAG
TSS_E_INVALID_ATTRIB_DATA
TSS_E_BAD_PARAMETER

TSS_E_INTERNAL_ERROR

Remarks

4.3.3.1.4 Tspi_GetAttribData

Start of informative comment:

This method gets a non 32-bit attribute of the object. The structure and size of the attribute data depends on the attribute.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_GetAttribData
(
   TSS_HOBJECT   hObject,           // in
   TSS_FLAG      attribFlag,        // in
   TSS_FLAG      subFlag,           // in
   UINT32*       pulAttribDataSize, // out
   BYTE**        prgbAttribData     // out
);
```

Parameters

hObject

Handle of the object where to retrieve the attribute.

attribFlag

Flag indicating the attribute to query (see table Defined Attributes).

subFlag

Sub flag indicating the attribute to query (see table Defined Attributes).

pulAttribDataSize

Receives the length (in bytes) of the *rgbAttribData* parameter. If the *rgbAttribData* parameter is a TSS_UNICODE string, the size includes the terminating null character.

prgbAttribData

On successful completion of the command, this parameter points to a buffer containing to the actual data of the specified attribute (see table Defined Attributes).

Defined Attributes

See table Defined Attributes of appropriate Class Definition section

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_INVALID_ATTRIB_FLAG
TSS_E_INVALID_ATTRIB_SUBFLAG
TSS_E_INVALID_ATTRIB_DATA

TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The Tspi_GetAttribData method allocates a memory block for the requested attribute data. This memory must be released utilizing the Tspi_Context_FreeMemory method.

4.3.3.1.5 Tspi_ChangeAuth

Start of informative comment:

This method changes the authorization data (secret) of an entity (object) and assigns the object to the policy object. All classes using secrets provide this method for changing their authorization data.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_ChangeAuth
(
    TSS_HOBJECT   hObjectToChange, // in
    TSS_HOBJECT   hParentObject,   // in
    TSS_HPOLICY   hNewPolicy       // in
);
```

Parameters

hObjectToChange

Handle of the object the authorization data should be changed.

hParentObject

Handle of the parent object wrapping the object addressed by *hObjectToChange*.

hNewPolicy

Handle of the policy object providing the new authorization data.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_INTERNAL_ERROR

Remarks

This command requires the parent object to unwrap the old authorization data and to create a shared secret, which is utilized to encrypt the new authorization data ensuring a secure authorization data transmission to the TPM. On successful completion of the command the object addressed by hObjectToChange is bound to the policy object addressed by hNewPolicy.

Special considerations for TPM Owner and SRK secrets

Owner authorization is required to change the Owner and SRK authorizations.

To change the TPM owner authorization: the ObjectToChange handle is the TPM Object handle and the parentObject will be NULL (0x00000000).

To change the SRK authorization: the ObjectToChange is the SRK Object handle and parentObject handle is the TPM Object handle.

4.3.3.1.6 Tspi_ChangeAuthAsym

Start of informative comment:

This method changes the authorization data (secret) of an entity (object) utilizing the asymmetric change protocol and assigns the object to the policy object. All classes using secrets provide this method for changing their authorization data.

This method changes the authorization data of an object ensuring that the parent of the object does not gain knowledge of the new secret.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_ChangeAuthAsym
(
    TSS_HOBJECT      hObjectToChange, // in
    TSS_HOBJECT      hParentObject,   // in
    TSS_HKEY         hIdentKey,       // in
    TSS_HPOLICY      hNewPolicy       // in
);
```

Parameters

hObjectToChange

Handle of the object the authorization data should be changed.

hParentObject

Handle of the parent object wrapping the object addressed by *hObjectToChange*.

hIdentKey

Handle of the identity key object required to proof the internally created temporary key.

hNewPolicy

Handle of the policy object providing the new authorization data.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_INTERNAL_ERROR

Remarks

The asymmetric change protocol requires creating a temporary asymmetric key pair. The creation of this key pair according to the rules of TCG is internally authenticated utilizing the identity key object. On successful completion of the command the object addressed by hObjectToChange is bound to the policy object addressed by hNewPolicy.

Special considerations for TPM Owner and SRK secrets

Owner authorization is required to change the Owner and SRK authorizations.

To change the TPM owner authorization: the ObjectToChange handle is the TPM Object handle and the parentObject will be NULL (0x00000000).

To change the SRK authorization: the ObjectToChange is the SRK Object handle and parentObject handle is the TPM Object handle.

4.3.3.1.7 Tspi_GetPolicyObject

Start of informative comment:

This method returns a policy object currently assigned to a working object. If an application does not create a policy object and does not assign it to the working object prior to this call, this function returns a handle to the default context policy. Setting a new secret of the default context policy will affect all future object operations associated with this policy.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_GetPolicyObject
(
    TSS_HOBJECT   hObject,   // in
    TSS_FLAG      policyType,// in
    TSS_HPOLICY*  phPolicy   // out
);
```

Parameters

hObject

Handle of the object.

policyTypet

Flag indicating the policy type of interest. (see table Defined Attributes)

phPolicy

Receives the handle to the assigned policy object.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR
TSS_E_KEY_NO_MIGRATION_POLICY

Remarks

In most cases a usage policy object is of interest (TSS_POLICY_USAGE). A few key object functions use a migration policy object (TSS_POLICY_MIGRATION).

If no usage policy object is associated with hObject, TSS_E_INERNAL_ERROR will be returned. If the policy type is invalid for hObject, or no policies can be associated with hObject, TSS_E_BAD_PARAMETER will be returned.

4.3.3.2 Tspi_Context Class Definition

Start of informative comment:

The Tspi_Context class represents a context of a connection to the TSS Core Service running on the local or a remote TCG system.

The focus of the Context object is:

- to provide a connection to a TSS Core Service. There might be multiple connections to the same or different core services.
- to provide functions for resource management and freeing of memory
- to create working objects.
- to establish a default policy for working objects as well as a policy object for the TPM object representing the TPM owner.
- to provide functionality to access the persistent storage database.

The TSP Context and the Policy-Objects provides a dynamic way to control the behavior for string terminating info in TSS_SECRET_MODE_POPUP method. The basic preference for all subsequently created Policy-Objects can be selected at the Context-Object and this arrangement will then be inherited. For fine tuning purposes the behavior can be changed for each Policy-Object instance if it is necessary.

The current selection can be retrieved by using GetAttribUint32 and can be modified by utilizing SetAttribUint32.

End of informative comment.

4.3.3.2.1 Tspi_Context_Create

Start of informative comment:

This method returns a handle to a new context object. The context handle is used in various functions to assign resources to it.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Context_Create
(
    TSS_HCONTEXT*                phContext           // out
);
```

Parameters

phContext

Receives the handle to the created context object.

Return Values

TSS_SUCCESS

TSS_E_INTERNAL_ERROR

Remarks

4.3.3.2.2 Tspi_Context_Close

Start of informative comment:

This method destroys a context and releases all assigned resources.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Context_Close
   (
   TSS_HCONTEXT  hContext   // in
   );
```

Parameters

hContext

Handle of the context object which is to be closed.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_INTERNAL_ERROR

Remarks

4.3.3.2.3 Tspi_SetAttribUint32

Start of informative comment:

This method sets a 32-bit attribute of the context object.

End of informative comment.

Definition:

See section 4.3.2 in this specification for definition.

Parameters

See section 4.3.2 in this specification for description.

Defined Attributes

Flag	SubFlag	Attribute Value	Description
TSS_TSPATTRIB_CONTEXT_SILENT_MODE		TSS_TSPATTRIB_CONTEXT_NOT_SILENT	TSP dialogs are shown (Default).
		TSS_TSPATTRIB_CONTEXT_SILENT	TSP dialogs are not shown.
TSS_TSPATTRIB_CONTEXT_VERSION_MODE		TSS_TSPATTRIB_CONTEXT_VERSION_V1_1	TSP sets the default version of the context to be 1.1 for the connection in this context object (Default). Since this is the default, it only needs to be used if the application has previously used either TSS_TSPATTRIB_CONTEXT_VERSION_V1_2 or TSS_TSPATTRIB_CONTEXT_VERSION_AUTO to change the default of the context to something else
		TSS_TSPATTRIB_CONTEXT_VERSION_V1_2	TSP switches the default version to 1.2 for the connection in this context object. Objects created after this will default to being 1.2 objects. (PCR objects will default to be long, and if other behavior is necessary, it must be set at object creation.)
		TSS_TSPATTRIB_CONTEXT_VERSION_AUTO	The TSP detects the underlying main version and sets the default for

			the context object to be the highest level consistent with both the TCS and the TPM. This avoids the necessity of software figuring this out on its own. No connection is interpretted as v1_1. If Auto sets the default to 1.2, it behaves the same as if it have been set to TSS_TS_ATTRIB_CONTEXT_VERSION_V1_2
TSS_TSPATTRIB_SECRET_HASH_MODE	TSS_TSPATTRIB_SECRET_HASH_MODE_POPUP	TSS_TSPATTRIB_HASH_MODE_NOT_NULL	TSP hashes the pass phrase excluding any terminating data.
		TSS_TSPATTRIB_HASH_MODE_NULL	TSP hashes the pass phrase including any terminating data.

Return Values

See section 4.3.2 for description.

Remarks

4.3.3.2.4 Tspi_GetAttribUint32

Start of informative comment:

This method gets a 32-bit attribute of the context object

End of informative comment.

Definition:

See section 4.3.3.1.2 for definition.

Parameters

See section 4.3.3.1.2 for description.

Defined Attributes

Flag	SubFlag	Attribute Value	Description
TSS_TSPATTRIB_CONTEXT_SILENT_MODE		TSS_TSPATTRIB_CONTEXT_NOT_SILENT	TSP dialogs are shown (Default).
		TSS_TSPATTRIB_CONTEXT_SILENT	TSP dialogs are not shown.
TSS_TSPATTRIB_CONTEXT_VERSION_MODE		TSS_TSPATTRIB_CONTEXT_VERSION_V1_1	The TSP default context (the default context used for objected created in this context) is 1.1 (This is the default for contexts.)
		TSS_TSPATTRIB_CONTEXT_VERSION_V1_2	The TSP default context (the default context used for objected created in this context) is 1.2
TSS_TSPATTRIB_CONTEXT_CONNECTION_VERSION		TSS_CONNECTION_VERSION_1_1	Indicates that the connection supports the functionality defined in the 1.1 TSS specification due to the version of the TCS, TPM, or both. Calls to 1.2 TSS functions may result in a TCS or TPM error.
		TSS_CONNECTION_VERSION_1_2	Indicates that the connection supports the functionality

			defined in the 1.2 TSS specification due to the version of the TCS, TPM, or both.
TSS_TSPATTRIB_SECRET_HASH_MODE	TSS_TSPATTRIB_SECRET_HASH_MODE_POPUP	TSS_TSPATTRIB_HASH_MODE_NOT_NULL	TSP hashes the pass phrase excluding any terminating data.
		TSS_TSPATTRIB_HASH_MODE_NULL	TSP hashes the pass phrase including any terminating data.

Return Values

See section 4.3.3.1.2 for description.

Remarks

The application can request that the TSS Service Provider implements the handling for a particular mode by selecting the mode at the Context-Object. Policy objects generated at the TSP inherits the info from the context object.

The selection is dynamically this means the application is able to change the attribute at the same context/policy on the fly or can open a different context/policy with separate settings.

4.3.3.2.5 Tspi_SetAttribData

Start of informative comment:

This method sets a non 32-bit attribute of the context object. The structure and size of the attribute data depends on the attribute.

End of informative comment.

Definition:

See section 4.3.3.1.3 for definition.

Parameters

See section 4.3.3.1.3 for parameters.

No Attributes Defined yet

Return Values

See section 4.3.3.1.3 for return values.

Remarks

4.3.3.2.6 Tspi_GetAttribData

Start of informative comment:

This method gets a non 32-bit attribute of the context object. The structure and size of the attribute data depends on the attribute.

End of informative comment.

Definition:

See section 4.3.3.1.4 for definition.

Parameters

See section 4.3.3.1.4 for description.

Defined Attributes

Flag	SubFlag	Data Description
TSS_TSPATTRIB_CONTEXT_MACHINE_NAME	0	Machine name of the TSS given as a null terminated TSS_UNICODE string.

Return Values

See section 4.3.3.1.4 for description.

Remarks

4.3.3.2.7 Tspi_Context_Connect

Start of informative comment:

This method establishes a connection to a local or remote TSS system.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Context_Connect
   (
   TSS_HCONTEXT    hContext,      // in
   TSS_UNICODE*    wszDestination // in
   );
```

Parameters

hContext

Handle of the context object

wszDestination

Pointer to a null terminated TSS_UNICODE string specifying the remote system which is to be connected.
If NULL, the context object is bound to the local system.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_NO_CONNECTION
TSS_E_INTERNAL_ERROR

Remarks

4.3.3.2.8 Tspi_Context_FreeMemory

Start of informative comment:

This method frees memory allocated by TSS Service Provider on a context base.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Context_FreeMemory
   (
   TSS_HCONTEXT  hContext,  // in
   BYTE*         rgbMemory  // in
   );
```

Parameters

hContext

Handle of the context object

rgbMemory

Pointer to the memory block to be freed.
If NULL, all allocated memory blocks bound to the context are freed.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_INTERNAL_ERROR
TSS_E_INVALID_RESOURCE

Remarks

If rgbMemory does not point to memory in use, the function will return TSS_E_INVALID_RESOURCE. (In 1.1 some implementations returned success or bad parameter as it was not defined.)

4.3.3.2.9 Tspi_Context_GetDefaultPolicy

Start of informative comment:

This method provides the default policy object of a context. (One per context).

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Context_GetDefaultPolicy
   (
   TSS_HCONTEXT  hContext,  // in
   TSS_HPOLICY*  phPolicy   // out
   );
```

Parameters

hContext

Handle of the context object

phPolicy

Receives the handle of the default policy object bound to the context.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_INTERNAL_ERROR

Remarks

4.3.3.2.10 Tspi_Context_CreateObject

Start of informative comment:

This method creates and initializes an empty object of the specified type and returns a handle addressing that object. The object is bound to an already opened context.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Context_CreateObject
   (
   TSS_HCONTEXT  hContext,  // in
   TSS_FLAG      objectType,// in
   TSS_FLAG      initFlags, // in
   TSS_HOBJECT*  phObject   // out
   );
```

Parameters

hContext

Handle of the context object

objectType

Flag indicating the object type to create (see section 2.3.2.1).

initFlags

Flag indicating the default attributes of the object (see section 2.3.2.2).

phObject

Receives the handle of the created object.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_INVALID_OBJECT_TYPE
TSS_E_INVALID_OBJECT_INIT_FLAG
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR
TSS_E_ENC_INVALID_TYPE
TSS_E_HASH_INVALID_ALG

Remarks

4.3.3.2.11 Tspi_Context_CloseObject

Start of informative comment:

This method destroys the object associated with the object handle. All allocated resources within the object are also released.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Context_CloseObject
   (
   TSS_HCONTEXT  hContext,  // in
   TSS_HOBJECT   hObject    // in
   );
```

Parameters

hContext

Handle of the context object

hObject

Handle of object to be closed.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_INTERNAL_ERROR

Remarks

4.3.3.2.12 Tspi_Context_GetCapability

Start of informative comment:

This method provides the capabilities of the TSS Core Service or TSS Service Provider.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Context_GetCapability
  (
  TSS_HCONTEXT  hContext,          // in
  TSS_FLAG      capArea,           // in
  UINT32        ulSubCapLength,    // in
  BYTE*         rgbSubCap,         // in
  UINT32*       pulRespDataLength,// out
  BYTE**        prgbRespData       // out
  );
```

Parameters

hContext

Handle of the context object

capArea

Flag indicating the attribute to query (see table Defined Attributes).

ulSubCapLength

The length (in bytes) of the *rgbSubCap* parameter.

rgbSubCap

Data indicating the attribute to query (see table Defined Attributes).

pulRespDataLength

Receives the length (in bytes) of the *prgbRespData* parameter.

prgbRespData

On successful completion of the command, this parameter points to a buffer containing the actual data of the specified capability (see table Defined Attributes).

Defined Attributes

Capability Area	SubCap Area	Response
TSS_TCSCAP_ALG	TSS_ALG_XX: A value of TSS Algorithm ID as defined in 2.3.2.19	Boolean value. TRUE indicates that the TCS supports the algorithm, FALSE indicates that the TCS does not support the algorithm.
	TSS_ALG_DEFAULT	Returns the default

		public key algorithm, A value of TSS Algorithm ID as defined in 2.3.2.19
	TSS_ALG_DEFAULT_SIZE	Returns the default public key length, a UINT32 value
TSS_TCSCAP_VERSION		Returns the TSS_VERSION structure that identifies the version of the TCS.
TSS_TCSCAP_MANUFACTURER	TSS_TCSCAP_MANUFACTURER_STR	Returns a TSS_UNICODE string of the TCS manufacturer. The contents of this string are determined by the manufacturer and are subject to change in subsequent releases of the TCS.
	TSS_TCSCAP_MANUFACTURER_ID	Returns the UINT32 value which identifies the TCS manufacturer as specified in the main specification by: Capability: TPM_CAP_PROPERTY; Sub-Capability: TPM_CAP_PROP_MANUFACTURER
TSS_TCSCAP_TRANSPORT	0	Queries the support of transport features. (TSS_BOOL)
	TSS_TCSCAP_PROP_TRANS_EXCLUSIVE	Queries whether the exclusive transport mode is supported. (TSS_BOOL)
TSS_TCSCAP_CACHING	TSS_TCSCAP_PROP_KEYCACHE	Boolean value. TRUE indicates that the TCS supports key caching, FALSE indicates that the TCS does not support key caching.
TSS_TCSCAP_CACHING	TSS_TCSCAP_PROP_AUTHCACHE	Boolean value. TRUE indicates that the TCS supports authorization session caching, FALSE indicates that the TCS does not support authorization session caching.
TSS_TCSCAP_PERSSTORAGE		Boolean value. TRUE

		indicates that the TCS supports persistent storage, FALSE indicates that the TCS does not support persisten storage.
TSS_TCSCAP_PLATFORM_CLASS	TSS_TCSCAP_PROP_HOST_PLATFORM	Returns a single TSS_PLATFORM_CLASS structure containing the definition of only the host platform's class
	TSS_TCSCAP_PROP_ALL_PLATFORMS	Returns an array of TSS_PLATFORM_CLASS strucutures which enumerates all the TCG_defined platforms associated with the Host Platform. The Host Platform MUST NOT be returned as one of these platform classes. There is no relationship required in the order of the platforms listed.
TSS_TSPCAP_ALG	TSS_ALG_DEFAULT	Returns the default public key algorithm
TSS_TSPCAP_ALG	TSS_ALG_DEFAULT_SIZE	Returns the default public key length
TSS_TSPCAP_ALG	TSS_ALG_XX: A value of TSS Algorithm ID as defined in 2.3.2.19	Boolean value. TRUE indicates that the TSP supports the algorithm, FALSE indicates that the TSP does not support the algorithm.
TSS_TSPCAP_VERSION		Returns the TSS_VERSION structure that identifies the version of the TSP.
TSS_TSPCAP_PERSSTORAGE		Boolean value. TRUE indicates that the TSP supports persistent storage, FALSE indicates that the TSP does not support persistent storage.
TSS_TSPCAP_MANUFACTURER	TSS_TSPCAP_PROP_MANUFACTURER_ID	UINT32 value. Returns the Identifier of the TSP manufacturer.
	TSS_TSPCAP_PROP_M	NULL terminated

	ANUFACTURER_STR	TSS_UNICODE string. Returns the label of the TSP manufacturer.
TSS_TSPCAP_RETURNVALUE_INFO	TSS_TSPCAP_PROP_RETURNVALUE_INFO	0 indicates ASN.1 encoding, 1 indicates byte stream
TSS_TSPCAP_PLATFORM_INFO	TSS_TSPCAP_PLATFORM_TYPE	Type of platform
	TSS_TSPCAP_PLATFORM_VERSION	Version of platform type (return 0 if not known)

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The Tspi_Context_GetCapability method allocates a memory block for the requested capability data. This memory must be released utilizing the Tspi_Context_FreeMemory method.

4.3.3.2.13 Tspi_Context_GetTPMObject

Start of informative comment:

This method retrieves the TPM object of a context. Only one instance of this object exists for a given context and implicitly represents a TPM owner.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Context_GetTpmObject
(
TSS_HCONTEXT  hContext,  // in
TSS_HTPM*     phTPM      // out
);
```

Parameters

hContext

Handle of the context object

phTPM

Receives the handle of the TPM object bound to the context.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

4.3.4 Encryption Transport Session

In the remote scenario (as well as in the local scenario) there's a need to protect secret data in plain text e.g. a session key during transmission from an application utilizing the TSP (in-process server) to the TPM and vice versa.

Basically the TPM commands

- TPM_Unbind
- TPM_Seal
- TPM_Unseal

will require data protection during transmission.

Another requirement of transport protection are situations where a new secret has to be established (e.g. in an OSAP session) and for that purpose gets protection by a parent entity's secret. The parent entity's secret may be well known or may be known to other users.

Basically the TPM commands

- TPM_CreateWrapKey
- TPM_ChangeAuth
- TPM_ChangeAuthOwner…

will require data protection for the new secret data to be established.

In addition the TPM commands

- TPM_CreateMigrationBlob
- TPM_ConvertMigrationBlob

if used in mode TPM_MS_MIGRATE may require protection for the one-time pad.

The TSP will process transport protection on behalf of the application in various ways. The TSP context object will expose configuration possibilities for encrypted transport and methods to turn logging on and off. In either case the TSP will hide all the burden of handling the transport session with transport HMACing, encryption etc. from the application the same way as it does with OIAP / OSAP sessions via the policy object mechanism.

An application may require an authentic, integrity secured communication channel with the TPM in the remote as well as in the local scenario, also with entities that don't require authorization by themselves e.g. unauthorized keys, and also with unauthorized TPM commands.

4.3.4.1 Extensions of TSP-Context-Object

By default the TSP uses a non-migrate-able storage key to establish the transport session (e.g. an ephemeral non-migrate-able storage key generated on behalf of the TSP for the lifetime of a TSP context object with random authorization data or a dedicated non-migrateable system key from the persistent storage).

If this default encryption key should not be used to shelter the transport session shared secret, any other key can be provided by the object reference (key object pointer, key handle) to the TSP context object via a specific function.

The transport session secret will be generated as a random number (i.e. Key) by the TSP, in either case of the transport session, in case the transport session secret is submitted in plain text to the TPM or in case it gets submitted encrypted. The TSP uses TPM default encryption algorithm for an established transport session.

4.3.4.1.1 Tspi_SetAttribUint32

Start of informative comment:

This method sets a 32-bit attribute of the policy object.

End of informative comment.

Definition:

See section 4.3.2 for definition.

Parameters

See section 4.3.2 for description.

Defined Attributes

Flag	SubFlag	Attribute Value	Description
TSS_TSPATTRIB_CONTEXT_TRANSPORT	TSS_TSPATTRIB_CONTEXTTRANS_CONTROL	TSS_TSPATTRIB_DISABLE_TRANSPORT	Disables the transport functionality.
		TSS_TSPATTRIB_ENABLE_TRANSPORT	Enables the transport functionality.
TSS_TSPATTRIB_CONTEXT_TRANSPORT	TSS_TSPATTRIB_CONTEXTTRANS_MODE	TSS_TSPATTRIB_TRANSPORT_NO_DEFAULT_ENCRYPTION	Turn off the default encrytion for the selected operations.
		TSS_TSPATTRIB_TRANSPORT_DEFAULT_ENCRYPTION	Turn on the default encrytion for the selected operations.
		TSS_TSPATTRIB_TRANSPORT_AUTHENTIC_CHANNEL	The logging mode is enabled for the transport session.
		TSS_TSPATTRIB_TRANSPORT_EXCLUSIVE	The exclusive command mode for the transport session is enabled.
		TSS_TSPATTRIB_TRANSPORT_STATIC_AUTH	Sets the transport session to use the reserved auth handle. See also TPM_KH_TRANSPORT.

Remarks

The condition of the default behavior for the TSP regarding the transport protection (i.e. enabled/disabled) functionality is TSS vendor orientated.

4.3.4.1.2 Tspi_GetAttribUint32

Start of informative comment:

This method gets a 32-bit attribute of the policy object

End of informative comment.

Definition:

See section 4.3.3.1.2 for definition.

Parameters

See section 4.3.3.1.2 for description.

Defined Attributes

Flag	SubFlag	Attribute Value	Description
TSS_TSPATTRIB_CONTEXT_TRANSPORT	TSS_TSPATTRIB_CONTEXTTRANS_CONTROL	TSS_TSPATTRIB_DISABLE_TRANSPORT	Gets if transport functionality is disabled.
		TSS_TSPATTRIB_ENABLE_TRANSPORT	Gets if the transport functionality is enabled.
TSS_TSPATTRIB_CONTEXT_TRANSPORT	TSS_TSPATTRIB_CONTEXTTRANS_MODE	TSS_TSPATTRIB_TRANSPORT_NO_DEFAULT_ENCRYPTION	Checks the default encrytion for the selected operations is off.
		TSS_TSPATTRIB_TRANSPORT_DEFAULT_ENCRYPTION	Checks the default encrytion for the selected operations is on.
		TSS_TSPATTRIB_TRANSPORT_AUTHENTIC_CHANNEL	Checks if the logging mode is enabled for the transport session.
		TSS_TSPATTRIB_TRANSPORT_EXCLUSIVE	Checks if the exclusive command mode for the transport session is enabled.
		TSS_TSPATTRIB_TRANSPORT_STATIC_AUTH	Checks if the transport session is using the reserved auth handle. See also TPM_KH_TRANSPORT.

Remarks

The condition of the default behavior for the TSP regarding the transport protection (i.e. enabled/disabled) functionality is TSS vendor orientated.

4.3.4.1.3 Tspi_Context_SetTransEncryptionKey

Start of informative comment:

This method can be used to set a specific transport session secret encryption key for this context object.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Context_SetTransEncryptionKey
   (
   TSS_HCONTEXT  hContext,  // in
   TSS_HKEY      hKey       // in
   );
```

Parameters

hContext

Handle of the context object

hKey

Handle of the key object addressing the encryption key for the transport session establishment.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The required key information must be set into the key object addressed by hKey via Tspi_SetAttribData( ) before this method is called.

4.3.4.1.4 Tspi_Context_CloseSignTransport

Start of informative comment:

This method completes a transport session, if logging for this session is turned on, then this command performs a signature verification of the hashed operations.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Context_CloseSignTransport
(
  TSS_HCONTEXT     hContext,          // in
  TSS_HKEY         hSigningKey,       // in
  TSS_VALIDATION*  pValidationData    // in, out (may be NULL)
);
```

Parameters

hContext

Handle of the context object

hSigningKey

Handle to the signing key used to sign the addressed logged transport data.

pValidationData

Validation data structure
NULL the validation is done by the TSP internally.
[OUT] On successful completion of the command, the structure provides a buffer containing the validation data and a buffer containing the data the validation data was computed from.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The required key information must be set into the key object addressed by hSigningKey via Tspi_SetAttribData( ) before this method is called.

4.3.4.2 Finding, Loading, and Registering Keys in a Context

4.3.4.2.1 Tspi_Context_LoadKeyByBlob

Start of informative comment:

This method creates a key object based on the information contained in the key blob and loads the key into the TPM which unwraps the key blob utilizing the key addressed by hUnwrappingKey. The key blob addressed by hUnwrappingKey must have been loaded previously into the TPM. The function returns a handle to the created key object by hKey.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Context_LoadKeyByBlob
(
   TSS_HCONTEXT           hContext,        // in
   TSS_HKEY               hUnwrappingKey,  // in
   UINT32                 ulBlobLength,    // in
   BYTE*                  rgbBlobData,     // in
   TSS_HKEY*              phKey            // out
) ;
```

Parameters

hContext

Handle of the context object

hUnwrappingKey

Handle of the key object addressing the key which should be used to the key information provided by the *rgbBlobData* parameter.

ulBlobLength

The length (in bytes) of the *rgbBlobData* parameter.

rgbBlobData

The wrapped key blob to load.

phKey

Receives the handle of the key object representing the loaded key.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

4.3.4.2.2 Tspi_Context_LoadKeyByUUID

Start of informative comment:

This method creates a key object based on the information contained in the key manager using the UUID and loads the key into the TPM. The persistent storage provides all information to load the parent keys required to load the key associated with the given UUID.

There are some subtle cases that need to be considered when using this command with parent keys that may require authorization.

If none of the registered keys require authorization, the application can use Tspi_Context_LoadKeyByUUID() as specified below without error or alerts.

If one of the registered keys requires authorization, and the application knows the registered key stack, it must get the key from the key database by Tspi_Context_GetKeyByUUID(), assign a policy to the key and load the key by Tspi_Key_LoadKey().

If one of the registered keys requires authorization, and the application doesn't know the key stack, it must first retrieve the key stack information from the key database by calling Tspi_Context_GetRegisteredKeysByUUID() and then continue as in the above paragraph.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Context_LoadKeyByUUID
(
   TSS_HCONTEXT                     hContext,              // in
   TSS_FLAG                         persistentStorageType, // in
   TSS_UUID                         uuidData,              // in
   TSS_HKEY*                        phKey                  // out
);
```

Parameters

hContext

Handle of the context object

persistentStorageType

Flag indicating the persistent storage (see section 2.3.2.22) the key is registered in.

uuidData

The UUID of the key by which the key was registered in the persistent storage (TSP or connected TCS).

phKey

Receives the handle of the key object representing the loaded key.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_PS_KEY_NOTFOUND
TSS_E_INTERNAL_ERROR

Remarks

4.3.4.2.3 Tspi_Context_RegisterKey

Start of informative comment:

This method registers a key in the TSS Persistent Storage database.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Context_RegisterKey
(
    TSS_HCONTEXT        hContext,                    // in
    TSS_HKEY            hKey,                        // in
    TSS_FLAG            persistentStorageType,           // in
    TSS_UUID            uuidKey,                     // in
    TSS_FLAG            persistentStorageTypeParent,     // in
    TSS_UUID            uuidParentKey                // in
    );
```

Parameters

hContext

Handle of the context object

hKey

Handle of the key object addressing the key to be registered.

persistentStorageType

Flag indicating the persistent storage (see section 2.3.2.22) the key is registered in.

uuidKey

The UUID by which the key is registered in the persistent storage (TSP or connected TCS).

persistentStorageTypeParent

Flag indicating the persistent storage (see section 2.3.2.22) the parent key is registered in.

uuidParentKey

The UUID by which the parent key was registered in the persistent storage (TSP or connected TCS).

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_PS_KEY_NOTFOUND
TSS_E_INTERNAL_ERROR

Remarks

The required key information must be set in the key object by Tspi_SetAttribData( ) before this method is called.

A registered key contains all information required for loading the key into the TPM plus additional information about its parent key.

4.3.4.2.4 Tspi_Context_UnregisterKey

Start of informative comment:

This method unregisters a key from the persistent storage database.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Context_UnregisterKey
(
   TSS_HCONTEXT                hContext,             // in
   TSS_FLAG                    persistentStorageType,     // in
   TSS_UUID                    uuidKey,              // in
   TSS_HKEY*                   phkey                      //
                               out
);
```

Parameters

hContext

Handle of the context object

persistentStorageType

Flag indicating the persistent storage (see section 2.3.2.22) the key is registered in.

uuidKey

The UUID of the key to be removed from the persistent storage (TSP or connected TCS).

phKey

Receives the handle of a key object containing the info from the archive.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_PS_KEY_NOTFOUND
TSS_E_INTERNAL_ERROR

Remarks

The key's usage authorization value is not required to unregister a key.

4.3.4.2.5 Tspi_Context_GetKeyByUUID

Start of informative comment:

This method searches the persistent storage for a registered key using the provided UUID and creates a key object initialized according to the found data. On successful completion of the method a handle to the created new key object is returned.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Context_GetKeyByUUID
   (
   TSS_HCONTEXT  hContext,                // in
   TSS_FLAG      persistentStorageType,   // in
   TSS_UUID      uuidData,                // in
   TSS_HKEY*     phKey                    // out
   );
```

Parameters

hContext

Handle of the context object

persistentStorageType

Flag indicating the persistent storage (see section 2.3.2.22) where the key is registered .

uuidData

The UUID of the key by which the key was registered in the persistent storage (TSP or connected TCS)

phKey

Receives the handle of the key object representing the key.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_PS_KEY_NOTFOUND
TSS_E_INTERNAL_ERROR

Remarks

4.3.4.2.6 Tspi_Context_GetKeyByPublicInfo

Start of informative comment:

This method searches the persistent storage for a registered key using the provided public key information and creates a key object initialized according to the found data. On successful completion of the method a handle to the created new key object is returned.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Context_GetKeyByPublicInfo
(
TSS_HCONTEXT       hContext,                // in
TSS_FLAG           persistentStorageType,   // in
TSS_ALGORITHM_ID   algID,                   // in
UINT32             ulPublicInfoLength,      // in
BYTE*              rgbPublicInfo,           // in
TSS_HKEY*          phKey                    // out
);
```

Parameters

hContext

Handle of the context object

persistentStorageType

Flag indicating the persistent storage (see section 2.3.2.22) where the key is registered.

algID

This parameter indicates the algorithm of the requested key.

ulPublicInfoLength

The length of the public key info provided at the parameter *rgbPublicInfo*.

rgbPublicInfo

The public key info provided to identify the key to be looked for in the persistent storage. In case *algID* equals TSS_ALG_RSA, this parameter contains the modulus of the public RSA key.

hKey

Receives the handle of the key object representing the key. In case the key hasn't been found, this value will be NULL.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_PS_KEY_NOTFOUND

TSS_E_INTERNAL_ERROR

Remarks

If the key identified by the public key info was not found at the persistent storage the method will return TSS_E_PS_KEY_NOTFOUND.

4.3.4.2.7 Tspi_Context_GetRegisteredKeysByUUID

Start of informative comment:

This method gets an array of TSS_KM_KEYINFO structures. This information reflects the registered key hierarchy. The keys stored in the persistent storage are totally independent from either the context provided in the function call or the context, which was provided while processing the key registration.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Context_GetRegisteredKeysByUUID
(
    TSS_HCONTEXT      hContext,                // in
    TSS_FLAG          persistentStorageType,   // in
    TSS_UUID*         pUuidData                // in
    UINT32*           pulKeyHierarchySize,     // out
    TSS_KM_KEYINFO**  ppKeyHierarchy           // out
);
```

Parameters

hContext

Handle of the context object

persistentStorageType

Flag indicating the persistent storage (see section 2.3.2.21) the key is registered in. This field will be ignored if pUuidData is NULL.

pUuidData

The UUID the key was registered in the persistent storage (TSP or connected TCS).

If this field is set to NULL, the returned array of TSS_KM_KEYINFO structures contains data reflecting the entire key hierarchy starting with storage root key. The array will include keys from both the user and the system TSS key store. The persistentStorageType field will be ignored.

If a certain key UUID is provided, the returned array of TSS_KM_KEYINFO structures only contains data reflecting the path of the key hierarchy regarding that key. The first array entry is the key addressed by the given UUID followed by its parent key up to and including the root key.

pulKeyHierarchySize

Receives the length (number of array entries) of the ppK*eyHierarchy* parameter.

ppKeyHierarchy

On successful completion of the command, this parameter points to a buffer containing the actual key hierarchy data.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The Tspi_Context_GetRegisteredKeysByUUID method allocates a memory block for the requested key hierarchy data. This memory must be released utilizing the Tspi_Context_FreeMemory method.

This array will return information of the whole registered key hierarchy independent from any context.

If no keys have been registered *pulKeyHierarchySize = 0 and *ppKeyHierarchy = NULL and the function returns with TSS_SUCCESS.

To determine the validity of the key found in the TSS, the application can load it into the TPM. TPM will attempt to decrypt it using the previously loaded parent and if the key is invalid or corrupt the process will fail with TPM_DECRYPT_ERROR error. This process can be used if the TPM ownership was reset and the SRK has changed since the key was created.

4.3.4.2.8 Tspi_Context_GetRegisteredKeysByUUID2

Start of informative comment:

This method gets an array of TSS_KM_KEYINFO2 structures. This information reflects the registered key hierarchy. The keys stored in the persistent storage are totally independent from either the context provided in the function call or the context, which was provided while processing the key registration. This method is identical to GetRegisteredKeysByUUID except that it returns an array of TSS_KM_KEYINFO2 structures.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Context_GetRegisteredKeysByUUID2
(
    TSS_HCONTEXT       hContext,                // in
    TSS_FLAG           persistentStorageType,   // in
    TSS_UUID*          pUuidData,               // in
    UINT32*            pulKeyHierarchySize,     // out
    TSS_KM_KEYINFO2**  ppKeyHierarchy           // out
);
```

Parameters

hContext

Handle of the context object

persistentStorageType

Flag indicating the persistent storage (see section 2.3.2.21) the key is registered in.

pUuidData

The UUID the key was registered in the persistent storage (TSP or connected TCS).

If this field is set to NULL, the returned array of TSS_KM_KEYINFO2 structures contains data reflecting the entire key hierarchy starting with root key. The array will include keys from both the user and the system TSS key store. The persistentStorageType field will be ignored.

If a certain key UUID is provided, the returned array of TSS_KM_KEYINFO structures only contains data reflecting the path of the key hierarchy regarding that key. The first array entry is the key addressed by the given UUID followed by its parent key up to and including the root key.

pulKeyHierarchySize

Receives the length (number of array entries) of the ppK*eyHierarchy* parameter.

ppKeyHierarchy

On successful completion of the command, this parameter points to a buffer containing the actual key hierarchy data.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The Tspi_Context_GetRegisteredKeysByUUIDs method allocates a memory block for the requested key hierarchy data. This memory must be released utilizing the Tspi_Context_FreeMemory method.

This array will return information of the whole registered key hierarchy independent from any context.

If no keys have been registered *pulKeyHierarchySize = 0 and *ppKeyHierarchy = NULL and the function returns with TSS_SUCCESS.

4.3.4.2.9 Tspi_TPM_KeyControlOwner

Start of informative comment:

This command controls some attributes of a loaded key. It requires owner authorization.

One of the defined attributes is TSS_TSPATTRIB_KEYCONTROL_OWNEREVICT If this bit is set to true, this key remains loaded in the TPM through all TPM_Startup events. The only way to evict this key is for the TPM Owner to execute this command again, setting the owner control bit to false and then executing Tspi_Key_UnloadKey.

This call will result in a TPM_KeyControlOwner operation for the specified key.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_KeyControlOwner
(
    TSS_HTPM        hTPM,           // in
    TSS_HKEY        hKey,           // in
    UINT32          attribName,     // in
    TSS_BOOL        attribValue,    // in
    TSS_UUID*       pUuidData       // out
);
```

Parameters

hTPM

Handle of the TPM object with an authorization session handle attached.

hKey

Handle of the key object addressing the key which property is to be modified.

attribName

The name of the bit to be modified.

attribValue

The value to set the bit to.

uUuidData

The UUID the key was registered as a TPM resident key. See remarks section for more details..

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR
TPM_AUTHFAIL

Remarks

Once the key is made resident in the TPM by this command, it is given one of the reserved UUIDs. This UUID is used to obtain a handle to this key at a later time. The UUID is volatile – associated with the key by the TSS at boot, so once used to obtain a handle, the key referenced should be checked against a public key to make certain it is the correct key the user wants to use. Alternatively an application can loop through the possible predefined set of UUIDs using GetKeybyUUID and comparing the results against a public key to determine the correct handle In most cases the UUID will not change, but if one of the keys resident in the TPM is removed, a reordering of the UUIDs could take place for other keys resident in the TPM.

4.3.4.3 TSPI_Policy Class Definition

Start of informative comment:

The Tspi_Policy class represents authorization data (secrets), authorization data handling and the assigned authorized objects like key objects or encrypted data objects.

Authorization data and secret will be used synonymously.

Secret Lifetime

If an application uses the mode TSS_SECRET_LIFETIME_COUNTER or TSS_SECRET_LIFETIME_TIMER, the application has to be aware that during a command processing the secret may be invalidated because of a time out or because the counter runs out.

TSPI Default Policy

Each context has its own default policy object that is automatically assigned to a new key or encrypted data object after its creation. If this policy object is not appropriate, a different policy object can be assigned with the function Tspi_Policy_AssignToObject(…).

The default policy object has following settings after initialization:

Secret mode = TSS_SECRET_MODE_POPUP

Secret lifetime mode = SECRET_LIFTIME_ALWAYS

Policy Callbacks

For applications that wish to be compatible with version 1.1. of the TSS specification, Tspi_SetAttribUint32 should be used to set all callbacks. For all others, Tspi_SetAttribData should be used with a TSS_CALLBACK structure.

Hash Termination

The TSP Context and the Policy-Objects provides a dynamic way to control the behavior for string terminating info in TSS_SECRET_MODE_POPUP method. The basic preference for all subsequently created Policy-Objects can be selected at the Context-Object and this arrangement will then be inherited. For fine tuning purposes the behavior can be changed for each Policy-Object instance if it is necessary.

The current selection can be retrieved by using GetAttribUint32 and can be modified by utilizing SetAttribUint32.

End of informative comment.

4.3.4.3.1 Tspi_SetAttribUint32

Start of informative comment:

This method sets a 32-bit attribute of the policy object.

End of informative comment.

Definition:

See section 4.3.2 for definition.

Parameters

See section 4.3.2 for description.

Defined Attributes

Flag	SubFlag	Attribute	Description
TSS_TSPATTRIB_POLICY_CALLBACK_HMAC	Application provided data.	Address of callback or NULL (disable)	Provided for TSS v1.1 compatibility
TSS_TSPATTRIB_POLICY_CALLBACK_XOR_ENC	Application provided data.	Address of callback or NULL (disable)	Provided for TSS v1.1 compatibility
TSS_TSPATTRIB_POLICY_CALLBACK_TAKEOWNERSHIP	Application provided data.	Address of callback or NULL (disable)	Provided for TSS v1.1 compatibility
TSS_TSPATTRIB_POLICY_CALLBACK_CHANGEAUHTASYM	Application provided data.	Address of callback or NULL (disable)	Provided for TSS v1.1 compatibility
TSS_TSPATTRIB_POLICY_SECRET_LIFETIME	TSS_TSPATTRIB_POLSECRET_LIFETIME_ALWAYS	Not important- it is the same as putting the secret in the policy object.	Secret will not be invalidated.
	TSS_TSPATTRIB_POLSECRET_LIFETIME_COUNTER	Counter value	Secret may be used n-times.
	TSS_TSPATTRIB_POLSECRET_LIFETIME_TIMER	Time value in seconds	Secret will be valid for n seconds.
TSS_TSPASTTRID_POLICY_DELEGATION_INFO	TSS_TSPATTRIB_POLDEL_OWNERBLOB		This data blob contains all the information necessary to externally store a set of owner delegation rights that can subsequently be used to create a policy object.
	TSS_TSPATTRIB_POLDEL_KEYBLOB		This data blob contains all the

			information necessary to externally store a set of key delegation rights that can subsequently be used to create a policy object.
	TSS_TSPATTRIB_POLDEL_PER1	UINT32	Permission bits. See TPM specification for the bitmap used for selecting command to delegate.
TSS_TSPATTRIB_POLICY_DELEGATION_PCR	TSS_TSPATTRIB_KEYPCR_DIGEST_ATRELEASE		Composite digest value of the PCR values, at the time when the delegation can be used.
	TSS_TSPATTRIB_KEYPCR_SELECTION		The selection of PCRs that specifies the digestAtRelease
TSS_TSPATTRIB_SECRET_HASH_MODE	TSS_TSPATTRIB_SECRET_HASH_MODE_POPUP	TSS_TSPATTRIB_HASH_MODE_NOT_NULL	TSP hashes the pass phrase excluding any terminating data.
		TSS_TSPATTRIB_HASH_MODE_NULL	TSP hashes the pass phrase including any terminating data.

Return Values

See section 4.3.2 for description.

Remarks

The lifetime is decreased as soon as the secret is set using the Tspi_Policy_SetSecret( ) method. After invalidation of a secret a further call to Tspi_Policy_SetSecret( ) will start the processing again.

4.3.4.3.2 Tspi_GetAttribUint32

Start of informative comment:

This method gets a 32-bit attribute of the policy object

End of informative comment.

Definition:

See section 4.3.3.1.2 for definition.

Parameters

See section 4.3.3.1.2 for description.

Defined Attributes

Flag	SubFlag	Attribute	Description
TSS_TSPATTRIB_POLICY_CALLBACK_HMAC	Application provided data.	Address of callback or NULL (disable)	Provided for TSS v1.1 compatibility
TSS_TSPATTRIB_POLICY_CALLBACK_XOR_ENC	Application provided data.	Address of callback or NULL (disable)	Provided for TSS v1.1 compatibility
TSS_TSPATTRIB_POLICY_CALLBACK_TAKEOWNERSHIP	Application provided data.	Address of callback or NULL (disable)	Provided for TSS v1.1 compatibility
TSS_TSPATTRIB_POLICY_CALLBACK_CHANGEAUHTASYM	Application provided data.	Address of callback or NULL (disable)	Provided for TSS v1.1 compatibility
TSS_TSPATTRIB_POLICY_SECRET_LIFETIME	TSS_TSPATTRIB_POLSECRET_LIFETIME_ALWAYS	TRUE if the flag is set in the policy object, FALSE if not.	Secret will not be invalidated.
	TSS_TSPATTRIB_POLSECRET_LIFETIME_COUNTER	Counter value	Secret may be used n-times.
	TSS_TSPATTRIB_POLSECRET_LIFETIME_TIMER	Time value in seconds	Secret will be valid for n seconds.
TSS_TSPASTTRIB_POLICY_DELEGATION_INFO	TSS_TSPATTRIB_POLDEL_TYPE	UINT32	Owner or key
	TSS_TSPATTRIB_POLDEL_INDEX	UINT32	Existing or new index
	TSS_TSPATTRIB_POLDEL_PER1	UINT32	Permission bits. See TPM specification for the bitmap used for selecting command to delegate.

	TSS_TSPATTRIB_POLDEL_PER2	UINT32	Permission bits. Reserved. Must be 0.
	TSS_TSPATTRIB_POLDEL_LABEL	BYTE	A byte that can be displayed or used by the applications. This MUST not contain sensitive information
	TSS_TSPATTRIB_POLDEL_FAMILYID	UINT32	The family ID that identifies which family the row belongs to.
	TSS_TSPATTRIB_POLDEL_VERCOUNT	UINT32	A copy of verificationCount from the associated family table.
TSS_TSPATTRIB_DELEGATION_PCR	TSS_TSPATTRIB_POLDELPCR_LOCALITY	UINT32	The locality modifier required for this delegation
TSS_TSPATTRIB_SECRET_HASH_MODE	TSS_TSPATTRIB_SECRET_HASH_MODE_POPUP	TSS_TSPATTRIB_HASH_MODE_NOT_NULL	TSP hashes the pass phrase excluding any terminating data.
		TSS_TSPATTRIB_HASH_MODE_NULL	TSP hashes the pass phrase including any terminating data.

Return Values

See section 4.3.3.1.2 for description.

The lifetime is decreased as soon as the secret is set using the Tspi_Policy_SetSecret( ) method. After invalidation of a secret a further call to Tspi_Policy_SetSecret( ) will start the processing again.

Remarks:

The application can request that the TSS Service Provider implements the handling for a particular mode by selecting the mode at the Context-Object. Policy objects generated at the TSP inherits the info from the context object.

The selection is dynamically this means the application is able to change the attribute at the same context/policy on the fly or can open a different context/policy with separate settings.

4.3.4.3.3 Tspi_SetAttribData

Start of informative comment:

This method sets a non 32-bit attribute of the policy object. The structure and size of the attribute data depends on the attribute.

End of informative comment.

Definition:

See section 4.3.3.1.3 for definition.

Parameters

See section 4.3.3.1.3 for description.

Defined Attributes

Flag	SubFlag	Attribute	Description
TSS_TSPATTRIB_POLICY_DELEGATION_INFO	TSS_TSPATTRIB_POLDEL_OWNERBLOB		This data blob contains all the information necessary to externally store a set of owner elegation rights that can subsequently be used to create a policy object.
	TSS_TSPATTRIB_POLDEL_KEYBLOB		This data blob contains all the information necessary to externally store a set of key delegation rights that can subsequently be used to create a policy object.
TSS_TSPATTRIB_POLICY_CALLBACK_HMAC	Ignored.	Address of a TSS_CALLBACK structure or NULL (disable)	The callback pointer in the TSS_CALLBACK structure shall contain the address of the callback.
TSS_TSPATTRIB_POLICY_CALLBACK_XOR_ENC	Ignored.	Address of a TSS_CALLBACK structure or NULL (disable)	The callback pointer in the TSS_CALLBACK structure shall contain the address of the callback.
TSS_TSPATTRIB_POLICY_CALLBACK_TAKEOWNERSHIP	Ignored.	Address of a TSS_CALLBACK structure	The callback pointer in the TSS_CALLBACK structure shall contain the address of the callback.

		or NULL (disable)	
TSS_TSPATTRIB_POLICY_CALLBACK_CHANGEAUHTASYM	Ignored.	Address of a TSS_CALLBACK structure or NULL (disable)	The callback pointer in the TSS_CALLBACK structure shall contain the address of the callback.
TSS_TSPATTRIB_POLICY_POPUPSTRING	0	POPUPSTRING	Text for popup
TSS_TSPATTRIB_POLICY_CALLBACK_SEALX_MASK	Ignored.	Address of a TSS_CALLBACK structure or NULL (disable)	Set the address of a callback function to be used to mask the data input to a Sealx command

Return Values

See section 4.3.3.1.3 for description.

Remarks

4.3.4.3.4 Tspi_GetAttribData

Start of informative comment:

This method gets a non 32-bit attribute of the policy object. The structure and size of the attribute data depends on the attribute.

End of informative comment.

Definition:

See section 4.3.3.1.4 for definition.

Parameters

See section 4.3.3.1.4 for description.

Defined Attributes

Flag	SubFlag	Attribute	Description
TSS_TSPASTTRIB_POLICY_DELEGATION_INFO	TSS_TSPATTRIB_POLDEL_OWNERBLOB		This data blob contains all the information necessary to externally store a set of owner elegation rights that can subsequently be used to create a policy object.
	TSS_TSPATTRIB_POLDEL_KEYBLOB		This data blob contains all the information necessary to externally store a set of key delegation rights that can subsequently be used to create a policy object.
TSS_TSPATTRIB_POLICY_DELEGATION_PCR	TSS_TSPATTRIB_KEYPCR_DIGEST_ATRELEASE		Composite digest value of the PCR values, at the time when the delegation can be used.
	TSS_TSPATTRIB_KEYPCR_SELECTION		The selection of PCRs that specifies the digestAtRelease
TSS_TSPATTRIB_POLICY_CALLBACK_HMAC	Ignored.	Address of a TSS_CALLBACK structure or NULL (disable)	The callback pointer in the TSS_CALLBACK structure shall contain the address of the callback.
TSS_TSPATTRIB_POLICY_CALLBACK_XOR_ENC	Ignored.	Address of a	The callback pointer in the TSS_CALLBACK

		TSS_CALLBACK structure or NULL (disable)	structure shall contain the address of the callback.
TSS_TSPATTRIB_POLICY_CALLBACK_TAKEOWNERSHIP	Ignored.	Address of a TSS_CALLBACK structure or NULL (disable)	The callback pointer in the TSS_CALLBACK structure shall contain the address of the callback.
TSS_TSPATTRIB_POLICY_CALLBACK_CHANGEAUHTASYM	Ignored.	Address of a TSS_CALLBACK structure or NULL (disable)	The callback pointer in the TSS_CALLBACK structure shall contain the address of the callback.
TSS_TSPATTRIB_POLICY_POPUPSTRING	0	POPUPSTRING	Text for popup
TSS_TSPATTRIB_POLICY_CALLBACK_SEALX_MASK	Ignored.	Address of a TSS_CALLBACK structure or NULL (disable)	Get the address of a callback function to be used to mask the data input to a Sealx command

Return Values

See section 4.3.3.1.4 for description.

Remarks

4.3.4.3.5 Tspi_Policy_SetSecret

Start of informative comment:

This method sets the authorization data of a policy object and defines the handling of its retrieval.

Some previous TSS implementations included the 16-bit null character at the end of the passphrase from TSS_SECRET_MODE_POPUP, and some did not. To allow migration of keys from any previous TSS implementation, the popup provides a mechanism to select whether the null character is included. Note that a key object's authorization can be changed after migration using Tspi_ChangeAuth (although the migration authorization cannot be changed this way).

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Policy_SetSecret
(
TSS_HPOLICY   hPolicy,         // in
TSS_FLAG      secretMode,      // in
UINT32        ulSecretLength,  // in
BYTE*         rgbSecret        // in
);
```

Parameters

hPolicy

Handle of the policy object

secretMode

Flag indicating the policy secret mode to set (see section 2.3.2.16).

ulSecretLength

The length (in bytes) of the *rgbSecret* parameter.

rgbSecret

The secret data blob.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

If the secret mode does not require any authorization data, the parameter *ulSecretLength* can be 0 and the parameter *rgbSecret* can be NULL.

If the *ulSecretLength* is set to zero and the *rgbSecret* is set to NULL a valid NULL password will be associated with the object. In other words, if a NULL password was used during key generation, the same NULL password will be required in order to generate a child of this key. Additionally, if the key was generated using TSS_KEY_AUTHORIZATION flag, the same NULL password will be required to use this key in TSS operation which require key usage authorization, such as signing.

Secret-Mode

TSS_SECRET_MODE_NONE:

- No authorization will be processed of all assigned working object (e.g. Key-Object);
- Different from a secret of 20 bytes of 0x00.
- (ulSecretLength == 0 and rgbSecret == NULL)

TSS_SECRET_MODE_SHA1:

- Secret string will not be touched by TSP and MUST be size of 20 bytes.
- (ulSecretLength == 20 and rgbSecret points to the hashed secret byte stream

TSS_SECRET_MODE_PLAIN:

- Secret string will be hashed by the TSP using SHA1.
- ulSecretLength contains the length of the secret string

(e.g. ulSecretLength = StringLen(strSecret);

- - rgbSecret points to the first byte of the secret string stream

TSS_SECRET_MODE_POPUP:

The TSP will prompt the user to enter a passphrase, which is represented as a TSS_UNICODE string. The TSP MUST hash the passphrase using SHA1 to create the authorization secret. A mechanism MUST be provided to select whether the 16-bit null character that terminates the TSS_UNICODE string is included in the SHA1 hash. The null character SHOULD ***NOT*** be included by default.

4.3.4.3.6 Tspi_Policy_FlushSecret

Start of informative comment:

The function flushes a cached secret.

End of informative comment.

Definition

```
TSS_RESULT Tspi_Policy_FlushSecret
  (
  TSS_HPOLICY   hPolicy // in
  );
```

Parameters

hPolicy

Handle of the policy object

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_INTERNAL_ERROR

Remarks

4.3.4.3.7 Tspi_Policy_AssignToObject

Start of informative comment:

This method assigns an object (working object) like TPM object, key object, encrypted data object to a certain policy object. Each of these working objects will utilize its assigned policy object to process an authorized TPM command.

By default each new initialized working object is assigned to the default policy, which is automatically created when a context object is created. When a working object is assigned to a policy object, the reference to the working object is added to the list of assigned objects stored in that policy object and the reference to the policy object is stored in the working object by internal object functions.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Policy_AssignToObject
   (
   TSS_HPOLICY      hPolicy,// in
   TSS_HOBJECT      hObject // in
   );
```

Parameters

hPolicy

Handle of the policy object

hObject

Handle of the object to be assigned

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_INTERNAL_ERROR

Remarks

Each TPM object, key object or data object can be assigned to one policy object of type TSS_POLICY_USAGE.

A key object or data object additionally can be assigned to one policy object of type TSS_POLICY_MIGRATION

The required policy object type must be set in the policy object by Tspi_SetAttribData( ) or on creation of the policy object.

4.3.4.4 Tspi_TPM_Class Definition

Start of informative comment:

Owner:

The Owner of the TPM has the right to perform special operations. The process of taking ownership is the procedure whereby the Owner inserts a shared secret into the TPM. For all future operations, knowledge of the shared secret is proof of Ownership. When the Owner wishes to perform one of the special operations then the Owner must use the authorization protocol to prove knowledge of the shared secret.

Identity:

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

To create a TSS identity that is recognized by the Privacy CA, the TPM must contain a private endorsement key. For this purpose there must be available: the endorsement credential, the platform credential, the conformance credential, and the public key of the Privacy CA. The process of obtaining evidence of TPM identity has three main phases: Create a new identity, contact a Privacy CA and activate this identity.

Credentials:

A Subsystem or its associated platform may store certificates. These are not essential for the Subsystem itself, but are useful because of their operational advantages when replying to integrity challenges. An Integrity Challenger requires the data in these certificates in order to judge the validity of integrity metrics measured in the platform. So, receiving the data from the TSS relieves the Challenger of the need to fetch it independently.

TPM Callbacks

For applications that wish to be compatible with version 1.1. of the TSS specification, Tspi_SetAttribUnit32 should be used to set all callbacks. For all others, Tspi_SetAttribData should be used with a TSS_CALLBACK structure.

End of informative comment.

4.3.4.4.1 Tspi_SetAttribUint32

Start of informative comment:

This method sets a 32-bit attribute of the TPM object.

End of informative comment.

Definition:

See section 4.3.2 for definition.

Parameters

See section 4.3.2 for description.

Defined Attributes

Flag	SubFlag	Attribute	Description
TSS_TSPATTRIB_TPM_CALLBACK_COLLATEIDENTITY	Application provided data.	Address of callback or NULL (disable)	Provided for TSS v1.1 compatibility.
TSS_TSPATTRIB_TPM_CALLBACK_ACTIVATEIDENTITY	Application provided data.	Address of callback or NULL (disable)	Provided for TSS v1.1 compatibility.

Return Values

See section 4.3.2 for description.

Remarks

4.3.4.4.2 Tspi_GetAttribUint32

Start of informative comment:

This method gets a 32-bit attribute of the TPM object

End of informative comment.

Definition:

See section 4.3.3.1.2 for definition.

Parameters

See section 4.3.3.1.2 for description.

Defined Attributes

Flag	SubFlag	Attribute	Description
TSS_TSPATTRIB_TPM_CALLBACK_COLLATEIDENTITY	Application provided data.	Address of callback or NULL (disable)	Provided for TSS v1.1 compatibility.
TSS_TSPATTRIB_TPM_CALLBACK_ACTIVATEIDENTITY	Application provided data.	Address of callback or NULL (disable)	Provided for TSS v1.1 compatibility.

Return Values

See section 4.3.3.1.2 for description.

Remarks

4.3.4.4.3 Tspi_SetAttribData

Start of informative comment:

This method sets a non 32-bit attribute of the TPM object. The structure and size of the attribute data depends on the attribute.

End of informative comment.

Definition:

See section 4.3.3.1.3 for definition.

Parameters

See section 4.3.3.1.3 for description.

Flag	SubFlag	Attribute	Description
TSS_TSPATTRIB _TPM_CALLBACK _COLLATEIDENT ITY	Ignored	Address of a TSS_CALLBACK structure or NULL (disable)	The callback pointer in the TSS_CALLBACK structure shall contain the address of the callback.
TSS_TSPATTRIB _TPM_CALLBACK _ACTIVATEIDEN TITY	Ignored	Address of a TSS_CALLBACK structure or NULL (disable)	The callback pointer in the TSS_CALLBACK structure shall contain the address of the callback.
TSS_TSPATTRIB _TPM_CREDENTI AL	TSS_TPMATTRIB _EKCERT	The Endorsement credential data blob	The endorsement credential information set
	TSS_TPMATTRIB _TPM_CC	The TPM conformance credential data blob	The TPM conformance credential data set
	TSS_TPMATTRIB _PLATFORMCERT	Platform credential data blob	The platform credential data set
	TSS_TPMATTRIB _PLATFORM_CC	Platform conformance credential data blob	The platform credential dat set

Return Values

See section 4.3.3.1.3 for description.

Remarks

- If the credential data are stored in the NV-Area of the TPM and the connected index needs authorization (other than owner auth), then the application must generate a NV-Object(s) to access the data and then transfer it into the TPM-Object via the attribute functions of these objects.
- Externally provided credential data take priority over the TSS internal access mechanisms so the application can use this way as kind of “overrule” instrument for these data elements.

4.3.4.4.4 Tspi_GetAttribData

Start of informative comment:

This method gets a non 32-bit attribute of the TPM object. The structure and size of the attribute data depends on the attribute.

End of informative comment.

Definition:

See section 4.3.3.1.4 for definition.

Parameters

See section 4.3.3.1.4 for description.

Flag	SubFlag	Attribute	Description
TSS_TSPATTRIB_TPM_CALLBACK_COLLATEIDENTITY	Ignored	Address of a TSS_CALLBACK structure or NULL (disable)	The callback pointer in the TSS_CALLBACK structure shall contain the address of the callback.
TSS_TSPATTRIB_TPM_CALLBACK_ACTIVATEIDENTITY	Ignored	Address of a TSS_CALLBACK structure or NULL (disable)	The callback pointer in the TSS_CALLBACK structure shall contain the address of the callback.

Return Values

See section 4.3.3.1.4 for description.

Remarks

4.3.4.5 Identity Management

4.3.4.5.1 Tspi_TPM_CreateEndorsementKey

Start of informative comment:

This method creates the endorsement key.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_CreateEndorsementKey
   (
   TSS_HTPM         hTPM,            // in
   TSS_HKEY         hKey,            // in
   TSS_VALIDATION*  pValidationData  // in, out
   );
```

Parameters

hTPM

Handle of the TPM object

hKey

Handle of the key object specifying the attributes of the endorsement key to create.

pValidationData

Validation data structure
[IN] Provide externalData information required to compute the checksum. - there is no signature, just a checksum.
[OUT] On successful completion of the command, the structure provides a buffer containing the validation checksum and a buffer containing the data the validation checksum was computed from.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The key information required for creating the endorsement key must be set in the key object by Tspi_SetAttribData( ) before this method is called.

On return the public endorsement key (PUBEK) can be retrieved by Tspi_GetAttribData( ) from the key object.

4.3.4.5.2 Tspi_TPM_GetPubEndorsementKey

Start of informative comment:

This function gets the public endorsement key.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_GetPubEndorsementKey
   (
   TSS_HTPM         hTPM,                // in
   TSS_BOOL         fOwnerAuthorized,    // in
   TSS_VALIDATION*  pValidationData,     // in, out
   TSS_HKEY*        phEndorsementPubKey// out
   );
```

Parameters

hTPM

Handle of the TPM object

fOwnerAuthorized

If TRUE, the TPM owner secret must be provided to get the endorsement public key.
If FALSE, no TPM owner secret must be provided to get the endorsement public key.

pValidationData

Validation data structure
[IN] Provide externalData information required to compute the signature.
[OUT] On successful completion of the command, the structure provides a buffer containing the validation data and a buffer containing the data the validation data was computed from.

phEndorsementPubKey

Receives a handle to a key object representing the endorsement public key.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The public key information of the endorsement key can be retrieved by calling *Tspi_GetAttribData*.

4.3.4.5.3 Tspi_TPM_CollateIdentityRequest

Start of informative comment:

This method creates an identity key, binds it to the label and returns a certificate request package. The privacy CA requires this certificate request to attest the identity key.

Only the Owner of the TPM has the privilege of creating a TPM identity key.

The symmetric session key is required to provide confidentiality of the "TCPA_IDENTITY_REQ" data structure, which should be sent to the Privacy CA chosen by the owner.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_CollateIdentityRequest
(
TSS_HTPM         hTPM,                         // in
TSS_HKEY         hKeySRK,                      // in
TSS_HKEY         hCAPubKey,                    // in
UINT32           ulIdentityLabelLength,        // in
BYTE*            rgbIdentityLabelData,         // in
TSS_HKEY         hIdentityKey,                 // in
TSS_ALGORITHM_ID algID ,                       // in
UINT32*          pulTCPAIdentityReqLength,     // out
BYTE**           prgbTCPAIdentityReq           // out
);
```

Parameters

hTPM

Handle of the TPM object

hKeySRK

Handle to the key object representing the SRK (Storage Root Key).

hCAPubKey

Handle to the key object representing the public key of the CA which signs the certificate of the created identity key.

ulIdentityLabelLength

Supplies the length (in bytes) of the *rgbIdentityLabelData* parameter.

rgbIdentityLabelData

Pointer to a memory block containing the identity label, which should be a TSS_UNICODE string.

hIdentityKey

Handle of the identity key object

algID

The type of symmetric algorithm to use as required by the Enhanced CA as defined in Algorithm ID Definitions 2.3.2.19.

pulTCPAIdentityReqLength

Receives the length (in bytes) of the *prgbTCPAIdentityReq* parameter.

prgbTCPAIdentityReq

Pointer to the memory block containing the certificate request structure TCPA_IDENTITY_REQ.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The Tspi_TPM_CollateIdentityRequest method allocates a memory block for the requested certificate request data. This memory must be released utilizing the Tspi_Context_FreeMemory method.

The key information required for creating the identity key must be set in the key object by Tspi_SetAttribData( ) before this method is called.

This method assembles all data necessary to request attestation for a Trusted Platform Module identity and exports this data by the output parameter prgbTCPAIdentityReq.

The structure "proof" (of type TCPA_IDENTITY_PROOF) contains fields that a privacy-CA requires in order to decide whether to attest to the TPM identity described by "proof".

Executing this method the TSS Service Provider performs two encryptions. The first is to symmetrically encrypt the information and the second is to encrypt the symmetric encryption key with an asymmetric algorithm. The symmetric key is a random nonce and the asymmetric key is the public key of the CA that will provide the identity credential.

For reasons of interoperability, publicCAKey SHOULD indicate TSS_ALG_RSA (RSA) with a key length of 2048 bits. CASymKey SHOULD be TSS_ALG_3DES (3DES in CBC mode and PKCS padding as defined in RFC 1423).

Action hints with credential handling:

- Validation of the input parameter (e.g. Public CA-Key/Label, Identity-Key-Template,…)
- Owner-Auth_Session establishment for identity key generation
- Calculate and assemble the TcsMakeIdenty2 parameter (e.g. CALabel, IdentityKey, AuthData(HMAC);
- Call MakeIdentity2 and verify the repsonse e.g. HMAC

- Access credential data:
  - If data set at the TPM-Object (i.e. SetAttribData) use this one
  - If data not available in the TPM-Object:
    - Check if the needed credential record is available via owner auth in the NV-Area. If so, then use this one
    - In conclusion, try to access the credential data of interest via the Tcsi_GetCredential operation (access mode derived from context version);
- Build the Identity-Generation related data structures (e.g. TCPA_IDENTITY_PROOF, TCPA_IDENTITY_REQ) and perform the necessary calculation steps.
- Return the identity request data record.

4.3.4.5.4 Tspi_TPM_ActivateIdentity

Start of informative comment:

This method proves the credential to be the credential of the identity key and returns the decrypted credential created by the privacy CA for that identity.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_ActivateIdentity
   (
   TSS_HTPM   hTPM,                         // in
   TSS_HKEY   hIdentKey,                    // in
   UINT32     ulAsymCAContentsBlobLength,   // in
   BYTE*      rgbAsymCAContentsBlob,        // in
   UINT32     ulSymCAAttestationBlobLength, // in
   BYTE*      rgbSymCAAttestationBlob,      // in
   UINT32*    pulCredentialLength,          // out
   BYTE**     prgbCredential                // out
   );
```

Parameters

hTPM

Handle of the TPM object

hIdentityKey

Handle of the identity key object

ulAsymCAContentsBlobLength

Supplies the length (in bytes) of the *rgbAsymCAContentsBlob* parameter.

rgbAsymCAContentsBlob

Pointer to a memory block containing the encrypted ASYM_CA_CONTENTS data structure got from the privacy CA.

ulSymCAAttestationBlobLength

Supplies the length (in bytes) of the *rgbSymCAAttestationBlob* parameter.

rgbSymCAAttestationBlob

Pointer to a memory block containing the encrypted SYM_CA_ATTESTATION data structure got from the privacy CA.

pulCredetialLength

Receives the length (in bytes) of the *prgbCredential* parameter.

prgbCredetial

Pointer to the memory block containing the decrypted credential data structure TCPA_IDENTITY_CREDENTIAL.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The Tspi_TPM_ActivateIdenty method allocates a memory block for the returned credential data. This memory must be released utilizing the Tspi_Context_FreeMemory method.

The TSP MUST support 3DES and MAY support other mechanisms.

4.3.4.6 New EK commands:

4.3.4.6.1 Tspi_TPM_CreateRevocableEndorsementKey

Start of informative comment:

This method creates the TPM revocable endorsement key. The input/output parameters specify whether the auth value to reset the EK will be generated by the TPM or set from the application.
It is the responsibility of the caller to properly protect and disseminate the revoke authorization data blob.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_CreateRevocableEndorsementKey
(
  TSS_HTPM         hTPM,                  // in
  TSS_HKEY         hKey,                  // in
  TSS_VALIDATION*  pValidationData,       // in, out
  UINT32*          pulEkResetDataLength,  // in, out
  BYTE**           prgbEkResetData        // in, out
);
```

Parameters

hTPM

Handle of the TPM object

hKey

Handle of the key object specifying the attributes of the endorsement key to create.

pValidationData

Validation data structure:
[IN] Provide externalData information required to compute the signature.
[OUT] On successful completion of the command, the structure provides a buffer containing the validation data and a buffer containing the data the validation data was computed from.

pulEkResetDataLength

If this data value referenced by the pointer is set to zero then the TSP uses the TPM to generate the authorization data to revoke the TPM endorsement key and after the operation this parameter carries the length info of the reset data blob. In the other case (not zero) this parameter specifies the size of the external created reset data blob (i.e. the pointer must always be valid.

rgbEkResetData

This parameter transports the EK revoke data for the TPM EK.
[IN] Externally created reset data blob

(i.e. Valid if *pulEKResetDataLength != 0 and *rgbEkResetData must be != NULL).
[OUT] Reset data blob generated by the TPM device

(i.e. Valid if *pulEKResetDataLength == 0 and *rgbEkResetData must be NULL).

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The key information required for creating the endorsement key must be set in the key object by Tspi_SetAttribData( ) before this method is called. The selected key parameters must be valid for the creation of a legal endorsement key (e.g. RSA key, minimum length 2048).

The user/calling application of this function should be aware that during the execution of this operation there is limited authorization environment available (e.g. no Owner).

On return the public endorsement key (PUBEK) can be retrieved by Tspi_GetAttribData( ) from the key object.

4.3.4.6.2 Tspi_TPM_RevokeEndorsementKey

Start of informative comment:

This method clears the TPM revocable endorsement key and executes the Clear Owner command. In effect, this erases all counters (except the base one), erases the Ek, the SRK, the owner auth and any NVRAM locked to the owner auth. It does not touch the delegation tables or other NVRAM.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_RevokeEndorsementKey
(
    TSS_HTPM   hTPM,                  // in
    UINT32     ulEkResetDataLength,  // in
    BYTE*      rgbEkResetData         // in
);
```

Parameters

hTPM

Handle of the TPM object

ulEkResetDataLength

Length info of the EK revoke data blob.

rgbEkResetData

This parameter transports the EK revoke data for the TPM EK.

Return Values

TSS_SUCCESS
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The authorization information required to revoke the TPM EK is generated during the creation process of a revocable endorsement key.

4.3.4.7 Setup and Takedown Commands

4.3.4.7.1 Tspi_TPM_TakeOwnership

Start of informative comment:

This method takes ownership of the TPM. The process of taking ownership is the procedure whereby the owner inserts a shared secret into the TPM. The Owner of the TPM has the right to perform special operations. When this command executes, under the covers it executes a Tspi_Context_RegisterKey for the SRK as a persistent key, with all normal information registered ***except*** the SRK public key, which must be requested directly.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_TakeOwnership
(
TSS_HTPM   hTPM,                // in
TSS_HKEY   hKeySRK,             // in
TSS_HKEY   hEndorsementPubKey // in
);
```

Parameters

hTPM

Handle of the TPM object

hKeySRK

Handle to the key object representing the SRK (Storage Root Key).

hEndorsementPubKey

Handle to the key object representing the endorsement public key required for encrypting the secret of SRK and the TPM owner secret. If NULL, the TSP internally queries the TPM for that endorsement public key.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_INTERNAL_ERROR

Remarks

4.3.4.7.2 Tspi_TPM_ClearOwner

Start of informative comment:

This method clears the TPM ownership.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_ClearOwner
   (
   TSS_HTPM   hTPM,         // in
   TSS_BOOL   fForcedClear  // in
   );
```

Parameters

hTPM

Handle of the TPM object.

fForcedClear

If FALSE, a clear ownership with proof of the TPM owner secret is done. If TRUE, a forced clear ownership with proof of physical access is done.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_INTERNAL_ERROR

Remarks

Please see the manual of your TCG system, to set the physical access state.

4.3.4.7.3 Tspi_TPM_CreateMaintenanceArchive

Start of informative comment:

This method creates the TPM Manufacturer specific maintenance archive data. Additionally it sets a flag in the TPM

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_CreateMaintenanceArchive
(
    TSS_HTPM    hTPM,                  // in
    TSS_BOOL    fGenerateRndNumber,    // in
    UINT32*     pulRndNumberLength,    // out
    BYTE**      prgbRndNumber,         // out
    UINT32*     pulArchiveDataLength,  // out
    BYTE**      prgbArchiveData        // out
);
```

Parameters

hTPM

Handle of the TPM object

fGenerateRndNumber

TRUE: a random number is generated by the TPM and returned. FALSE: a random number is calculated based on the owner secret.

pulRndNumberLength

Receives the length (in bytes) of the *prgbRndNumber* parameter. 0, if *fGenerateRndNumber* is FALSE.

prgbRndNumber

Receives pointer to the random number data Attributes). NULL, if *fGenerateRndNumber* is FALSE.

pulArchiveDataLength

Receives the length (in bytes) of the *prgbArchiveData* parameter.

prgbArchiveData

Receives pointer to the archive data.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_NOTIMPL
TSS_E_INTERNAL_ERROR

Remarks

The *Tspi_TPM_CreatMaintenaceArchive* method allocates memory blocks for the requested output data. This memory must be released utilizing the *Tspi_Context_FreeMemory* method.

4.3.4.7.4 Tspi_TPM_KillMaintenanceFeature

Start of informative comment:

This method disables the functionality of creating a maintenance archive

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_KillMaintenanceFeature
(
  TSS_HTPM   hTPM  // in
);
```

Parameters

hTPM

Handle of the TPM object

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_NOTIMPL
TSS_E_INTERNAL_ERROR

Remarks

After disabling the functionality of creating a maintenance archive, this functionality can only be enabled again by releasing the TPM ownership.

4.3.4.7.5 Tspi_TPM_LoadMaintenancePubKey

Start of informative comment:

This method loads the public maintenance key into the TPM.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_LoadMaintenancePubKey
(
    TSS_HTPM          hTPM,             // in
    TSS_HKEY          hMaintenanceKey,  // in
    TSS_VALIDATION*   pValidationData   // in, out
);
```

Parameters

hTPM

Handle of the TPM object

hMaintenanceKey

Handle of the maintenance key object

pValidationData

Validation data structure
[IN] Provide externalData information required to compute the signature.
[OUT] On successful completion of the command, the structure provides a buffer containing the validation data and a buffer containing the data the validation data was computed from.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_NOTIMPL
TSS_E_INTERNAL_ERROR

Remarks

The maintenance public key can only be loaded once. Subsequent calls to *Tspi_TPM_LoadMaintenancePubKey* will fail.

The key information required for loading the maintenance public key must be set in the key object by *Tspi_SetAttribData()* before this method is called.

If pValidationData != NULL: The caller has to check the digest on his own.
If pValidationData = NULL: The TSS Service Provider checks the digest obtained from the TPM internally.

Calculation of hash value for the validation data:
SHA1 hash of the concatenated data of <maintenance public key>|| <externalData>
See the definition of the ordinal in TCG 1.1b Main Specification

4.3.4.7.6 Tspi_TPM_CheckMaintenancePubKey

Start of informative comment:

This method proofs the maintenance public key.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_CheckMaintenancePubKey
(
    TSS_HTPM          hTPM,            // in
    TSS_HKEY          hMaintenanceKey, // in
    TSS_VALIDATION*   pValidationData  // in, out
);
```

Parameters

hTPM

Handle of the TPM object

hMaintenanceKey

Handle of the maintenance key object

pValidationData

Validation data structure
[IN] Provide externalData information required to compute the signature.
[OUT] On successful completion of the command, the structure provides a buffer containing the validation data and a buffer containing the data the validation data was computed from.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_NOTIMPL
TSS_E_INTERNAL_ERROR

Remarks

If hMaintenanceKey = NULL, pValidationData must not be NULL. The caller has to proof the digest on his own.
If hMaintenanceKey != NULL, pValidationData must be NULL. The TSS Service Provider checks the digest obtained from the TPM internally. The key information required for proofing the maintenance public key must be set in the key object by *Tspi_SetAttribData()* before this method is called.

Calculation of hash value for the validation data:
SHA1 hash of the concatenated data of <maintenance public key>|| <externalData>
See the definition of the ordinal in TCG 1.1b Main Specification

4.3.4.7.7 Tspi_TPM_SetOperatorAuth

Start of informative comment:

This function sets the operator authorization value in the TPM.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_SetOperatorAuth
(
   TSS_HTPM                             hTPM,             // in
   TSS_HPOLICY                          hOperatorPolicy   // in
);
```

Parameters

hTPM

Handle of the tpm object

hOperatorPolicy

Handle to the policy object holding the new operator authorization value.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

On successful completion of this function, hOperatorPolicy is automatically assigned to the hTPM object as the operator authorization policy. Note that command sends the operator authorization value to the TPM in plaintext. If hOperatorPolicy has its secret mode set to `TSS_SECRET_MODE_CALLBACK` the TSP will not have access to the secret and so will fail with a `TSS_E_POLICY_NO_SECRET` error.

4.3.4.8 TPM Get and Set Status Commands

4.3.4.8.1 Tspi_TPM_SetStatus

Start of informative comment:

This method modifies the TPM status.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_SetStatus
(
TSS_HTPM   hTPM,          // in
TSS_FLAG   statusFlag,    // in
TSS_BOOL   fTpmState      // in
);
```

Parameters

hTPM

Handle of the TPM object.

statusFlag

Flag indicating the status to be set (see section 2.3.2.18 for description).

fTpmState

Status value to set.

Defined Attributes:

Flag	fTpmState	Description
TSS_TPMSTATUS_ DISABLEOWNERCLEAR	Ignored	Permanently disable the TPM owner authorized clearing of TPM ownership. The method Tspi_TPM_ClearOwner( ) with fForcedClear = FALSE is not available any longer. Owner authorization is required.
TSS_TPMSTATUS_ DISABLEFORCECLEAR	Ignored	Prevent temporarily (until next power on) a forced clear of the TPM ownership. The method Tspi_TPM_ClearOwner( ) with fForcedClear = TRUE is temporarily not available.
TSS_TPMSTATUS_ OWNERSETDISABLE	TSS_BOOL	fTpmState = TRUE: Disable the TPM. Owner authorization is required.
TSS_TPMSTATUS_	TSS_BOOL	fTpmState = TRUE: Disable the

PHYSICALDISABLE		TPM. Proof of physical access is required.
TSS_TPMSTATUS_ PHYSICALSETDEACTIVATED	TSS_BOOL	fTpmState = TRUE: Deactivate the TPM. Proof of physical access is required
TSS_TPMSTATUS_ SETTEMPDEACTIVATED	Ignored	Temporarily deactivate (until next power on) the TPM.
TSS_TPMSTATUS_ SETOWNERINSTALL	TSS_BOOL	fTpmState = TRUE: Set the ability to take TPM ownership utilizing the method Tspi_TPM_TakeOwnership( ). Proof of physical access is required.
TSS_TPMSTATUS_ DISABLEPUBEKREAD	1.1 TPMs: Ignored 1.2 TPMs: TSS_BOOL	Permanently disable (1.1 TPMs) Disable or enable (not 1.1 TPMs) the ability to read the endorsement public key without required TPM owner authorizition. fTpmState = TRUE: The method Tspi_TPM_GetPubEndorsementKey( ) with fOwnerAuthorized = FALSE is not available any longer. Owner authorization is required.
TSS_TPMSTATUS_DISABLEPUBSRKREAD Not valid for 1.1 TPMs	TSS_BOOL	Disable or enable the ability to read the public portion of the SRK. fTpmState = TRUE: The method Tspi_Key_GetPubKey( ) with hKey = the handle of the SRK is not available any longer. (Instead should use Tspi_TPM_GetSRKPubKey) Owner authorization is required.
TSS_TPMSTATUS_ALLOWMAINTENANCE	TSS_BOOL	Disables the ability for the TPM owner to create a maintenance archive utilizing the method Tspi_TPM_CreateMaintenanceArchive( ).
TSS_TPMSTATUS_DISABLED	TSS_BOOL	Sets the TPM to disabled or not (TSS 1.1 will not allow setting this flag)
TSS_TPMSTATUS_DEACTIVATED	TSS_BOOL	Sets the TPM to deactivated or not
TSS_TPMSTATUS_RESETLOCK	TRUE	Resets the effects of multiple authorization failures

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

Please see the manual of your TCG system, to set the physical access state.

For information about which functionality is still available if the TPM is disabled or deactivated, see section 2.3.2.18.

4.3.4.8.2 Tspi_TPM_GetStatus

Start of informative comment:

This method queries the TPM status.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_GetStatus
(
TSS_HTPM   hTPM,          // in
TSS_FLAG   statusFlag,    // in
TSS_BOOL*  pfTpmState     // out
);
```

Parameters

hTPM

Handle of the TPM object.

statusFlag

Flag indicating the status to retrieve (see section 2.3.2.18 for description).

pfTpmState

The value referenced by *pfTpmState* contains the queried status value.

Defined Attributes:

Flags below has descriptions that contain specific information for this function. For other flags see the general description of the flags in TPM Status Flags Definitions 2.3.2.18.

Flag	Description
TSS_TPMSTATUS_DISABLEOWNERCLEAR	The method Tspi_TPM_ClearOwner( ) with fForcedClear = FALSE is not available any longer.
TSS_TPMSTATUS_DISABLEFORCECLEAR	TPM Prevented temporarily (until next power on) to perform a forced clear of the TPM ownership. The method Tspi_TPM_ClearOwner( ) with fForcedClear = TRUE is temporarily not available.
TSS_TPMSTATUS_OWNERSETDISABLE	fTpmState = TRUE: Disabled TPM. Supported for backward compatibilty. Recommend using TSS_TPMSTATUS_DISABLED flag
TSS_TPMSTATUS_PHYSICALDISABLE	fTpmState = TRUE: TPM Disabled Supported for backward

	compatibilty. Recommend using TSS_TPMSTATUS_DISABLED flag
TSS_TPMSTATUS_PHYSICALSETDEACTIVATED	fTpmState = TRUE: Deactivated TPM. Supported for backward compatibilty. Recommend using TSS_TPMSTATUS_DEACTIVATED flag
TSS_TPMSTATUS_SETTEMPDEACTIVATED	Temporarily deactivated (until next power on) the TPM.
TSS_TPMSTATUS_SETOWNERINSTALL	fTpmState = TRUE: Get the ability to take TPM ownership utilizing the method Tspi_TPM_TakeOwnership( ).
TSS_TPMSTATUS_DISABLEPUBEKREAD	Query if TPM is Permanently disabled (1.1 TPMs) Disabled (not 1.1 TPMs) from the ability to read the endorsement public key without required TPM owner authorizition. The method Tspi_TPM_GetPubEndorsementKey ( ) with fOwnerAuthorized = FALSE is not available any longer.
TSS_TPMSTATUS_DISABLED	Query whether TPM is disabled or enabled. (TSS 1.1b will not allow setting this flag)
TSS_TPMSTATUS_DEACTIVATED	Query whether the TPM is deactivated or activated.
TSS_TPMSTATUS_ALLOWMAINTENANCE	Query whether the TPM owner may create a maintenance archive utilizing the method Tspi_TPM_CreateMaintenanceArchive( ) or not.
TSS_TPMSTATUS_MAINTENANCEUSED	Query whether the TPM owner has already created a maintenance archive for the current SRK
TSS_TPMSTATUS_PHYSPRES_LIFTIMELOCK	Query whether both physicalPresenceHWEnable and physicalPresenceCMDEnable flags are locked and cannot be changed for the life of the TPM.
TSS_TPMSTATUS_PHYSPRES_HWENABLE	Query whether the TPM hardware signal <physical presence> is enabled to provide proof of physical presence.
TSS_TPMSTATUS_PHYSPRES_CMDENABLE	Query whether the TPM command TSC_PhysicalPresence is enabled to provide proof of physical presence.
TSS_TPMSTATUS_CEKP_USED	Query whether the endorsement

	key pair was created using the method Tspi_TPM_CreateEndorsementKey( ) or it was created using a manufacturers process.
TSS_TPMSTATUS_PHYSPRESENCE	Query whether physical presence has been asserted.
TSS_TPMSTATUS_PHYSPRES_LOCK	Query whether changes to the physicalPresence flag are permitted.
TSS_TPMSTATUS_POSTINITIALISE	Indicates that the TPM is between the TPM_Init state and the execution of the TPM_Startup command.
TSS_TPMSTATUS_TPMPOST	Queries if the TPM is set to force a full selftest before allowing commands to be performed.
TSS_TPMSTATUS_TPMPOSTLOCK	Queries if lock has been set the state of the TSS_TPMSTATUS_TPMPOST flag for the lifetime of the TPM
TSS_TPMSTATUS_DISABLEPUBSRKREAD Not valid for 1.1 TPMs	Indicates the ability to read the public portion of the SRK using Tspi_Key_GetPubKey( ) with hKey = the handle of the SRK.
TSS_TPMSTATUS_OPERATOR_INSTALLED Not valid for 1.1 TPMs TSS_TPMSTATUS_FIPS TSS_TPMSTATUS_ENABLE_REVOKEEK TSS_TPMSTATUS_NV_LOCK TSS_TPMSTATUS_TPM_ESTABLISHED	Indicates whether or not the operator authorization has been set. Indicates whether or not the TPM operates in FIPS mode Indicates whether or not the ability to revoke EK is enabled Indicates whether or not the authorization is active to access NV area. TRUE – authorization active FALSE – no authorization is active, (except for maxNVWrites) Indicates whether or not the dynamic root of trust of measurement has been executed.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

Because reading any TPM flags requires Owner authorization, this command requires Owner authorization for any flags if using a 1.1 TPM because 1.1 TPMs require Owner authorization to read any TPM flags.

For information about which functionality is still available if the TPM is disabled or deactivated, see section 2.3.2.18.

4.3.4.9 Get TPM Capabilities

4.3.4.9.1 Tspi_TPM_GetCapability

Start of informative comment:

This method provides the capabilities of the TPM.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_GetCapability
   (
   TSS_HTPM   hTPM,               // in
   TSS_FLAG   capArea,            // in
   UINT32     ulSubCapLength,     // in
   BYTE*      rgbSubCap,          // in
   UINT32*    pulRespDataLength,  // out
   BYTE**     prgbRespData        // out
   );
```

Parameters

hTPM

Handle of the TPM object

capArea

Flag indicating the attribute to query (see table Defined Attributes).

ulSubCapLength

The length (in bytes) of the *rgbSubCap* parameter.

rgbSubCap

Data indicating the attribute to query (see table Defined Attributes).

pulRespDataLength

Receives the length (in bytes) of the *prgbRespData* parameter.

prgbRespData

Receives pointer to the actual data of the specified attribute (see table Defined Attributes).

Defined Attributes

Capability Area	SubCap Area	Response
TSS_TPMCAP_ORD	Value of the ordinal	Boolean value. TRUE indicates that the TPM supports the ordinal. FALSE indicates that the TPM does not

		support the ordinal.
TSS_TPMCAP_FLAG	Ignored	Bit map of persistent and volatile flags.
TSS_TPMCAP_ALG	TSS_ALG_XX: A value of TSS Algorithm ID as defined in 2.3.2.19	Boolean value. TRUE indicates that the TPM supports the algorithm, FALSE indicates that the TPM does not support the algorithm.
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_PCR	UINT32 value. Returns the number of PCR registers supported by the TPM
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_PCRMAP	Returns the array of TPM_PCR_ATTRIBUTES
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_DIR	UINT32 value. Returns the number of DIR registers supported by the TPM.
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_MANUFACTURER	UINT32 value. Returns the Identifier of the TPM manufacturer.
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_SLOTS or TSS_TPMCAP_PROP_KEYS	UINT32 value. Returns the maximum number of 2048 bit RSA keys that the TPM is capable of loading. This MAY vary with time and circumstances.
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_MIN_COUNTER	UINT32. The minimum amount of time in 10ths of a second that must pass between invoations of incrementing the monotonic counter.
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_FAMILYROWS	UINT32. Maximum number of rows in the family table
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_DELEGATEROWS	UINT32. Maximum number of rows in the delegate table.

TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_OWNER	TSS_BOOL. Returning a value of TRUE indicates that the TPM has successfully installed an owner.
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_MAXKEYS	UINT32. Returns the maximum number of 2048-bit RSA keys that the TPM can support. The number does not include the EK or SRK.
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_AUTHSESSIONS	UINT32. Returns the number of available authorization sessions. This MAY vary with time and circumstances.
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_MAXAUTHSESSIONS	UINT32. Returns the maximum number of loaded authorization sessions the TPM supports.
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_TRANSESSIONS	UINT32. Returns the number of available transport sessions. This MAY vary with time and circumstances.
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_MAXTRANSESSIONS	UINT32. Returns the maximum number of loaded transport sessions the TPM supports.
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_SESSIONS	UNIT32. Returns the number of available sessions from the pool. Pool sessions include authorization and transport sessions. This MAY vary with time and circumstances.
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_MAXSESSIONS	UINT32. Returns the maximum number of sessions (authorization and transport) the TPM supports.

TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_CONTEXTS	UINT32. Returns the number of available saved session slots. This MAY vary with time and circumstances.
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_MAXCONTEXTS	UINT32. Returns the maximum number of saved session slots.
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_DAASESSIONS	UINT32. Returns the number of available DAA sessions. This MAY vary with time and circumstances.
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_MAXDAASESSIONS	UINT32. Returns the maximum number of DAA sessions (join or sign) that the TPM supports.
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_DAA_INTERRUPT	TSS_BOOL. Returning a value of TRUE indicates that the TPM will accept ANY command while executing a DAA Join or Sign. Returning a value of FALSE indicates that the TPM will invalidate the DAA Join or Sign upon the receipt of any command other than the next join/sign in the session or a TPM_SaveContext.
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_COUNTERS	UINT32. Returns the number of available monotonic counters. This MAY vary with time and circumstances.
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_MAXCOUNTERS	UINT32. Returns the maximum number of monotonic counters under control of TPM_CreateCounter.
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_ACTIVECOUNTER	TPM_COUNT_ID. Returns the ID of the current counter. 0xff..ff is returned if no

		counter is active.
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_MINCOUNTERINCTIME	UINT32. Returns the minimum amount of time in 10ths of a second that must pass between invocations of incrementing the monotonic counter.
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_TISTIMEOUTS	Returns a 4-element array of UINT32 values each denoting the timeout value in microseconds for the following in this order: TIMEOUT_A, TIMEOUT_B, TIMEOUT_C, TIMEOUT_D Where these timeouts are to be used is determined by the platform-specific TPM Interface Specification.
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_STARTUPEFFECTS	Returns the TPM_STARTUP_EFFECTS structure.
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_MAXCONTEXTCOUNTDIST	UINT32. Returns the maximum distance between context count values. This MUST be at least 2^16-1.
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_CMKRESTRICTION	TSS_BOOL Returns TPM_Permanent_Data -> restrictDelegate
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_DURATION	Returns a 3-element array of UINT32 values each denoting the value in microseconds of the duration of the three classes of commands in the following order: SMALL_DURATION, MEDIUM_DURATION, LONG_DURATION
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_MAXNVAVAILABLE	UINT32. Returns the maximum number of NV space that can be allocated. This

		MAY vary with time and circumstances.
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_INPUTBUFFERSIZE	UINT32. Returns the size of the TPM input buffer in bytes.
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_MAXNVWRITE	UINT32 The count of NV writes that have occurred when there is not TPM owner
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_REVISION	BYTE: This is the TPM major and minor revision indicator in the standard structure.
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_LOCALITIES_AVAIL	Type? The number of localities available in the TPM
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_INPUTBUFFERSIZE	UINT32. Returns the size of the TPM input buffer in bytes
TSS_TPMCAP_VERSION	Ignored	Returns the TSS_VERSION structure that identifies the version of the TPM.
TSS_TPMCAP_VERSION_VAL	Ignored	Queries the TPM_VERSION_VAL for a 1.2 or later TPM
TSS_TPMCAP_NV_LIST	Ignored	Retrieves the list of indices for defined NV storage areas.
TSS_TPMCAP_NV_INDEX	UINT32 (TPM_NV_INDEX)	Retrieves a TPM_NV_DATA_PUBLIC structure that indicates the values for the specified NV area.
TSS_TPMCAP_MFR	Ignored	Retrieves manufacturer specific TPM and TPM state information.
TSS_TPMCAP_SYM_MODE	One of the TPM_SYM_MODE_* type of encryption	Bool Queries whether or not the TPM supports a particular type of

TCG Software Stack (TSS) Specification

		a symmetric encryption
TSS_TPMCAP_HANDLE	One of the TPM_RT_* values specifying whether a list of keys, auth sessions, or transport session should be returned	Array of UINT32s (TPM handles) Returns list of handles of objects currently loaded in the TPM
TSS_TPMCAP_TRANS_ES	One of the TSS_ES_* values	Bool Queries whether the TPM supports a particular encryption scheme in the transport session.
TSS_TPMCAP_AUTH_ENCRYPT	One of the TSS_ALGORITHM_ID values	Bool Queries whether the TPM supports a particular encryption scheme in the OSAP encryption of the AuthData values.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The Tspi_TPM_GetCapability method allocates a memory block for the requested capability data. This memory must be released utilizing the Tspi_Context_FreeMemory method.

Some capability information can only be requested if owner authorization data is provided by the policy object bound to the TPM object.

Information about capArea and rgbSubCap is transmitted to the TPM without any interpretation by TCS. The TPM will return an appropriate error on wrong values.

4.3.4.9.2 Tspi_TPM_OwnerGetSRKPubKey

Start of informative comment:

This method returns the public key of the key object using owner authorization. This can only be used on 1.2 TPMs, and only to return the public key of the SRK.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_OwnerGetPubKey
(
   TSS_HTPM   hTPM,            // in
   UINT32*    pulPubKeyLength, // out
   BYTE**     prgbPubKey       // out
);
```

Parameters

hTPM

Handle of the TPM object.

pulPubKeyLength

Receives the length (in bytes) of the *prgbPubKey* parameter.

prgbPubKey

Receives a pointer to the memory block containing the public key blob retrieved for the key object referenced by *hKey*.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR
TSS_E_TPM_UNSUPPORTED

Remarks

When this method is used to return the SRK pubkey, it shall call *Tcsip_OwnerReadInternalPub.* This method can succeed only on version 1.2 and later TPMs. When inovked with 1.1 TPMs, this method should return TSS_E_TPM_UNSUPPORTED.

4.3.4.9.3 Tspi_TPM_GetCapabilitySigned

Start of informative comment:

NOTE: The TPM function TPM_GetCapabilitySigned that actually performs this functions was found to contain a vulnerability that makes its security questionable therefore its use unadvised. Since the final TPM specification contained this function and products have shipped with this function it is exposed at the TPM layer. However, the TSS Working Group has decided that TSS should not require the implementation of this function for any TSS. However, if a TSS provider should decide to include this function the TSS WG recommends the implementation contained here.

This method provides the capabilities of the TPM and returns a signature to proof the TPM as originator of the capability data.

C Definition:

```
TSS_RESULT Tspi_TPM_GetCapabilitySigned
   (
   TSS_HTPM         hTPM,                 // in
   TSS_HKEY         hKey,                 // in
   TSS_FLAG         capArea,              // in
   UINT32           ulSubCapLength,       // in
   BYTE*            rgbSubCap,            // in
   TSS_VALIDATION*  pValidationData,      // in, out
   UINT32*          pulRespDataLength,    // out
   BYTE**           prgbRespData          // out
   );
```

Parameters

hTPM

Handle of the TPM object

hKey

Handle of the signature key object

capArea

Flag indicating the attribute to query (see table Defined Attributes).

ulSubCapLength

The length (in bytes) of the *rgbSubCap* parameter.

rgbSubCap

Data indicating the attribute to query (see table Defined Attributes).

pValidationData

Validation data structure
[IN] Provide externalData information required to compute the signature.
[OUT] On successful completion of the command, the structure provides a buffer containing the validation data and a buffer containing the data the validation data was computed from.

pulRespDataLength

Receives the length (in bytes) of the *prgbRespData* parameter.

prgbRespData

Receives pointer to the actual data of the specified attribute (see table Defined Attributes).

Defined Attributes

Capability Area	SubCap Area	Response
TSS_TPMCAP_ALG	TSS_ALG_XX: A value of TSS Algorithm ID as defined in 2.3.2.11	Boolean value. TRUE indicates that the TPM supports the algorithm, FALSE indicates that the TPM does not support the algorithm.
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_PCR	UINT32 value. Returns the number of PCR registers supported by the TPM
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_DIR	UINT32 value. Returns the number of DIR registers supported by the TPM.
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_MANUFACTURER	UINT32 value. Returns the Identifier of the TPM manufacturer.
TSS_TPMCAP_PROPERTY	TSS_TPMCAP_PROP_SLOTS	UINT32 value. Returns the maximum number of 2048 bit RSA keys that the TPM is capable of loading. This MAY vary with time and circumstances.
TSS_TPMCAP_VERSION	Ignored	Returns the TSS_VERSION structure that identifies the version of the TPM.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The Tspi_TPM_GetCapabilitySigned method allocates a memory block for the requested capability data. This memory must be released utilizing the Tspi_Context_FreeMemory method.

Calculation of hash value for the validation data:
SHA1 hash of the concatenated data of <respData">|| <externalData>
See the definition of the ordinal in TCG 1.1b Main Specification

Information about capArea and rgbSubCap is transmitted to the TPM without any interpretation by TCS. The TPM will return an appropriate error on wrong values.

4.3.4.10 Test Commands

4.3.4.10.1 Tspi_TPM_SelfTestFull

Start of informative comment:

This method performs a self-test of each internal TPM function.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_SelfTestFull
    (
    TSS_HTPM   hTPM  // in
    );
```

Parameters

hTPM

Handle of the TPM object.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_INTERNAL_ERROR

Remarks

4.3.4.10.2 Tspi_TPM_CertifySelfTest

Start of informative comment:

This method performs a self-test of each internal TPM function and returns an authenticated value (signature) if the test has passed. If the key pointed to by hKey has a signature scheme that is not TPM_SS_RSASSAPKCS1v15_SHA1, the return value can be either TSS_E_BAD_PARAMETER or TSS_SUCCESS with a vendor specific signature.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_CertifySelfTest
   (
   TSS_HTPM         hTPM,             // in
   TSS_HKEY         hKey,             // in
   TSS_VALIDATION*  pValidationData   // in, out
   );
```

Parameters

hTPM

Handle of the TPM object.

hKey

Handle of the signature key object.

pValidationData

Validation data structure
[IN] Provide externalData information required to compute the signature.
[OUT] On successful completion of the command, the structure provides a buffer containing the validation data and a buffer containing the data the validation data was computed from.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

Calculation of hash value for the validation data:
SHA1 hash of the concatenated data of <the null terminated string of “Test Passed”>|| <externalData> || <ordinal>.
See the definition of the ordinal in TCG 1.1b Main Specification

4.3.4.10.3 Tspi_TPM_GetTestResult

Start of informative comment:

The method provides manufacturer specific information regarding the results of the self test

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_GetTestResult
    (
    TSS_HTPM    hTPM,                  // in
    UINT32*     pulTestResultLength,   // out
    BYTE**      prgbTestResult         // out
    );
```

Parameters

hTPM

Handle of the TPM object.

pulTestResultLength

Receives the length (in bytes) of the *prgbTestResult* parameter.

prgbTestResult

Pointer to the memory block containing the TPM manufacturer specific information.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The *Tspi_TPM_GetTestResult* method allocates a memory block for the requested data. This memory must be released utilizing the *Tspi_Context_FreeMemory* method.

Vendor specific
See the definition of the ordinal in TCG 1.2 Main Specification Part 2

4.3.4.11 Random Numbers

4.3.4.11.1 Tspi_TPM_GetRandom

Start of informative comment:

This method gets a random number from the TSS Service Provider utilizing the TPM.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_GetRandom
(
    TSS_HTPM    hTPM,                // in
    UINT32      ulRandomDataLength,  // in
    BYTE**      prgbRandomData       // out
);
```

Parameters

hTPM

Handle of the TPM object

ulRandomDataLength

Number of random bytes requested.

prgbRandomData

Receives a pointer to memory containing the random data.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The maximum length of the random number is 4096 Bytes.

The Tspi_TPM_GetRandom method allocates a memory block for the requested random data. This memory must be released utilizing the Tspi_Context_FreeMemory method

4.3.4.11.2 Tspi_TPM_StirRandom

Start of informative comment:

This method adds entropy to the TPM Random Number Generator

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_StirRandom
(
    TSS_HTPM    hTPM,                   // in
    UINT32      ulEntropyDataLength,    // in
    BYTE*       rgbEntropyData          // in
);
```

Parameters

hTPM

Handle of the TPM object

ulEntropyDataLength

The length (in bytes) of the *rgbEntropyData* parameter.

rgbEntropyData

Pointer to the entropy data.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

4.3.4.12 Old PCR Commands

4.3.4.12.1 Tspi_TPM_GetEvent

Start of informative comment:

This method provides a PCR event for a given PCR index and event number.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_GetEvent
(
    TSS_HTPM         hTPM,           // in
    UINT32           ulPcrIndex,     // in
    UINT32           ulEventNumber,  // in
    TSS_PCR_EVENT*   pPcrEvent       // out
);
```

Parameters

hTPM

Handle of the TPM object

ulPcrIndex

Index of the PCR to request.

ulEventNumber

Index of the event to request.

pPcrEvent

Receives the PCR event data.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

4.3.4.12.2 Tspi_TPM_GetEvents

Start of informative comment:

This method provides a specific number of PCR events for a given index.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_GetEvents
(
    TSS_HTPM          hTPM,             // in
    UINT32            ulPcrIndex,       // in
    UINT32            ulStartNumber,    // in
    UINT32*           pulEventNumber,   // in, out
    TSS_PCR_EVENT**   prgPcrEvents      // out
);
```

Parameters

hTPM

Handle of the TPM object

ulPcrIndex

Index of the PCR to request.

ulStartNumber

Index of the first event to request.

pulEventNumber

[IN] Number of elements to request.
[OUT] Receives number of returned event data structures in prg*PcrEvents* parameter

prgPcrEvents

Receives a pointer to an array of PCR event data.
If NULL, only the number of elements is returned in pulEventNumber parameter.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The Tspi_TPM_GetEvents method allocates a memory block for the requested event data. This memory must be released utilizing the Tspi_Context_FreeMemory method.

4.3.4.12.3 Tspi_TPM_GetEventLog

Start of informative comment:

This method provides the whole event log.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_GetEventLog
(
  TSS_HTPM        hTPM,            // in
  UINT32*         pulEventNumber,  // out
  TSS_PCR_EVENT** prgPcrEvents     // out
);
```

Parameters

hTPM

Handle of the TPM object

pulEventNumber

Receives number of returned event data structures in prg*PcrEvents* parameter

prgPcrEvents

Receives a pointer to an array of PCR event data.
If NULL, only the number of elements is returned in pulEventNumber parameter.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The Tspi_TPM_GetEventLog method allocates a memory block for the requested event data. This memory must be released utilizing the Tspi_Context_FreeMemory method.

4.3.4.12.4 Tspi_TPM_Quote

Start of informative comment:

This method quotes a TCG system.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_Quote
(
    TSS_HTPM         hTPM,             // in
    TSS_HKEY         hIdentKey,        // in
    TSS_HPCRS        hPcrComposite,    // in
    TSS_VALIDATION*  pValidationData   // in, out
    );
```

Parameters

hTPM

Handle of the TPM object

hIdentKey

Handle of the signature key object

hPcrComposite

Handle of the PCR composite object; the structure type must be TSS_PCRS_STRUCT_INFO.

[IN] Selected PCRs to be quoted

pValidationData

Validation data structure
[IN] Provide externalData information required to compute the signature.
[OUT] On successful completion of the command, the structure provides a buffer containing the validation data and a buffer containing the data the validation data was computed from.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The required information about which PCRs should be quoted must be set in the PcrComposite object before calling this method. On return each element of the collection has its pcrValue set.

If structure type other than TSS_PCRS_STRUCT_INFO is used in the hPcrComposite object, the error TSS_E_INVALID_OBJ_ACCESS will be returned.

The returned signature is computed over the TCPA_QUOTE_INFO structure as defined in the TCG 1.1b Main Specification.

The Tspi_TPM_Quote method allocates memory blocks for the requested validation data. This memory must be released utilizing the Tspi_Context_FreeMemory method.

4.3.4.12.5 Tspi_TPM_PcrExtend

Start of informative comment:

This method extends a PCR register and writes the PCR event log.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_PcrExtend
(
   TSS_HTPM        hTPM,              // in
   UINT32          ulPcrIndex,        // in
   UINT32          ulPcrDataLength,   // in
   BYTE*           pbPcrData,         // in
   TSS_PCR_EVENT*  pPcrEvent,         // in
   UINT32*         pulPcrValueLength, // out
   BYTE**          prgbPcrValue       // out
);
```

Parameters

hTPM

Handle of the TPM object

ulPcrIndex

Index of the PCR to extend.

ulPcrDataLength

Parameter contains the length of data to be extended.

pbPcrData

Data pointer to the data blob for the PCR extend operation. If a pPcrEvent is not NULL, this data will be used in a hash created according to the description of the rgbPcrValue parameter of the TSS_PCR_EVENT structure in section . If pPcrEvent is parameter of the TSS_PCR_EVENT structure in section . If pPcrEvent is NULL, this data will be extended into the TPM without being touched by the TSP.

pPcrEvent

Pointer to a TSS_PCR_EVENT structure containing the info for an event entry. If this pointer is NULL no event entry is created and the function only executes an extend operation. If non NULL, members of this struct will be set by the TSP according to the rules in section .

pulPcrValueLength

Receives the length (in bytes) of the *prgbPcrValue* parameter.

prgbPcrValue

Receives a pointer to the memory block containing the PCR data after the extend operation.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The Tspi_TPM_PcrExtend method allocates a memory block for the prgbPcrValue data.
This memory must be released utilizing the Tspi_Context_FreeMemory method.

4.3.4.12.6 Tspi_TPM_PcrRead

Start of informative comment:

This methods reads a PCR register.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_PcrRead
(
    TSS_HTPM    hTPM,                  // in
    UINT32      ulPcrIndex,            // in
    UINT32*     pulPcrValueLength,     // out
    BYTE**      prgbPcrValue           // out
);
```

Parameters

hTPM

Handle of the TPM object

ulPcrIndex

Index of the PCR to read.

pulPcrValueLength

Receives the length (in bytes) of the *prgbPcrValue* parameter.

prgbPcrValue

Receives a pointer to the memory block containing the PCR data.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The Tspi_TPM_PcrRead method allocates a memory block for the prgbPcrValue data. This memory must be released utilizing the Tspi_Context_FreeMemory method

4.3.4.13 Tspi_Data Class Definition for Seal and PCRs

4.3.4.13.1 Tspi_GetAttribUint32 / Tspi_SetAttribUint32

Flag	SubFlag	Attribute	Description
TSS_TSPATTRIB_ENCDATA_SEAL	TSS_TSPATTRIB_ENCDATASEAL_PROTECT_MODE	TSS_TSPATTRIB_ENCDATA_SEAL_PROTECT	Enable the usage of the Sealx functionality.
	TSS_TSPATTRIB_ENCDATA_SEAL_PROTECT_MODE	TSS_TSPATTRIB_ENCDATA_SEAL_NO_PROTECT	Disable the usage of the Sealx functionality.

4.3.4.14 Tspi_PcrComposite Class Definition

4.3.4.14.1 Tspi_SetAttribUint32

Start of informative comment:

This method sets a 32-bit attribute of the PcrComposite object.

End of informative comment.

Definition:

See section 4.3.2 for definition.

Parameters

See section 4.3.2 for description.

Defined Attributes

Flag	SubFlag	Attribute	Description
TSS_TSPATTRIB_PCRS_INFO	TSS_TSPATTRIB_PCRSINFO_PCRSTRUCT	TSS_PCRS_STRUCT XX	Value indicating the type of PCR structure used as defined in 2.3.2.31.

Return Values

See section 4.3.2 for description.

Remarks

Attributes of the TSS_TSPATTRIB_PCRS_INFO flag cannot be set after any of the following methods have been called to set values in the PcrComposite object: Tspi_PcrComposite_SelectPcrIndex(), Tspi_PcrComposite_SelectPcrIndexEx(), Tspi_PcrComposite_SetPcrValue(), Tspi_PcrComposite_SetPcrLocality(); attempting to do so will result in TSS_E_INVALID_ATTRIB_FLAG being returned.

4.3.4.14.2 Tspi_GetAttribUint32

Start of informative comment:

This method gets a 32-bit attribute of the PcrComposite object

End of informative comment.

Definition:

See section 4.3.3.1.2 for definition.

Parameters

See section 4.3.3.1.2 for description.

Defined Attributes

Flag	SubFlag	Attribute	Description
TSS_TSPATTRIB_PCRS_INFO	TSS_TSPATTRIB_PCRSINFO_PCRSTRUCT	TSS_PCRS_STRUCT_XX	Value indicating the type of PCR structure used as defined in 2.3.2.31.

Return Values

See section 4.3.3.1.2 for description.

Remarks

4.3.4.14.3 Tspi_PcrComposite_SelectPcrIndex

Start of informative comment:

This method selects a PCR index inside a PCR composite object using the 1.1 TCPA_PCR_INFO structure. The PCR composite object must be created with the function Tspi_Context_CreateObject(). An example for the usage is the selection of PCR registers before calling Tspi_TPM_Quote().

End of informative comment.

Definition:

```
TSS_RESULT Tspi_PcrComposite_SelectPcrIndex
(
    TSS_HPCRS  hPcrComposite,   // in
    UINT32     ulPcrIndex       // in
);
```

Parameters

hPcrComposite

Handle to the PCR composite object instance where the index should be selected.

ulPcrIndex

This parameter indicates the index of the PCR to select.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The PCR composite object must have been created by the method *Tspi_Context_CreateObject()* and be set to use the 1.1 TCPA_PCR_INFO structure. An example for the usage of this method is the selection of PCR registers prior to calling Tspi_TPM_Quote(). Multiple PCRs with different indexes can be selected by calling the method multiple times on the same PCR composite object.

If the PcrComposite object is using other than the 1.1 TCPA_PCR_INFO structure, this function shall return with TSS_E_INVALID_OBJ_ACCESS.

4.3.4.14.4 Tspi_PcrComposite_SetPcrValue

Start of informative comment:

This method sets the digest for a given PCR index inside the PCR composite object.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_PcrComposite_SetPcrValue
(
    TSS_HPCRS  hPcrComposite,      // in
    UINT32     ulPcrIndex,         // in
    UINT32     ulPcrValueLength,   // in
    BYTE*      rgbPcrValue         // in
);
```

Parameters

hPcrComposite

Handle to the PCR composite object instance where a PCR value should be set.

ulPcrIndex

This parameter indicates the index of the PCR to set.

ulPcrValueLength

The length (in bytes) of the *rgbPcrValue* parameter

rgbPcrValue

Pointer to memory containing the actual value which should be set for the PCR indicated by *ulPcrIndex*.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

An example for the usage is the preparation of a PCR composite object before calling *Tspi_Key_CreateKey()*. The PCR composite object must have been created by the method *Tspi_Context_CreateObject()*.Multiple PCRs with different indexes can be set by calling this method multiple times on the same PCR composite object.

This method may be used to set PCR values in a PCR composite object regardless of the type of PCR structure (TCPA_PCR_INFO, TCPA_PCR_INFO_LONG, TCPA_PCR_INFO_SHORT) the object is using.

When a TCPA_PCR_INFO_LONG structure is used, this method sets the PCR values for the DigestAtRelease.

4.3.4.14.5 Tspi_PcrComposite_GetPcrValue

Start of informative comment:

This method returns the digest value of a given PCR index inside a PCR composite object.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_PcrComposite_GetPcrValue
(
   TSS_HPCRS  hPcrComposite,       // in
   UINT32     ulPcrIndex,          // in
   UINT32*    pulPcrValueLength,   // out
   BYTE**     prgbPcrValue         // out
);
```

Parameters

hPcrComposite

Handle to the PCR composite object instance from which the PCR value should be returned.

ulPcrIndex

This parameter indicates the index of the PCR to read.

pulPcrValueLength

Receives the length (in bytes) of the *prgbPcrValue* parameter.

prgbPcrValue

After successful completion this parameter receives a pointer to the memory block containing the PCR value of the PCR indicated by *ulPcrIndex*.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

An example for the usage of this method is for retrieving the value of a PCR after a *Tspi_TPM_Quote()* call. Multiple PCRs vales for different indexes can be retrieved by calling this method multiple times on a PCR composite object.

This method may be used to get PCR values in a PCR composite object regardless of the type of PCR structure (TCPA_PCR_INFO, TCPA_PCR_INFO_LONG, TCPA_PCR_INFO_SHORT) the object is using.

When a TCPA_PCR_INFO_LONG structure is used, this method gets the PCR values for the DigestAtRelease.

The Tspi_PcrComposite_GetPcrValue method allocates a memory block for the prgbPcrValue data. This memory must be released utilizing the Tspi_Context_FreeMemory method.

4.3.4.15 New PCR commands:

4.3.4.15.1 *Tspi_TPM_PcrReset*

Start of informative comment:

This method resets a PCR register. Whether or not it succeeds may depend on the locality executing the command. PCRs can be defined in a platform specific specification to allow reset of certain PCRs only for certain localities. The one exception to this is PCR 16, which can always be reset in a 1.2 implementation. (This is to allow for software testing).

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_PcrReset
(
    TSS_HTPM    hTPM,             // in
    TSS_HPCRS   hPcrComposite     // in
);
```

Parameters

hTPM

Handle of the TPM object

hPcrComposite

Handle to the PCR composite object instance specifying the PCRs to be reset; this must be a TCPA_PCR_INFO structure.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_TPM_NOT_RESETABLE
TSS_E_WRONG_LOCALITY
TSS_E_INTERNAL_ERROR

Remarks

4.3.4.15.2 Tspi_Data_Seal

Start of informative comment:

This method is an "overloaded" function when executed on a 1.2 TSS stack.

This method encrypts a data blob in a manner that is only decryptable by Tspi_Data_Unseal on the same system. The data blob is encrypted using a public key operation with the nonmigratable key addressed by the given encryption key object.

Additionally the Tspi_Data_Seal operation allows software to explicitly state the future "trusted" configuration that the platform must be in for the encrypted data to be revealed and explicitly includes a list of relevant Platform Configuration Register (PCR) values when the Tspi_Data_Seal operation was performed.

When used as a 1.1 function, a single subset of PCR registers is selected in a PCR_Object, and those registers are both recorded and used for matching the state of the system at release.

Beginning with the 1.2 specification, the PCR object passed in to this function may contain a locality for release and also two sets of PCRs, once which identifies the subset of PCRs to be recorded at Seal, and the other which specifies a (potentially different) subset of PCRs and their values which must match in order to perform UnSeal.

If the Tspi_Data_Unseal operation succeeds, proof of the platform configuration that was in effect when the Tspi_Data_Seal operation was performed is returned to the caller, as well as the secret data. This proof may, or may not, be of interest. If the SEALed secret is used to authenticate the platform to a third party, a caller is normally unconcerned about the state of the platform when the secret was SEALed, and the proof may be of no interest. On the other hand, if the SEALed secret is used to authenticate a third party to the platform, a caller is normally concerned about the state of the platform when the secret was SEALed. Then the proof is of interest.

For example, if SEAL is used to store a secret key for a future configuration (probably to prove that the platform is a particular platform that is in a particular configuration), the only requirement is that that key can be used only when the platform is in that future configuration. Then there is no interest in the platform configuration when the secret key was SEALed. An example of this case is when SEAL is used to store a network authentication key.

On the other hand, suppose an OS contains an encrypted database of users allowed to log on to the platform. The OS uses a SEALED blob to store the encryption key for the user-database. However, the nature of SEAL is that *any* SW stack can SEAL a blob for any other software stack. Hence the OS can be attacked by a second OS replacing both the SEALED-blob encryption key, *and* the user database itself, allowing untrusted parties access to the services of the OS. To thwart such attacks, SEALED blobs include the *past* SW configuration. Hence, if the OS is concerned about such attacks, it may check to see whether the past configuration is one that is known to be trusted.

To seal data that is larger than the RSA public key modulus it is the responsibility of the caller to perform the blocking and subsequent combination of data.

It is important to note what the effect of the various commands that result in either a TCPA_PCR_INFO or TCPA_PCR_INFO_LONG structure will have when they are used with either 1.1 or 1.2 commands.

For example, in 1.1, if the commands

Tspi_PcrComposite_SelectPcrIndex and

Tspi_PcrComposite_SetPcrValue are used,

Then the TCPA_PCR_INFO structure which is eventually created, when used with a SEAL command, will select values that will both record at creation and check at release values sleected by the PcrIndex.

The same thing must happen in 1.2 if a TCPA_PCR_INFO_LONG is eventually generated. The newly defined function Tspi_PcrComposite_SelectPcrIndexX is a better function to use when using the 1.2 ability to select different values for recording at creation and checking at release.

How this is done internally in the TSS is left to the developer. If they want to always carry a 1.2 structure and then compress the structure to a 1.1 whenever it is possible before use, they can do that. If they want to carry two versions they can do that. If they want to do something completely different, they can do that, too.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Data_Seal
(
    TSS_HENCDATA  hEncData,        // in
    TSS_HKEY      hEncKey,         // in
    UINT32        ulDataLength,    // in
    BYTE*         rgbDataToSeal,   // in
    TSS_HPCRS     hPcrComposite    // in
);
```

Parameters

hEncData

Handle of the data object which contains the sealed data on successful completion of the command.

hEncKey

Handle to the key object addressing the nonmigratable key which is used to encrypt the data.

ulDataLength

The length (in bytes) of the *rgbDataToSeal* parameter.

rgbDataToSeal

Pointer to memory containing the data to be encrypted.

hPcrComposite

Handle of the PCR composite object, which can either contain TCPA_PCR_INFO or TCPA_PCR_INFO_LONG.

TCPA_PCR_INFO will contain only a single subset of PCRs and values that must be matched upon unseal. TCPA_PCR_INFO_LONG contains two subsets of PCRs, one identifying the PCRs to be recorded in the Sealed blob and one identifying the PCRs which must be matched at unseal – as well as the values they must match. In addition, the TCPA_PCR_INFO_LONG object may specify the locality that must be matched to unseal the data. Set to NULL, if the encrypted data should only be bound to the system and PCRs are not of interest.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_ENC_INVALID_LENGTH
TSS_E_ENC_NO_DATA
TSS_E_ENC_INVALID_TYPE
TSS_E_INTERNAL_ERROR

Remarks

The sealed data blob is stored in the data object addressed by hEncData and can be exported from that object by GetAttribData( ). The caller gets this exported encrypted data blob according the rules of TCG.

The information about the used PCRs must be set in the PCR composite object addressed by hPcrComposite before calling this method. hPcrComposite may use either TCPA_PCR_INFO (1.1 format) or TCPA_PCR_INFO_LONG (1.2 format). hPcrComposite MUST be set to NULL, if PCR values are not of interest.

Tspi_Data_Seal maximum data input size*

s = key size (bytes)

TSS_KEY_TYPE_STORAGE	s-(40-2)-65

* 65 bytes accounts for the size of the TPM_SEALED_DATA structure

4.3.4.15.3 Tspi_TPM_Quote2

Start of informative comment:

This method quotes a TCG system using TPM_Quote2, which provides the requester with a more complete view of the current platform configuration than TPM_Quote.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_Quote2
(
    TSS_HTPM          hTPM,             // in
    TSS_HKEY          hIdentKey,        // in
    TSS_BOOL          fAddVersion,      // in
    TSS_HPCRS*        hPcrComposite,    // in
    TSS_VALIDATION*   pValidationData,  // in, out
    UINT32*           versionInfoSize,  //out
    BYTE**            versionInfo      // out
```

Parameters

hTPM

Handle of the TPM object

hIdentKey

Handle of the signature key object

fAddVersion

If TRUE, the TPM version is added to the output
If FALSE, the TPM version is not added to the output

hPcrComposite

Handle of the PCR composite object; the structure type must be TSS_PCRS_STRUCT_INFO_SHORT.

[IN] Selected PCRs to be quoted

pValidationData

Validation data structure
[IN] Provide externalData information required to compute the signature.
[OUT] On successful completion of the command, the structure provides a buffer containing the validation data and a buffer containing the data the validation data was computed from.

This differs from Tspi_TPM_Quote in that the data contains a TCPA_PCR_INFO_SHORT as opposed to a TCPA_PCR_COMPOSITE, and the version information is returned in a TPM_QUOTE_INFO2 instead of a TPM_QUOTE_INFO

versionInfoSize

The size of the byte stream returned by versionInfo . If the fAddVersion is False this is zero.

versionInfo

The version information returned as a byte stream reflecting the data in TSS_CAP_VERSION_INFO if the fAddVersion is TRUE. Else it is NULL.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The required information about which PCRs should be quoted must be set in the PCR_Composite object before calling this method. On return the validation data contains the information about how the collection has its locality and PCR digest set.

If structure type other than TSS_PCRS_STRUCT_INFO_SHORT is used in the hPcrComposite object, the error TSS_E_INVALID_OBJ_ACCESS will be returned.

The returned signature is computed over the TCPA_QUOTE_INFO2 structure concatenated with the TPM-CAP_VERSION_INFO structure which is returned separately.

The Tspi_TPM_Quote2 method allocates memory blocks for the requested validation data. This memory must be released utilizing the Tspi_Context_FreeMemory method.

4.3.4.15.4 Tspi_PcrComposite_SetPcrLocality

Start of informative comment:

This method sets the LocalityAtRelease inside the PCR composite object using a 1.2 TCPA_PCR_INFO_LONG or TCPA_PCR_INFO_SHORT structure.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_PcrComposite_SetPcrLocality
   (
   TSS_HPCRS  hPcrComposite,   //in
   UINT32     LocalityValue    //in
   );
```

Parameters

hPcrComposite

Handle to the PCR composite object instance where a locality value should be set.

LocalityValue

LocalityAtRelease value to set

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_INVALID_OBJ_ACCESS
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

If the PcrComposite object is using the 1.1 TCPA_PCR_INFO structure, this function shall return with TSS_E_INVALID_OBJ_ACCESS.

4.3.4.15.5 Tspi_PcrComposite_GetPcrLocality

Start of informative comment:

This method gets the LocalityAtRelease from a PCR composite object using either a 1.2 TCPA_PCR_INFO_LONG structure, or a 1.2 TCPA_PCR_INFO_SHORT structure.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_PcrComposite_GetPcrLocality
(
   TSS_HPCRS  hPcrComposite,   //in
   UINT32*    pLocalityValue   //out
);
```

Parameters

hPcrComposite

Handle to the PCR composite object instance from which a locality value should be returned

pLocalityValue

Receives the locality value requested

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_INVALID_OBJ_ACCESS
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

If the PcrComposite object is using the 1.1 TCPA_PCR_INFO structure, this function shall return with TSS_E_INVALID_OBJ_ACCESS.

4.3.4.15.6 Tspi_PcrComposite_GetCompositeHash

Start of informative comment:

This method gets the digestAtRelease from a PCR composite object using either a 1.2 TCPA_PCR_INFO_LONG structure, or a 1.2 TCPA_PCR_INFO_SHORT structure.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_PcrComposite_GetCompositeHash
(
   TSS_HPCRS  hPcrComposite,//in
   UINT32*    pLen,         //out
   BYTE**     ppbHashData   //out
);
```

Parameters

hPcrComposite

Handle to the PCR composite object instance from which a composite hash digest should be returned

pLen

Receives the length (in bytes) of the *ppbHashData* parameter

ppbHashData

Digest at Creation or Digest at Release

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_INVALID_OBJ_ACCESS
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

If the PcrComposite object is using the 1.1 TCPA_PCR_INFO structure, this function shall return with TSS_E_INVALID_OBJ_ACCESS.

The Tspi_PcrComposite_GetCompositeHash method allocates a memory block for the ppbHashData data. This memory must be released utilizing the Tspi_Context_FreeMemory method.

4.3.4.15.7 Tspi_PcrComposite_SelectPcrIndexEx

Start of informative comment:

This method selects a PCR index inside a PCR composite object containing a TCPA_PCR_INFO_LONG or TCPA_PCR_INFO_SHORT structure. For the TCPA_PCR_INFO_LONG structure, the index may be selected for Creation or Release; for TCPA_PCR_INFO_SHORT, the index may be selected only for Release.

The PCR composite object must be created with the function Tspi_Context_CreateObject(). An example for the usage is the selection of PCR registers before calling Tspi_TPM_Quote2().

End of informative comment.

Definition:

```
TSS_RESULT Tspi_PcrComposite_SelectPcrIndexEx
(
    TSS_HPCRS  hPcrComposite,   //in
    UINT32     ulPcrIndex,      //in
    UINT32     Direction        //in
);
```

Parameters

hPcrComposite

Handle to the PCR composite object instance where a PCR value should be set.

ulPcrIndex

Parameter indicating the index of the PCR to select

Direction

Chooses whether the index selected is for a PCR at creation or a PCR at release

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

If the PcrComposite object is using the 1.1 TCPA_PCR_INFO structure, this function shall return with TSS_E_INVALID_OBJ_ACCESS.

If the PcrComposite object is using the 1.2 TCPA_PCR_INFO_SHORT structure and the selection is for Creation, this function shall return with TSS_E_INVALID_OBJ_ACCESS.

4.3.4.16 Keys: Create, load, manage

4.3.4.16.1 Tspi_ChangeAuth

Start of informative comment:

This method changes the authorization data (owner secret) of the TPM object and assigns the TPM object to the policy object.

End of informative comment.

Definition:

See section 4.3.3.1.5 for definition.

Parameters

See section 4.3.3.1.5 for definition.

Return Values

See section 4.3.3.1.5 for description.

Remarks

4.3.4.16.2 Tspi_GetPolicyObject

Start of informative comment:

This method returns a policy object currently assigned to the TPM object

End of informative comment.

Definition:

See section 4.3.3.1.7 for definition.

Parameters

See section4.3.3.1.7 for definition.

Return Values

See section 4.3.3.1.7 for description.

Remarks

4.3.4.17 Tspi_Key Class Definition

Start of informative comment

The particular creation command for CMK (CMKCreateKey) can be completely covered by the Key-Class of the TSP.

End of informative comment

4.3.4.17.1 Tspi_SetAttribUint32

Start of informative comment:

This method sets a 32-bit attribute of the key object.

End of informative comment.

Definition:

See section 4.3.2 for definition.

Parameters

See section 4.3.2 for description.

Defined Attributes

Flag	SubFlag	Attribute	Description
TSS_TSPATTRIB_KEY_REGISTER	0	TSS_TSPATTRIB_KEYREGISTER_USER	Deleted – mistake in 1.1
	0	TSS_TSPATTRIB_KEYREGISTER_SYTEM	Deleted – mistake in 1.1
	0	TSS_TSPATTRIB_KEYREGISTER_NO	Deleted – mistake in 1.1
TSS_TSPATTRIB_KEY_INFO	TSS_TSPATTRIB_KEYINFO_USAGE	TSS_KEYUSAGE_XX	TSS Key usage value indicating the usage type of the key as defined in 2.3.2.24.
	TSS_TSPATTRIB_KEYINFO_MIGRATABLE	Boolean value.	If TRUE, key is migratable.
	TSS_TSPATTRIB_KEYINFO_REDIRECTED	Boolean value.	If TRUE, key is redirected. Refer to main spec for details.
	TSS_TSPATTRIB_KEYINFO_VOLATILE	Boolean value.	If TRUE, key is volatile.
	TSS_TSPATTRIB_KEYINFO_AUTHDATAUSAGE	Boolean value.	If TRUE, authorization is required to use the key.
	TSS_TSPATTRIB_KEYINFO_ALGORITHM	TSS_ALG_XX	TSS algorithm ID value indicating the algorithm of the key as defined in

			2.3.2.19.
	TSS_TSPATTRIB _KEYINFO_ENCSCHEME	TSS_ES_XX	TSS encryption scheme value that the key uses to encrypt information as defined in 2.3.2.29
	TSS_TSPATTRIB _KEYINFO_SIGSCHEME	TSS_SS_XX	TSS signature scheme value that the key uses to perform digital signatures as defined in 2.3.2.30.
	TSS_TSPATTRIB _KEYINFO_SIZE		The key size in bits.
	TSS_TSPATTRIB _KEYINFO_KEYFLAGS		Contains the TCG key flag info.
	TSS_TSPATTRIB _KEYINFO_AUTHUSAGE		Direct set of the authDataUsage in the TCG-KeyParams.
	TSS_TSPATTRIB _KEYINFO_KEYSTRUCT	TSS_KEY_STRUCT _XX	Value indicating the type of key structure used as defined in 2.3.2.26.
TSS_TSPATTRIB _RSAKEY_INFO	TSS_TSPATTRIB _KEYINFO_RSA_ PRIMES		The number of prime factors used by the RSA key.

Return Values

See section 4.3.2 for description.

Remarks

4.3.4.17.2 Tspi_GetAttribUint32

Start of informative comment:

This method gets a 32-bit attribute of the key object

End of informative comment.

Definition:

See section 4.3.3.1.2 for definition.

Parameters

See section 4.3.3.1.2 for description.

Defined Attributes

Flag	SubFlag	Attribute	Description
TSS_TSPATTRIB_KEY_REGISTER	0	TSS_TSPATTRIB_KEYREGISTER_USER	Key is registered automatically in PS.
	0	TSS_TSPATTRIB_KEYREGISTER_SYTEM	Key is registered automatically in PS.
	0	TSS_TSPATTRIB_KEYREGISTER_NO	Key is not registered in PS.
TSS_TSPATTRIB_KEY_INFO	TSS_TSPATTRIB_KEYINFO_USAGE	TSS_KEYUSAGE_XX	TSS Key usage value indicating the usage type of the key as defined in 2.3.2.24.
	TSS_TSPATTRIB_KEYINFO_MIGRATABLE	Boolean value.	If TRUE, key is migratable.
	TSS_TSPATTRIB_KEYINFO_CMK	Boolean value.	If TRUE, then key is certified migratable
	TSS_TSPATTRIB_KEYINFO_REDIRECTED	Boolean value.	If TRUE, key is redirected. Refer to main spec for details.
	TSS_TSPATTRIB_KEYINFO_VOLATILE	Boolean value.	If TRUE, key is volatile.
	TSS_TSPATTRIB_KEYINFO_AUTHDATAUSAGE	Boolean value.	If TRUE, authorization is required to use the key.
	TSS_TSPATTRIB_KEYINFO_ALOGRITHM	TSS_ALG_XX	TSS algorithm ID value indicating the algorithm of the key as defined in 2.3.2.19.
	TSS_TSPATTR	TSS_ES_XX	TSS encryption

	IB_KEYINFO_ ENCSCHEME		scheme value that the key uses to encrypt information as defined in 2.3.2.29
	TSS_TSPATTR IB_KEYINFO_ SIGSCHEME	TSS_SS_XX	TSS signature scheme value that the key uses to perform digital signatures as defined in 2.3.2.30
	TSS_TSPATTR IB_KEYINFO_ KEYFLAGS		Contains the TCG key flag info.
	TSS_TSPATTR IB_KEYINFO_ AUTHUSAGE		Returns the content of the authDataUsage.
	TSS_TSPATTR IB_KEYINFO_ KEYSTRUCT	TSS_KEY_STRUCT_XX	Value indicating the type of key structure used as defined in 2.3.2.26.
	TSS_TSPATTR IB_KEYINFO_ SIZE		The key size in bits.
TSS_TSPATTRI B_KEY_PCR_LO NG	TSS_TSPATTR IB_KEYPCRLO NG_LOCALITY _ATCREATION	The locality modifier when the blob was created	
	TSS_TSPATTR IB_KEYPCRLO NG_LOCALITY _ATRELEASE	The locality modifier required for using the key	
TSS_TSPATTRI B_RSAKEY_INF O	TSS_TSPATTR IB_KEYINFO_ RSA_KEYSIZE		The size of the RSA key in bits
	TSS_TSPATTR IB_KEYINFO_ RSA_PRIMES		The number of prime factors used by the RSA key.

Return Values

See section 4.3.3.1.2 for description.

Remarks

4.3.4.17.3 Tspi_SetAttribData

Start of informative comment:

This method sets a non 32-bit attribute of the key object. The structure and size of the attribute data depends on the attribute.

End of informative comment.

Definition:

See section 4.3.3.1.3 for definition.

Parameters

See section 4.3.3.1.3 for description.

Defined Attributes

Flag	SubFlag	Data Description
TSS_TSPATTRIB_KEY_BLOB	TSS_TSPATTRIB_KEYBLOB_BLOB	Key information as a key blob.
	TSS_TSPATTRIB_KEYBLOB_PUBLIC_KEY	Public key information as a key blob.
	TSS_TSPATTRIB_KEYBLOB_PRIVATE_KEY	Encrypted private key information as private key blob.
TSS_TSPATTRIB_CMK_INFO	TSS_TSPATTRIB_CMK_INFO_MA_APPROVAL	HMAC of the migration authority approval
	TSS_TSPATTRIB_CMK_INFO_MA_DIGEST	Migration authority digest data
TSS_TSPATTRIB_RSAKEY_INFO	TSS_TSPATTRIB_KEYINFO_RSA_EXPONENT	The public exponent of the key.
	TSS_TSPATTRIB_KEYINFO_RSA_MODULUS	The RSA public modulus.

Return Values

See section 4.3.3.1.3 for description.

Remarks

4.3.4.17.4 Tspi_GetAttribData

Start of informative comment:

This method gets a non 32-bit attribute of the key object. The structure and size of the attribute data depends on the attribute. Note: if the SRK public key is asked for, a BadParameter error message will be returned. The SRK public key must be obtained directly from the TPM using the Tspi_Key_PubKey

End of informative comment.

Definition:

See section 4.3.3.1.4 for definition.

Parameters

See section 4.3.3.1.4 for description.

Defined Attributes

Flag	SubFlag	Data Description
TSS_TSPATTRIB_KEY_BLOB	TSS_TSPATTRIB_KEYBLOB_BLOB	Key information returned as a key blob.
	TSS_TSPATTRIB_KEYBLOB_PUBLIC_KEY	Public key information as public key blob.
	TSS_TSPATTRIB_KEYBLOB_PRIVATE_KEY	Encrypted private key information as private key blob.
TSS_TSPATTRIB_KEY_INFO	TSS_TSPATTRIB_KEYINFO_VERSION	Version info returned as TSS_VERSION structure
TSS_TSPATTRIB_RSAKEY_INFO	TSS_TSPATTRIB_KEYINFO_RSA_EXPONENT	The public exponent of the key.
	TSS_TSPATTRIB_KEYINFO_RSA_MODULUS	The RSA public modulus.
TSS_TSPATTRIB_KEY_UUID	0	TSS_UUID structure containing the UUID the key is assigned to.
TSS_TSPATTRIB_KEY_PCR_LONG	TSS_TSPATTRIB_KEYPCRLONG_CREATION_PCRLONG_SELECTION	The selection of PCRs active when the blob was created
	TSS_TSPATTRIB_KEYPCRLONG_RELEASE_PCRLONG_SELECTION	The selection of PCRs required for use of key
	TSS_TSPATTRIB_KEYPCRLONG_DIGEST_ATCREATION	The digest of the PCRs corresponding to the creation PCR selection
	TSS_TSPATTRIB_KEYPCRLONG_DIGEST_ATRELEASE	The digest of the PCRs corresponding ot the release PCR selection necessary for use of the key
TSS_TSPATTRIB_KEY_PCR	TSS_TSPATTRIB_KEYPCR_DIGEST_ATCREATION	Composite digest value of the PCR values, at the time when the sealing was

		performed.
	TSS_TSPATTRIB_KEYPCR_DIGEST_ATRELEASE	Composite digest value of the PCR values, at the time when the unsealing should be performed.
	TSS_TSPATTRIB_KEYPCR_SELECTION	A bit map that indicates if a PCR is active or not.
TSS_TSPATTRIB_CMK_INFO	TSS_TSPATTRIB_CMK_INFO_MA_APPROVAL	HMAC of the migration authority approval
	TSS_TSPATTRIB_CMK_INFO_MA_DIGEST	Migration authority digest data

Return Values

See section 4.3.3.1.4 for description.

Remarks

4.3.4.17.5 Tspi_Key_LoadKey

Start of informative comment:

The Tspi_Key_LoadKey method loads the key blob of the object into the TPM. The TPM will unwrap the key when it is loaded.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Key_LoadKey
(
    TSS_HKEY   hKey,            // in
    TSS_HKEY   hUnwrappingKey   // in
);
```

Parameters

hKey

Handle of the key object to load.

hUnwrappingKey

Handle of the key which should be used to unwrap the key addressed by *hKey*.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The key information for the key to load is set by calling *Tspi_SetAttribData*. The key blob addressed by *hUnwrappingKey* must have been loaded into the TPM previously.

4.3.4.17.6 Tspi_Key_UnloadKey

Start of informative comment:

The Tspi_Key_UnloadKey method unloads the key referenced by the key object from the TPM. This call will result in a TPM_EvictKey operation for the specified key.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Key_UnloadKey
(
  TSS_HKEY   hKey          // in
);
```

Parameters

hKey

Handle of the key object to unload.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR
TPM_KEY_OWNER_CONTROL

Remarks

The Tspi_Key_UnloadKey method unloads the key referenced by the key object from the TPM. This call will result in a TPM_EvictKey operation for the specified key.

4.3.4.17.7 Tspi_Key_GetPubKey

Start of informative comment:

This method returns the public key of the key object.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Key_GetPubKey
(
    TSS_HKEY    hKey,             // in
    UINT32*     pulPubKeyLength,  // out
    BYTE**      prgbPubKey        // out
);
```

Parameters

hKey

Handle of the key object.

pulPubKeyLength

Receives the length (in bytes) of the *prgbPubKey* parameter.

prgbPubKey

Receives a pointer to the memory block containing the public key blob retrieved for the key object referenced by *hKey*.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

4.3.4.17.8 Tspi_Key_CertifyKey

Start of informative comment:

This method signs a public key using TPM_SS_RSASSAPKCS1v15_SHA1 . It requires some extra data if the certification procedure has to do with CMK (see remark).

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Key_CertifyKey
(
  TSS_HKEY        hKey,           // in
  TSS_HKEY        hCertifyingKey, // in
  TSS_VALIDATION* pValidationData // in, out
);
```

Parameters

hKey

Handle of the key object where the public key should be signed.

hCertifyingKey

Handle to the certifying key used to sign the key addressed by *hKey*.

pValidationData

Pointer addresses a TSS_VALIDATION structure. After successful completion of the call the member rgbValidationData of this structure contains the signature data of the command. The member prgbData of the structure points to a buffer containing a TCPA_CERTIFY_INFO2 data stream as specified within the TCG TPM 1.2 Main Specification or a TCPA_CERTIFY_INFO data stream as specified within the TCG 1.1b Main Specification.

TCPA_CERTIFY_INFO2: On TPM v1.2 with usage of CMK keys and keys with locality restrictions (TPM_KEY12 key complex).

TCPA_CERTIFY_INFO: Is returned if the TSS is connected to a TPM v1.1 or the key which is used on a TPM v1.2 is a legacy one (PCR's without locality -> TPM_KEY key complex).

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

This method calls the TPM command *TPM_CertifyKey* where the public key information to be signed is addressed by *hKey* and the signing key is addressed by *hCertifyingKey*. Memory allocated by this method for the members of the structure *TSS_VALIDATION* must be deallocated by calling *Tspi_Context_FreeMemory*.

The decision which TPM operation should be used will be done by the TSP depending on the condition and information set in the associated Key-Objects. The Key-Object carries the key type information and can determine the connection info from context. If the involved key is a Certified Migratable Key (CMK) key then the certification procedure requires the MSA-Composite data which is a digest of the TPM-MSA_COMPOSITE structure, containing at least one public key of a Migration Authority. This data can be set with SetAttribData at the Key-Object.

4.3.4.17.9 Tspi_Key_CreateKey

Start of informative comment:

The *Tspi_Key_CreateKey* method creates a key pair within the TPM and wraps it with the key addressed by *hWrappingKey*. After this function returns successfully, no attributes of the key object (other than the key blob itself) can be set without affecting the usability of the key. Calling Tspi_SetAttribData or Tspi_SetAttribUint32 on a created key object SHOULD result in an error being returned, except in the case of setting a new blob.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Key_CreateKey
(
   TSS_HKEY   hKey,          // in
   TSS_HKEY   hWrappingKey, // in
   TSS_HPCRS  hPcrComposite // in, may be NULL
);
```

Parameters

hKey

Handle of the key object to create.

hWrappingKey

Handle to the key used to wrap the newly created key.

hPcrComposite

Handle to an object of the type *Tspi_PcrComposite*. If the value of the handle doesn't equal to NULL, the newly created key will be bound to the PCR values described with this object.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_KEY_NO_MIGRATION_POLICY
TSS_E_INTERNAL_ERROR

Remarks

This method calls the TPM command *TPM_CreateWrapKey*. If a PCR composite object is provided at the parameter *hPcrComposite* (*hPcrComposite* is not NULL) the created key blob is bound to these PCR values.

If *hKey* is using a TCPA_KEY structure (1.1), then *hPcrComposite* must use a TCPA_PCR_INFO structure. If *hKey* is using a TCPA_KEY12 structure (1.2), then *hPcrComposite* must use a TCPA_PCR_INFO_LONG structure. If the wrong combination of objects is used, then error TSS_E_INVALID_OBJ_ACCESS will be returned.

The key object addressed by *hKey* must contain the key information needed for key creation, previously set with *Tspi_SetAttribXXX()*. On return the object pointed to by *hKey* contains the wrapped key blob, which can be retrieved by calling *GetAttribData()*.

When a migratable key will be created the command requires a migration secret provided with a policy. If there was no secret set for this policy it will be retrieved via pop-up or callback function.

4.3.4.17.10 Tspi_Key_WrapKey

Start of informative comment:

This method wraps a key with the key addressed by *hWrappingKey*. After this function returns successfully, no attributes of the key object (other than the key blob itself) can be set without affecting the usability of the key. Calling Tspi_SetAttribData or Tspi_SetAttribUint32 on a created key object SHOULD result in an error being returned, except in the case of setting a new blob.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Key_WrapKey
(
    TSS_HKEY    hKey,           // in
    TSS_HKEY    hWrappingKey,   // in
    TSS_HPCRS   hPcrComposite   // in, may be NULL
);
```

Parameters

hKey

Handle of the key object to create.

hWrappingKey

Handle to the key used to wrap the key addressed by *hKey*.

hPcrComposite

Handle to an object of the type *Tspi_PcrComposite*. If the value of the handle doesn't equal to NULL, the key addressed by *hKey* will be bound to the PCR values described with this object.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

If a PCR composite object is provided at the parameter *hPcrComposite* (*hPcrComposite* is not NULL) the created key blob is sealed to this PCR values.

If *hKey* is using a TCPA_KEY structure (1.1), then *hPcrComposite* must use a TCPA_PCR_INFO structure. If *hKey* is using a TCPA_KEY12 structure (1.2), then *hPcrComposite* must use a TCPA_PCR_INFO_LONG structure. If the wrong combination of objects is used, then error TSS_E_INVALID_OBJ_ACCESS will be returned.

The key object addressed by *hKey* must contain the key information required for key creation, previously set with *Tspi_SetAttribXXX()*. On return the object pointed to by *hKey* contains the wrapped key blob, which can be retrieved by calling Tspi_*GetAttribData()*.

4.3.4.17.11 Tspi_TPM_AuthorizeMigrationTicket

Start of informative comment:

This method provides the migration ticket required for the migration process.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_AuthorizeMigrationTicket
(
  TSS_HTPM            hTPM,               // in
  TSS_HKEY            hMigrationKey,         // in
  TSS_MIGRATE_SCHEME migrationScheme ,  // in
  UINT32*             pulMigTicketLength,// out
  BYTE**              prgbMigTicket          // out
);
```

Parameters

hTPM

Handle of the TPM object

hMigrationKey

Handle of the key object representing the migration key.

migrationScheme

Flag indicating the migration scheme to be used.

pulMigTicketLength

Receives the length (in bytes) of the *prgbMigTicket* parameter.

prgbMigTicket

Receives a pointer to the memory block containing the migration ticket blob.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The Tspi_TPM_AuthorizeMigrationTicket method allocates a memory block for the requested ticket data. This memory must be released utilizing the Tspi_Context_FreeMemory method.

4.3.4.17.12 Tspi_Key_CreateMigrationBlob

Start of informative comment:

This command was written so that a TPM can be used by a migration authority.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Key_CreateMigrationBlob
(
   TSS_HKEY hKeyToMigrate,          // in
   TSS_HKEY hParentKey,             // in
   UINT32   ulMigTicketLength,      // in
   BYTE*    rgbMigTicket,           // in
   UINT32*  pulRandomLength,        // out
   BYTE**   prgbRandom,             // out
   UINT32*  pulMigrationBlobLength, // out
   BYTE**   prgbMigrationBlob       // out
);
```

Parameters

hKeyToMigrate

Handle of the key object to migrate.

hParentKey

Handle to the parent key related to the key addressed by *hKeyToMigrate*.

ulMigTicketLength

The length (in bytes) of the *rgbMigTicket* parameter

rgbMigTicket

Pointer to memory containing the migration ticket (migration public key and its authorization digest). This data previously have been returned by the method *Tspi_TPM_AuthorizeMigrationTicket()*.

pulRandomLength

On successful completion this parameter returns the random data length returned at the parameter *prgbRandom*.

prgbRandom

On successful completion this parameter returns the random data.

pulMigrationBlobLength

On successful completion this parameter returns the length of the migration blob data returned at the parameter *prgbMigrationBlob*.

prgbMigrationBlob

On successful completion this parameter returns the migration data blob.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_KEY_NO_MIGRATION_POLICY
TSS_E_INTERNAL_ERROR

Remarks

The function returns a key blob containing an encrypted part, which will be different depending on the migration scheme indicated within the migration ticket previously created by the method *Tspi_TPM_AuthorizeMigrationTicket()*.

Migration scheme: TSS_MS_REWRAP

The returned key blob can be loaded into another TPM without further actions. No random number is returned.

Migration scheme: TSS_MS_MIGRATE

The method returns a random number and a migration blob which must be converted by calling *Tspi_Key_ConvertMigrationBlob()*.

This method calls the TPM command TPM_CreateMigrationBlob().

The Tspi_Key_CreateMigrationBlob method allocates a memory block for the allocated data. This memory must be released utilizing the Tspi_Context_FreeMemory method.

4.3.4.17.13 Tspi_Key_ConvertMigrationBlob

Start of informative comment:

This method takes the migration blob built by Tspi_Key_CreateMigrationBlob using the migration scheme TSS_MS_MIGRATE and creates a normal wrapped key. The resulting normal wrapped key blob is stored in the instance associated with hKeyToMigrate and may be retrieved from that instance by Tspi_GetAttribData().

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Key_ConvertMigrationBlob
(
    TSS_HKEY hKeyToMigrate,          // in
    TSS_HKEY hParentKey,             // in
    UINT32   ulRandomLength,         // in
    BYTE*    rgbRandom,              // in
    UINT32   ulMigrationBlobLength,  // in
    BYTE*    rgbMigrationBlob        // in
);
```

Parameters

hKeyToMigrate

Handle of the key object to convert.

hParentKey

Handle to the parent key related to the key addressed by *hKeyToMigrate*.

ulRandomLength

Length of random data provided at the parameter *rgbRandom*.

rgbRandom

Random data as returned together with the migration blob by the method *Tspi_Key_CreateMigrationBlob()*.

ulMigrationBlobLength

Length of the migration blob data provided at the parameter *rgbMigrationBlob*.

rgbMigrationBlob

Migration blob data as returned by a previously called method *Tspi_Key_CreateMigrationBlob()*.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

4.3.4.17.14 Tspi_ChangeAuth

Start of informative comment:

This method changes the authorization data (owner secret) of the key object and assigns the key object to the policy object.

End of informative comment.

Definition:

See section 4.3.3.1.5 for definition.

Parameters

hObjectToChange

Handle of the key object.

hParentObject

Handle to the parent object related with the object addressed by *hObjectToChange*.

hNewPolicy

Handle to the new policy object.

Return Values

See section 4.3.3.1.5 for description.

Remarks

The TSP's key manager SHALL replace key blobs associated with *hObjectToChange with the new key blobs that contain the new authorization value.*

Changing the owner secret of the SRK (Storage Root Key) the parameter *hObjectToChange* must refer to the SRK object and the parameter *hParentObject* must refer to the TPM object

4.3.4.17.15 Tspi_ChangeAuthAsym

Start of informative comment:

This method changes the authorization data (secret) of the key object utilizing the asymmetric change protocol and assigns the key object to new policy object.

This method changes the authorization data of the key object ensuring that the parent of the key object does not get knowledge of the new secret.

End of informative comment.

Definition:

See section 4.3.3.1.6 for definition.

Parameters

See section 4.3.3.1.6 for description.

Return Values

See section 4.3.3.1.6 for description.

Remarks

4.3.4.17.16 Tspi_GetPolicyObject

Start of informative comment:

This method returns a policy object currently assigned to the key object

End of informative comment.

Definition:

See section 4.3.3.1.7 for definition.

Parameters

See section 4.3.3.1.7 for description.

Return Values

See section 4.3.3.1.7 for description.

Remarks

See section 4.3.3.1.7 for description.

4.3.4.18 CMK commands:

Start of informative comment:

Key Class	Available in 1.2	Security Properties	Usability
Non-Migratable	NO	Those using key can be assured private component always protected by TPM	Cannot backup key
Migratable	NO	Private component is protected while used and store but no assurance it has never been migrated to an untrusted destination	Keys easily backed up
Certified Migratable	YES	Private component is protected while used and store AND provides assurance it has can only be migrated to trusted destinations	Keys easily backed up

In version 1.1 there were two key types, non-migratable and migratable keys. The TPM would only certify non-migrating keys. There is a need for a key that allows migration but allows for certification. This proposal is to create a key that allows for migration but still has properties that the TPM can certify.

These new keys are "certifiable migratable keys" or CMK. This designation is to separate the keys from either the normal migration or non-migration types of keys. The TPM Owner is not required to use these keys.

Two entities may participate in the CMK process. The first is the Migration-Selection Authority and the second is the Migration Authority (MA).

Migration Selection Authority (MSA)

The MSA controls the migration of the key but does not handle the migration itself. Instead it selects the Migration Authority that is allowed to handle the migration.

Migration Authority (MA)

A Migration Authority actually handles the migrated key.

Use of MSA and MA

Migration of a CMK occurs using TPM_CMK_CreateBlob (TPM_CreateMigrationBlob cannot be used). The TPM Owner authorizes the migration destination (as usual), and the key owner authorizes the migration transformation (as usual). An MSA authorizes the migration destination as well. If the MSA is the migration destination, no MSA authorization is required.

End of informative comment.

4.3.4.18.1 Tspi_TPM_CMKSetRestrictions

Start of informative comment:

This method is used by the owner to globally dictate the usage of a certified migration key with delegated authorization.

This command can't be owner delegated.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_CMKSetRestrictions
(
    TSS_HTPM          hTPM,          // in
    TSS_CMK_DELEGATE CmkDelegate    // in
);
```

Parameters:

hTPM

Handle of the TPM object.

CmkDelegate

Bit mask to determine the restrictions on certified-migration-keys.

Return Values:

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks:

The default value is that no restrictions are active.

4.3.4.18.2 Tspi_TPM_CMKApproveMA

Start of informative comment:

This method creates an authorization ticket, to allow the TPM owner to specify which Migration Authorities they approve and allow users to create certified-migration-keys without further involvement with the TPM owner.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_CMKApproveMA
(
    TSS_HTPM       hTPM,            // in
    TSS_HMIGDATA   hMaAuthData      // in
);
```

Parameters:

hTPM

Handle of the TPM object.

hMaAuthData

Migration data properties object to transfer the input and output data blob during the migration process. For this command this object calculates the digest of the selected MSA (Migration Selection Authority) which are imported into this object.

Return Values:

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks:

The hMaAuthData contains an arbitrary number of public keys belonging to Migration Authorities imported via multiple calls of SetAttribData(). The Migration-Data-Property-Object internally calculates the digest structure which will be used for the approval process. The HMAC data blob of the migration authority can be exported with GetAttribData().

4.3.4.18.3 Tspi_TPM_CMKCreateTicket

Start of informative comment:

This method uses a public key to verify the signature over a digest. The output ticket data can be used to prove the same TPM for signature verification.

This operation requires owner authorization which can be delegated.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_CMKCreateTicket
(
   TSS_HTPM      hTPM,            // in
   TSS_HKEY      hVerifyKey,      // in
   TSS_HMIGDATA  hSigData         // in
);
```

Parameters:

hTPM

Handle of the TPM object.

hVerifyKey

Handle of the Key object containing the public key used to check the signature value.

hSigData

Migration data properties object to transfer the input and output data blob during the migration process. For this command the object includes the data proper to be signed and the signature value to be verified. The caller can access the ticket/signature date via GetAttribData().

Return Values:

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks:

Before the execution the hSigData parameter references an object which contains the signature data (digest) and the signature value (Byte-Stream).

After the call the hSigData object holds the ticket data (HMAC) and the caller can access it via GetAttribData().

4.3.4.19 Tspi_MigData Class Definition

Start of informative comment:

If an application intends to support the CMK migration feature then this class represents a data access layer and container for information relevant in the CMK migration flow.

When designing a CMK application this class contains the data associated with the CMK migration process: e.g.

- Public-Key data from a MA (Migration Authority); MSA (Migration Selection Authority)
- CMK migration ticket data
- Construct the MSA-List containing the linked digest info

In addition to the pure container functionality the class offers some CMK migration data manipulation service: e.g.

- HMAC and DIGEST calculation for the Public-Key data blobs

End of informative comment.

4.3.4.19.1 Tspi_SetAttribUint32

Start of informative comment:

This method sets a 32bit attribute of the MigData object.

End of informative comment.

Definition:

See section 4.3.2 for definition.

Parameters

See section 4.3.2 for description.

Defined Attributes

Flag	SubFlag	Attribute	Description

Return Values

See section 4.3.2 for description.

Remarks

4.3.4.19.2 Tspi_GetAttribUint32

Start of informative comment:

This method gets a 32bit attribute of the MigData object

End of informative comment.

Definition:

See section 4.3.3.1.2 for definition.

Parameters

See section 4.3.3.1.2 for description.

Defined Attributes

Flag	SubFlag	Attribute	Description

Return Values

Remarks

4.3.4.19.3 Tspi_SetAttribData

Start of informative comment:

This method sets a non 32bit attribute of the MigData object. The structure and size of the attribute data depends on the attribute.

End of informative comment.

Definition:

See section 4.3.3.1.3 for definition.

Parameters

See section 4.3.3.1.3 for description.

Flag	SubFlag	Data Description	Usage
TSS_MIGATTRIB_MIGRATIONBLOB	TSS_MIGATTRIB_MIG_MSALIST_PUBKEY_BLOB	Public key information from a migration authority as a key blob.	Tspi_TPM_CMKApproveMA;
	TSS_MIGATTRIB_MIG_AUTHORITY_PUBKEY_BLOB	Public key belonging to migration authority;	Tspi_Key_CMKCreateBlob
	TSS_MIGATTRIB_MIG_DESTINATION PUBKEY_BLOB	Approved destination public key;	Tspi_Key_CMKCreateBlob
	TSS_MIGATTRIB_MIG_SOURCE_PUBKEY_BLOB	Public key to be migrated;	Tspi_Key_CMKCreateBlob
TSS_MIGATTRIB_AUTHORITY_DATA	TSS_MIGATTRIB_AUTHORITY_DIGEST	Digest of the selected migration selection authorities.	Tspi_TPM_CMKApproveMA; Tspi_Key_CreateKey
	TSS_MIGATTRIB_AUTHORITY_APPROVAL_HMAC	Approved migration authority ticket data.	Tspi_TPM_CMKApproveMA; Tspi_Key_CreateKey
	TSS_MIGATTRIB_AUTHORITY_MSALIST	Digest-List of the public key belonging to a migration authority.	Tspi_Key_CMKCreateBlob; Tspi_TPM_CMKApproveMA

TSS_MIGATTRIB_TICKET_DATA	TSS_MIGATTRIB_TICKET_SIG_DIGEST	Data portion do be verfied that signature is valid.	Tspi_TPM_CMKCreateTicket
	TSS_MIGATTRIB_TICKET_SIG_VALUE	Signature value to be verified.	Tspi_TPM_CMKCreateTicket
	TSS_MIGATTRIB_TICKET_SIG_TICKET	Signature ticket to prove the creation on a specific TPM.	Tspi_TPM_CMKCreateTicket; Tspi_Key_CMKCreateBlob
	TSS_MIGATTRIB_TICKET_RESTRICT_TICKET	Containing the digests of the public keys belonging to the migration authority	Tspi_Key_CMKCreateBlob
TSS_MIGATTRIB_MIGRATIONTICKET	0	Accesses the migration ticket data from the authorize migration key process.	Tspi_Key_CMKCreateBlob; Tspi_TPM_AuthorizeMigrationTicket
TSS_MIGATTRIB_MIG_AUTH_DATA	TSS_MIGATTRIB_MIG_AUTH_AUTHORITY_DIGEST	Digest (public key) of the selected migration selection authorities.	
	TSS_MIGATTRIB_MIG_AUTH_DESTINATION_DIGEST	Digest of a public key for the approved destination.	
	TSS_MIGATTRIB_MIG_AUTH_SOURCE_DIGEST	Digest of a public key for the key to be migrated.	

Return Values

See section 4.3.3.1.3 for description.

Remarks

4.3.4.19.4 Tspi_GetAttribData

Start of informative comment:

This method gets a non 32bit attribute of the MigData object. The structure and size of the attribute data depends on the attribute.

End of informative comment.

Definition:

See section 4.3.3.1.4 for definition.

Parameters

See section 4.3.3.1.4 for description.

Defined Attributes

Flag	SubFlag	Data Description	Usage
TSS_MIGATTRIB_MIGRATIONBLOB	TSS_MIGATTRIB_MIGRATION_XOR_BLOB	Output data packet from CreateBlob operation.	Tspi_Key_CMKCreateBlob;
TSS_MIGATTRIB_AUTHORITY_DATA	TSS_MIGATTRIB_AUTHORITY_DIGEST	Digest of the selected migration selection authorities.	Tspi_TPM_CMKApproveMA; Tspi_Key_CreateKey
	TSS_MIGATTRIB_AUTHORITY_APPROVAL_HMAC	Approved migration authority ticket data.	Tspi_TPM_CMKApproveMA; Tspi_Key_CreateKey
	TSS_MIGATTRIB_AUTHORITY_MSALIST	Digest-List of the public key belonging to a migration authority.	Tspi_Key_CMKCreateBlob; Tspi_TPM_CMKApproveMA
TSS_MIGATTRIB_TICKET_DATA	TSS_MIGATTRIB_TICKET_SIG_TICKET	Signature ticket to prove the creation on a specific TPM.	Tspi_TPM_CMKCreateTicket

TSS_MIGATTRIB_MIG_AUTH_DATA	TSS_MIGATTRIB_MIG_AUTH_AUTHORITY_DIGEST	Digest (public key) of the selected migration selection authorities.	
	TSS_MIGATTRIB_MIG_AUTH_DESTINATION_DIGEST	Digest of a public key for the approved destination.	
	TSS_MIGATTRIB_MIG_AUTH_SOURCE_DIGEST	Digest of a public key for the key to be migrated.	

Return Values

See section 4.3.3.1.4 for description.

Remarks

4.3.4.19.5 Tspi_Key_MigrateKey

Start of informative comment:

This method decrypts with assistance of the TPM the input package (e.g. Key) and then re-encrypts it with the input public key.

This command exists to allow the TPM to be a migration authority.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Key_MigrateKey
(
    TSS_HKEY      hMaKey,            // in
    TSS_HKEY      hPublicKey,        // in
    TSS_HKEY      hMigData           // in
);
```

Parameters:

hMaKey

Handle of the key object to be migrated

hPublicKey

Handle to the public key to which the blob is to be migrated.

hMigData

Migration data key object to transfer the input and output data blob during the migration process. The input data blob is from the previous call of the function *CreateMigrationBlob* or *CMK_CreateBlob*.

Return Values:

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks:

The data blob is stored in the key data object addressed by hMigData and can be exported from that object by GetAttribData( ) and imported via SetAttribData().

4.3.4.19.6 Tspi_Key_CMKCreateBlob

Start of informative comment:

This method implements the first step in the process of moving a certified migrateable key to a new parent platform.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Key_CMKCreateBlob
(
    TSS_HKEY        hKeyToMigrate,      // in
    TSS_HKEY        hParentKey,         // in
    TSS_HMIGDATA    hMigrationData,     // in
    UINT32*         pulRandomLength,    // out
    BYTE**          prgbRandom          // out
);
```

Parameters:

hKeyToMigrate

Handle of the key object to migrate.

hParentKey

Handle to the parent key related to the key addressed by *hKeyToMigrate*.

hMigrationData

Migration data properties object to transfer the input and output data blob during the migration process.

pulRandomLength

On successful completion this parameter returns the random data length returned at the parameter prgbRandom.

prgbRandom

On successful completion this parameter returns the random data.

Return Values:

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_KEY_NO_MIGRATION_POLICY
TSS_E_INTERNAL_ERROR

Remarks:

The function returns a key blob containing an encrypted part, which will be different depending on the migration scheme.

In addition the method expects different input information depending on the selected migration scheme.

Migration scheme: TSS_MS_RESTRICT_MIGRATE:

This scheme is used for the migration of a CMK directly to a MSA.

Migration scheme: TSS_MS_RESTRICT_APPROVE_DOUBLE:

This scheme is used for the migration of a CMK to a destination other than the MSA (i.e. MA). This mode requires further checks of the selected migration destination and needs therefore the “sigTicket” and “restrictTicket” data to be set at the object. This ticket can contain a composite structure which contains a list of multiple MA’s.

4.3.4.19.7 Tspi_Key_CMKConvertMigration

Start of informative comment:

This method completes the migration of a certified migration process. This function takes a certified migration blob and creates a normal wrapped key blob which must be loaded into the TPM using the normal LoadKey operation.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Key_CMKConvertMigration
(
    TSS_HKEY      hKeyToMigrate,      // in
    TSS_HKEY      hParentKey,         // in
    TSS_HMIGDATA  hMigrationData,     // in
    UINT32        ulRandomLength,     // in
    BYTE*         rgbRandom           // in
);
```

Parameters:

hKeyToMigrate

Handle of the key object to migrate.

hParentKey

Handle to the parent key related to the key addressed by *hKeyToMigrate*.

hMigrationData

Migration data properties object to transfer the input and output data blob during the migration process.

ulRandomLength

Length of random data provided at the parameter *rgbRandom*.

rgbRandom

Random data as returned together with the migration blob by the method *Tspi_Key_CMKCreateBlob()*.

Return Values:

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_KEY_NO_MIGRATION_POLICY
TSS_E_INTERNAL_ERROR

Remarks:

The related TPM command migrates private keys only. The migration of the associated public keys is not specified by the TPM and will be handled by the TSS this is equivalent to the non CMK migration procedure.

In addition the TPM checks that one of the MAs is listed in the "migrationAuth" of the target key has approved migration to the destination key, this data set is managed by the MigrationData-Object (hMigrationData).

4.3.4.20 Tspi_Hash Class Definition

4.3.4.20.1 Tspi_SetAttribData

Start of informative comment:

This method sets a non 32-bit attribute of the data object. The structure and size of the attribute data depends on the attribute. We note that programmers should be cautious in their selection of hashs, as many of the more commonly used ones have recently been found to be weak to one degree or another.

End of informative comment.

Definition:

See section 4.3.3.1.3 for definition.

Parameters

See section 4.3.3.1.3 for description.

Defined Attributes

Flag	SubFlag	Data Description
TSS_TSPATTRIB_HASH_IDENTIFIER	0	Deprecated, use TSS_TSPATTRIB_ALG_IDENTIFIER instead.
TSS_TSPATTRIB_ALG_IDENTIFIER	0	Sets the AlgorithmIdentifier as defined in the PKCS#1 v2.1 standard (see remarks)

Return Values

See section 4.3.3.1.3 for description.

Remarks

Some of the commonly used algorithm identifier values are listed here:

Hash Algorithm	OID	ulAttribData Size	rgbAttribData
SHA1	1.3.14.3.2.26	11	0x30, 0x09, 0x06, 0x05, 0x2b, 0x0e, 0x03, 0x02, 0x1a, 0x05, 0x00
MD5	1.2.840.113549.2.5	14	0x30, 0x0c, 0x06, 0x08, 0x2a, 0x86, 0x48, 0x86, 0xf7, 0x0d, 0x02, 0x05, 0x05, 0x00
MD4	1.2.840.113549.2.4	14	0x30, 0x0C, 0x06, 0x08, 0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D, 0x02, 0x04, 0x05, 0x00
MD2	1.2.840.113549.2.2	14	0x30, 0x0C, 0x06, 0x08, 0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D,

			0x02, 0x02, 0x05, 0x00

Other algorithm identifiers not listed in the table can be used. If an application wishes to sign data without explicitly specifying any OID, it may do so by setting the raw hash value using Tspi_Hash_SetHashValue() function and then calling Tspi_Hash_Sign().

Example Use Cases:

PKCS#1 v2.1 standard defines `AlgorithmIdentifier as follows:`

```
AlgorithmIdentifier { ALGORITHM-IDENTIFIER:InfoObjectSet } ::= SEQUENCE
{
    algorithm
            ALGORITHM-IDENTIFIER.&id({InfoObjectSet}),
    parameters
            ALGORITHM-IDENTIFIER.&Type({InfoObjectSet}{@.algorithm})  OPTIONAL
}
```

It defines DigestInfo as follows:

```
DigestInfo ::= SEQUENCE
{
            digestAlgorithm AlgorithmIdentifier,
            digest Digest
}
```

And it defines the PKCS#1 signature padding algorithm as follows:

Let T be the DER encoding of the DigestInfo value. The sequence T is converted to the sequence {0x00 0x01, PS, 0x00, T}, where PS is a sequence of 8 or more bytes with the value 0xff, long enough to make the length of the entire sequence appropriate for the RSA modulus size.

Signing flow:

1. Application sets AlgorithmIdentifier using Tspi_SetAttribData() (optional).
2. Application sets Digest value using Tspi_SetHashValue() or Tspi_UpdateHashValue().
3. In Tspi_Hash_Sign() the TSS constructs a blob and passes it to the TPM:

- If the key's signature scheme is PKCS1v15_SHA1 then just the Digest value is passed to the TPM. The TPM constructs the DER-endoded DigestInfo, T, performs PKCS#1 signature padding, and performs the RSA signature algorithm.
- If the key's signature scheme is PKCS1v15_DER, the hash object's algorithm attribute is set to TSS_HASH_OTHER, and the AlgorithmIdentifier is not set, then the Digest value is assumed to be T and just the Digest is passed to the TPM. The TPM performs PKCS#1 signature padding on the value and performs the RSA signature algorithm.
- If the key's signature scheme is PKCS1v15_DER, and either the hash object's algorithm attribute is set to TSS_HASH_SHA1 or the AlgorithmIdentifier is set, the

TSS constructs the DER-encoded DigestInfo sequence, T, and passes it to the TPM. The TPM performs PKCS#1 signature padding on the value and performs the RSA signature algorithm.

Signature verification flow:

1. Application sets AlgorithmIdentifier using Tspi_SetAttribData() (optional).
2. Application sets Digest value using Tspi_SetHashValue() or Tspi_UpdateHashValue().
3. In Tspi_Hash_VerifySignature() the TSS constructs the blob to be verified:

- The TSS forms T:
    - If the key's signature scheme is PKCS1v15_SHA1, or the key's signature scheme is PKCS1v15_DER and the hash object's attribute is TSS_HASH_SHA1, then the TSS sets T to be a DER-encoded DigestInfo sequence, where AlgorithmIdentifier is defined as in the table above and the Digest value was set by the application in step 2.
    - If the key's signature scheme is PKCS1v15_DER, the hash object's algorithm attribute is set to TSS_HASH_OTHER, and the AlgorithmIdentifier is not set, then the TSS sets T to be the Digest value set in step 2.
    - If the key's signature scheme is PKCS1v15_DER, the hash object's algorithm attribute is set to TSS_HASH_OTHER, and the AlgorithmIdentifier is set, then the TSS sets T to be a DER-encoded DigestInfo sequence using the AlgorithmIdentifier and Digest value set in steps 1 and 2.
- The TSS performs PKCS#1 signature padding on T, then runs the RSA verification algorithm on the result.

4.3.4.20.2 Tspi_Hash_Sign

Start of informative comment:

This method signs the hash data of the object with the provided signing key. Note, that while the parameter hHash implies this is a hash value, it is, in fact, just an opaque value assigned by the caller that created the TSS_HHASH object.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Hash_Sign
(
    TSS_HHASH  hHash,              // in
    TSS_HKEY   hKey,               // in
    UINT32*    pulSignatureLength,// out
    BYTE**     prgbSignature      // out
);
```

Parameters

hHash

Handle to the hash object instance which hash value should be signed.

hKey

Handle to the key object which should be used for the signature.

pulSignatureLength

On successful completion this parameter indicates the length of the signature data returned at the parameter *prgbSignature*.

prgbSignature

On successful completion this parameter points to the signature data.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_HASH_INVALID_LENGTH
TSS_E_HASH_NO_DATA
TSS_E_HASH_NO_IDENTIFIER
TSS_E_INTERNAL_ERROR

Remarks

The data to be signed must be set at the hash instance associated with *hHash* by calling *Tspi_Hash_SetHashValue()* or *Tspi_Hash_UpdateHash()*.

The Tspi_Hash_Sign method allocates a memory block for the prgbSignature data. This memory must be released utilizing the Tspi_Context_FreeMemory method.

4.3.4.20.3 Tspi_Hash_VerifySignature

Start of informative comment:

This method verifies the hash value of the hash object with a given signature

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Hash_VerifySignature
(
    TSS_HHASH  hHash,              // in
    TSS_HKEY   hKey,               // in
    UINT32     ulSignatureLength,  // in
    BYTE*      rgbSignature        // in
);
```

Parameters

hHash

Handle to the hash object instance which hash value should be verified.

hKey

Handle to the key object which should be used for the signature verification.

ulSignatureLength

This parameter indicates the length of the signature data provided at the parameter *rgbSignature*.

rgbSignature

This parameter points to the signature data.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_HASH_INVALID_LENGTH
TSS_E_HASH_NO_DATA
TSS_E_INVALID_SIGSCHEME
TSS_E_INTERNAL_ERROR

Remarks

The data to be verified must be set at the hash instance associated with *hHash* by calling *Tspi_Hash_SetHashValue()* or *Tspi_Hash_UpdateHash()*.

4.3.4.20.4 Tspi_Hash_SetHashValue

Start of informative comment:

This method sets the hash value of the hash object.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Hash_SetHashValue
(
    TSS_HHASH  hHash,              // in
    UINT32     ulHashValueLength, // in
    BYTE*      rgbHashValue        // in
);
```

Parameters

hHash

Handle to the hash object instance which hash value should be set.

ulHashValueLength

This parameter indicates the length of the hash value data provided at the parameter *rgbHashValue*.

rgbHashValue

This parameter points to the hash value data.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_HASH_INVALID_LENGTH
TSS_E_HASH_NO_DATA
TSS_E_INTERNAL_ERROR

Remarks

If the object was created with the flag TSS_HASH_OTHER then the hash algorithm identifier has to be set by calling *Tspi_SetAttribData()* to perform the sign operation.

4.3.4.20.5 Tspi_Hash_GetHashValue

Start of informative comment:

This method retunes the hash value of the hash object.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Hash_GetHashValue
(
    TSS_HHASH  hHash,              // in
    UINT32*    pulHashValueLength,// out
    BYTE**     prgbHashValue      // out
);
```

Parameters

hHash

Handle to the hash object instance which hash value should be returned.

pulHashValueLength

On successful completion this parameter indicates the length of the hash data returned at the parameter *prgbHashValue*.

prgbSignature

On successful completion this parameter points to the hash data.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_HASH_INVALID_LENGTH
TSS_E_HASH_NO_DATA
TSS_E_INTERNAL_ERROR

Remarks

The Tspi_Hash_GetHashValue method allocates a memory block for the prgbHashValue data. This memory must be released utilizing the Tspi_Context_FreeMemory method.

4.3.4.20.6 Tspi_Hash_UpdateHashValue

Start of informative comment:

This method updates the hash object with new data.

Supported Hash Algorithm:

- SHA1

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Hash_UpdateHashValue
(
    TSS_HHASH  hHash,         // in
    UINT32     ulDataLength,  // in
    BYTE*      rgbData        // in
);
```

Parameters

hHash

Handle to the hash object instance which hash value should updated.

ulDataLength

This parameter indicates the length of the data provided at the parameter *rgbData*.

rgbData

This parameter points to the data.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_HASH_INVALID_LENGTH
TSS_E_HASH_NO_DATA
TSS_E_INTERNAL_ERROR

Remarks

The object can't be modified after *Tspi_Hash_SetHashValue()*, *Tspi_Hash_GetHashValue()*, *Tspi_Hash_Sign()* or *Tspi_Hash_VerifySignature()* have been called on it. If the object was created with the flag TSS_HASH_OTHER then this method will return an error.

4.3.4.21 Tspi_Data Class Definition

4.3.4.21.1 Tspi_SetAttribUint32

Start of informative comment:

This method sets a 32-bit attribute of the data object.

End of informative comment.

Definition:

See section 4.3.2 for definition.

Parameters

See section 4.3.2 for description.

No Attributes Defined yet

Return Values

See section 4.3.2 for description.

Remarks

4.3.4.21.2 Tspi_GetAttribUint32

Start of informative comment:

This method gets a 32-bit attribute of the data object

End of informative comment.

Definition:

See section 4.3.3.1.2 for definition.

Parameters

See section 4.3.3.1.2 for description.

Defined Attributes

TSS_TSPATTRIB_ENCDATA_PCR_LONG	TSS_TSPATTRIB_ENCDATAPCRLONG_LOCALITY_ATCREATION	Get the locality value at the time the sealing was performed.
	TSS_TSPATTRIB_ENCDATAPCRLONG_LOCALITY_ATRELEASE	Get the locality value for the time the unsealing is to be performed.

Return Values

See section 4.3.3.1.2 for description.

Remarks

4.3.4.21.3 Tspi_SetAttribData

Start of informative comment:

This method sets a non 32-bit attribute of the data object. The structure and size of the attribute data depends on the attribute.

End of informative comment.

Definition:

See section 4.3.3.1.3 for definition.

Parameters

See section 4.3.3.1.3 for description.

Defined Attributes

Flag	SubFlag	Data Description
TSS_TSPATTRIB_ENCDATA_BLOB	TSS_TSPATTRIB_ENCDATABLOB_BLOB	Data blob that represents the encrypted data depending on its type (seal, bind or legacy).

Return Values

See section 4.3.3.1.3 for description.

Remarks

4.3.4.21.4 Tspi_GetAttribData

Start of informative comment:

This method gets a non 32-bit attribute of the data object. The structure and size of the attribute data depends on the attribute.

End of informative comment.

Definition:

See section 4.3.3.1.4 for definition.

Parameters

See section 4.3.3.1.4 for description.

Defined Attributes

Flag	SubFlag	Data Description
TSS_TSPATTRIB_ENCDATA_BLOB	TSS_TSPATTRIB_ENCDATABLOB_BLOB	Data blob that represents the encrypted data depending on its type (seal, bind or legacy).
TSS_TSPATTRIB_ENCDATA_PCR_LONG	TSS_TSPATTRIB_ENCDATAPCRLONG_CREATION_SELECTION	Get the bit map indicating the active PCRs at the time the sealing was performed.
	TSS_TSPATTRIB_ENCDATAPCRLONG_RELEASE_SELECTION	Get the bit map indicating the active PCRs for the time the unsealing is to be performed.
	TSS_TSPATTRIB_ENCDATAPCRLONG_DIGEST_ATCREATION	Get the composite digest of the PCRs selected at the time the sealing was performed.
	TSS_TSPATTRIB_ENCDATAPCRLONG_DIGEST_ATRELEASE	Get the composite digest of the PCRs selected for the time the unsealing is to be performed.
TSS_TSPATTRIB_ENCDATA_PCR	TSS_TSPATTRIB_ENCDATAPCR_DIGEST_ATCREATION	Composite digest value of the PCR values, at the time when the sealing was performed.
	TSS_TSPATTRIB_ENCDATAPCR _DIGEST_ATRELEASE	Composite digest value of the PCR values, at the time when the unsealing should be performed.
	TSS_TSPATTRIB_ENCDATAPCR _SELECTION	A bit map that indicates if a PCR is active or not.

Return Values

See section 4.3.3.1.4 for description.

Remarks

4.3.4.21.5 Tspi_Data_Bind

Start of informative comment:

This method encrypts a data blob in a manner that is decryptable by Tspi_Data_Unbind. The data blob is encrypted using a public key operation with the key addressed by the given encryption key object.

To bind data that is larger than the RSA public key modulus it is the responsibility of the caller to perform the blocking and subsequent combination of data.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Data_Bind
(
    TSS_HENCDATA  hEncData,      // in
    TSS_HKEY      hEncKey,       // in
    UINT32        ulDataLength,  // in
    BYTE*         rgbDataToBind  // in
);
```

Parameters

hEncData

Handle of the data object which contains the encrypted data on successful completion of the command.

hEncKey

Handle to the key object addressing the public key which is used to encrypt the data.

ulDataLength

The length (in bytes) of the *rgbDataToBind* parameter.

rgbDataToBind

Pointer to memory containing the data to be encrypted.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INVALID_ENCSCHEME
TSS_E_ENC_INVALID_LENGTH
TSS_E_ENC_NO_DATA
TSS_E_ENC_INVALID_TYPE
TSS_E_INTERNAL_ERROR

Remarks

The bound data blob is stored in the data object addressed by hEncData and can be exported from that object by GetAttribData( ). The caller gets this exported encrypted data blob according the rules of TCG in order to ensure interoperability between different TCG systems.

Use of Tspi_Data_Bind is not restricted to a key generated by the resident TPM, as it is performed entirely in software. It may be used with any appropriate public key. However the TSS may not be able to provide Tspi_Data_Unbind services to an application in this case.

Tspi_Data_Bind maximum data input sizes*

s = key size (bytes)

	TSS_ES_RSAESPKCSV15	TSS_ES_RSAESOAEP_SHA1_MGF1
TSS_KEY_TYPE_BIND	s-11-(4-1)	s-(40-2)-(4-1)
TSS_KEY_TYPE_LEGACY	s-11	s-(40-2)-(4-1)

* (4-1) bytes accounts for the size of the TPM_BOUND_DATA structure

4.3.4.21.6 Tspi_Data_Unbind

Start of informative comment:

This method takes the encrypted data blob that was exported from the data object used in the Tspi_Data_Bind command and decrypts it

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Data_Unbind
(
    TSS_HENCDATA  hEncData,             // in
    TSS_HKEY      hKey                  // in
    UINT32*       pulUnboundDataLength, // out
    BYTE**        prgbUnboundData       // out
);
```

Parameters

hEncData

Handle of the data object which addresses the encrypted data.

hKey

Handle of the key object addressing the private key which is used to decrypt the data.

pulDataLength

Receives the length (in bytes) of the *prgbUnboundData* parameter.

prgbUnboundData

On successful completion of the command, this parameter points to a buffer containing the plaintext data.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_ENC_INVALID_LENGTH
TSS_E_ENC_NO_DATA
TSS_E_ENC_INVALID_TYPE
TSS_E_INTERNAL_ERROR

Remarks

The Tspi_Data_Unbind method allocates a memory block for the decrypted data. This memory must be released utilizing the Tspi_Context_FreeMemory method.

The encrypted data blob must be imported to the object addressed by hEncData by SetAttribData( ) before calling this method.

This method operates on a block-by-block basis, and has no notion of any relation between one block and another.

If the key used in Tspi_Data_Bind is not available to the TPM, or is of the wrong type, the TPM may not be able to provide Tspi_Data_Unbind services.

4.3.4.21.7 Tspi_Data_Unseal

Start of informative comment:

This method reveals data encrypted by Tspi_Data_Seal only if it was encrypted on the same platform and the current configuration (as defined by the named PCR contents of the encrypted data blob) is the one named as qualified to decrypt it. This is internally proofed and guaranteed by the TPM.

If the Tspi_Data_Unseal operation succeeds, proof of the platform configuration that was in effect when the Tspi_Data_Seal operation was performed is returned to the caller, as well as the secret data. This proof may, or may not, be of interest. If the SEALed secret is used to authenticate the platform to a third party, a caller is normally unconcerned about the state of the platform when the secret was SEALed, and the proof may be of no interest. On the other hand, if the SEALed secret is used to authenticate a third party to the platform, a caller is normally concerned about the state of the platform when the secret was SEALed. Then the proof is of interest.

For example, if SEAL is used to store a secret key for a future configuration (probably to prove that the platform is a particular platform that is in a particular configuration), the only requirement is that that key can be used only when the platform is in that future configuration. Then there is no interest in the platform configuration when the secret key was SEALed. An example of this case is when SEAL is used to store a network authentication key.

On the other hand, suppose an OS contains an encrypted database of users allowed to log on to the platform. The OS uses a SEALED blob to store the encryption key for the user-database. However, the nature of SEAL is that *any* SW stack can SEAL a blob for any other software stack. Hence the OS can be attacked by a second OS replacing both the SEALED-blob encryption key, *and* the user database itself, allowing untrusted parties access to the services of the OS. To thwart such attacks, SEALED blobs include the *past* SW configuration. Hence, if the OS is concerned about such attacks, it may check to see whether the past configuration is one that is known to be trusted.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Data_Unseal
(
    TSS_HENCDATA  hEncData,                 // in
    TSS_HKEY      hKey,                     // in
    UINT32*       pulUnsealedDataLength,    // out
    BYTE**        prgbUnsealedData          // out
);
```

Parameters

hEncData

Handle of the data object which contains the sealed data.

hKey

Handle to the key object addressing the nonmigratable key which is used to decrypt the data.

pulUnsealedDataLength

The length (in bytes) of the *prgbUnsealedData* parameter.

prgbUnsealedData

On successful completion of the command, this parameter points to a buffer containing the plaintext data.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_ENC_INVALID_LENGTH
TSS_E_ENC_NO_DATA
TSS_E_ENC_INVALID_TYPE
TSS_E_INTERNAL_ERROR

Remarks

The Tspi_Data_Unseal method allocates a memory block for the decrypted data. This memory must be released utilizing the Tspi_Context_FreeMemory method.

The sealed data blob must be imported to the object addressed by hEncData utilizing SetAttribData( ) before calling this method.

The platform configuration status at the time when the Tspi_Data_Seal method has sealed the data can be retrieved from the object addressed by hEncData utilizing GetAttribData( )after this method was called.

This method operates on a block-by-block basis, and has no notion of any relation between one block and another.

4.3.4.21.8 Tspi_ChangeAuth

Start of informative comment:

This method changes the authorization data (secret) of the data object and assigns the data object to the policy object.

End of informative comment.

Definition:

See section 4.3.3.1.5 for definition.

Parameters

See section 4.3.3.1.5 for description.

Return Values

See section 4.3.3.1.5 for description.

Remarks

4.3.4.21.9 *Tspi_ChangeAuthAsym*

Start of informative comment:

This method changes the authorization data (secret) of the data object utilizing the asymmetric change protocol and assigns the data object to the policy object.

This method changes the authorization data of the data object ensuring that the parent of the data object does not get knowledge of the new secret.

End of informative comment.

Definition:

See section 4.3.3.1.6 for definition.

Parameters

See section 4.3.3.1.6 for description.

Return Values

See section 4.3.3.1.6 for description.

Remarks

4.3.4.21.10 Tspi_GetPolicyObject

Start of informative comment:

This method returns a policy object currently assigned to the data object

End of informative comment.

Definition:

See section 4.3.3.1.7 for definition.

Parameters

See section 4.3.3.1.7 for description.

Return Values

See section 4.3.3.1.7 for description.

Remarks

See section 4.3.3.1.7 for description.

4.3.4.22 Monotonic Counter functions

Start of informative comment:

The monotonic counter is likely to be used to tag data as corresponding to a particular counter value. It is important that the counter itself not be incremented except as authorized by the owner of the platform, as otherwise it could really mess up software that counts on knowing every time the counter is incremented. Therefore a number of the commands that speak directly to the counter will have to have restricted access. Implementations on how to control this restricted access will no doubt vary from OS to OS. Specifically the create, increment, and release counter arguments need to be controlled.

End of informative comment.

4.3.4.22.1 Tspi_TPM_ReadCurrentCounter

Start of informative comment:

This method reads the current value of the current active counter register.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_ReadCounter
(
  TSS_HTPM          hTPM,           // in
  UINT32 *          counterValue    // out
);
```

Parameters

hTPM

Handle of the TPM object.

counterValue

Current value of the counter.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

4.3.4.23 Time Stamping Function Definitions

4.3.4.23.1 Tspi_TPM_ReadCurrentTicks

Start of informative comment:

This method reads the current tick out of the TPM.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_ReadCurrentTicks
(
   TSS_HTPM            hTPM          //in
   TPM_CURRENT_TICKS*  tickCount     // out
);
```

Parameters

hTPM

The handle of the TPM object

tickCount

After successful completion this parameter, holds the current TPM tick counter value

Return Values

TSS_SUCCESS
TSS_E_INVALID_TPM_HANDLE
TSS_E_INTERNAL_ERROR

Remarks

The Tspi_TPM_ReadCurrentTick method reads the current value of the tick counter in the TPM.

4.3.4.23.2 Tspi_Hash_TickStampBlob

Start of informative comment:

This method is similar to a time stamp: it associates a tickvalue with a blob, indicating that the blob existed at some point earlier than the time corresponding to the tickvalue.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_Hash_TickStampBlob
(
    TSS_HHASH         hHash,            // in
    TCS_KEY_HANDLE    hIdentKey,        // in
    TSS_VALIDATION* pValidationData   //in
);
```

Parameters

hHash

handle of 20 byte hash of blob to be tickstamped

hIdentKey

Identity key used to perform tickstamp

pValidationData

The data necessary to validate the signature

Return Values

TSS_SUCCESS
TSS_E_INVALID_TPM_HANDLE
TSS_E_INVALID_KEY_HANDLE
TSS_E_INTERNAL_ERROR

Remarks

Tspi_TickStampBlob can be used to link an external timestamp to the tick / ticknonce inside the TPM

The hHash object's algorithm must be a TSS_HASH_SHA1 because the underlying TPM function only accepts 20-byte values to sign.

On entry, the caller should provide the antiReplay nonce in pValidationData->ExternalData. On successful completion pValidationData->ValidationDataLength and pValidationData->ValidationData will hold the signature from the TPM and pValidationData->DataLength and pValidationData->Data will hold the data that was signed, specifically the buffer will be the byte sequence encoding a TPM_SIGN_INFO, with TPM_SIGN_INFO.data set to (hash value from hHash || TPM_CURRENT_TICKS).

4.3.4.24 DIR Commands

4.3.4.24.1 Tspi_TPM_DirWrite

Start of informative comment:

This method writes a Data Integrity Register. It is deprecated, as one can also do this by using the NV RAM commands.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_DirWrite
(
    TSS_HTPM    hTPM,             // in
    UINT32      ulDirIndex,       // in
    UINT32      ulDirDataLength,  // in
    BYTE*       rgbDirData        // in
);
```

Parameters

hTPM

Handle of the TPM object

ulDirIndex

Index of the DIR to write.

ulDirDataLength

The length (in bytes) of the *rgbDirData* parameter

rgbDirData

Pointer to memory containing the data to be written to the DIR.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

4.3.4.24.2 Tspi_TPM_DirRead

Start of informative comment:

This method reads a Data Integrity Register.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_DirRead
(
    TSS_HTPM    hTPM,               // in
    UINT32      ulDirIndex,         // in
    UINT32*     pulDirDataLength,   // out
    BYTE**      prgbDirData         // out
);
```

Parameters

hTPM

Handle of the TPM object

ulDirIndex

Index of the DIR to read.

pulDirDataLength

Receives the length (in bytes) of the *prgbDirData* parameter.

prgbDirData

Receives a pointer to the memory block containing the the DIR data.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The Tspi_TPM_DirRead method allocates a memory block for the prgbDirData data. This memory must be released utilizing the Tspi_Context_FreeMemory method.

4.3.4.25 Tspi_NV Class Definition

The Tspi_NV class is used to store the attributes of a region of Non volatile RAM inside the TPM, for use when defining, releasing, reading or writing such a region. This class establishes the size of the data space, the index, the various authorizations required to either read or write that area. Those authorizations can be based on PCR values or authorization data, but not locality. The various attributes of the class are used to establish what is requested before Tspi_NV_DefineSpace is called (similar to the way a key is created).

4.3.4.25.1 Tspi_SetAttribUint32

Start of informative comment:

This method sets a 32-bit attribute of the NV Storage object.

End of informative comment.

See section 4.3.2 for definition.

Parameters

See section 4.3.2 for description.

Defined Attributes

Flag	SubFlag	Attribute	Description
TSS_TSPATTRIB_NV_INDEX	0	UINT32	Index of the NV Storage area associated with this object
TSS_TSPATTRIB_NV_PERMISSIONS	0	UINT32	The value of the permissions
TSS_TSPATTRIB_NV_DATASIZE	0	UINT32	Size of the defined NV storage area.

Return Values

See section 4.3.2 for description.

Remarks

4.3.4.25.2 Tspi_GetAttribUint32

Start of informative comment:

This method gets a 32-bit attribute of the NV Storage object

End of informative comment.

See section 4.3.3.1.2 for definition.

Parameters

See section 4.3.3.1.2 for description.

Defined Attributes

Flag	SubFlag	Attribute	Description
TSS_TSPATTRIB_NV_INDEX	0	UINT32	Index of the NV Storage area associated with this object
TSS_TSPATTRIB_NV_PERMISSIONS	0	UINT32	The value of the permissions
TSS_TSPATTRIB_NV_DATASIZE	0	UINT32	Size of the defined NV storage area.
TSS_TSPATTRIB_NV_STATE	TSS_TSPATTRIB_NVSTATE_READSTCLEAR	Boolean	Set to FALSE on each TPM_Startup(ST_Clear) and set to TRUE after a ReadValuexxx with datasize of 0
	TSS_TSPATTRIB_NVSTATE_WRITESTCLEAR	Boolean	Set to FALSE on each TPM_Startup(ST_CLEAR) and set to TRUE after a WriteValuexxx with a datasize of 0
	TSS_TSPATTRIB_NVSTATE_WRITEDEFINE	Boolean	Set to FALSE after TPM_NV_DefineSpace and set to TRUE after a successful WriteValue with a datasize of 0
TSS_TSPATTRIB_NV_PCR	TSS_TSPATTRIB_NVPCR_READLOCALITYATRELEASE	BYTE	The locality mask for the PCR read restrictions of the NV space.
	TSS_TSPATTRIB_NVPCR_WRITELOCALITYATRELEASE	BYTE	The locality mask for the PCR write restrictions of the NV space.

Return Values

See section 4.3.3.1.2 for description.

Remarks

4.3.4.25.3 Tspi_SetAttribData

Start of informative comment:

This method sets a non 32-bit attribute of the NV Storage object. The structure and size of the attribute data depends on the attribute.

End of informative comment.

Definition:

See section 4.3.3.1.3 for description.

Parameters

See section 4.3.3.1.3 for description.

Defined Attributes

No attributes defined for this section.

Return Values

See section 4.3.3.1.3 for description.

Remarks

4.3.4.25.4 Tspi_GetAttribData

Start of informative comment:

This method gets a non 32-bit attribute of the NV Storage object. The structure and size of the attribute data depends on the attribute.

End of informative comment.

Definition:

See section 4.3.3.1.4 for definition.

Parameters

See section 4.3.3.1.4 for description.

Defined Attributes

Flag	SubFlag	Description
TSS_TSPATTRIB_NV_PCR	TSS_TSPATTRIB_NVPCR_READPCRSELECTION	The PCR selection mask for the PCR read restrictions of the NV space.
	TSS_TSPATTRIB_NVPCR_READDIGESTATRELEASE	The digestAtRelease for the PCR read restrictions of the NV space
	TSS_TSPATTRIB_NVPCR_WRITEPCRSELECTION	The PCR selection mask for the PCR write restrictions of the NV space
	TSS_TSPATTRIB_NVPCR_WRITEDIGESTATRELEASE	The digestAtRelease for the PCR write restrictions of the NV space

Return Values

See section 4.3.3.1.4 for description.

Remarks

4.3.4.25.5 Tspi_NV_DefineSpace

Start of informative comment:

This function establishes the space necessary for the indicated NVStore .

End of informative comment.

Definition:

```
TSS_RESULT Tspi_NV_DefineSpace
(
    TSS_HNVSTORE  hNVStore,           // in
    TSS_HPCRS     hReadPcrComposite,  // in, may be NULL
    TSS_HPCRS     hWritePcrComposite  // in, may be NULL
);
```

Parameters

hNVStore

Handle of the NV storage object to define.

hReadPcrComposite

Handle to an object of the type *Tspi_PcrComposite*. If the is value is NULL, no PCR values are associated with reading from the NV Space. If the value of the handle contains a Tspi_PcrComposite object, the newly created NV Storage area will require the PCR values described with this object for a read from this object to succeed.

hWritePcrComposite

Handle to an object of the type *Tspi_PcrComposite*. If the is value is NULL, no PCR values are associated with writing to the NV Space. If the value of the handle contains a Tspi_PcrComposite object, the newly created NV Storage area will require the PCR values described with this object for a write to this object to succeed.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR
TSS_E_NV_AREA_NOT_EXIST
TPM_BAD_INDEX
TPM_AUTH_CONFLICT
TPM_AUTHFAIL
TPM_OWNERSET
TPM_BAD_DATASIZE
TPM_MAXNVWRITE
TPM_INVALID_STRUCTURE
TPM_NOWRITE

Remarks

This function is used to define NV Storage space. Note, this will call the same TPM function as Tspi_NV_ReleaseSpace because the TPM function is overloaded with both the definition and releasing of NV storage space. Because of the nature of the TSPI it will very easy in some circumstance for a program error in the allocation of NV storage, to call this function twice with the same object thereby deleting it. This function, therefore, MUST first check to see if the NV Storage area reference by hNVStore is already defined. If it is, rather than call the underlying TPM operation, this function MUST fail with the error TSS_E_NV_AREA_EXIST.

If the TSS_TSPATTRIB_NV_PERMISSTIONS attribute indicates that the NV space requires NV-specific auth (i.e. if either the TPM_NV_PER_AUTHREAD, or TPM_NV_PER_AUTHWRITE bit is set), then the auth value will come form the NV object's usage policy and the TSP should return an error if that secret is not available. If none of these bits are set, then the TSP should not access the NV object's usage policy during this operation; in particular, no popups should be shown and it is not an error if the secret has expired or the policy has been set to TSS_SECRET_MODE_NONE. Note that if the application has not explicitly assigned a usage policy to the NV object, then the context's default policy will be used.

Besides the authorization data for the NV-Storage space, the operation requires in addition Owner authorization. The handling for this secret data is performed by the policy object at the TPM object from the current TSP-Context. This policy object can be retrieved directly from the TPM object by calling Tspi_GetPolicyObject.

4.3.4.25.6 Tspi_NV_ReleaseSpace

Start of informative comment:

This function releases the space for the indicated NVStore

End of informative comment.

Definition:

```
TSS_RESULT Tspi_NV_ReleaseSpace
(
  TSS_HNVSTORE  hNVStore   // in
);
```

Parameters

hNVStore

Handle of the NV storage object to release.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR
TSS_E_NV_AREA_NOT_EXIST
TPM_AREA_LOCKED

Remarks

This function is used to release NV Storage space. Note, this will call the same TPM function as Tspi_NV_DefineSpace. See remark in Tspi_NV_DefineSpace for explanation. This function MUST first check to see if the NV Storage area reference by hNVStore is already defined. If it is not, rather than call the underlying TPM operation, this function MUST fail with the error TSS_E_NV_AREA_NOT_EXIST.

This command requires Owner authorization. The handling for this secret data is performed by the policy object at the TPM object from the current TSP-Context. This policy object can be retrieved directly form the TPM object by calling Tspi_GetPolicyObject.

4.3.4.25.7 Tspi_NV_WriteValue

Start of informative comment:

This command writes the value to a defined area.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_NV_WriteValue
(
    TSS_HNVSTORE  hNVStore,        // in
    UINT32        offset,          // in
    UINT32        ulDataLength,    // in
    BYTE*         rgbDataToWrite   // in
);
```

Parameters

hNVStore

Handle of the NV storage object in which to perform this write.

offset

Offset within the NV area to begin writing.

ulDataLength

The Length (in bytes) of the rgbDataToWrite parameter. This is also the length of the data area to write to the object.

rgbDataToWrite

Pointer to memory containing the data to be written.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR
TPM_BAD_INDEX
TPM_MAXNVWRITE
TPM_AUTH_CONFLICT
TPM_AUTHFAIL
TPM_AREA_LOCKED
TPM_BAD_LOCALITY
TPM_BAD_PRESENCE
TPM_DISABLED_CMD
TPM_NOSPACE
TPM_NOT_FULLWRITE
TPM_WRONGPCRVALUE

Remarks

The policy object assigned to this NV object will control the authorization of the command. If authData is available from the policy, it will be used to authorize the operation. If the policy is set to SECRETMODENONE, then an unauthenticated write will be performed.

4.3.4.25.8 Tspi_NV_ReadValue

Start of informative comment:

This command reads the value from a defined area.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_NV_ReadValue
(
    TSS_HNVSTORE  hNVStore,      // in
    UINT32        offset,        // in
    UINT32*       pulDataLength,  // in, out
    BYTE**        prgbDataRead  // out
);
```

Parameters

hNVStore

Handle of the NV storage object from which to perform this read.

offset

Offset within the NV area to begin reading.

ulDataLength

The Length (in bytes) of the rgbDataRead parameter.

rgbDataRead

Pointer to memory containing the data that was read from the object.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR
TPM_BAD_INDEX
TPM_AUTH_CONFLICT
TPM_AUTHFAIL
TPM_BAD_LOCALITY
TPM_BAD_PRESENCE
TPM_DISABLED_CMD
TPM_NOSPACE
TPM_WRONGPCRVALUE

Remarks

The policy object assigned to this NV object will control the authorization of the command. If authData is available from the policy, it will be used to authorize the operation. If the policy is set to SECRETMODENONE, then an unauthenticated read will be performed.

4.3.4.26 GPIO

TPM_NV_INDEX_GPIO_xx	0x00011600
PC Client GPIO_Express_00	00
PC Client reserved	01-7F
PC Client VendorSpecified	80-FF
PC Client GPIO_Express_00_bit	0x01

GPIO Usage

Start of informative comment:

General purpose I/O (GPIO) provides an interface to and from between the TPM's command interface and an external device. Access to the TPM's GPIO is done using the TPM NV RAM mechanism. From a software perspective, there is no difference between writing to and reading from a GPIO than from any other NV RAM area. For this reason, there is no specific set of functions dedicated for GPIO. To use the TPM's GPIO features, applications will simply call the appropriate NV RAM functions using helper definitions.

The TPM Main Specification designates a range of NV RAM indices that are to be used for GPIO by all classes of platforms. Within each platform class (e.g., PC Client) there exits a set of defined mappings between the NV RAM indices and one or more physical GPIO pins that perform a specific function for the class of platform. Because there is a single area dedicated for all classes of platforms, it is important that applications understand the class of the platform as well as the particular function and meaning of the bit for the GPIO before accessing the TPM's GPIO area. Further note that even within a platform class the function of a GPIO may not be relevant for all instantiations of that platform's class. This information is documented in the platform-specific specification for the target class of platforms.

For example: The PC Client designates the GPIO named PC_Client_GPIO_Express_00 as a signal targeted at the PCI Express root complex in some implementations. First the application would need to determine the nature of the GPIO by determining the platform class (e.g., referencing the platform's credentials or a derivative of it). Then, if the GPIO is optional, again referring the application knowledge of the instantiation of the platform class (i.e., is the PC_Client_GPIO_Express_00 even needed on this platform being referenced. Sometime during the setup of the platform (should be before access by a user) the TPM Owner calls Tspi_NV_DefineSpace to allow access to the area designated for the GPIO. After that, application may call Tspi_NV_WriteValue to send or Tspi_NV_ReadValue or receive a signal from this particular pin.

Because GPIO can be used for security or privacy functions, it must not be open, by default, for public access. For this reason it is required that the NV RAM area that is mapped to the GPIO be "defined" like any other NV RAM area prior to allowing its use. When defining this area, the TPM Owner may elect to assign rights per the normal NV RAM permissions as defined in section NV Constants. Note that this is done only once until the TPM Owner is removed at which time the area returns to undefined and this area must again be defined before use. The reason for this is the new TPM Owner may have different security and privacy requirements for this GPIO.

Note to implementers:

Careful examination of the NV RAM Permissions demonstrates that if none of the read or writes permission bits (1-2 and 16-18) are set, this area is set to public reads and writes.

End of informative comment.

4.3.4.27 Delegation TSPI functions

These functions do not take control away from an owner/user, but rather are used to give control to a particular entity or piece of executing software in a way that can be revoked. This differs from giving the password to the entity of the executing software (which action is hard to revoke).

4.3.4.27.1 Tspi_SetAttribUint32

Start of informative comment:

This method sets a 32-bit attribute of the Delegate Family object.

End of informative comment.

Definition:

See section 4.3.2 for definition.

Parameters

See section 4.3.2 for description.

Defined Attributes

Flag	SubFlag	Attribute	Description
TSS_DELTABLE_FAMILY_STATE	TSS_DELTABLE_FAMILY_LOCKED	Boolean	Used to prevent further management of the delegation family tables until an Owner is installed, or until the Owner is removed from the TPM. (Note that the Physical Presence command TPM_ForceClear always enables further management, even if TPM_ForceClear is used when no Owner is installed.) If TRUE some delegation commands are locked and return TPM_DELEGATE_LOCK. If FALSE delegation commands are available. Default is FALSE
	TSS_DELTABLE_FAMILY_ENABLED	Boolean	Enable/disable use of the family and all the rows of the delegate table belonging to that family. When TRUE the delegation table is enabled. The default vaue is FALSE.

Return Values

TSS_SUCCESS
TSS_E_TSP_AUTHFAIL
TPM_DELEGATE_FAMILY
TPM_DELEGATE_LOCK
TPM_MAXNVWRITES
TPM_BADINDEX
TSS_E_INVALID_HANDLE

Remarks

4.3.4.27.2 Tspi_GetAttribUint32

Start of informative comment:

This method gets a 32-bit attribute of the Delegate Family object

End of informative comment.

Definition:

See section 4.3.2 for definition.

Parameters

See section 4.3.2 for description.

Defined Attributes

Same as Tspi_SetAttribUint32 for this class with the addition of the following:

Flag	SubFlag	Attribute	Description
TSS_DELATTRIB_FAMILY_INFO	TSS_DELATTRIB_FAMILY_LABEL	BYTE	A sequence number that software can map to a string of bytes that can be displayed or used by the applications. This MUST not contain sensitive information.
	TSS_DELATTRIB_FAMILY_VERCOUNT	UINT32	Verification count value used to identify when a row in the delegate table was last verified.
	TSS_DELTABLE_FAMILY_ID	UINT32	Family ID assigned by the TPM.
TSS_DELTABLE_FAMILY_STATE	TSS_DELTABLE_FAMILY_LOCKED	Boolean	If TRUE some delegation commands are locked and return TPM_DELEGATE_LOCK. If FALSE delegation commands are available. Default is FALSE
	TSS_DELTABLE_FAMILY_ENABLED	Boolean	If TRUE the delegation family table is enabled. The default vaue is FALSE.

Return Values

See section 4.3.2 for return values.

Remarks

Under the covers this calls the Tcsip_TPM_Deleate_Manage function to change these values.

4.3.4.27.3 Tspi_SetAttribData

Start of informative comment:

This method sets a non 32-bit attribute of the Delegate Family object. The structure and size of the attribute data depends on the attribute.

End of informative comment.

Definition:

See section 4.3.3.1.3 for definition.

Parameters

See section 4.3.3.1.3 for description.

Defined Attributes

No attributes defined for this section.

Return Values

See section 4.3.3.1.3 for description.

Remarks

4.3.4.27.4 Tspi_GetAttribData

Start of informative comment:

This method gets a non 32-bit attribute of the Delegate Family object. The structure and size of the attribute data depends on the attribute.

End of informative comment.

Definition:

See section 4.3.3.1.4 for definition.

Parameters

See section 4.3.3.1.4 for description.

Defined Attributes

No attributes defined for this section.

Return Values

See section 4.3.3.1.4 for description.

Remarks

4.3.4.27.5 Tspi_TPM_Delegate_AddFamily

Start of informative comment:

Tspi_TPM_Delegate_AddFamily command is authorized either by the TPM Owner or by physical presence. If no Owner is installed, Tspi_TPM_Delegate_AddFamily requires no privilege to execute. (uses TPM_Delegate_Manage with TPM_FAMILY_CREATE)

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_Delegate_AddFamily
(
  TSS_HTPM        hTPM,         // in, must not be NULL
  BYTE            bLabel,       // in
  TSS_HDELFAMILY* phFamily      // out
);
```

Parameters

hTPM

Handle of the TPM object with an authorization session handle attached. May be NULL if there is no owner present.

bLabel

Family table entry label. A sequence number that software can map to a string of bytes that can be displayed or used by the applications. This MUST not contain sensitive information.

phFamily

Receives the handle to the assigned delegation family object.

Return Values

TSS_SUCCESS
TSS_E_TSP_AUTHFAIL
TPM_MAXNVWRITES
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER

Remarks

This command uses the underlying TCS command TCS_Delegate_Manage, with opcode TPM_FAMILY_CREATE. It is used to make a define delegation family active. hTPM must not be NULL. If hTPM's usage policy is set to secret mode TSS_SECRET_MODE_NONE then the operation will be performed without authorization. This will only succeed if there is no owner installed. If the policy is set to any other secret mode the operation will proceed as an owner authorized command.

4.3.4.27.6 Tspi_TPM_Delegate_GetFamily

Start of informative comment:

Tspi_TPM_Delegate_GetFamily returns a handle to a previously created delegation family. (uses TPM_Delegate_ReadTable).

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_Delegate_GetFamily
(
    TSS_HTPM          hTPM,         // in, must not NULL
    UINT32            ulFamilyID,   // in
    TSS_HDELFAMILY*   phFamily      // out
);
```

Parameters

hTPM

Handle of the TPM object. No owner authorization is necessary.

ulFamilyID

The delegation family ID.

phFamily

Receives the handle to the existing delegation family object.

Return Values

TSS_SUCCESS
TSS_E_TSP_AUTHFAIL
TPM_BADINDEX
TPM_MAXNVWRITES
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER

Remarks

hTPM must not be NULL. If hTPM's usage policy is set to secret mode TSS_SECRET_MODE_NONE then the operation will be performed without authorization. This will only succeed if there is no owner installed. If the policy is set to any other secret mode the operation will proceed as an owner authorized command.

4.3.4.27.7 Tspi_TPM_Delegate_InvalidateFamily

Start of informative comment:

Tspi_TPM_Delegate_InvalidateFamily command is authorized either by the TPM Owner or by physical presence. If no Owner is installed, Tspi_TPM_Delegate_InvalidateFamily requires no privilege to execute.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_Delegate_InvalidateFamily
(
    TSS_HTPM         hTPM,         // in, must not be NULL
    TSS_HDELFAMILY   hFamily       // in
);
```

Parameters

hTPM

Handle of the TPM object with the owner authorization policy attached.

hFamily

Handle to the existing delegation family object to invalidate.

Return Values

TSS_SUCCESS
TSS_E_TSP_AUTHFAIL
TPM_MAXNVWRITES
TPM_DELEGATE_LOCK
TPM_SELFTEST_FAILED
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER

Remarks

This command uses the underlying TCS command TCS_Delegate_Manage, with opcode TPM_FAMILY_INVALIDATE. It is used to make a define delegation family active. hTPM must not be NULL. If hTPM's usage policy is set to secret mode TSS_SECRET_MODE_NONE then the operation will be performed without authorization. This will only succeed if there is no owner installed. If the policy is set to any other secret mode the operation will proceed as an owner authorized command.

4.3.4.27.8 Tspi_TPM_Delegate_CreateDelegation

Start of informative comment:

This command creates a key usage or an owner delegation policy.

Tspi_TPM_Delegate_CreateDelegation includes the ability to void all existing owner delegations (by selection incrementVerificationCount flag. This ensures that the new delegation will be the only delegation that can operate at Owner privilege in this family. This new delegation could be used to enable a security monitor (a local separate entity, or remote separate entity, or local host entity) to reinitialize a family and perhaps perform external verification of delegation settings.

If the verification count is incremented and the new delegation does not delegate any privileges (to any ordinals) at all, or uses an authorization value that is then discarded, this family's delegations are all void and delegation must be managed using actual Owner authorization.

Once a delegation is associated with a policy object, none of its properties can be changed without invalidating the delegation blob (and if the delegation blob is referenced by an index in the TPM's delegation table, it is not even possible for the TSS to modify the delegation blob). As a result, if a policy object has a fixed delegation blob, using Tspi_SetAttribData or Tspi_SetAttribUint32 to change any of the TSS_TSPATTRIB_POLICY_DELEGATION_INFO attributes should result in the TSS returning an error with the following exceptions:

- TSS_TSPATTRIB_POLDEL_TYPE can be used to set TSS_DELEGATION_NONE, disabling the use of delegation for this policy object and clearing all delegation attributes.
- TSS_TSPATTRIB_POLDEL_KEYBLOB inserts a new key delegation blob, overwriting any existing delegation information.
- TSS_TSPATTRIB_POLDEL_OWNERBLOB inserts a new owner delegation blob, overwriting any existing delegation information.
- TSS_TSPATTRIB_POLDEL_INDEX associated the policy object with a delegation blob stored in the TPM, and the delegation properties and attributes of the policy are based on that entry form the delegation table.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_Delegate_CreateDelegation
(
    TSS_HOBJECT      hObject,        // in
    BYTE             bLabel,         // in
    UINT32           ulFlags,        // in
    TSS_HPCRS        hPcr,           // in, may be NULL
    TSS_HDELFAMILY   hFamily,        // in
    TSS_HPOLICY      hDelegation     // in
);
```

Parameters

hObject

Handle of an object. Could be hTPM with owner authorization policy attached or a hKey, loaded key with key usage authorization policy attached. Type of this object determines the type of the delegation that will be created: owner or key.

bLabel

Delegation table entry label. A sequence number that software can map to a string of bytes that can be displayed or used by the applications. This MUST not contain sensitive information.

ulFlags

Flags. Currently supported: incrementVerification for owner delegations only.

hPcr

Handle to the PCRobject, which must contain PCR_info (not PCR_infoLong). . If the value of the handle doesn't equal to NULL, the newly created delegation will be bound to the PCR values described with this object.

hFamily

Handle to an existing delegation family object.

hDelegation

A handle to the previously created policy object. This object must have the permissions attribute set using SetAttribUINT32() command. This object must also have the delegated secret value set using Tspi_Policy_SetSecret() command.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR
TSS_E_NV_AREA_EXIST
TPM_AUTHFAIL

Remarks

If *hPcr* is non-NULL, it must use a TCPA_PCR_INFO_SHORT structure.

On return the hDelegation with contain a complete owner or key delegation blob which could be retrieved using Tspi_GetAttribData command. The resulting hDelegation may also be assigned to the TSS_HKEY or TSS_HTPM object using Tspi_Policy_AssignObject command.

Note: internally this uses either the TCS_Delegate_CreateOwnerDelegation or the TCS_Delegate_CreateKeyDelegation.

- hTPM must not be NULL. If hObject is a TPM object and its policy has secret mode TSS_SECRET_MODE_NONE, the delegation blob will be created for a TPM without an owner (note this means the authorization secret will appear unencrypted in the delegation blob). If the policy is set to any other secret mode then the operation will proceed as an owner authorized command.

4.3.4.27.9 Tspi_TPM_Delegate_CacheOwnerDelegation

Start of informative comment:

This command loads a delegate table row blob into a non-volatile delegate table in the TPM. Tspi_TPM_Delegate_CacheOwnerDelegation can be used during manufacturing or on first boot (when no Owner is installed), or after an Owner is installed. If an Owner is installed, Tspi_TPM_Delegate_CacheOwnerDelegation requires Owner authorization.

Burn-out of TPM non-volatile storage by inappropriate use is mitigated by the TPM's normal limits on NV-writes in the absence of an Owner. Tables can be locked after loading using Tspi_SetAttiUint32 command with TSS_DELTABLE_FAMILY_LOCKED attribute, to prevent subsequent tampering.

The calling application may choose to overwrite any previously stored delegation table entries by selecting the appropriate flags.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_Delegate_CacheOwnerDelegation
(
  TSS_HTPM        hTPM,         // in, must not be NULL
  TSS_HPOLICY     hDelegation,  // in
  UINT32          ulIndex,      // in
  UINT32          ulFlags       // in
);
```

Parameters

hTPM

Handle of a TPM object with owner authorization policy attached. Could be NULL if no owner is installed.

hDelegation

A handle to the previously created policy object with delegation attributes set.

ulIndex

The index of the delegate row to be written.

ulFlags

Flags. Currently supported: OverwriteExisting

.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER

Remarks

hTPM must not be NULL. If hTPM's usage policy is set to secret mode TSS_SECRET_MODE_NONE then the operation will be performed without authorization. This will only succeed if there is no owner installed. If the policy is set to any other secret mode the operation will proceed as an owner authorized command.

4.3.4.27.10 Tspi_TPM_Delegate_UpdateVerificationCount

Start of informative comment:

Tspi_TPM_Delegate_UpdateVerificationCount sets the verificationCount in an entity (a blob or a delegation row) to the current family value, in order that the delegations represented by that entity will continue to be accepted by the TPM.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_Delegate_UpdateVerificationCount
(
    TSS_HTPM        hTPM,         // in
    TSS_HPOLICY     hDelegation   // in
);
```

Parameters

hTPM

Handle of a TPM object with owner authorization policy attached

hDelegation

A handle to the previously created policy object with delegation attributes set.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TPM_AUTHFAIL

Remarks

4.3.4.27.11 Tspi_TPM_Delegate_VerifyDelegation

Start of informative comment:

Tspi_TPM_Delegate_VerifyDelegation verifies a delegate blob and returns success or failure, depending on whether the blob is currently valid.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_Delegate_VerifyDelegation
(
    TSS_HPOLICY          hDelegation   // in
);
```

Parameters

hDelegation

A handle to the previously created policy object with delegation attributes set.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_FAILURE

Remarks

4.3.4.27.12 Tspi_TPM_Delegate_ReadTables

Start of informative comment:

This command is used to read from the TPM the public contents of the family and delegate tables that are stored on the TPM. Such data is required during external verification of tables.

There are no restrictions on the execution of this command; anyone can read this information regardless of the state of the PCRs, regardless of whether they know any specific authorization value and regardless of whether or not the enable and admin bits are set one way or the other.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_Delegate_CreateDelegation
(
TSS_HCONTEXT                  hContext,             // in
UINT32*                       pulFamilyTableSize,   // out
TSS_FAMILY_TABLE_ENTRY**      ppFamilyTable,        // out
UINT32*                       pulDelegateTableSize, // out
TSS_DELEGATION_TABLE_ENTRY**  ppDelegateTable       // out
);
```

Parameters

hDelegation

A handle to the previously created policy object with delegation attributes set.

pulFamilyTableSize

Receives the length (number of array entries) of the *ppFamilyTable* parameter.

ppFamilyTable

On successful completion of the command, this parameter points to a buffer containing the actual Family Table data.

pulDelegateTableSize

Receives the length (number of array entries) of the *ppDelegateTable* parameter.

ppDelegateTable

On successful completion of the command, this parameter points to a buffer containing the actual Delegate Table data.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR
TSS_E_NV_AREA_EXIST
TPM_AUTHFAIL

Remarks

4.3.4.28 DAA Commands

4.3.4.28.1 Introduction

This document specifies the TSS specific functions of Direct Anonymous Attestation scheme (DAA).

"..Because the TPM has limited resources, a requirement for direct anonymous attestation was that the operations carried out on the TPM be minimal and, if possible, be outsourced to TSS. Of course, security must be maintained, i.e., a (corrupted) host/TSS should not be able to authenticate without interacting with the TPM. However, privacy/anonymity needs only be guaranteed if the host/TSS is not corrupted: as the host controls all the communication of the TPM to the outside, a corrupted host/TSS can always break privacy/anonymity by just adding some identifier to each message sent by the TPM. In fact, our scheme satisfies an even stronger requirement: when the corrupted software is removed, the privacy/anonymity properties are restored."[9]

Compared to other TSS functions, the DAA functions will do some computations for reasons explained above.

Besides the TPM and the Host, DAA interacts with a DAA Issuer, DAA Verifier, DAA Mediator and DAA Anonymity Revocation Authority which do not need a TPM themselves. Their behavior is specified in this document as optional functions.

4.3.4.28.2 Components

The basic DAA scheme knows 4 parties: The TPM, its Platform TSS, the DAA Issuer and the DAA Verifier. Depending on the setup the TPM Platform will have a TCG Application. The TCG application interfaces the DAA Issuer and DAA Verifier with its TSS. A DAA Issuer issues DAA Credentials to a platform by means of the DAA join protocol. The issued DAA Credential can be shown to a DAA Verifier for verification by means of the DAA sign protocol. The DAA Anonymity Revocation Authority is a Trusted Third Party (TTP), which can revoke[1] the anonymity or pseudonym of a DAA Credential show depending on how the credential was shown. There is no need for the DAA Anonymity Revocation Authority to be (directly) involved in the DAA Sign protocol.

[1] Policies will determine under what circumstances revocation will take place. Use cases to be defined by IWG.

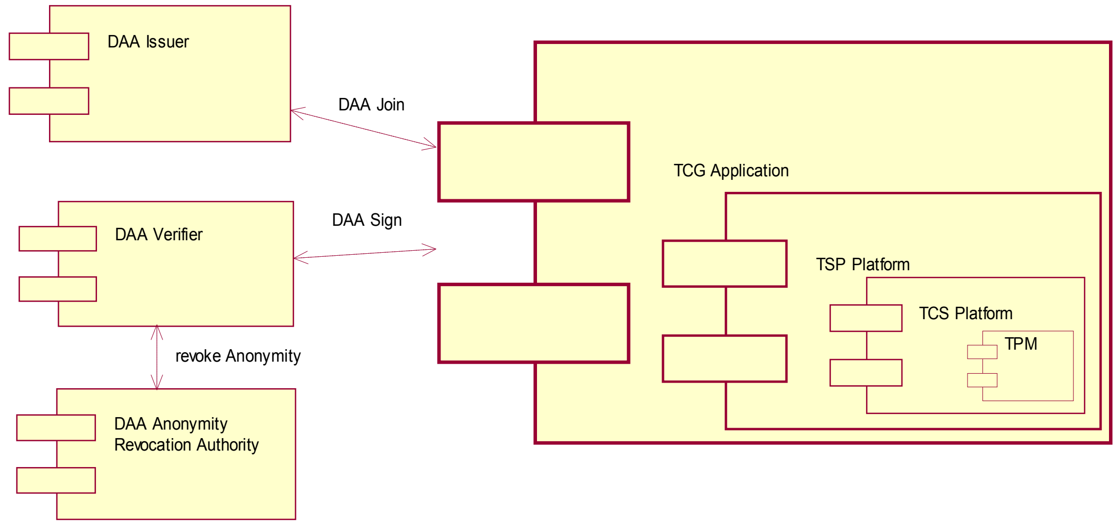


Figure 8 : UML component diagram of DAA components and entities

4.3.4.29 DAA Protocols

DAA defines two protocols, DAA Join and DAA Sign.

4.3.4.29.1 DAA Join

In the DAA join protocol a DAA Issuer interacts with a TCG platform to issue a DAA Credential for that platform. The credential can contain attributes visible or non visible to the DAA Issuer. The latter allows issuing credentials that can be used only once or binding a credential to another one while still guaranteeing the anonymity feature of DAA.

As an initial setup it is assumed that the TCG platform knows the DAA Issuer public key and has verified its authenticity.

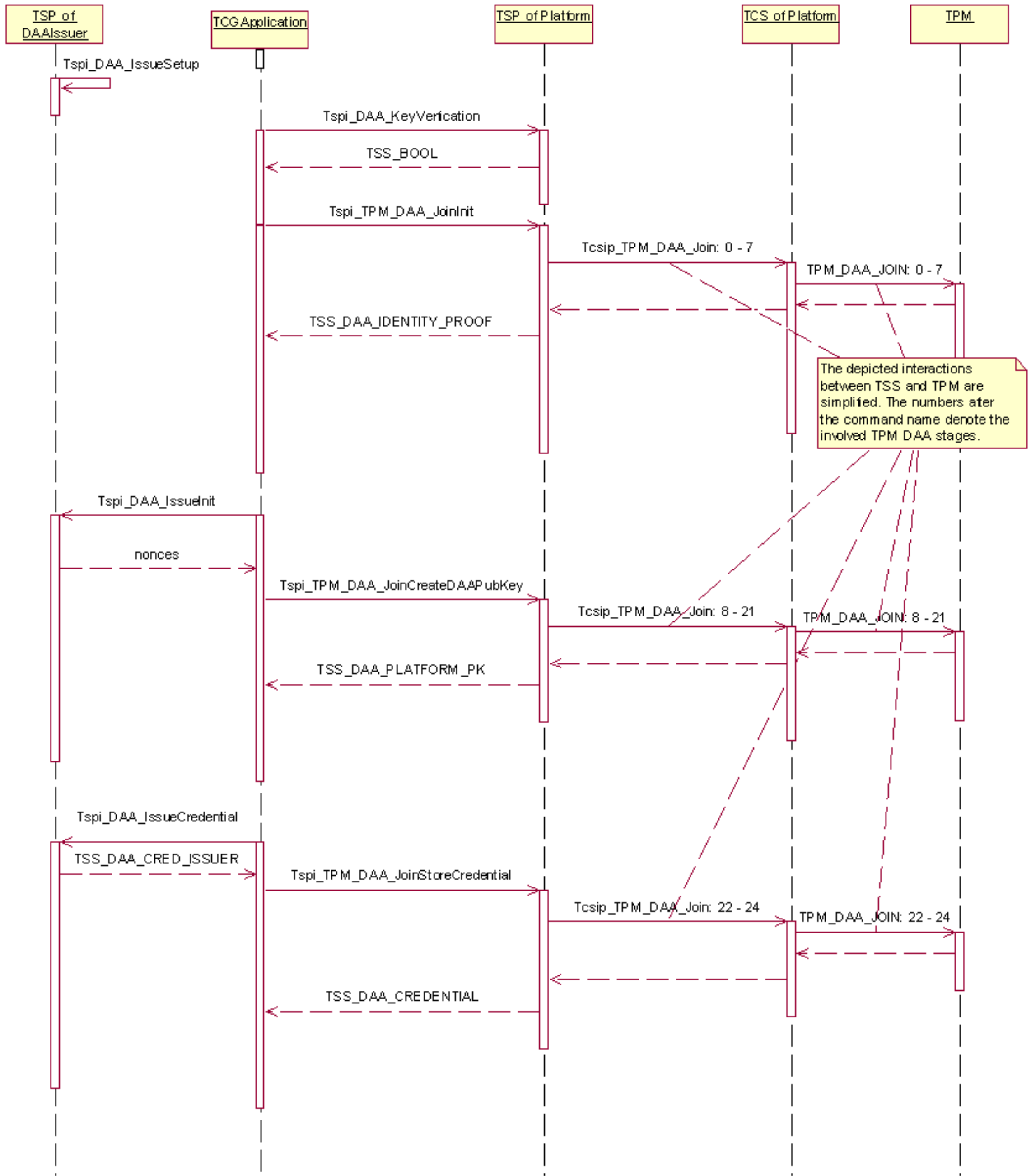


Figure 9 UML Sequence diagram of DAA Join

4.3.4.29.2 Limitation

For security reasons (non-concurrent zero knowledge proof of TPM), the DAA Join protocol can not be run arbitrarily interleaved with different platforms using the same DAA public key. Concretely, this applies to the execution of the following three functions.

- Tspi_DAA_IssueInit
- Tspi_TPM_DAA_JoinCreateDaaPubKey
- Tspi_DAA_IssueCredential

However, different executions of the protocol can be batched as follows:

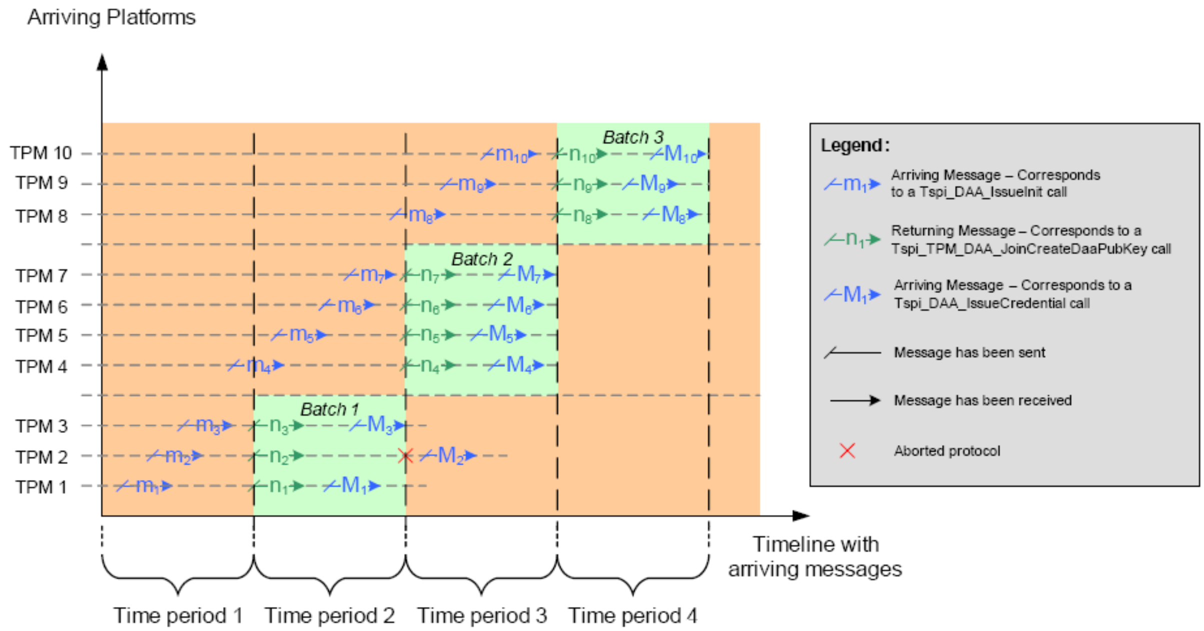


Figure 10 : Batch execution of DAA Join

All first messages *m* received from Platforms by the DAA Issuer in a certain time period, will be handled in one batch. With the start of a batch, the platforms receive returning messages *n* (nonces), to which they have to respond with messages *M* in a certain time period. If a platform is unable to respond in that time period, it has to restart the protocol.

For example, the messages *m* from the Platforms 1-3 have been received in time to be handled by the first batch. Platform 1 and 3 are able to respond during the time period 2, hence the DAA Join protocol will be continued. However, the protocol with Platform 2 will be aborted.

Concretely, the first message m is the input parameters to a Tspi_DAA_IssueInit call. The returning message *n* corresponds to a Tspi_TPM_DAA_JoinCreateDaaPubKey call. And the responding message *M* to *n* corresponds to a Tspi_DAA_IssueCredential call.

The number of different platforms handled by one batch is unlimited. The length of the time periods can be chosen arbitrarily.

4.3.4.29.3 DAA Sign

In the DAA Sign protocol a TCG platform interacts with a DAA Verifier to produce a signature with a DAA Credential whereby its attributes can be selectively revealed, encrypted in a verifiable manner or properties of it can be proven. An example for the latter would be to prove that the expiration date of a DAA Credential has not been reached, without revealing it, namely that the expiration date is greater than today's date.

The DAA Credential (respectively its key) allows signing of either an AIK of the TPM or an external message.

Further, revocation of the anonymity or pseudonym of the DAA Credential can be enabled when using the DAA Issuer's name base or a fixed name base chosen by the DAA Verifier, respectively. Revoking the anonymity of the DAA Credential means to reveal the identity of the DAA Credential owner as known to the DAA Issuer.

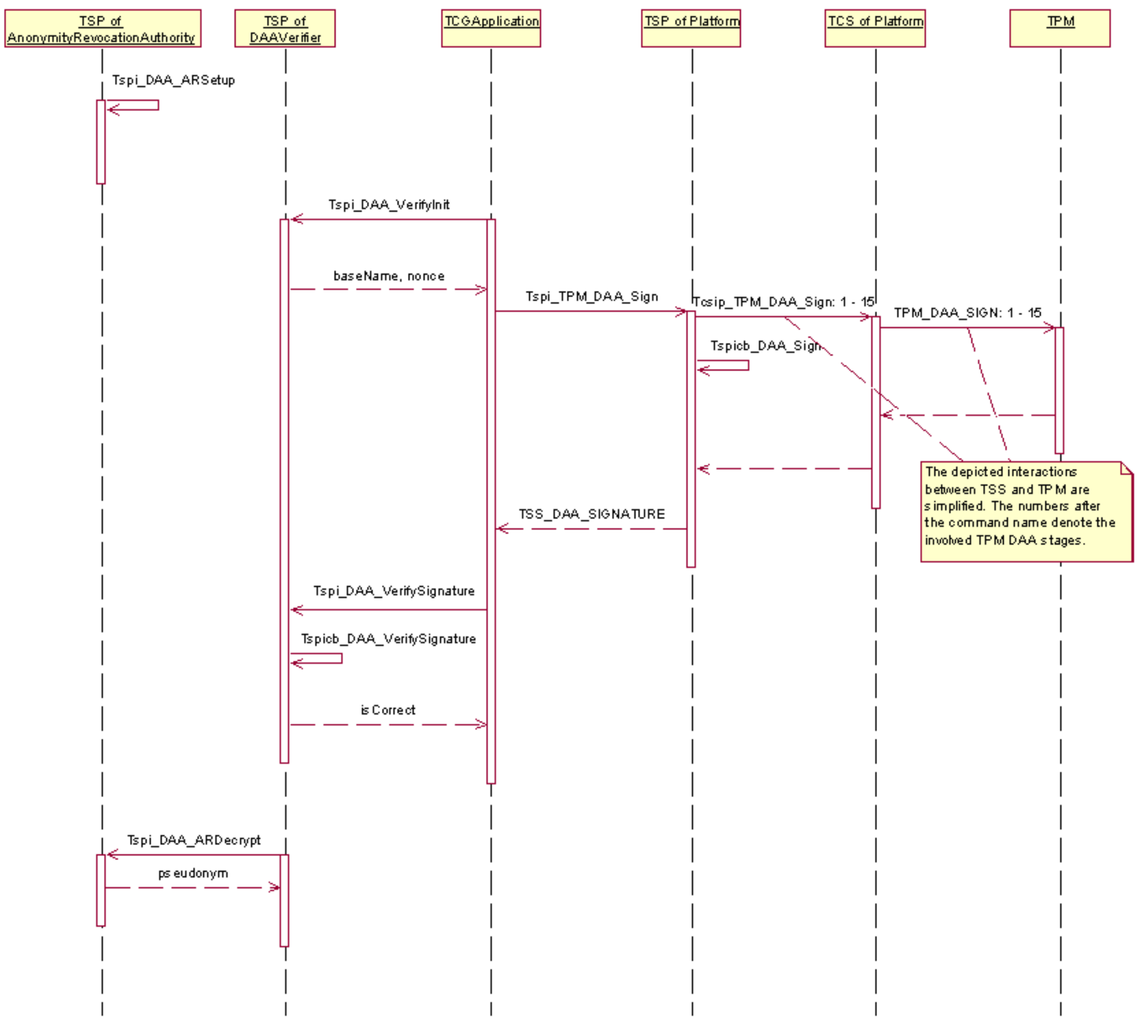


Figure 11 : UML Sequence diagram of DAA Sign

4.3.4.29.4 Keys of DAA Issuer

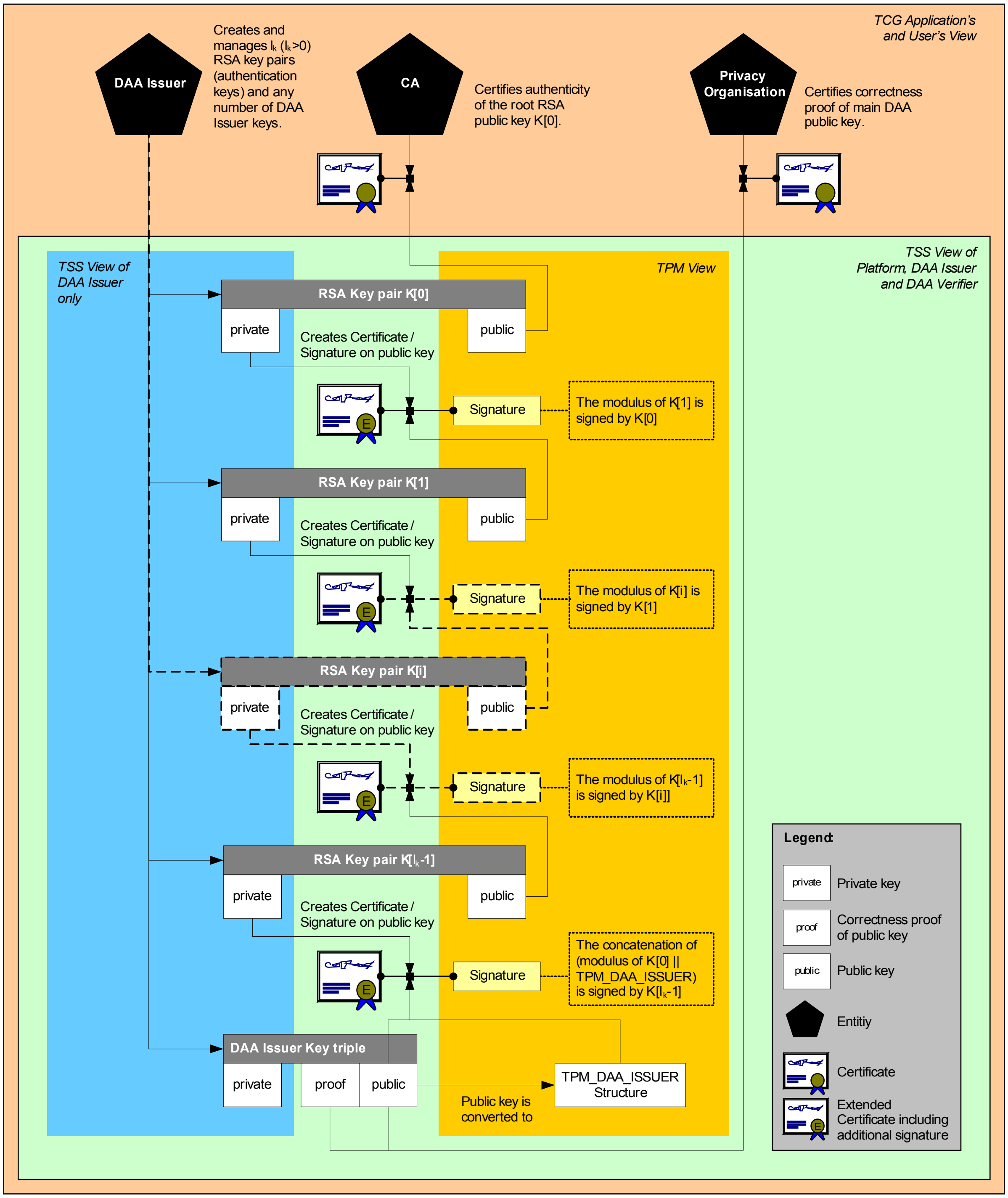


Figure 12 : Keys of DAA Issuer

A DAA Issuer might issue several thousands DAA Credentials using its private DAA Issuer key, hence that key will be a high volume key. For security reasons one should be able to change this key periodically. In order to support this and to meet the revocation requirements of TCG, a DAA Issuer will have a root authentication key (RSA key) for authentication of main DAA Issuer keys. Further, a DAA Issuer can have many intermediate authentication keys (key chain) that authenticate themselves and main DAA Issuer keys. This allows keeping the exposure period of a key to a minimum.

To meet the revocation requirements of TCG, a platform has to use the same private DAA Platform key each time it requests a DAA Credential from the same DAA Issuer. Therefore the root authentication key is used as input to compute the private key of the Platform. Further, the private key is used to compute the pseudonym with respect to a DAA Issuer and a DAA Verifier. A different private key, would allow establishing new pseudonyms with DAA Issuers and DAA Verifiers. This affects frequency checks by a DAA Verifier and Anonymity Revocation of a DAA Credential. The private key with respect to a DAA Issuer can only be changed, when changing the `daaCounter` that is used as input to the DAA Join protocol. A change of the `daaCounter` is appropriate if a platform changes owner to protect the privacy of the owners.

The keys are authenticated using certificates, e.g. X.509 certificates, and simple signatures. While the TSS will verify the certificates, the TPM will only be able to read and verify simple signatures on parts of a public key, e.g. RSA modulus. Such a signature can be stored in an extended attribute of a certificate as illustrated in figure 12.

4.3.4.29.5 Notation

Generally, all numbers are positive unsigned integers, unless otherwise noted.

$\{0,1\}^l$ Denotes the set of all binary strings of length l. It denotes also the set $[0,2^l-1]$ of integers.

$\pm\{0,1\}^l$ Denotes the set of all binary strings of length l. It denotes also the set $[-2^{l-1},2^{l-1}-1]$ of integers.

$PKDAA_I$ Is the DAA Issuer public key, the (main) public key of the DAA Issuer and is an instance of `TSS_DAA_PK` . It consists of:

- n Modulus .
- $R_{0,} R_1$ Logarithm bases used by TPM to encode its key into the credential.
- Y_i $Y_{0,..}, Y_{l_h+l_i-1}$ Logarithm bases used to encode attributes into the credential.
- l_h Number of attributes that the Platform can choose for a DAA Credential.
- l_i Number of attributes that the DAA Issuer can choose for a DAA Credential.
- S Generator of the group of quadratic residues modulo .
- Z Generator of the group .
- γ Gamma.
- Γ Capital Gamma.
- ρ Rho.
- bsn_I Is the DAA Issuer's long-term base name.

S' $S':=s^{2^{l_s}} mod\ n$ ·

γ_i $\gamma_{0,..},\gamma_{l_h+l_i-1}$ Are elements of $\langle\gamma\rangle$ and are used to compute commitments

$\gamma_i :=($`MGF1(` $i\|\gamma$, `LENGTH_MGF1_GAMMA` $))^{\frac{(\Gamma-1)}{\rho}} mod\ \Gamma$ [2]

PK_I $PK_{I_0},\ldots,PK_{I_{l_k-1}}$ Is the public key chain of the DAA Issuer used to authenticate $PKDAA_I$, see also Figure 12.

$PKSig_I$ $PKSig_{I_0},\ldots,PKSig_{I_{l_k-1}}$ Is the signature chain created by PK_I and $PKDAA_I$, see also Figure 12.

[2] The variable i is an index of an attribute to which a commitment will be computed. Its length is 4 bytes (UINT32).

l_k Length of key chain PK_I, see also Figure 12.

a_i Are attribute values of the DAA Credential where $a_i \in \pm\{0,1\}^{l_f}$, $a_{0,} .. a_{l_h-1}$ are attribute values that the Platform has chosen and $a_{l_h}, \ldots, a_{l_h+l_i-1}$ are the attributes that the DAA Issuer has chosen.

l_c Denotes the number of commitments to attributes of the DAA Credential.

b_2 Boolean value, if TRUE Anonymity Revocation is enabled for DAA Signature.

b_b Boolean value, if TRUE Callback mechanism is enabled in DAA Sign protocol.

C Is a set of indices of attributes used in conjunction with commitments. For DAA Sign these are the attributes which the credential owner wants to commit to. It is a subset of indices of the attributes the owner does not want to reveal to the DAA Verifier. For DAA Join these are attributes not known by the DAA Issuer, but for which he knows a commitment.

$\{x\}_b$ $\{x\}_b = \{ \begin{matrix} x, if\ b = true \\ \perp, if\ b = false \end{matrix} \}$
x is a term that is ignored (NULL) if the Boolean value b is FALSE.

$H_{TPM}()$ Hash function defined and used by TPM. The algorithm is `SHA-1` .

`MGF1`(s, l) Mask Generation Function 1 using $H_{TPM}()$ as the hash function. It has two input parameters s (seed) and l (length). The length denotes the number of bytes of the output.

|| Concatenation operator

`issuerSettings`
Is an instance of `TPM_DAA_ISSUER` structure containing $(n, R_{0,} R_{1,} S, S', \rho, \Gamma)$ of $PKDAA_I$

`sessionHandle`
A TPM supports at least one concurrent DAA Join or DAA Sign session. Such a session is referenced by the `sessionHandle` variable. The session handle is generated by both TPM DAA commands of stage equal to zero and input to each command of each stage that is not equal to zero.

`Tcsip_TPM_DAA_Join(i, inputData0, inputData1)`
Reflects the TCS call of the DAA Join command (`Tcsip_TPM_DAA_Join`) of stage `i` with the most relevant input parameters `inputData0` and `inputData1`.

`Tcsip_TPM_DAA_Sign(i, inputData0, inputData1)`
Reflects the TCS call of the DAA Sign command (`Tcsip_TPM_DAA_Sign`) of stage `i` with the most relevant input parameters `inputData0` and `inputData1`.

4.3.4.29.6 Join Protocol

`daaCounter`
Is the current value of the counter keeping track of the number of times the TPM has run the Join protocol (however, the TPM is allowed to re-run the Join protocol with the same `daaCounter` value many times).

4.3.4.29.7 Sign Protocol

X	Is the set of the indices of the attributes that the credential owner wants to reveal. Thus the Platform sends the DAA Verifier also these attributes, i.e., the list $\{a_i\}_{i \in X}$. The other attributes, i.e. $\{a_i\}_{i \notin X}$, remain hidden from the DAA Verifier.
L	Is a string describing the condition under which anonymity can be revoked.
bsn_v	Is the base name value provided by the DAA Verifier.

4.3.4.29.8 Definitions

- l_n := 2048 Is the size of the RSA modulus
- l_f := 104 Is the size of the $f_i's$ (information encoded into the certificate).
- l_e := 368 Is the size of the e's (exponents, part of the certificate).
- l'_e := 120 Is the size of the interval the e's are chosen from.
- l_v := 2536 Is the size of the $v's$ (random value, part of the certificate).
- l_Φ := 80 Is the security parameter controlling the statistical zero-knowledge property.
- l_H := 160 Is the output length of the hash function (SHA-1) used for the Fiat-Shamir heuristic.
- l_s := 1024 Is the size to split large exponent for easier computations on the TPM.
- l_Γ := 1632 Is the size of the modulus
- l_{rho} := 208 Is the size of the order ρ of the subgroup of $\mathbb{Z}^*_\Gamma$ that is used for rogue-tagging.

Note that the sizes l_h and l_i denote the number of possible attributes of a DAA Credential and is defined per DAA Issuer public key. The size l_k denotes the length of the key chain of the DAA Issuer, concretely the number of authentication (RSA) keys. The variable l_c denotes the number of commitments to attributes of the DAA credential.

4.3.4.29.9 Lengths

The following lengths denote the number of bytes:

`LENGTH_MGF1_GAMMA` = 214 Is the output length of MGF1 in conjunction with the modulus Γ and is equal to $\lfloor \frac{(l_\Gamma + l_\Phi)}{8} \rfloor$.

`LENGTH_MGF1_AR` = 25 Is the output length of MGF1 used for anonymity revocation and is equal to $\lfloor \frac{(l_\rho - 1)}{8} \rfloor$.

4.3.4.29.10 Input to hash functions

The algorithm specifications for DAA contain a couple of hash functions with long input parameters that are a concatenation of multiple variables. It is important that the byte representation including the length of these variables is well defined.

"Each structure MUST use big endian bit ordering, which follows the Internet standard and requires that the low-order bit appear to the far right of a word, buffer, wire format, or other area and the high-order bit appear to the far left.

All structures MUST be packed on a byte boundary."[3]

Mathematical variables (in order of their appearance)	Byte length
i	4
γ	$\ell_\Gamma /8$
$U, U', \tilde{U}, \tilde{U}'$	$\ell_n /8$
$N_I, \tilde{N}_I$	$\ell_\Gamma /8$
$\beta, \tilde{\beta}$	$\ell_\Gamma /8$
n_i	$\ell_H /8$
v''	$\ell_v /8$
$A, \tilde{A}$	$\ell_n /8$
n_h	$\ell_H /8$
n_v	$\ell_H /8$
ℓ_c	4
b_2, b_b (Boolean values, where TRUE→1, FALSE → 0)	1
ζ	$\ell_\Gamma /8$
$T, \tilde{T}$	$\ell_n /8$
$\eta, \lambda_1, \lambda_2, \lambda_3$	$\ell_\Gamma /8$
$\delta_{1,} \delta_{2,} \delta_{3,} \delta_{4,} \tilde{\delta}_{1,} \tilde{\delta}_{2,} \tilde{\delta}_{3,} \tilde{\delta}_4$	$\ell_\Gamma /8$
c_b	$\ell_H /8$
L	$\ell_H /8$
$n, S, Z, R_0, R_1, Y_0, \ldots, Y_{\ell_h + \ell_i - 1}$	$\ell_n /8$
c_t, n_e	$\ell_H /8$

[3] See TPM Main - Part 2 TPM Structures specification. Per default the Java virtual machine uses big endian bit order and the Intel x86 processors little endian, for instance.

4.3.4.30 DAA Functions

The provided algorithm specification describes how a TSS implementation has to behave.

4.3.4.30.1 Tspi_SetAttribUint32

Start of informative comment:

This method sets a 32-bit attribute of a DAA Credential object.

End of informative comment.

Definition:

See section 4.3.3.1.1 of the TSS specification for definition.

Parameters

See section 4.3.3.1.1 of the TSS specification for description.

Defined Attributes

None.

Return Values

See section 4.3.3.1.1 for description.

4.3.4.30.2 Tspi_GetAttribUint32

Start of informative comment:

This method gets a 32-bit attribute of a DAA Credential object.

End of informative comment.

Definition:

See section 4.3.3.1.2 of the TSS specification for definition.

Parameters

See section 4.3.3.1.2 of the TSS specification for description.

Defined Attributes

Flag	SubFlag	Attribute	Description
TSS_TSPATTRIB_DAACRED_COMMIT	TSS_TSPATTRIB_DAACOMMIT_NUMBER	Integer value.	Get the number of commitments to selective attributes of or for the DAA Credential that will be used during the DAA Join or DAA Sign protocol. If zero (default) no commitments will be used.

Return Values

See section 4.3.3.1.2 for description.

4.3.4.30.3 Tspi_SetAttribData

Start of informative comment:

This method sets a non-32-bit attribute of a DAA Credential object. The structure and size of the attribute data depend on the attribute.

End of informative comment.

Definition:

See section 4.3.3.1.3 of the TSS specification for definition.

Parameters

See section 4.3.3.1.3 of the TSS specification for description.

Defined Attributes

Flag	SubFlag	Data Description
TSS_TSPATTRIB_DAACRED_COMMIT	TSS_TSPATTRIB_DAACOMMIT_SELECTION	A selection of attributes of or for a DAA Credential. The data type is TSS_DAA_SELECTED_ATTRIB.
	TSS_TSPATTRIB_DAA_COMMITMENT	This data can be shared by a DAA Sign and a DAA Join command. It contains the commitments on selected attributes of the DAA Credential and the commitment randomness to open a commitment. The data type is an array of TSS_DAA_ATTRIB_COMMIT. The length of the array is TSS_TSPATTRIB_DAA_ATTRIB_COMMIT_NUMBER.
TSS_TSPATTRIB_DAACRED_ATTRIB_GAMMAS	TSS_TSPATTRIB_DAAATTRIBGAMMAS_BLOB	Array of gammas.
TSS_TSPATTRIB_DAACRED_CREDENTIAL_BLOB	0	The DAA credential.
TSS_TSPATTRIB_DAACRED_CALLBACK_SIGN	Application provided data.	Address of callback or NULL (disable)
TSS_TSPATTRIB_DAACRED_CALLBACK_VERIFYSIGNATURE	Application provided data.	Address of callback or NULL (disable)

Return Values

See section 4.3.3.1.3 of the TSS specification for description.

4.3.4.30.4 Tspi_GetAttribData

Start of informative comment:

This method gets a non-32-bit attribute of a DAA Credential object. The structure and size of the attribute data depend on the attribute.

End of informative comment.

Definition:

See section 4.3.3.1.4 of the TSS specification for definition.

Parameters

See section 4.3.3.1.4 of the TSS specification for description.

Defined Attributes

Flag	SubFlag	Data Description
TSS_TSPATTRIB_DAACRED_COMMIT	TSS_TSPATTRIB_DAA_SELECTED_ATTRIB	A set of selection of attributes of or for a DAA Credential. The data type is an array of TSS_DAA_SELECTED_ATTRIB. The length of the array is TSS_TSPATTRIB_DAACOMMIT_NUMBER.
	TSS_TSPATTRIB_DAA_COMMITMENT	This data can be shared by a DAA Sign and a DAA Join command. It contains the commitments on selected attributes of the DAA Credential and the commitment randomness to open a commitment. The data type is an array of TSS_DAA_ATTRIB_COMMITMENT. The length of the array is TSS_TSPATTRIB_DAACOMMIT_NUMBER.
TSS_TSPATTRIB_DAACRED_ATTRIB_GAMMAS	TSS_TSPATTRIB_DAAATTRIBGAMMAS_BLOB	Array of gammas.
TSS_TSPATTRIB_DAACRED_CREDENTIAL_BLOB	0	The DAA credential.
TSS_TSPATTRIB_DAACRED_CALLBACK_SIGN	Application provided data.	Address of callback or NULL (disable)
TSS_TSPATTRIB_DAACRED_CALLBACK_VERIFYSIGNATURE	Application provided data.	Address of callback or NULL (disable)

Return Values

See section 4.3.3.1.4 of the TSS specification for description.

4.3.4.30.5 Tspi_SetAttribUint32

Start of informative comment:

This method sets a 32-bit attribute of a DAA Issuer Key object.

End of informative comment.

Definition:

See section 4.3.3.1.1 of the TSS specification for definition.

Parameters

See section 4.3.3.1.1 of the TSS specification for description.

Defined Attributes

Flag	SubFlag	Attribute	Description
TSS_TSPATTRIB_DAAISSUERKEY_PUBKEY	TSS_TSPATTRIB_DAAISSUERPUBKEY_NUM_PLATFORM_ATTRIBS	Integer value.	The number of platform attributes for a credential: l_h
	TSS_TSPATTRIB_DAAISSUERPUBKEY_NUM_ISSUER_ATTRIBS	Integer value.	The number of issuer attributes for a credential: l_i

Return Values

See section 4.3.3.1.1 of the TSS specification for description.

4.3.4.30.6 Tspi_GetAttribUint32

Start of informative comment:

This method gets a 32-bit attribute of a DAA Issuer Key object.

End of informative comment.

Definition:

See section 4.3.3.1.2 of the TSS specification for definition.

Parameters

See section 4.3.3.1.2 of the TSS specification for description.

Defined Attributes

Flag	SubFlag	Attribute	Description

TSS_TSPATTRIB_DAAISSUERKEY_PUBKEY	TSS_TSPATTRIB_DAAISSUERPUBKEY_NUM_ATTRIBS	Integer value.	The total number of credential attributes: $l_h + l_i$
	TSS_TSPATTRIB_DAAISSUERPUBKEY_NUM_PLATFORM_ATTRIBS	Integer value.	The number of platform attributes for a credential: l_h
	TSS_TSPATTRIB_DAAISSUERPUBKEY_NUM_ISSUER_ATTRIBS	Integer value.	The number of issuer attributes for a credential: l_i

Return Values

See section 4.3.3.1.2 of the TSS specification for description.

4.3.4.30.7 Tspi_SetAttribData

Start of informative comment:

This method sets a non-32-bit attribute of a DAA Issuer Key object.

End of informative comment.

Definition:

See section 4.3.3.1.3 of the TSS specification for definition.

Parameters

See section 4.3.3.1.3 of the TSS specification for description.

Defined Attributes

Flag	SubFlag	Data Description
TSS_TSPATTRIB_DAAISSUERKEY_BLOB	TSS_TSPATTRIB_DAAISSUERKEYBLOB_PUBLIC_KEY	An Issuer's public key as a TSS_DAA_PK structure.
	TSS_TSPATTRIB_DAAISSUERKEYBLOB_SECRET_KEY	An Issuer's secret key as a TSS_DAA_SK structure.

	TSS_TSPATTRIB_DAAISSUERKEYBLOB_KEYBLOB	An Issuer's key pair as a TSS_DAA_KEY_PAIR structure.
	TSS_TSPATTRIB_DAAISSUERKEYBLOB_PROOF	An Issuer's public key proof as a TSS_DAA_PK_PROOF structure.

Return Values

See section 4.3.3.1.3 of the TSS specification for description.

4.3.4.30.8 Tspi_GetAttribData

Start of informative comment:

This method gets a non-32-bit attribute of a DAA Issuer Key object.

End of informative comment.

Definition:

See section 4.3.3.1.4 of the TSS specification for definition.

Parameters

See section 4.3.3.1.4 of the TSS specification for description.

Defined Attributes

Flag	SubFlag	Data Description
TSS_TSPATTRIB_DAAISSUERKEY_BLOB	TSS_TSPATTRIB_DAAISSUERKEYBLOB_PUBLIC_KEY	An Issuer's public key as a TSS_DAA_PK structure.
	TSS_TSPATTRIB_DAAISSUERKEYBLOB_SECRET_KEY	An Issuer's secret key as a TSS_DAA_SK structure.
	TSS_TSPATTRIB_DAAISSUERKEYBLOB_KEYBLOB	An Issuer's key pair as a TSS_DAA_KEY_PAIR structure.
	TSS_TSPATTRIB_DAAISSU	An Issuer's public key

Flag	SubFlag	Data Description
	ERKEYBLOB_PROOF	proof as a TSS_DAA_PK_PROOF structure.

Return Values

See section 4.3.3.1.4 of the TSS specification for description.

4.3.4.30.9 Tspi_SetAttribUint32

Start of informative comment:

This method sets a 32-bit attribute of a DAA ARA Key object.

End of informative comment.

Definition:

See section 4.3.3.1.1 of the TSS specification for definition.

Parameters

See section 4.3.3.1.1 of the TSS specification for description.

Defined Attributes

None.

Return Values

See section 4.3.3.1.1 of the TSS specification for description.

4.3.4.30.10 Tspi_GetAttribUint32

Start of informative comment:

This method gets a 32-bit attribute of a DAA ARA Key object.

End of informative comment.

Definition:

See section 4.3.3.1.2 of the TSS specification for definition.

Parameters

See section 4.3.3.1.2 of the TSS specification for description.

Defined Attributes

None.

Return Values

See section 4.3.3.1.2 of the TSS specification for description.

4.3.4.30.11 Tspi_SetAttribData

Start of informative comment:

This method sets a non-32-bit attribute of a DAA ARA Key object.

End of informative comment.

Definition:

See section 4.3.3.1.3 of the TSS specification for definition.

Parameters

See section 4.3.3.1.3 of the TSS specification for description.

Defined Attributes

Flag	SubFlag	Data Description
TSS_TSPATTRIB_DAAARAKEY_BLOB	TSS_TSPATTRIB_DAAARAKEYBLOB_PUBLIC_KEY	Get/Set an ARA's public key as a TSS_DAA_AR_PK structure.
	TSS_TSPATTRIB_DAADARAKEYBLOB_SECRET_KEY	Get/Set an ARA's secret key as a TSS_DAA_AR_SK structure.
	TSS_TSPATTRIB_DAAARAKEYBLOB_KEYBLOB	Get/Set an ARA's key pair as a TSS_DAA_AR_KEY_PAIR structure.

Return Values

See section 4.3.3.1.3 of the TSS specification for description.

4.3.4.30.12 Tspi_GetAttribData

Start of informative comment:

This method gets a non-32-bit attribute of a DAA ARA Key object.

End of informative comment.

Definition:

See section 4.3.3.1.4 of the TSS specification for definition.

Parameters

See section 4.3.3.1.4 of the TSS specification for description.

Defined Attributes

Flag	SubFlag	Data Description
TSS_TSPATTRIB_DAAARAKEY_BLOB	TSS_TSPATTRIB_DAAARAKEYBLOB_PUBLIC_KEY	Get/Set an ARA's public key as a TSS_DAA_AR_PK structure.
	TSS_TSPATTRIB_DAADARAKEYBLOB_SECRET_KEY	Get/Set an ARA's secret key as a TSS_DAA_AR_SK structure.
	TSS_TSPATTRIB_DAAARAKEYBLOB_KEYBLOB	Get/Set an ARA's key pair as a TSS_DAA_AR_KEY_PAIR structure.

Return Values

See section 4.3.3.1.4 of the TSS specification for description.

4.3.4.30.13 Tspi_TPM_DAA_JoinInit

Start of informative comment:

This is the first out of 3 functions to execute in order to receive a DAA Credential. It verifies the keys of the DAA Issuer and computes the TPM DAA public key.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_DAA_JoinInit
(
    TSS_HTPM             hTPM,                              // in
    TSS_HDAA_ISSUER_KEY  hIssuerKey,                        // in
    UINT32               daaCounter,                        // in
    UINT32               issuerAuthPKsLength,               // in
    TSS_HKEY*            issuerAuthPKs,                     // in
    UINT32               issuerAuthPKSignaturesLength,      // in
    UINT32               issuerAuthPKSignaturesLength2,// in
    BYTE**               issuerAuthPKSignatures,            // in
    UINT32*              capitalUprimeLength,               // out
    BYTE**               capitalUprime,                     // out
    TSS_DAA_IDENTITY_PROOF** identityProof,                 // out
    UINT32*                  joinSessionLength,             // out
    BYTE**                   joinSession                    // out
);
```

Parameters

hTPM

Handle of the TPM object. This object provides owner authorization for the TPM commands.

hIssuerKey

Handle of the DAA Issuer Key object

daaCounter

DAA counter

issuerAuthPKsLength

Number of keys in the issuerAuthPKs array (l_k , see Figure 12).

issuerAuthPKs

Handle of an array of RSA public keys (key chain) of the DAA Issuer used to authenticate the DAA Issuer public key. The size of the modulus must be DAA_SIZE_issuerModulus[4] (256)

issuerAuthPKSignaturesLength

The number of signatures in the array issuerAuthPKSignatures. It is equal to issuerAuthPKsLength (l_k , see Figure 12).

[4] See TPM Main - Part 2 TPM Structures specification.

issuerAuthPKSignaturesLength2

The length of each individual signature in the array issuerAuthPKSignatures. For 1.2 TPMs this value must be DAA_SIZE_issuerModulus (256)

issuerAuthPKSignatures

An array of byte arrays representing signatures on the modulus of the above key chain (issuerAuthPKs). In more details, the array has the following content (S(K[1],K[0]), S(K[2],N[1]),..S(K[l_k]],K[l_k -1]), S(TPM_DAA_ISSUER,K[l_k])), where S(msg,privateKey) denotes the signature function with msg being signed by the privateKey. Each array elements contains a signature on one piece of the authentication public key chain, except the last array element contains the signature on the TPM_DAA_ISSUER structure.

See Figure 12 that illustrates the key chain.

capitalUprimeLength

Length of capitalUprime which is $\frac{l_n}{8}$

capitalUprimeLength

Length of capitalUprime which is $\frac{l_n}{8}$

capitalUprime

U'

identityProof

Returns the endorsement, platform and conformance credential.

joinSessionLength

Returns the length of the join session buffer.

joinSession

This buffer used by the TSS to temporarily store data and intermediate calculations for the remaining 2 DAA Join function calls. Its format is TSS-specific. When the application is finished with the DAA Join protocol this buffer should be freed via Tspi_Context_FreeMemory.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR
TSS_E_DAA_ISSUER_KEY_ERROR

Algorithm specification

JoinInit(daaCounter, issuerPK, issuerAuthPKs, issuerAuthPKSignatures)

Input:

		daaCounter
PK_I	:=	issuerAuthPKs
$PKSig_I$	:=	issuerAuthPKSignatures

Input via TSS Attributes:

$PKDAA_I$	:=	TSS_TSPATTRIB_DAAISSUERKEYBLOB_PUBLIC_KEY

Output:

capitalUPrime	:=	U'
joinSession	:=	(U', $PKDAA_I$, daaCounter, sessionHandle)

Steps:

1. Convert $PKDAA_I$ to a TPM_DAA_ISSUER structure

   TPM_DAA_ISSUER issuerSettings:= $PKDAA_I$

2. The TPM and Platform verify that $PKDAA_I$ and issuerSettings are authenticated by PK_I and $PKSig_{I.}$

   (a) If $l_k > 0$, verify whether $PKDAA_I$ and issuerSettings are authenticated by PK_I and $PKSig_{I.}$; otherwise return the error code TSS_E_DAA_ISSUER_KEY_ERROR

   (b) Send l_k to the TPM (Verifies $l_k > 0$ as well) and get session handle:

   sessionHandle:= Tcsip_TPM_DAA_Join(0, l_k, null).

   (c) Let the TPM verify that issuerSettings is authenticated by PK_I and $PKSig_{I.}$. send only the modulus of the key PK_I to the TPM:

   i. Tcsip_TPM_DAA_Join(1, PK_{I_0}, null),

   ii. Tcsip_TPM_DAA_Join(1, $PK_{I_{i+1}}, PKSig_{I_i}$) for $i = 0, \ldots, l_k - 2$,

   iii. Tcsip_TPM_DAA_Join(2, issuerSettings, $PKSig_{I_{l_k-1}}$).

3. Send the DAA counter to the TPM: Tcsip_TPM_DAA_Join(3, daaCounter, null).

4. Call the TPM to compute the first part of the credential request (DAA public key of the TPM):

   (a) Tcsip_TPM_DAA_Join(4, R_0, n),

   (b) Tcsip_TPM_DAA_Join(5, R_1, n),

   (c) Tcsip_TPM_DAA_Join(6, S, n),

   (d) U':= Tcsip_TPM_DAA_Join(7, S', n).

5. Save $PKDAA_I$, U', daaCounter, and sessionHandle in joinSession.
6. Output U' and joinSession

Remarks

The DAA Issuer uses a key chain for key management which is verified by the TPM and which root key N0 is used by the TPM to compute its private DAA key. See figures 11 DAA Sign and12 .

Additional verification of the DAA Issuer's key chain by TSS before loading it into the TPM is recommended since it allows detecting problems faster due to the general performance advantage of the TSS over TPM. The signing algorithm used by the DAA Issuer is "TPM_SS_RSASSAPKCS1v15_SHA1" as defined in the TPM Main specification .

4.3.4.30.14 Tspi_TPM_DAA_JoinCreateDaaPubKey

Start of informative comment:

This is the second out of 3 functions to execute in order to receive a DAA Credential. It computes the credential request for the DAA Issuer, which also includes the Platforms's DAA public key and the attributes that were chosen by the Platform, and which are not visible to the DAA Issuer. The Platform can commit to the attribute values it has chosen.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_DAA_JoinCreateDaaPubKey
(
   TSS_HTPM                     hTPM,                          // in
   TSS_HDAA_CREDENTIAL          hDAACredential,                // in
   UINT32                       authenticationChallengeLength,// in
   BYTE*                        authenticationChallenge,       // in
   UINT32                       issuerNonceLength,             // in
   BYTE*                        issuerNonce,                   // in
   UINT32                       attributesPlatformLength,      // in
   UINT32                       attributesPlatformLength2,     // in
   BYTE**                       attributesPlatform,            // in
   UINT32                       blindingValueLength,           // in
   UINT32                       joinSessionLength,             // in
   BYTE*                        joinSession,                   // in
   TSS_DAA_CREDENTIAL_REQUEST**  credentialRequest             // out
);
```

Parameters

hTPM

Handle of the TPM object. This object provides owner authorization for the TPM commands.

hDAACredential

Handle of the DAA Credential object.

authenticationChallengeLength

Length of authenticationChallenge (256 bytes - - DAA_SIZE_NE[5]).

authenticationChallenge

Second nonce of the DAA Issuer that is encrypted by the endorsement public key. It is used as a challenge to authenticate the underlying TPM.

issuerNonceLength

Length of nonceIssuer (20 bytes - - $\frac{l_H}{8}$).

issuerNonce

[5] See TPM Main - Part 2 TPM Structures specification

Nonce of the DAA Issuer.

attributesPlatformLength

The number of attributes in the attributesPlatform array, which is determined by the DAA Issuer public key (l_h).

attributesPlatformLength2

The length of an individual attribute. This value should be $\frac{l_f}{8}$.

attributesPlatform

An array of attributes to be encoded into the DAA Credential not visible to the DAA Issuer

blindingValueLength

This is l_μ , the length (in bytes) of each blinding value μ_j . If the commitment values are provided through the hDAACredential object this parameter is ignored. Otherwise, if this value is 0 the μ_j s are chosen from $\mathbb{Z}_\rho$ and if it is non-zero the μ_j s are chosen randomly from $\{0,1\}^{l_\mu}$.

joinSessionLength

This is the length of the join session temporary data..

joinSession

This buffer contains DAA Join session information.

credentialRequest

Credential request of the Platform, it contains the blinded DAA public key of the platform on which the DAA Issuer will issue the credential with the blinded attributes chosen by the Platform.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Algorithm specification:

JoinCreateDaaPubKey($authenticationChallenge, nonceIssuer, attributesPlatform, joinSession$)

Input:

n_i	:=	issuerNonce
α	:=	authenticationChallenge
$a_{0,} \ldots, a_{l_h-1}$	:=	attributesPlatform
l_μ	:=	blindingValueLength
$(U', issuerPK, daaCounter, sessionHandle)$	:=	joinSession

Input via TSS Attributes:

l_c	:=	TSS_TSPATTRIB_DAACOMMIT_NUMBER
$C_{0,} \ldots, C_{l_c-1}$	:=	TSS_TSPATTRIB_DAACOMMIT_SELECTION
$(\beta_{0,} \mu_0), \ldots, (\beta_{l_c-1}, \mu_{l_c-1})$	:=	TSS_TSPATTRIB_DAACOMMIT_COMMITMENTS
$\gamma_i \; for \; i \in C_0 \cup \ldots C_{l_c-1}$	:=	TSS_TSPATTRIB_DAACRED_ATTRIB_GAMMAS

$$\gamma_i := (MGF1(i \| \gamma, LENGTH\_MGF1\_GAMMA))^{(\Gamma-1)/\rho} mod\ \Gamma \ for\ i \in C_0 \cup \ldots C_{l_c-1}$$

Output:

credentialRequest :=

$$(U, N_I, a_{U'}, c, n_t, n_h, s_{f_0}, s_{f_1}, s_{v'}, s_{\tilde{v}'}, s_{a_0}, \ldots, s_{a_{l_h-1}}, \beta_{0,} .. \beta_{l_c-1}, s_{\mu_0}, \ldots s_{\mu_{l_c-1}})$$

joinSession :=

$$(n_h, U, U', \tilde{v}', a_{0,} .., a_{l_h-1}, issuerPK, daaCounter, sessionHandle)$$

Steps:

1. Compute second part of the credential request (The variable $\tilde{v}'$ randomizes the attributes):
   (a) $\tilde{v}' \in_R \{0,1\}^{l_n + l_\Phi}$
   (b) $U := U' S^{\tilde{v}'} * \prod_{i=0}^{l_h-1} Y_i^{a_i} mod\ n$
2. Call the TPM to compute authentication proof with U':
   (a) $a_{U'}$:= Tcsip_TPM_DAA_Join $(8, \alpha, null)$.
3. Call the TPM to compute $\tilde{U}'$ (first part of correctness proof of the credential request):
   (a) Tcsip_TPM_DAA_Join(9, R_0 , n),

(b) Tcsip_TPM_DAA_Join(10, R_1 , n),

(c) Tcsip_TPM_DAA_Join(11, S , n),

(d) $\tilde{U}' :=$ Tcsip_TPM_DAA_Join(12, S' , n)

4. Compute pseudonym with respect to the DAA issuer

(a) Project Issuer's base name into group Γ :

$$\zeta := (MGF1(bsn_I, LENGTH\_MGF1\_GAMMA))^{\frac{\Gamma-1}{\rho}} \bmod \Gamma$$

(b) Let the TPM compute the pseudonym of the credential:

Extend the byte representation of ζ to DAA_SIZE_w bytes.

i. Tcsip_TPM_DAAJoin(13, Γ, ζ)

ii. $N_I := Tcsip\_TPM\_DAA\_Join(14, \Gamma, null)$.

(c) TPM now computes the correctness proof for the pseudonym:

i. $\tilde{N}_I$:=Tcsip_TPM_DAA_Join(15, Γ , null).

5. Compute the second part of the correctness proof of the credential request (with attributes not visible to Issuer):

(a) $r_{a_0}, \ldots, r_{a_{l_h}-1} \in_R \pm\{0,1\}^{l_f + l_\Phi + l_H}$

(b) Compute the commitment related part:

for (j=0,..., $l_c - 1$)

i. set $\gamma, \gamma_i, \Gamma, \rho, l_\mu$ according to TSS_ATTRIB_DAA_COMMIT_PARAM[j]

ii. if $\{\mu_j, \beta_j\} = \perp$ (the values have not been set by TSS attributes) then

A. If $\rho \neq \perp$ then choose random $\mu_j \in_R \mathbb{Z}_\rho$ else choose random $\mu_j \in_R \{0,1\}^{l_\mu}$

B. Compute commitments $\beta_j := \gamma^{\mu_j} \prod_{i \in C_j}^{\cdot} \gamma_i^{a_i} \bmod \Gamma$

iii. If $\rho \neq \perp$ then choose random $r_{\mu_j} \in_R \mathbb{Z}_\rho$ else choose random $r_{\mu_j} \in_R \{0,1\}^{l_\mu}$

iv. Compute the correctness proof of commitments $\tilde{\beta}_j := \gamma^{r_{\mu_j}} \prod_{i \in C_j}^{\cdot} \gamma_i^{r_{a_i}} \bmod \Gamma$

(c) $r_{\tilde{v}'} \in_R \{0,1\}^{l_f + 2l_\Phi + l_H}$

(d) $\tilde{U} := S^{r_{\tilde{v}'}} * \prod_{i=0}^{l_h - 1} Y_i^{r_{a_i}} \bmod n$

(e) $c_h := H_{TPM}(issuerPK \| U \| U' \| \tilde{U} \| \tilde{U}' \| N_I \| \tilde{N}_I \| l_c \| (\beta_0 \| \tilde{\beta}_0) \| \ldots \| (\beta_{t_c - 1} \| \tilde{\beta}_{t_c - 1}) \| n_i)$

(f) $n_t :=$ Tcsip_TPM_DAA_Join(16, c_h , null),

(g) s_{f_0}:= Tcsip_TPM_DAA_Join(17, null, null),

(h) s_{f_1}:= Tcsip_TPM_DAA_Join(18, null, null),

(i) $s_{v_1'}$:= Tcsip_TPM_DAA_Join(19, null, null),

(j) c:= Tcsip_TPM_DAA_Join(20, null, null),

(k) $s_{v_2'}$:= Tcsip_TPM_DAA_Join(21, null, null),

(l) $s_{v'} := s_{v_1'} + 2^{l_s} s_{v_2'}$

(m) $s_{a_i} := r_{a_i} + c * a_i \quad for\ i = 1, \ldots, l_h,$

(n) $s_{\tilde{v}'} := r_{\tilde{v}'} + c * \tilde{v}'$

(o) for (j=0,..., $l_c - 1$)

i. If $\rho \neq \perp$ then compute $s_{\mu_j} := r_{\mu_j} + c\mu_j \, mod\ \rho$ else compute $s_{\mu_j} := r_{\mu_j} + c\mu_j$

6. Compute the nonce for the Issuer to prove correctness of the credential: $n_h \in_R \{0,1\}^{l_H}$

7. Assign commitments

(a) If $l_c > 0$ then set

i. Update

TSS_TSSATTRIB_DAACOMMIT_COMMITMENT:= $(\beta_{0,} \mu_0), \ldots (\beta_{l_c - 1}, \mu_{l_c - 1}),$

(b) else set

i. TSS_TSPATTRIB_DAACOMMIT_COMMITMENT:= $\perp$

8. Save additionally $n_h, U, \tilde{v}'$, and $a_o, \ldots, a_{l_h - 1}$ in joinSession

9. Output credentialRequest :=
$(U, N_I, a_{U'}, c, n_t, n_h, s_{f_0}, s_{f_1}, s_{v'}, s_{\tilde{v}'}, s_{a_0}, \ldots, s_{a_{l_h-1}}, \beta_{0,} .. \beta_{l_c - 1}, s_{\mu_0}, \ldots s_{\mu_{l_c-1}})$

Remarks

Check the limitations of DAA Join

TCG Software Stack (TSS) Specification

4.3.4.30.15 Tspi_TPM_DAA_JoinStoreCredential

Start of informative comment:

This is the last out of 3 functions to execute in order to receive a DAA Credential. It verifies the issued credential from the DAA Issuer and computes the final DAA Credential.

End of informative comment.

```
TSS_RESULT Tspi_TPM_DAA_JoinStoreCredential
(
    TSS_HTPM                  hTPM,                // in
    TSS_HDAA_CREDENTIAL       hDAACredential,      // in
    TSS_DAA_CRED_ISSUER*      credIssuer,          // in
    UINT32                    joinSessionLength,   // in
    BYTE*                     joinSession          // in
);
```

Parameters

hTPM

Handle of the TPM object. This object provides owner authorization for the TPM commands.

hDAACredential

Handle of the DAA Credential object holding the credential parameters. This object receives the final credential.

credIssuer

The DAA Credential issued by the DAA Issuer including proof of correctness.

joinSessionLength

This is the length of the join session temporary data..

joinSession

This buffer contains DAA Join session information.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR
TSS_E_DAA_CREDENTIAL _PROOF_ERROR

Algorithm Specification

JoinStoreCredential(*hDAACredential* , *credIssuer* , *joinSession*)

Input:

$((A, e, v''), a_{l_h}, \ldots, a_{l_h+l_i-1}, (c', S_e))$:= credIssuer,

$(n_h, U, U', \tilde{v}', a_{0,}.., a_{l_h-1}, issuerPK, daaCounter, sessionHandle)$:= joinSession.

Output:via TSS attributes

TSS_TSPATTRIB_DAACRED_CREDENTIAL_BLOB := $(A, e, \bar{v}_{0,} \bar{v}_{1,} a_{0,}.., a_{l_h+l_i-1}, issuerPK, tpmSpecificEnc)$

Steps:

1. Verify the correctness of the issued credential. If one of the verifications fails return the error code TSS_E_DAA_CREDENTIAL_PROOF_ERROR.
   (a) Verify whether e is a prime and lies in $\left[2^{l_e-1}, 2^{l_e-1}+2^{l_e'-1}\right]$
   (b) Compute

$$\tilde{A} := A^{c'} \left(\frac{Z}{US^{v''} * \prod_{i=l_h}^{l_h+l_i-1} Y_i^{a_i}} \right)^{s_e} mod\ n$$

   (c) Check whether the following holds
      i. $c' = H_{TPM}(issuerPK \| v'' \| A \| \tilde{A} \| n_h)$
      ii. $Z \equiv A^e * US^{v''} * \prod_{i=l_h}^{l_h+l_i-1} Y_i^{a_i} (mod\ n)$
2. Compute the final credential[2] with the help of the TPM:
   (a) $v_0'' := LSB((v''+\tilde{v}'), l_s)$
   (b) $\bar{v}_0 :=$ Tcsip_TPM_DAA_Join(22, v''_0, null),
   (c) $v_1'' := CAR((v''+\tilde{v}'), l_s)$
   (d) Extend the byte representation of v_1'' to DAA_SIZE_v1 bytes
   (e) $\bar{v}_1 :=$ Tcsip_TPM_DAA_Join(23, v_1'', null)
   (f) tpmSpecificEnc:=Tcsip_TPM_DAA_Join(24, null, null).
3. Store the DAA Credential $(A, e, \bar{v}_{0,} \bar{v}_{1,} a_{0,}.., a_{l_h+l_i-1}, issuerPK, tpmSpecificEnc)$ in the object hDAACredential.

Remarks

4.3.4.30.16 Tspi_TPM_DAA_Sign

Start of informative comment:

This function creates a DAA Signature that proofs ownership of the DAA Credential and includes a signature on either a public AIK or a message.

If anonymity revocation is enabled, the value Nv is not provided in the clear anymore but encrypted under the public key of anonymity revocation authority, a trusted third party (TTP). Thus the DAA Verifier cannot check for revocation or link a transaction/signature to prior ones. Depending on how ζ is chosen, the protocol either allows implementing anonymity revocation (i.e., using the DAA Issuer's long-term base name bsn_I as the DAA Verifier's base name bsn_v), or having the TTP doing the linking of different signatures for the same DAA Verifier (i.e., using the DAA Verifier's base name bsn_v).

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_DAA_Sign
(
   TSS_HTPM                  hTPM,                        // in
   TSS_HDAA_CREDENTIAL       hDAACredential,              // in
   TSS_HDAA_ARA_KEY          hARAKey,                     // in
   TSS_HHASH                 hARACondition,               // in
   TSS_DAA_SELECTED_ATTRIB* revealAttributes,             // in
   UINT32                    verifierNonceLength,         // in
   BYTE*                     verifierNonce,               // in
   UINT32                    verifierBaseNameLength,      // in
   BYTE*                     verifierBaseName,            // in
   TSS_HOBJECT               signData,                    // in
   TSS_DAA_SIGNATURE**       daaSignature                 // out
);
```

Parameters

hTPM

Handle of the TPM object

hDAACredential

Handle of the DAA Credential object holding the credential.

hARAKey

Handle of the DAA Anonymous Revocation Authority Key object. This contains the public key of the DAA ARA as an attribute. If this handle is NULL, anonymous revocation is not used.

hARACondition

Handle of a hash object containing a digest representing the conditions under which the ARA should reveal the signer's pseudonym. This handle should be NULL if and only if hARAKey is NULL.

revealAttributes

The attributes which the credential owner wants to reveal to the DAA Verifier.

verifierNonceLength

Length of verifierNonceName (20 bytes - - $\frac{l_H}{8}$).

verifierNonce

Nonce created by the DAA Verifier

verifierBaseNameLength

Length of verifierBaseName

verifierBaseName

Base name chosen by the DAA Verifier. If it equals null, the platform chooses a random base name.

signData

The data to be signed. This can be either a TSS_HHASH or a TSS_HKEY object. If it is a TSS_HKEY object, the key must be an identity key and must be loaded.

daaSignature

DAA signature contains proof of ownership of the DAA Credential, as well as a signature on either an AIK or a message.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Algorithm Specification

Sign(hDAACredential, revealAttributes, verifierBaseName, verifierNonce,signData)

Input:

hDAACredential,

X := revealAttributes,

bsn_v := verifierBaseName,

n_v := verifierNonce,

Further, set the following mathematical variables accordingly

b_2 := `TRUE (1) if hARAKey is non-NULL, FALSE (0) if hARAKey is NULL`

`anonymityRevocationEnabled :=` b_2

Input via TSS attributes:

signData,

Commitment related input from hDAACredential

l_c := TSS_TSSATTRIB_DAACOMMIT_NUMBER

$C_{0,..}, C_{l_c-1}$:= TSS_TSPATTRIB_DAACOMMIT_SELECTION

$(\beta_{0,}\mu_0), \ldots, (\beta_{l_c-1}, \mu_{l_c-1})$:= TSS_TSPATTRIB_DAACOMMIT_COMMITMENTS

Further, set the following mathematical variables accordingly

$\gamma_i := (MGF1(i \| \gamma, LENGTH\_MGF1\_GAMMA))^{(\Gamma-1)/\rho} mod\ \Gamma\ for\ i \in C_0 \cup \ldots C_{l_c-1}$

If Anonymity Revocation is enabled (hARAKey is not NULL)

$(\eta, \lambda_{1,} \lambda_{2,} \lambda_3)$:= TSS_DAAARAKEYBLOB_PUBLIC_KEY from hARAKey

L := hash value from hARACondition

Output daaSignature:=($\zeta, T, c, n_t, s_v, S_{f_0}, s_{f_1}, s_e, (s_{a_i})_{i \in X}$, signedPseudonym).

If Commitments are enabled (TSS_TSPATTRIB_DAACOMMIT_NUMBER>0)

then set TSS_TSPATTRIB_DAACOMMIT_COMMITMENTS:= $(\beta_{0,}\mu_0), \ldots, (\beta_{l_c-1}, \mu_{l_c-1})$,

Output: $\sigma :=$ daaSignature

Steps:

1. Load the credential information ($A, e, a_{0,..}, a_{l_h+l_i-1}, \bar{v}_0, \bar{v}_1, PKDAA_I$, tpmSpecific) from TSS_TSPATTRIB_DAACRED_CREDENTIAL_BLOB.
2. Convert $PKDAA_I$ to a TPM_DAA_ISSUER structure

   TPM_DAA_ISSUER issuerSettings:= $PKDAA_{I.}$
3. Initialize the TPM with the credential context (keys) and get session handle:
   (a) sessionHandle :=Tcsip_TPM_DAA_Sign(0, issuerSettings, null)
   (b) Tcsip_TPM_DAA_Sign(1, tpmspecificEnc, null)
4. Let the TPM compute the first part of the DAA Signature:
   (a) Tcsip_TPM_DAA_Sign(2, $R_{0,}$ n),
   (b) Tcsip_TPM_DAA_Sign(3, $R_{1,}$ n),
   (c) Tcsip_TPM_DAA_Sign(4, $S, n,$),
   (d) $\tilde{T}_t$ Tcsip_TPM_DAA_Sign(5, S', n),
5. Compute the pseudonym with respect to which the credential will be shown together with the TPM:
   (a) Depending on the verifier's request the platform computes ζ as follows:

   If $bsn_V \neq \perp$ then project bsn_V into group Γ by computing

   $$\zeta := (MGF1(bsn_V, LENGTH\_MGF1\_GAMMA))^{\frac{\Gamma-1}{\rho}} \bmod \Gamma$$

   else choose[3] a random element of group Γ: $\zeta \in_R \langle \gamma \rangle$

   (b) Let the TPM check whether ζ is a member of group Γ:

   Extend the byte representation of ζ to DAA_SIZE_w bytes

   Tcsip_TPM_DAA_Sign(6, γ, ζ)

   (c) Let the TPM compute the pseudonym:

   $N_V :=$ Tcsip_TPM_DAA_Sign(7, γ , null).

   (d) Let the TPM compute the correctness proof of the pseudonym:

   : $\tilde{N}_V :=$ Tcsip_TPM_DAA_Sign(8, γ , null).
6. Anonymity Revocation (Encrypt the pseudonym with respect to either the verifier's bsn_v or the issuer's bsn_I)

   If `anonymityRevocationEnabled` $\equiv$ TRUE

   (a) Choose a random $\tau \in_R \mathbb{Z}_\rho$

   (b) Encrypt the pseudonym by computing

   $$\delta_1 := \gamma^\tau \bmod \Gamma$$

$$\delta_2 := \eta^{\tau} \bmod \Gamma$$

$$\delta_3 := \lambda_3^{\tau} N_V \bmod \Gamma$$

$$u := MGF1((\delta_1 \| \delta_2 \| \delta_3 \| L), LENGTH\_MGF1\_AR)$$

$$\delta_4 := \lambda_1^{\tau} \lambda_2^{\tau u} \bmod \Gamma$$

(c) Choose a random $r_\tau \in_R \mathbb{Z}_\rho$

(d) Compute the correctness proof of the encryption

i. $\tilde{\delta}_1 := \gamma^{r_\tau} \bmod \Gamma$

ii. $\tilde{\delta}_2 := \eta^{r_\tau} \bmod \Gamma$

iii. $\tilde{\delta}_3 := \lambda_3^{r_\tau} \tilde{N}_V \bmod \Gamma$

iv. $\tilde{\delta}_4 := \lambda_1^{r_\tau} \lambda_2^{r_\tau u} \bmod \Gamma$

7. Prove ownership of the credentials

(a) Choose a random integer $w \in \{0,1\}^{l_n + l_\Phi}$

(b) Compute T: = $AS^w \bmod n$

(c) Compute random integers

i. $r_e \in_R \{0,1\}^{l_e' + l_\Phi + l_H}$

ii. $r_v \in_R \{0,1\}^{l_e + l_n + 2l_\Phi + l_H + 1}$

iii. $r_{a_i} \in_R \pm \{0,1\}^{l_f + l_\Phi + l_H}$ $for\, i \notin X$

(d) Compute (randomize the set of attributes not being revealed)

$$\tilde{T} := \tilde{T}_t T^{r_e} S^{r_v} * \prod_{i \notin X} Y_i^{r_{a_i}} \bmod n$$

8. Commitments (Compute part of signature related to commitments): for (j=0,..., $l_c - 1$)

(a) set $\gamma, \gamma_i, \Gamma, \rho, l_\mu$ according to TSS_ATTRIB_DAA_COMMIT_PARAM[j]

(b) if $\{\mu_j, \beta_j\} = \perp$ (the values have not been set by TSS attributes) then

i. If $\rho \neq \perp$ then choose random $\mu_j \in_R \mathbb{Z}_\rho$ else choose random $\mu_j \in_R \{0,1\}^{l_\mu}$

ii. Compute commitments $\beta_j := \gamma^{\mu_j} \prod_{i \in C_j} \gamma_i^{a_i} \bmod \Gamma$

(c) If $\rho \neq \perp$ then choose random $r_{\mu_j} \in_R \mathbb{Z}_\rho$ else choose random $r_{\mu_j} \in_R \{0,1\}^{l_\mu}$

(d) Compute the correctness proof of commitments $\tilde{\beta}_j := \gamma^{r_{\mu_j}} \prod_{i \in C_j} \gamma_i^{r_{a_i}} \bmod \Gamma$

9. Call the callback function Tspicb_DAA_Sign if the callback mechanism is set in the DAA object.

(a) AdditionalProof:=Tspcib_DAA_Sign($PKDAA_I$,

$((\gamma_{0,}.., \gamma_{l_h+l_i}), (a_{0,}..a_{l_h+l_i}), (r_{a_i}, ... r_{a\, l_h+l_i}), (\beta_{0,}\mu_0), ..., (\beta_{l_c}, \mu_{l_c}), (\tilde{\beta}_{0,} r_{\mu_0}), ..., (\tilde{\beta}_{l_c}, r_{\mu_{l_c}}),,$

$\{N_V, \tilde{N}_V, \tau, \delta_{1,}\delta_{2,}\delta_{3,}\delta_{4,} r_\tau, \tilde{\delta}_{1,}\tilde{\delta}_{2,}\tilde{\delta}_{3,}\tilde{\delta}_4\}_{b_2})$

(b) c_b :=additionalProof.challenge

(c) b_b := TRUE

else

(d) additionalProof := $\perp$

(e) $c_b = \perp$

(f) $b_b = FALSE$

10. Compute “challenge”

(a) Compute (The Boolean values b_1, b_2 and b_b are represented as byte values when input to the hash function. FALSE =0, and TRUE=1)

$$c_h := H_{TPM}(PKDAA_I \| n_v \| X \| l_c \| b_2 \| b_b \| \zeta \| T \| \tilde{T} \| (C_0 \| \beta_0 \| \tilde{\beta}_0) \| ... \| (C_{l_c-1} \| \beta_{l_c-1} \| \tilde{\beta}_{l_c-1})$$

$$\| \{N_v \| \tilde{N}_v\}_{not\, b_2} \| \{(\eta \| \lambda_1 \| \lambda_2 \| \lambda_3) \| (\delta_1 \| \delta_2 \| \delta_3 \| \delta_4) \| (\tilde{\delta}_1 \| \tilde{\delta}_2 \| \tilde{\delta}_3 \| \tilde{\delta}_4)\}_{b_2} \| \{c_b\}_{b_b})$$

(b) Send c_h to the TPM and let the TPM choose a nonce

n_t:= Tcsip_TPM_DAA_Sign(9, c_h , null)

(c) Choose the payload selector and payload:

```
if (signData is a TSS_HHASH)
    selector := 1
    payload := signData hash value
else if (signData is a TSS_HKEY)
    selector := 0
    payload := the TCS handle for the signData key.
else
    return error TSS_E_BAD_PARAMETER.
```

(d) Let the TPM hash a public AIK or a hashed message as part of the DAA Signature.

c:= Tcsip_TPM_DAA_Sign(10, selector, payload)

11. Compute “response” to “challenge” c:

(a) Let the TPM compute

i. s_{f_0}:= Tcsip_TPM_DAA_Sign(11,null,null),

ii. s_{f_1}:= Tcsip_TPM_DAA_Sign(12,null,null),

iii. s_{v_0}:= Tcsip_TPM_DAA_Sign(13, $\bar{v}_0$,null),

iv. Tcsip_TPM_DAA_Sign(14, $\bar{v}_0$,null),

v. s_{v_1}:= Tcsip_TPM_DAA_Sign(15, $\bar{v}_1$,null),

(b) Compute

i. $s_e := r_e + c * (e - 2^{l_e - 1})$

ii. $s_v := s_{v_0} + 2^{l_s} s_{v_1} + r_v - c * w * e$

iii. $s_{a_i} := r_{a_i} + c * a_i \; for \; i \notin X$

(c) for (j=0,..., $l_c - 1$)

i. If $\rho \neq \perp$ then compute $s_{\mu_j} := r_{\mu_j} + c \mu_j mod \rho$ else compute $s_{\mu_j} := r_{\mu_j} + c \mu_j$

(d) If `anonymityRevocationEnabled` $\equiv$ TRUE, compute

$s_\tau := r_\tau + c * \tau \, (mod \; \rho)$

12. Output the DAA signature daaSignature

(a) Assign Commitments

if $l_c > 0$ then set

i. TSS_DAA_ATTRIB_COMMIT

attributeCommitments:= $(\beta_{0,} s_{\mu_0}), \ldots, (\beta_{l_c - 1}, s_{\mu_{l_c - 1}})$

ii. Update

TSS_TSPATTRIB_DAACOMMIT_COMMITMENTS:= $(\beta_{0,} \mu_0), \ldots, (\beta_{l_c - 1}, \mu_{l_c - 1})$

else set

iii. TSS_DAA_COMMIT attributeCommitments:= $\perp$

iv. TSS_TSPATRIB_DAA_COMMITMENT := $\perp$

(b) If `anonymityRevocationEnabled` $\equiv$ TRUE then set

i. TSS_DAA_PSEUDONYM_ENCRYPTED

payload:= $\{(\delta_{1,} \delta_{2,} \delta_{3,} \delta_4), s_\tau\}$

else set

ii. TSS_DAA_PSEUDONYM_PLAIN payload := $\{N_V\}$

(c) Assign above payload to new structure

TSS_DAA_PSEUDONYM signedPseudonym.payload:=payload

(d) Set

daaSignature:=($\zeta, T, c, n_t, s_v, s_{f_0}, s_{f_1}, s_e, (s_{a_i})_{i \notin X}$, attributeCommitments, signedPseudonym).

Remarks

Anonymity revocation and computation of commitments might be defined optional for low resource devices

4.3.4.30.17 Tspi_DAA_IssuerKeyVerify

Start of informative comment:

This function verifies the DAA public key of a DAA Issuer with respect to its associated proof.

This is a resource intensive task. It can be done by trusted third party (certification).

This is an optional function and does not require a TPM or a TCS.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_DAA_IssuerKeyVerify
(
   TSS_HDAA_CERTIFICATE  hDAACertificate,         // in
   TSS_HDAA_ISSUER_KEY   hIssuerKey,              // in
   TSS_BOOL*             isCorrect                // out
);
```

Parameters

hDAACertificate

Handle of the DAA Certificate object.

hIssuerKey

Handle of the DAA Issuer key object. Contains the issuer PK proof.

isCorrect

Proves the correctness of the DAA Issuer public key.

Return Values

TSS_SUCCESS
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Algorithm Specification

IssuerKeyVerify(`issuerKey`)

Input via TSS attributes:

$n, R_{0,} R_{1,} Y_{0,} .., Y_{l_h+l_i-1}, S, Z, \gamma, \Gamma, \rho, bsn_I :=$
TSS_TSPATTRIB_DAAISSUERKEYBLOB_PUBLIC_KEY

$c, \hat{x}_{(0,0)}, \ldots, \hat{x}_{(0,l_H-1)}, \ldots, \hat{x}_{(i,j)}, \ldots, \hat{x}_{(l_g-1,0)}, \ldots, \hat{x}_{(l_g-1,l_H-1)} :=$
TSS_TSPATTRIB_DAAISSUERKEYBLOB_PROOF

Output: isCorrect

Steps:

1. Check whether the following holds, otherwise return with isCorrect=FALSE
   (a) Γ and ρ are primes
   (b) ρ is a divisor of $(\Gamma - 1)$
   (c) ρ *is not a divisor of* $\frac{(\Gamma - 1)}{\rho}$
   (d) $\gamma^{\rho} \equiv 1 (mod\, \Gamma)$
2. Check whether all public key parameters have the required length, otherwise return isCorrect:=FALSE
3. Verify the proof
   issuerPkProof:= $(c, \hat{x}_{(0,0)}, \ldots, \hat{x}_{(0,l_H-1)}, \ldots \hat{x}_{(i,j)}, \ldots, \hat{x}_{(Il_g-1,0)}, \ldots, \hat{x}_{(l_g-1,l_h-1)})$
   that $R_{0,} R_{1,} Y_{0,.} .. Y_{l_h+l_i-1}$ and Z are correctly formed, namely
   Z, $R_{0,} R_{1,} Y_{0,.} .. Y_{l_h+l_i-1} \in \langle S \rangle$, as follows:
   (a) Set the number of elements to be verified to $l_g := 3 + l_h + l_i$
   (b) Compute $P_{(i,j)}$ where c_j is the j^{th} bit of c (c:= $c_{0,} c_{1,.} .., c_{l_h-1}$)
   for (j=0,…, $l_H - 1$)
      i. $P_{(0,j)} := Z^{c_j} * S^{\hat{x}_{(0,j)}} mod\, n$
      ii. $P_{(1,j)} := R_0^{c_j} * S^{\hat{x}_{(1,j)}} mod\, n$
      iii. $P_{(2,j)} := R_1^{c_j} * S^{\hat{x}_{(2,j)}} mod\, n$
      iv. for (i=3,…, $l_g - 1$)
         A. $P_{(i,j)} := Y_{i-3}^{c_j} * S^{\hat{x}_{(i,j)}} mod\, n$
   (c) Concatenate all $P_{(i,j)}$ together
   $c' := (P_{(0,0)} \| \ldots \| P_{(0,l_h-1)} \| \ldots \| P_{(i,j)} \| \ldots P_{(l_g-1,0)} \| \ldots \| \ldots P_{(l_g-1,l_H-1)})$
   (d) Verify if the following holds, otherwise return with isCorrect:=False
   $c = H(n \| S \| Z \| R_0 \| R_1 \| Y_0 \| \ldots \| Y_{l_h+l_i-1} \| c')$
4. Output isCorrect:=TRUE

Remarks

When using the DAA Issuer public key during the DAA join or sign protocol, Verification of that key is important. It assures the properties of the key which guarantees the privacy of the TCG platform.

4.3.4.30.18 Tspi_DAA_Issuer_GenerateKey

Start of informative comment:

This function is part of the **DAA Issuer** component. It defines the generation of a DAA Issuer public and private key. Further it defines the generation of a non-interactive proof (using the Fiat-Shamir heuristic) that the public keys were chosen correctly. The latter will guarantee the security requirements of the platform (and of its user), i.e., that the privacy and anonymity of signatures will hold.

The generation of the authentication keys of the DAA Issuer, which are used to authenticate (main) DAA Issuer keys, is not defined by this function.

For further information about the DAA Issuer's keys see Figure 12.

This is an optional function and does not require a TPM or a TCS.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_DAA_Issuer_GenerateKey
(
    TSS_HDAA_ISSUER_KEY   hIssuerKey,                  // in
    UINT32                issuerBaseNameLength,        // in
    BYTE*                 issuerBaseName,              // in
);
```

Parameters

hIssuerKey

Handle of the DAA Issuer key object. This object must have its TSS_TSPATTRIB_DAAISSUERPUBKEY_NUM_PLATFORM_ATTRIBS and TSS_TSPATTRIB_DAAISSUERPUBKEY_NUM_ISSUER_ATTRIBS sub-attributes set when this function is invoked. This function will load the sub-attributes for the private and public portions of the key as well as the proof of the DAA Issuer public key.

issuerBaseNameLength

Length of issuerBaseName

issuerBaseName

Unique name of the DAA Issuer

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Algorithm Specification

Issuer_GenerateKey(

issuerBaseName, numberPlatformAttributes, numberIssuerAttributes)

Input: issuerBaseName

Input via TSS attributes:

l_h:= TSS_TSPATTRIB_DAAISSUERPUBKEY_NUM_PLATFORM_ATTRIBS,

l_i:= TSS_TSPATTRIB_DAAISSUERPUBKEY_NUM_ISSUER_ATTRIBS

Output via TSS attributes:

TSS_TSPATTRIB_DAAISSUERKEYBLOB_PUBLIC_KEY:=issuer public key

TSS_TSPATTRIB_DAAISSUERKEYBLOB_SECRET_KEY:=issuer secret key

TSS_TSPATTRIB_DAAISSUERKEYBLOB_PROOF:=public key proof

Steps:

1. Choose an RSA modulus n = pq with p= 2p'+1, q=2q'+1 such that p,p',q,q' are all primes and n has l_n bits.
2. Choose a random generator S of QR_n (the group of quadratic residues modulo n) Do:
   (a) Choose a random $r \in_R \mathbb{Z}_n$
   (b) compute $S := r^2 \, mod \, n$
   WHILE (S=1 or $gcd(n, S-1) \neq 1$)
3. Compute $Z, R_{0,} R_{1,} Y_{0,..}, Y_{l_h+l_i-1} \in \langle S \rangle$
   (a) Set the number of elements to be computed to $l_g := 3 + l_h + l_i$
   (b) Choose random integers for each element to be computed
      i. $x_i \in [1, p'q'] \; for \, i = 0, \ldots, l_g - 1$
   (c) Compute the elements of $\langle S \rangle$
      i. $Z := S^{x_0} mod \, n$,
      ii. $R_0 := S^{x_1} mod \, n$
      iii. $R_1 := S^{x_2} mod \, n$
      iv. $Y_{i-3} := S^{x_i} mod \, n \qquad for \, i = 3, \ldots, l_g - 1$
4. Produce a non-interactive zero-knowledge proof that $Z, R_{0,} R_{1,} Y_{0,..}, Y_{l_h+l_i-1} \in \langle S \rangle$
   (a) Compute "commitment"
      i. Choose random $x_{\tilde{(i,0)}}, \ldots, x_{(i,\tilde{l_H}-1)} \in_R [1, p'q'] \; for \, i = 0, \ldots, l_g - 1$
      ii. Compute for (i=0,…, $l_g - 1$), for (j=0,…, $l_H - 1$, $P_{(i,j)} := S^{\hat{x}_{(i,j)}} mod \, n$
   (b) Compute "challenge"

i. Concatenate all P(i,j) together
$c' := (P_{(0,0)} \| \ldots \| P_{(0,l_H-1)} \| \ldots \| P_{(i,j)} \| \ldots P_{(l_g-1,0)} \| \ldots \| \ldots P_{(l_g-1,l_H-1)})$

ii. Compute hash $c := H(n \| S \| Z \| R_0 \| R_1 \| Y_0 \| \ldots \| Y_{l_h+l_i-1} \| c')$

(c) Compute "response", where c_j is the j^{th} bit of c $(c := c_{0,} c_{1,} \ldots, c_{l_H-1})$

for $(i=0,\ldots,l_g-1)$, $for(j=0,\ldots,l_H-1)$ $\hat{x}_{(i,j)} := \hat{x}_{(i,j)} - c_j * x_i \bmod p'q'$

(d) Compose the proof of the challenge and response

Set TSS_TSPATTRIB_DAAISSUERKEYBLOB_PROOF:=(
$c, \hat{x}_{(0,0)}, \ldots, \hat{x}_{(0,l_H-1)}, \ldots, \hat{x}_{(i,j)}, \ldots, \hat{x}_{(l_g-1,0)}, \ldots, \hat{x}_{(l_g-1,l_H-1)}$)

5. Generate a group of prime order

(a) Choose random primes

ρ *with length* l_ρ

Γ *with length* l_Γ

such that $\Gamma = r\rho + 1$ for some r and ρ is not a divisor of r

(b) Choose a random $\gamma' \in_R \mathbb{Z}_\Gamma^*$ such that $\gamma'^{\frac{(\Gamma-1)}{\rho}} \neq 1 (mod\ \Gamma)$ and set

$\gamma := \gamma'^{\frac{(\Gamma-1)}{\rho}} \bmod \Gamma$

6. Set TSS_TSPATTRIB_DAAISSUERKEYBLOB_PUBLIC_KEY:=(
$n, R_{0,} R_{1,} Y_{0,} \ldots, Y_{l_h+l_i-1}, S, Z, \gamma, \Gamma, \rho, bsn_I$)

7. Set TSS_TSPATTRIB_DAAISSUERKEYBLOB_SECRET_KEY:=(p'q')

Remarks

4.3.4.30.19 Tspi_DAA_Issuer_InitCredential

Start of informative comment:

This function is part of the **DAA Issuer** component. It's the first function out of 2 in order to issue a DAA Credential for a TCG Platform. It assumes that the endorsement key and its associated credentials are from a genuine and valid TPM. (Verification of the credentials is a process defined by the TCG Infrastructure WG.)

This is an optional function and does not require a TPM or a TCS.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_DAA_Issuer_InitCredential
(
   TSS_HDAA_ISSUER_KEY        hIssuerKey,                     // in
   TSS_HKEY                   hIssuerAuthPK,                  // in
   TSS_DAA_IDENTITY_PROOF*    identityProof,                  // in
   UINT32                     capitalUprimeLength,            // in
   BYTE*                      capitalUprime,                  // in
   UINT32                     daaCounter,                     // in
   UINT32*                    nonceIssuerLength,              // out
   BYTE**                     nonceIssuer,                    // out
   UINT32*                    authenticationChallengeLength,// out
   BYTE**                     authenticationChallenge,        // out
   UINT32*                    issuerJoinSessionLength         // out
   BYTE**                     issuerJoinSession               // out
);
```

Parameters

hIssuerKey

Handle of the DAA Issuer key (including the private and public portions).

hIssuerAuthPK

Root authentication (public) key of DAA Issuer

identityProof

Structure containing endorsement, platform and conformance credential of the TPM requesting the DAA Credential.

capitalUprimeLength

Length of capitalUprime which is $\frac{l_n}{8}$

capitalUprime

U'

daaCounter

DAA counter.

nonceIssuerLength

Length of nonceIssuer (20 bytes - - $\frac{l_H}{8}$).

nonceIssuer

Nonce of the DAA Issuer.

authenticationChallengeLength

Length of authenticationChallenge (256 bytes - - DAA_SIZE_NE[6]).

authenticationChallenge

Second nonce of the DAA Issuer that is encrypted by the endorsement public key. It is used as a challenge to authenticate the TPM.

issuerJoinSessionLength

Length of issuerJoinSession.

issuerJoinSession

On successful return, this buffer contains DAA Issuer join session information.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

Check the limitations of DAA Join, section 4.3.4.29.2.

Algorithm Specification

Issuer_InitCredential(

`hIssuerKey`, `hIssuerAuthPK`, `identityProof`,

`capitalUprime`, `daaCounter`)

Input: PK_{I_0} `:=hIssuerAuthPK`,
EK `:=identityProof`,
U' `:=capitalUprime`,
`daaCounter` .

Input via TSS attributes:

$PKDAA_I$ `:= TSS_TSPATTRIB_DAAISSUERKEYBLOB_PUBLIC_KEY`

$(p'q')$ `:= TSS_TSPATTRIB_DAAISSUERKEYBLOB_SECRET_KEY`

Output:

[6] See TPM Main - Part 2 TPM Structures specification

`nonceIssuer` $:= n_i$,
`authenticationChallenge` $:= \alpha$,
`issuerJoinSession` $:= (PK_{I_0}, EK, U', daaCounter, n_i, n_e)$

Steps:

1. Verify *EK* (and associated credentials) of the platform.
2. Determine if the platform has previously received a DAA Credential from this DAA Issuer with the same root authentication (RSA) key and if it has changed the `daaCounter` value. If both are true, accept or reject the request according to the DAA Issuer's policy. A different `daaCounter` value will allow establishing new pseudonyms with this DAA Issuer and any DAA Verifier. This affects frequency checks by a DAA Verifier and Anonymity Revocation of a DAA Credential.
   a. Permanently store *EK* and `daaCounter` with respect to the root authentication key, in order to do future verifications of the above.[7]
3. Choose a random nonce for the platform:
   $n_i \in \{0,1\}^{l_H}$.
4. Choose a random nonce n_e and encrypt it under the *EK* (The TPM will authenticate itself by its capability to decrypt the nonce):
   $n_e \in \{0,1\} l_H$,
   $\alpha := enc_{EK}(n_e)$.
5. Save $(PK_{I_0}, EK, U', daaCounter, n_i, n_e)$ in `issuerJoinSession`.
6. Output n_i, α .

[7] It is expected that the daaCounter check and storage of EK and daaCounter be done outside the scope of this API. This is to allow flexibility to the management of the data required for these operations. The actual check of daaCounter and storage of daaCounter and EK are marked as optional for implementers of the TSS.

4.3.4.30.20 Tspi_DAA_Issuer_IssueCredential

Start of informative comment:

This function is part of the **DAA Issuer** component. It's the last function out of 2 in order to issue a DAA Credential for a TCG Platform. It detects rogue TPM according to published rogue TPM DAA keys.

This is an optional function and does not require a TPM or a TCS.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_DAA_Issuer_IssueCredential
(
   TSS_HDAA_ISSUER_KEY          hIssuerKey,                // in
   TSS_DAA_CREDENTIAL_REQUEST*  credentialRequest,         // in
   UINT32                       issuerJoinSessionLength,   // in
   BYTE*                        issuerJoinSession,         // in
   TSS_DAA_CRED_ISSUER**        credIssuer                 // out
);
```

Parameters

hIssuerKey

Handle of the DAA Issuer key. This contains the Issuer's public and secret keys, and the Issuer attributes and the number of of them.

credentialRequest

Credential request of the Platform, it contains the blinded DAA public key of the platform on which the DAA Issuer will issue the credential with the blinded attributes chosen by the Platform.

issuerJoinSessionLength

Length of issuerJoinSession.

issuerJoinSession

This buffer contains DAA Issuer Join session information.

credIssuer

This structure contains the DAA Credential issued by the DAA Issuer, the proof of correctness of the credential and the attributes chosen by the DAA Issuer.

Return Values

TSS_SUCCESS
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR
TSS_E_DAA_AUTHENTICATION_ERROR
TSS_E_DAA_PSEUDONYM_ERROR
TSS_E_DAA_CREDENTIAL_REQUEST_PROOF_ERROR

Remarks

Check the limitations of DAA Join, section 4.3.4.29.2.

Algorithm Specification

Issuer_IssueCredential(hIssuerKey, credentialRequest, issuerJoinSession)

Input: :=attributesIssuer ,

$(U, N_I, a_{U'}, c, n_t, n_h, s_{f_0}, s_{f_1}, s_{\nu'}, s_{\tilde{\nu}'}, s_{a_0}, \ldots, s_{a_{l_h-1}}, \beta_{0,} .. \beta_{l_c-1}, s_{\mu_0}, \ldots s_{\mu_{l_c-1}})$

:=credentialRequest,

(PK_{I_0}, EK, U', daaCounter, n_i, n_e)

:=issuerJoinSession .

Input via TSS attributes:

issuerPK := TSS_TSPATTRIB_DAAISSUERKEYBLOB_PUBLIC_KEY

$(p'q')$:= TSS_TSPATTRIB_DAAISSUERKEYBLOB_SECRET_KEY

$a_{l_k}, \ldots, a_{l_k+l_i-1}$

:= TSS_TSPATTRIB_DAAISSUERPUBKEY_NUM_ISSUER_ATTRIBS

Commitment related input

l_c :=TSS_TSPATTRIB_DAACOMMIT_NUMBER

$C_{0,} .. C_{l_c-1}$:=TSS_TSPATTRIB_DAACOMMIT_SELECTION ,

γ_i :=(MGF1($i \| \gamma$, LENGTH_MGF1_GAMMA)) $^{\frac{\Gamma-1}{\rho}}$ for $\mod \Gamma$

$for\ i \in C_0 \cup \ldots C_{l_c-1}$. [8]

*Output:*daaCredIssuer := $((A, e, \nu''), a_{l_h}, \ldots, a_{l_h+l_i-1}, (c', s_e))$.

Steps:

1. Check if the TPM is rogue[9]
    a. Check for all $(f_{0,} f_1)$ on the rogue list whether $N_I \notin (\zeta^{f_0+f_1*2^{l_f}})(\mod \Gamma)$. Also check this using the *EK* for the N_I (pseudonym) this platform had used previously. If either of it is true return the error code TSS_E_DAA_PSEUDONYM_ERROR.

---

[8] The variable i is an index of an attribute to which a commitment will be computed. Its length is 4 bytes (UINT32). Computation of γ_i is only required for attributes that are part of a commitment.

[9] It is expected that the rogue TPM check and the storage of Ni and EK be done outside the scope of this API. This is to allow flexibility to the management of the data required for these operations. The actual check for rogueness and storage of Ni and EK is marked as optional for implementers of TSS.

b. Permanently store N_I and EK in order to do future verifications.

2. Verify if the authentication proof of the TPM is correct, otherwise return the error code `TSS_E_DAA_AUTHENTICATION_ERROR`.

a. Compute[10]
$C_t := H_{TPM}(U' \| daaCounter \| H_{TPM}(PK_{I_0}))$

b. Verifiy
$a_{U'} = H_{TPM}(c_t \| n_e)$

3. Verify the correctness proof of the credential request. Compute[11]:

a. Compute the commitment related part
for $(j=0,\ldots,l_c-1)$

i. set γ, γ_i, Γ as per TSS_ATTRIB_DAA_COMMIT_PARAM[j]

ii. $\tilde{\beta}_j := \beta_j^{-c} \gamma_{s_{\mu_j}} \prod_{i \in C_j} \gamma_i^{s_{a_i}} \bmod \Gamma$,

b. $\tilde{U}' := U'^{-c} R_0^{s_{f_0}} R_1^{s_{f_1}} S^{s_{v'}} \bmod n$,

c. $\tilde{U} := \left(\frac{U}{U'}\right)^{-c} S^{s_{\tilde{v}'}} \prod_{i=0}^{l_h-1} Y_i^{s_{a_i}} \bmod n$,

d. $\tilde{N}_I := N_I^{-c} \zeta_I^{s_{f_0}+2^{l_f} * s_{f_1}} \bmod \Gamma$.

e. $c_h := H_{TPM}$ (issuerPk
$\| U \| U' \| \tilde{U} \| \tilde{U}' \| N_I \| \tilde{N}_i \| l_c \| (\beta_0 \| \tilde{\beta}_0) \| \ldots \| (\beta_{l_c-1} \| \tilde{\beta}_{l_c-1}) \| n_i)$,

f. Verify if the following holds, otherwise return the error code `TSS_E_DAA_CREDENTIAL_REQUEST_PROOF_ERROR`:

i. $c = H_{TPM}(c_h \| n_t) \in [0, 2^{l_h} - 1]$,

ii. $s_{f_0}, s_{f_1}, s_{a_1}, \ldots, s_{a_{l_h}} \in \{0,1\}^{l_f + l_\Phi + l_H + 1}$,

iii. $s_{v'}, s_{\tilde{v}'} \in \{0,1\}^{l_n + 2l_\Phi + l_H - 1}$.

4. Compute the credential for the TCG platform.

a. Compute a random number v'' with length l_v:
$\hat{v} \in \{0,1\}^{l_v - 1}$,
$v'' := \hat{v} + 2^{l_v - 1}$.

---

[10] Extend the byte representation of U' to 256 bytes (Size of `TPM_DAA_CONTEXT.DAA_Scratch`). The length of the variable `daaCounter` is 4 bytes. The variable PK_I means the 256 bytes of the RSA modulus of the first key in the key chain PK_I.

[11] The fraction can be computed as a multiplication with the multiplicative inverse of the divisor modulo n.

b. Choose a prime number:
$e \in \left[2^{l_e - 1}, 2^{l_e - 1} + 2^{l'_e - 1} \right]$.

c. Compute[12]

$$A := \left(\frac{Z}{U S^{v''} \prod_{i=l_h}^{l_h + l_i - 1} Y_i^{a_i}} \right)^{\frac{1}{e}} mod\ n$$

5. Prove that A was computed correctly:

a. Choose random number
$r_e \in \left[0, p'q' \right]$.

b. Compute

$$\tilde{A} := \left(\frac{Z}{U S^{v''} \prod_{i=l_h}^{l_h + l_i - 1} Y_i^{a_i}} \right)^{r_e} mod\ n ,$$

c. $c' = H_{TPM}($`issuerPk` $\| v'' \| A \| \tilde{A} \| n_h$,

d. $s_e := r_e - \frac{c'}{e} mod\ p'q'$.

6. Output `daaCredIssuer:=` $((A, e, v''), a_{l_h}, \ldots, a_{l_h + l_i - 1}, (c', s_e))$.

4.3.4.30.21 Tspi_DAA_Verifier_Init

Start of informative comment:

This function is part of the **DAA Verifier** component. It's the first function out of 2 in order to verify a DAA Credential of a TCG platform. It creates a challenge for the TCG platform.

This is an optional function and does not require a TPM or a TCS.

End of informative comment.

[12] Note: 1/e in the exponent is computed as the multiplicative inverse of e modulo p'q'. The division in the big fraction can be computed as a multiplication with the multiplicative inverse of the denominator modulo n.

Definition:

```
TSS_RESULT Tspi_DAA_Verifier_Init
(
    TSS_HDAA_CREDENTIAL    hDAACredential,           // in
    UINT32*                nonceVerifierLength,      // out
    BYTE**                 nonceVerifier             // out
    UINT32*                baseNameLength,           // out
    BYTE**                 baseName,                 // out
);
```

Parameters

hDAACredential

Handle of a DAA Credential object. For this function, no attributes of this object need to be set.

nonceVerifierLength

Length of nonceVerifier

nonceVerifier

A challenge for the platform

baseNameLength

Length of baseName

baseName

The base name that was chosen for the DAA Signature.

Return Values

TSS_SUCCESS
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Algorithm Specification

Verifier_Init(hDAACredential)

Input: $\perp$.

Output: `nonceVerifier`. `baseName`

Steps:

1. Compute a random nonce
   `nonceVerifier` $\in \{0,1\}^{l_H}$
2. Choose `baseName` to be either `baseName` $\in [0,1]$ (Preferably a unique string of the verifier, which can change often) or

   `baseName` = $\perp$. (The platform will use a random base name).
3. Output `nonceVerifier`.

4.3.4.30.22 Tspi_DAA_VerifySignature

Start of informative comment:

This function is part of the **DAA Verifier** component. It's the last function out of 2 in order to verify a DAA Credential of a TCG platform. It verifies the DAA Credential and detects public rogue TPMs.

This is an optional function and does not require a TPM or a TCS.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_DAA_VerifySignature
(
    TSS_HDAA_CREDENTIAL   hDAACredential,             // in
    TSS_HDAA_ISSUER_KEY   hIssuerKey,                 // in
    TSS_HDAA_ARA_KEY      hARAKey,                    // in
    TSS_HHASH             hARACondition,              // in
    UINT32                attributesLength,           // in
    UINT32                attributesLength2,          // in
    BYTE**                attributes,                 // in
    UINT32                verifierNonceLength,        // in
    BYTE*                 verifierNonce,              // in
    UINT32                verifierBaseNameLength,     // in
    BYTE*                 verifierBaseName,           // in
    TSS_HOBJECT           signData,                   // in
    TSS_DAA_SIGNATURE*    daaSignature,               // in
    TSS_BOOL*             isCorrect                   // out
);
```

Parameters

hDAACredential

Handle of the DAA Credential object holding the credential. For this function only the set of attributes committed to in the credential (but not their values) need to be provided.

hIssuerKey

Handle of the DAA Issuer Key object. This contains the public key of the DAA Issuer as an attribute.

hARAKey

Handle of the DAA Anonymous Revocation Authority Key object. This contains the public key of the DAA ARA as an attribute. If this handle is NULL, anonymous revocation is not used.

hARACondition

Handle of a hash object containing a digest representing the conditions under which the ARA should reveal the signer's pseudonym. This handle should be NULL if and only if hARAKey is NULL.

attributesLength

The number of elements in the attributes array. This value should match the value in the DAA Issuer public key $(l_h + l_i)$.

attributesLength2

The number of bytes in each revealed attribute. This value should be $\frac{l_f}{8}$.

attributes

Array of attributes for the DAA Credential. Each element will be NULL if the attribute is not revealed to the verifier, and will be a pointer to an array of attributessLength2 BYTEs if the attribute is revealed to the verifier.

verifierNonceLength

Length of verifierNonce (20 bytes : $\frac{l_H}{8}$).

verifierNonce

The nonce that was computed in the previous function.

verifierBaseNameLength

Length of baseName

verifierBaseName

The base name that was chosen for the DAA Signature.

signData

The data that was signed. This can be either a TSS_HHASH or a TSS_HKEY object. If it is a TSS_HKEY object, the key must be an identity key and only the public part of the key is required.

daaSignature

DAA signature contains proof of ownership of the DAA Credential, as well as a signature on either an AIK or a message.

isCorrect

Denotes if the verification of the DAA Signature was successful.

Return Values

TSS_SUCCESS
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Algorithm Specification

Verification of a DAA Signature `daaSignature` and a message `signData` with respect to the public keys $PKDAA_I$ and $(\eta, \lambda_1, \lambda_2, \lambda_3)$ is defined as follows:

VerifySignature(`hDAACrednetial, hIssuerKey, hARAKey, hARACondition, attributes, verifierNonce, verifierBaseName, signData, daaSignature`)

Input: $(\zeta, T, c, n, s_v, s_{f_0}, s_{f_1}, s_e, (s_{a_i})_{i \notin X})$ `attributeCommitments, signedPseudonym`)

:= `daaSignature` ,

$a_{0,..}, a_{l_h+l_i-1}$:= `attributes` ,

n_v := `nonceVerifier`,

bsn_v := `baseName`.

X : Is the set of the indices of the attributes that the credential owner has revealed, i.e., the set of i where $a_i \neq \perp$.

b_2 := `TRUE (1) if hARAKey is non-NULL, FALSE (0) if hARAKey is NULL.`

Input via TSS attributes:

Issuer public key

$PKDAA_I$:= `TSS_TSPATTRIB_DAAISSUERKEYBLOB_PUBLIC_KEY`

Commitment related input

l_c := `TSS_TSPATTRIB_DAACOMMIT_NUMBER`

$C_{0,..}, C_{l_c-1}$:= `TSS_TSPATTRIB_DAACOMMIT_SELECTION`

γ_i :=(`MGF1`($i \| \gamma$, `LENGTH_MGF1_GAMMA`)) $^{\frac{(\Gamma-1)}{\rho}}$ $mod\, \Gamma$

for [13] $i \in C_0 \cup \ldots C_{l_c-1}$

`signData`,

If Anonymity Revocation is enabled

($hARAKey \neq \perp$) then set

$(\eta, \lambda_1, \lambda_2, \lambda_3)$:= `TSS_TSPATTRIB_DAAARAKEYBLOB_PUBLIC_KEY`,

L := `hARACondition` .

Output: `isCorrect` .

[13] The variable i is an index of an attribute to which a commitment will be computed. Its length is 4 bytes (UINT32). Computation of γ_i is only required for attributes that are part of a commitment.

Steps:

1. If $bsn_v \neq \perp$ then check whether the following holds, otherwise output `FALSE`

   $\zeta =$ (`MGF1(` bsn_v `, LENGTH_MGF1_GAMMA))` $^{\frac{(\Gamma-1)}{\rho}}$ $mod\, \Gamma$.

2. Compute

   $$\tilde{T} := \left(\frac{Z}{\prod_{i \in X} Y_i^{a_i}} \right)^{-c} T^{s_e + c2^{l_e-1}} R_0^{s_{f_0}} R_1^{s_{f_1}} S^{s_v} \prod_{i \notin X} Y_i^{s_{a_i}} \bmod n \quad .$$

3. Compute the commitment related part for $(j = 0, \ldots, l_c - 1)$

   i. set γ, γ_i, Γ according to TSS_ATTRIB_DAA_COMMIT_PARAM[j]

   ii. $\tilde{\beta}_j := \beta_j^{-c} \gamma_{s_{\mu_j}} \prod_{i \in C_j} \gamma_i^{s_{a_i}} \bmod \Gamma$, .

4. Verify the pseudonym related part of the DAA Signature:

   a. Compute
   $s_f := s_{f_0} + s_{f_1} 2^{l_f}$

   b. If b_2 = `FALSE` then

      i. Verify[14] whether the following holds, otherwise output `FALSE`
      $N_v, \zeta \in \langle \gamma \rangle$.

      ii. Compute
      $\tilde{N}_v := N_v^{-c} \zeta^{s_f} \bmod \Gamma$.

   c. Else (b_2 = `TRUE`) :

      i. Verify[29] whether the following holds, otherwise output `FALSE`

      $\delta_1, \delta_2, \delta_3, \zeta \in \langle \gamma \rangle$.

      ii. Compute
      $\tilde{\delta}_1 := \delta_1^{-c} \gamma^{s_\tau} \bmod \Gamma$,
      $\tilde{\delta}_2 := \delta_2^{-c} \eta^{s_\tau} \bmod \Gamma$,
      $\tilde{\delta}_3 := \delta_3^{-c} \lambda_3^{s_\tau} \zeta^{s_f} \bmod \Gamma$,
      $\tilde{u} :=$ `MGF1(` $(\tilde{\delta}_1 \| \tilde{\delta}_2 \| \tilde{\delta}_3 \| L)$, `LENGTH_MGF1_AR`)
      $\tilde{\delta}_4 := \delta_4^{-c} \lambda_1^{s_\tau} \lambda_2^{s_\tau \tilde{u}} \bmod \Gamma$.

5. Call the callback function `Tspicb_DAA_VerifySignature` if the callback mechanism is set in the DAA object[15].

---

[14] The check $x \in \langle \gamma \rangle$ can be done by raising x to the order of γ (which is ρ) and checking whether the result is 1.

[15] This callback function allows extending the proofs realized by the commitments and anonymity revocation. Therefore at least `TSS_TSPATTRIB_DAA_SIGN_COMMIT` or

a. `isCorrect` :=`Tspicb_DAA_VerifySignature`(`c`, `additionalProof`, hPubKeyIssuer,
$((\gamma_{0,}..,\gamma_{l_h+l_i}),(s_{a_i},\ldots,s_{a_{l_h+l_i}}),(\beta_{0,}s_{\mu_0}),\ldots,(\beta_{l_c-1},s_{\mu_{l_c-1}}),(\tilde{\beta}_{0,}\tilde{\beta_{l_c-1}}))$,
$\{\zeta,s_f,\delta_{1,}\delta_{2,}\delta_{3,}\delta_{4,}s_\tau,\tilde{\delta_{1,}}\tilde{\delta_{2,}}\tilde{\delta_{3,}}\tilde{\delta_4}\}_{b_2}$) ,

b. If `isCorrect` = `FALSE`, then output `FALSE` .

6. Verify the “response” to the “challenge”:

a. Compute[16]
$c_h := H_{TPM}$ ($PKDAA_I \| n_v \| X \| l_c \| b_2 \| b_b \| \zeta \| T \| \tilde{T}$
$\| (C_0 \| \beta_0 \| \tilde{\beta}_0) \| \ldots \| (C_{l_c-1} \| \beta_{l_c-1} \| \tilde{\beta_{l_c-1}})$
$\| \{N_v \| \tilde{N}_V\}_{\neg b_2} \| \{(\eta \| \lambda_1 \| \lambda_2 \| \lambda_3) \| (\delta_1 \| \delta_2 \| \delta_3 \| \delta_4) \| (\tilde{\delta}_1 \| \tilde{\delta}_2 \| \tilde{\delta}_3 \| \tilde{\delta}_4)\}_{b_2}$
$\| \{c_b\}_{b_0}$).

b. Compute
$c := H_{TPM}(c_h \| n_t)$.

c. Set the payload and selector:

if (signData is a TSS_HHASH)
  selector := 1
  payload := signData hash value
else if (signData is a TSS_HKEY)
  selector := 0
  payload := the RSA modulus for the signData key.
else
  return error TSS_E_BAD_PARAMETER.

d. Verify whether the following holds, otherwise output `FALSE`

i. $c = H_{TPM}$ ($c_t \|$ `selector` $\|$ `payload`),

ii. $s_{f_0}, s_{f_1}, (s_{a_i})_{i \in X} \in \{0,1\}^{l_f + l_\Phi + l_H + 1}$,

iii. $s_e \in \{0,1\}^{l'_e + l_\Phi + l_H + 1}$.

7. Verify if the “TPM DAA key” is on the rogue list.
If b_2 = `FALSE` then for all $(f_{0,} f_1)$ on the rogue list check[17] whether the following holds, otherwise output `FALSE`
$N_v \neq \zeta^{f_0 + f_1 2^{l_f}} \mathit{mod}\, \Gamma$. [18]

8. Output `TRUE` .

`anonymityRevocationEnabled` has to be equal to `TRUE`.

[16] The Boolean values $b_{1,} b_2$ and b_b are represented as byte values when input to the hash function. `FALSE` is equal to 0 and `TRUE` equals to 1.

[17] In case §is random, one can apply so called batch verification techniques to obtain a considerable speed-up of the verification step. Also note that the involved exponents are relatively small. Finally, if ζ §is not random, one could precompute $\zeta^{f_0 + f_1 2^{l_f}}$ §for all $(f_{0,} f_1)$ § on the rogue list.

4.3.4.30.23 Tspi_DAA_ARA_GenerateKey

Start of informative comment:

This function is part of the **DAA Anonymity Revocation Authority** component. It creates the public and private key to verifiably encrypt and to decrypt the pseudonym with respect to a DAA Signature. The keys can only be used with a certain DAA public key.

Verifiable encryption allows the DAA Verifier of the DAA Signature to verify that indeed the pseudonym was encrypted and that the correct public key was used .

This is an optional function and does not require a TPM or a TCS.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_DAA_ARA_GenerateKey
(
    TSS_HDAA_ISSUER_KEY         hIssuerKey,            // in
    TSS_HDAA_ARA_KEY            hARAKey                // in
    );
```

Parameters

hIssuerKey

Handle of the DAA Issuer Key object, which contains the issuer's public key as an attribute.

hARAKey

Handle of the DAA ARA Key object. Use *Tspi_GetAttribData* to retrieve the public key or secret key on a successful return.

Return Values

TSS_SUCCESS
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

[18] It is expected that the rogue check be done outside the scope of this API. This is to allow flexibility to the management of the data required for these operations. The actual check for rogueness is marked as optional for implementers of TSS.

Algorithm Specification

ARA_GenerateKey(`IssuerKey`)

Input via TSS attributes:

γ, Γ, ρ := TSS_TSPATTRIB_DAA_ISSUER_PK

Output via TSS attributes:

TSS_TSPATTRIB_DAAARAKEYBLOB_SECRET_KEY := $(x_{0,}\ldots, x_5)$.

TSS_TSPATTRIB_DAAARAKEYBLOB_PUBLIC_KEY := $(\eta, \lambda_{1,} \lambda_{2,} \lambda_3)$.

Steps:

1. Compute the secret key by choosing the following integers randomly $x_{0,}\ldots, x_5 \in_R \mathbb{Z}_\rho$
2. Compute the public key
   a. $\eta := \gamma^{x_0} mod\ \Gamma$,
   b. $\lambda_1 := \gamma^{x_1} \eta^{x_2} mod\ \Gamma$,
   c. $\lambda_2 := \gamma^{x_3} \eta^{x_4} mod\ \Gamma$,
   d. $\lambda_3 := \gamma^{x_5} mod\ \Gamma$.
3. Set TSS_TSPATTRIB_DAAARAKEYBLOB_SECRET_KEY := $(x_{0,}\ldots, x_5)$ and set TSS_TSPATTRIB_DAAARAKEYBLOB_PUBLIC_KEY := $(\eta, \lambda_{1,} \lambda_{2,} \lambda_3)$.

4.3.4.30.24 Tspi_DAA_ARA_RevokeAnonymity

Start of informative comment:

This function is part of the **DAA Anonymity Revocation Authority** component. It defines the Cramer-Shoup decryption algorithm to revoke the anonymity of a DAA Signature. The pseudonym, with respect to either the DAA Verifier's base name, the DAA Issuer's base name or (just for completeness) a random base name, can be revealed.

The pseudonym with respect to a DAA Signature and the used base name is N_v. An encryption of N_v is the tuple $(\delta_{1,} \delta_{2,} \delta_{3,} \delta_4)$ and is decrypted using the secret key ($x_{0,} .., x_5$), the decryption condition and the DAA public key.

This is an optional function and does not require a TPM or a TCS.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_DAA_ARA_RevokeAnonymity
(
    TSS_HDAA_ARA_KEY              hARAKey,                  // in
    TSS_HHASH                     hARACondition,            // in
    TSS_HDAA_ISSUER_KEY           hIssuerKey,               // in
    TSS_DAA_PSEUDONYM_ENCRYPTED*  encryptedPseudonym,       // in
    TSS_DAA_PSEUDONYM_PLAIN**     pseudonym                 // out
);
```

Parameters

hARAKey

Handle of the DAA ARA Key object. This contains the private key of the DAA Anonymity Revocation Authority as an attribute.

hARACondition

Handle of the hash object containing a digest representing the conditions under which anonymity can be revoked, and therefore the conditions for the decryption of the pseudonym.

hIssuerKey

Handle of the DAA Issuer Key object. This contains the public key of the DAA Issuer as an attribute.

encryptedPseudonym

The encrypted pseudonym.

pseudonym

On successful return, this parameter returns the decrypted pseudonym structure.

Return Values

TSS_SUCCESS
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR
TSS_E_DAA_AR_DECRYPTION_ERROR

Remarks

The Tspi_DAA_ARA_RevokeAnonymity method allocates memory blocks for the decrypted pseudonym. This memory must be released utilizing the Tspi_Context_FreeMemory method.

Algorithm Specification

ARA_RevokeAnonymity(`ARAKey`, `arCondition`, `IssuerKey`, `encryptedPseudonym`):

Input: $(\delta_{1,}\delta_{2,}\delta_{3,}\delta_4)$:=`encryptedPseudonym`,
L :=`arCondition`

Input via TSS attributes:
$x_{0,..}, x_5$:=TSS_TSPATTRIB_DAA_ARA_SK,
Γ, ρ :=TSS_TSPATTRIB_DAA_ISSUER_PK

Output:`pseudonym` := N_v .

Steps:

1. Compute
   $u := \text{MGF1}(\delta_1 \| \delta_2 \| \delta_3 \| L, \text{LENGTH\_MGF1\_AR})$
2. Check whether the following holds:
   $\delta_1^{x_1+x_3u}\delta_2^{x_2+x_4u} \equiv \delta_4 (mod\,\Gamma)$ and $\delta_2 \equiv \delta_1^{x_0} (mod\,\Gamma)$
3. Decrypt if above condition is true:
   a. Compute
      $$N_v := \frac{\delta_3}{\delta_1^{x_5}} mod\,\Gamma$$
   b. Output N_v .

else

   a. Return `TSS_E_DAA_AR_DECRYPTION_ERROR`.

4.3.4.31 Audit Commands:

4.3.4.31.1 Tspi_SetAttribUint32

Start of informative comment:

This method sets a 32-bit attribute of the TPM object.

End of informative comment.

Definition:

See section 4.3.2 for definition.

Parameters

See section 4.3.2 for description.

Defined Attributes

Flag	SubFlag	Attribute	Description
TSS_TSPATTRIB_TPM_ORDINAL_AUDIT_STATUS	TPM_CAP_PROP_TPM_SET_ORDINAL_AUDIT	TPM_COMMAND_CODE	The value of the ordinal which will be added to the audit list.
	TPM_CAP_PROP_TPM_CLEAR_ORDINAL_AUDIT	TPM_COMMAND_CODE	The value of the ordinal which will be cleared from the audit list.

4.3.4.31.2 Tspi_TPM_GetAuditDigest

Start of informative comment:

This method returns the current audit digest. This value may be unique to an individual TPM.

End of informative comment.

Definition:

```
TSS_RESULT Tspi_TPM_GetAuditDigest
   (
   TSS_HTPM             hTPM,                // in
   TSS_HKEY             hKey,                // in
   TSS_BOOL             closeAudit,          // in
   TPM_DIGEST*          pAuditDigest,        // out
   TPM_COUNTER_VALUE*   pCounterValue,       // out
   TSS_VALIDATION*      pValidationData,     // out
   UINT32*              ordSize,             // out
   UINT32**             ordList              // out
   );
```

Parameters

hTPM

Handle of the TPM object

hKey

Handle of the key that may be used sign the audit data. If this value is NULL the TPM function called will be TPM_GetAuditDigest. If this value is non-NULL this is the handle of the key used to sign the audit digest.

closeAudit

A flag indicating whether or not to close the current audit digest after it is signed. This value is only used if hKey is not NULL.

pAuditDigest

The digest of all audited events.

pCounterValue

The current value of the audit monotonic counter.

pValidationData

This value is ignored if hKey is NULL. If hKey is non-NULL this field holds the information necessary to verify the signature. On input the ExternalData field should be the antiReplay nonce for the signing operation. On successful completion of this function the pValidationData->Data field will contain the data that was signed (a byte-array encoding of the TPM_SIGN_INFO structure with the TPM_SIGN_INFO.data field set to (auditDigest || counterValue || auditedOrdinalDigest)), and TSS_ pValidationData->ValidationData will be set to the signature returned by the TPM.

ordSize

The number of the audited ordinals in the ordinals list. This is only returned if hKey is NULL.

ordList

The list of the audited ordinals. This is only returned if hKey is NULL.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

This command is used to retrieve the audit digest. The digest may be signed, in which case the current audit digest, the current audit counter, the hash of the audited ordinal list, and a signature are returned, or the digest may be unsigned, in which case the current audit digest, the current audit counter, the full list of audited ordinals is returned.

If the unsigned audit digest is returned, the TSP may need to make multiple calls to the TCS to retrieve the full audited ordinal list. The TSP MUST ensure that it returns the complete audited ordinal list.

The audit functions are optional in the TPM specification, so this command may not be supported by all TPMs.

4.3.4.32 Callback Function Definitions

4.3.4.32.1 Tspicb_CallbackHMACAuth

Start of informative comment:

This method is called each time when authorized TPM commands are called and the callback mechanism is set in the assigned policy object. In functions where there is only one object with a usage policy, this usage policy object will have the necessary attribute for the callback. In some cases, the TSP may internally use an OSAP session, as required by the function. Section Tspicb_CallbackXorEnc 4.3.4.32.2 defines which functions *require* OSAP and also defines which policy must hold the callback pointer for the xor encrypt. When one of these functions is called, then the same policy object will hold the pointer for this function as well if desired.

When a change of authorization is being done, then it is possible that this function may be registered to two different policy objects. Before a change of authorization, the callback, when required, must be registered to the usage policy of the existing object. For the verify, the callback should be registered to the *new* usage policy if a callback is required.

When the parameter ReturnOrVerify is TRUE, the callback must calculate the HMAC data, if FALSE the callback must verify the HMAC data returned from the TPM.

The two pointers rgbNonceEvenOSAP, rgbNonceOddOSAP are only valid, if the service provider uses an internally OSAP session. In this case the shared secret must be used for the HMAC.

End of informative comment.

Definition:

```
TSS_RESULT Tspicb_CallbackHMACAuth
   (
   PVOID          lpAppData,          // in
   TSS_HOBJECT    hAuthorizedObject,  // in
   TSS_BOOL       ReturnOrVerify,     // in
   UINT32         ulPendingFunction , // in
   TSS_BOOL       ContinueUse,        // in
   UINT32         ulSizeNonces,       // in
   BYTE*          rgbNonceEven,       // in
   BYTE*          rgbNonceOdd,        // in
   BYTE*          rgbNonceEvenOSAP,   // in
   BYTE*          rgbNonceOddOSAP,    // in
   UINT32         ulSizeDigestHmac ,  // in
   BYTE*          rgbParamDigest,     // in
   BYTE*          rgbHmacData         // in, out
   );
```

Parameters

lpAppData

Pointer to application provided data as provided on registration of callback function

hAuthorizedObject

Handle to the object authorization is required

ReturnOrVerify

Flag indicating authorization or verification is required.

(CalculateHMACData)

TRUE: the callback must calculate the HMAC data

FALSE: callback must verify the HMAC data returned from the TPM.

ulPendingFunction

Ordinal number of TPM command for which the HMAC must be calculated.

ContinueUse

The continue use flag for the authorization session. Required to calculate or verify the rgbHmacData

ulSizeNonces

The size of the nonces rgbNonceEven, rgbNonceOdd, rgbNonceEvenOSAP and rgbNonceOddOSAP

rgbNonceEven

Even nonce previously generated by TPM to cover inputs. Required to calculate or verify the rgbHmacData

rgbNonceOdd

Nonce generated by TSP associated with the authorization session. Required to calculate or verify the rgbHmacData

rgbNonceEvenOSAP

Nonce generated by TPM and associated with shared secret. Required to calculate the shared secret for the OSAP session.

rgbNonceOddOSAP

The nonce generated by the caller associated with the shared secret. Required to calculate the shared secret for the OSAP session.

ulSizeDigestHmac

The size of the parameter rgbParamDigest and rgbHmacData

rgbParamDigest

SHA1 digest of the TPM function parameters.

If ReturnOrVerify = TRUE, digest of incoming parameters.

If ReturnOrVerify = FALSE, digest of ingoing parameters.

rgbHmacData

The authorization digest for inputs or returned parameters.

If ReturnOrVerify = TRUE, authorization digest required to process the TPM command

If ReturnOrVerify = FALSE, authorization digest returned from the TPM

Example:

```
TSS_RESULT Tspicb_CallbackHMACAuth
(
   PVOID       lpAppData,            // in
   TSS_HOBJECT   hAuthorizedObject, // in
   TSS_BOOL    ReturnOrVerify,       // in
   UINT32      ulPendingFunction,    // in
   TSS_BOOL    ContinueUse,          // in
   UINT32      ulSizeNonces,         // in
   BYTE*       rgbNonceEven,         // in
   BYTE*       rgbNonceOdd,          // in
   BYTE*       rgbNonceEvenOSAP,     // in
   BYTE*       rgbNonceOddOSAP,      // in
   UINT32      ulSizeDigestHmac,     // in
   BYTE*       rgbParamDigest,       // in
   BYTE*       rgbHmacData           // in, out

{
   // Get secret from user for hAuthorizedObject
   // (e.g. via application dialog)

   BYTE *pbSecret = SHA1(…)
   BYTE *pHmacDataTemp= HMAC(pbSecret, DataToHMAC);

   if (ReturnOrVerify)
   {
        memcpy(rgbHmacData,pHmacDataTemp, ulSizeDigestHmac);
   }
   else
   {
        if (memcmp(rgbHmacData, pHmacDataTemp, ulSizeDigestHmac))
             return TSS_E_FAIL;
   }

   return TSS_SUCCESS;
}
```

4.3.4.32.2 Tspicb_CallbackXorEnc

Start of informative comment:

This method is called when one of the following TSPI function is called and the callback mechanism is set in the assigned policy object. A flag indicates the purpose of the call. That means, in one case a new secret must be inserted and in the other case an existing secret must be changed.

Some functions require OSAP sessions and for others it is optional to use OIAP or OSAP. For the functions that require OSAP, there is potential confusion as to which policy object should hold the callback pointer when desired. This section defines which policy object will hold the callback pointer when desired.

TSPI functions:

Tspi_Data_Seal

The callback is registered to the usage policy of the enc data object.

Tspi_Key_CreateKey (NEW; Usage and migration secret)

The callback is registered to the usage policy of the parent key object

Tspi_TPM_CollateIdentityRequest

The callback is registered to the usage policy of the TPM object.

Tspi_ChangeAuth

The callback is registered to the usage policy of the object being changed.

Refer to TCG 1.1b Main Specification for information about the encryption process

End of informative comment.

Definition:

```
TSS_RESULT Tspicb_CallbackXorEnc
(
  PVOID        lpAppData,            // in
  TSS_HOBJECT   hOSAPObject,         // in
  TSS_HOBJECT   hObject,             // in
  TSS_FLAG     PurposeSecret,        // in
  UINT32       ulSizeNonces,         // in
  BYTE*        rgbNonceEven,         // in
  BYTE*        rgbNonceOdd,          // in
  BYTE*        rgbNonceEvenOSAP,     // in
  BYTE*        rgbNonceOddOSAP,      // in
  UINT32       ulSizeEncAuth,        // in
  BYTE*        rgbEncAuthUsage,      // out
  BYTE*        rgbEncAuthMigration   // out
);
```

Parameters

lpAppData

Pointer to application provided data as provided on registration of callback function

hOSAPObject

Handle to the object authorization is required

hObject

Handle to the object the secret should be set or changed

PurposeSecret

Flag indicating the whether a new secret must be inserted or an existing secret must be changed.

(NewSecret)

TRUE: New secret must be inserted

FALSE: Existing secret must be changed

ulSizeNonces

The size of the nonces rgbNonceEven, rgbNonceOdd, rgbNonceEvenOSAP and rgbNonceOddOSAP

rgbNonceEven

Even nonce previously generated by TPM to cover inputs. Required to calculate the pattern the new secrets are XORed or encrypted with.

rgbNonceOdd

Nonce generated by TSP associated with the authorization session. This value is provided to allow the application to keep track of the nonces in use.

rgbNonceEvenOSAP

Nonce generated by TPM and associated with shared secret. Required to calculate the shared secret for the OSAP session.

rgbNonceOddOSAP

The nonce generated by the caller associated with the shared secret. Required to calculate the shared secret for the OSAP session.

ulSizeEncAuth

The size of the parameter rgbEncAuthUsage and rgbEncAuthMigration

rgbEncAuthUsage

Encrypted usage secret

rgbEncAuthMigration

Encrypted migration secret

Example:

```
TSS_RESULT Tspicb_CallbackXorEnc(
```

```
   PVOID       lpAppData,           // in
   TSS_HOBJECT   hOSAPObject,       // in
   TSS_HOBJECT   hObject,           // in
   TSS_FLAGS  PurposeSecret,        // in
   UINT32     ulSizeNonces,         // in
   BYTE*      rgbNonceEven,         // in
   BYTE*      rgbNonceOdd,          // in
   BYTE*      rgbNonceEvenOSAP,     // in
   BYTE*      rgbNonceOddOSAP,      // in
   UINT32     ulSizeEncAuth,        // in
   BYTE*      rgbEncAuthUsage,      // out
   BYTE*      rgbEncAuthMigration   // out

{
   // Get secret from user for hOSAPObject (e.g. via application
   dialog)
   BYTE *pbSecretFromUser= SHA1(…)

   BYTE *pbSessionSecret= HMAC(pbSecretFromUser,
              rgbNonceEvenOSAP,
              rgbNonceOddOSAP);

   BYTE *pbEncValue= SHA1(pbSessionSecret, rgbNonceEven);

   BYTE *pbEncAuthUsageTemp= XOR(pbSecretToEnc, pbEncValue);
   memcpy(rgbEncAuthUsage, pbEncAuthUsageTemp, ulSizeEncAuth);

   return TSS_SUCCESS;
}
```

4.3.4.32.3 Tspicb_CallbackTakeOwnership

Start of informative comment:

This method is called when the function Tspip_TPM_TakeOwnership is used and the callback mechanism is set in the assigned policy object of the object.

Both the SRK and TPM secrets are being set in Tspip_TPM_TakeOwnership. Both the SRK and TPM polices need appropriate information to obtain the encrypted secrets before the command is executed. If the SRK authorization is to be encrypted by the calling application, then the policy object of the SRK usage policy will have the callback registered to it. If the TPM authorization must also be encrypted by the calling application, then the usage policy of the TPM object will have the registered callback.

End of informative comment.

It is the application writer's responsibility to display or mask any pop-up windows that may result from this function.

Definition:

```
TSS_RESULT Tspicb_CallbackTakeOwnership
(
   PVOID       lpAppData,          // in
   TSS_HOBJECT  hObject,           // in
   TSS_HKEY    hObjectPubKey ,     // in
   UINT32      ulSizeEncAuth,      // in
   BYTE*       rgbEncAuth          // out
);
```

Parameters

lpAppData

Pointer to application provided data as provided on registration of callback function

hObject

Handle to the TPM object

hObjectPubKey

Handle to the key object representing the endorsement public key required for encrypting the secret of SRK and the TPM owner secret.

ulSizeEncAuth

The size of the parameter *rgbEncAuthOwner* and rgbEncAuthSrk

rgbEncAuth

The encrypted authorization. If the callback is registered to the usage policy of the TPM object, then this will be the encrypted owner authorization. If it is registered to the usage policy of the SRK object, then this will be the encrypted SRK usage policy.

Example:

```
TSS_RESULT Tspicb_CallbackTakeOwnership(
   PVOID       lpAppData        // in
   TSS_HOBJECT   hObject,       // in
   TSS_HKEY   hObjectPubKey     // in
   UINT32     ulSizeEncAuth,    // in
   BYTE*      rgbEncAuth        // out
{
   // Get secrets (e.g. via application dialog)
   BYTE *pbSecretOwnerToEnc = SHA1(…)
   BYTE *pbSecretSrkToEnc = SHA1(…)

   Tspi_GetAttribData(hObjectPubKey, …, pubKey);

   BYTE *pbEncAuthTemp = Encrypt(pubKey, pbSecretToEnc);

   memcpy(rgbEncAuth, pbEncAuthTemp, ulSizeEncAuth);

   return TSS_SUCCESS;
}
```

4.3.4.32.4 Tspicb_CallbackSealxMask

Start of informative comment:

This callback masks or unmasks the data during a Sealx or Unseal operation.

End of informative comment.

Definition:

```
typedef TSS_RESULT (*Tspicb_CallbackSealxMask)
(
    PVOID                    lpAppData,        // in
    TSS_HKEY                 hKey,             // in
    TSS_HENCDATA             hEncData,         // in
    TSS_ALGORITHM_ID         algID,            // in
    UINT32                   ulSizeNonces,     // in
    BYTE*                    rgbNonceEven,     // in
    BYTE*                    rgbNonceOdd,      // in
    BYTE*                    rgbNonceEvenOSAP, // in
    BYTE*                    rgbNonceOddOSAP,  // in
    UINT32                   ulDataLength,     // in
    BYTE*                    rgbDataToMask,    // in
    BYTE*                    rgbMaskedData     // out
);
```

Parameters

lpAppData

Application-supplied pointer that was registered with the callback function.

hKey

The key performing the Sealx or Unseal

hEncData

The data object that will hold the ciphertext (if the operation is a Sealx) or the object that holds the ciphertext (if the operation is Unseal).

algID

The symmetric algorithm that should be used to mask/unmask the data.

ulSizeNonces

The size, in bytes, of the nonce buffers.

rgbNonceEven

The current even nonce for the authorization session.

rgbNonceOdd

The current odd nonce for the authorization session.

rgbNonceEvenOSAP

The current even nonce for the OSAP session.

rgbNonceOddOSAP

The current odd nonce for the OSAP session.

ulDataLength

The size of the data to be masked.

rgbDataToMask

The buffer to hold the data to be masked. Its length is equal to ulDataLength.

rgbMaskedData

The output buffer for the masked data. Its length is equal to ulDataLength.

Return Values

TSS_SUCCESS
TSS_E_INVALID_HANDLE
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

Remarks

The masking operation during the Sealx and Unseal are the same, so there is no parameter indicating which operation is taking place. In both cases, the source data is provided in rgbDataToMask and the generated data should be placed in rgbMaskedData.

4.3.4.32.5 Tspicb_CallbackChangeAuthAsym

Start of informative comment:

This method is called when the function Tspip_ChangeAuthAsym is used and the callback mechanism is set in the assigned policy object of the object.

The service provider uses the HMAC calculation as parameter for the TPM command TPM_ChangeAuthAsymFinish(…). The HMAC links the old and new authorization values together (see Main Specification).

The usage policy of the object being changed will have this registered callback.

End of informative comment.

Definition:

```
TSS_RESULT CallbackChangeAuthAsym
(
    PVOID       lpAppData,        // in
    TSS_HOBJECT   hObject,        // in
    TSS_HKEY    hObjectPubKey,    // in
    UINT32      ulSizeEncAuth,    // in
    UINT32      ulSizeAuthLink,   // in
    BYTE*       rgbEncAuth,       // out
    BYTE*       rgbAuthLink       // out
);
```

Parameters

lpAppData

Pointer to application provided data as provided on registration of callback function

hObject

Handle to the object the secret should changed

hObjectPubKey

Handle to the key object representing the public key required for encrypting the secret of SRK and the TPM owner secret.

ulSizeEncAuth

The size of the parameter rgbEncAuth

ulSizeAuthLink

The size of the parameter rgbAuthLink

rgbEncAuth

New authorization data encrypted with ephemeral key

rgbAuthLink

HMAC digest that links the old and new authorization values together

Example:

```
TSS_RESULT Tspicb_CallbackChangeAuthAsym
(
   PVOID       lpAppData,        // in
   TSS_HOBJECT   hObject,        // in
   TSS_HKEY    hObjectPubKey,    // in
   UINT32      ulSizeEncAuth,    // in
   UINT32      ulSizeAuthLink,   // in
   BYTE*       rgbEncAuth,       // out
   BYTE*       rgbAuthLink       // out
{
   // Get old and new secret (e.g. via application dialog)
   BYTE *pbOldSecret = SHA1(…)
   BYTE *pbNewSecret = SHA1(…)

   Tspi_GetAttribData(hObjectPubKey, …, pubKey);

   BYTE* pbEncAuthTemp = Encrypt(pubKey, pbNewSecret);
   memcpy(rgbEncAuth, pbEncAuthTemp, ulSizeEncAuth);

   BYTE* pbAuthLinkTemp = HMAC(pbOldSecret, pbNewSecret);
   memcpy(rgbAuthLink, pbAuthLinkTemp, ulSizeAuthLink);

   return TSS_SUCCESS;
   }
```

4.3.4.32.6 Tspicb_CollateIdentity

Start of informative comment:

This method is called when the function Tspi_CollateIdentity is used. The intent is to allow application writers to use any symmetric key algorithm they choose when encrypting the identity request packet.

This callback is registered as an attribute of the TPM object within the calling context.

The TSS must allocate the memory needed for the rgbSessionKey and rgbTCPAIdentityProof parameters of the callback before calling it. The size of the allocated memory should also be passed to the callback in the ulSessionKeyLength and pulTCPAIdentityProofLength parameters. If the amount of memory allocated by the TSS isn't sufficient, the callback should return an error.

End of informative comment.

It is the application writer's responsibility to display or mask any pop-up windows that may result from this function. This callback MAY be provided by default by the TSP. A call to

Definition:

```
TSS_RESULT Tspicb_CollateIdentity
(
    PVOID            lpAppData,                         // in
    UINT32           ulTCPAPlainIdentityProofLength,    // in
    BYTE*            rgbTCPAPlainIdentityProof,         // in
    TSS_ALGORITHM_ID algID,                             // in
    UINT32           ulSessionKeyLength,                // out
    BYTE*            rgbSessionKey,                     // out
    UINT32*          pulTCPAIdentityProofLength,        // out
    BYTE*            rgbTCPAIdentityProof               // out
);
```

Parameters

lpAppData

Pointer to application provided data as provided on registration of callback function

ulTCPAPlainIdentityProofLength

Size of the Identity Proof in plain text.

rgbTCPAPlainIdentityProof

Pointer containing the plain text identity request structure `TCPA_IDENTITY_PROOF`.

algID

The encryption algorithm to use to encrypt the identity proof.

ulSessionKeyLength

Length of the symmetric key

rgbSessionKey

Pointer containing the session key EncTcpaSymmetricKey as documented in section 9.4.1 of the TCG 1.1b Main Specification of ‘ulSessionKeyLength’ bytes

pulTCPAIdentityProofLength

Size of the encrypted identity proof.

rgbTCPAIdentityProof

Pointer containing the encrypted identity proof.

4.3.4.32.7 Tspicb_ActivateIdentity

Start of informative comment:

This method is called when the function Tspi_ActivateIdentity is used. The intent is to allow application writers to use any symmetric key algorithm they choose when decrypting the credential received from the privacy CA.

This callback is registered as an attribute of the TPM object within the calling context.

The TSS must allocate the memory needed for the rgbSessionKey and rgbTCPAIdentityProof parameters of the callback before calling it. The size of the allocated memory should also be passed to the callback in the ulSessionKeyLength and pulTCPAIdentityProofLength parameters. If the amount of memory allocated by the TSS isn't sufficient, the callback should return an error.

End of informative comment.

It is the application writer's responsibility to display or mask any pop-up windows that may result from this function.

Definition:

```
TSS_RESULT Tspicb_ActivateIdentity
(
   PVOID    lpAppData                      // in
   UINT32   ulSessionKeyLength,            // in
   BYTE     *rgbSessionKey,                // in
   UINT32   ulSymCAAttestationBlobLength   // in
   BYTE     *rgbSymCAAttestationBlob,      // in
   UINT32   *pulCredentialLength,          // out
   BYTE     *rgbCredential                 // out
);
```

Parameters

lpAppData

Pointer to application provided data as provided on registration of callback function.

ulSessionKeyLength

Length of the symmetric key

rgbSessionKey

pointer containing the session key of 'ulSessionKeyLength' bytes

ulSymCAAttestationBlobLength

Size of the encrypted credential received from the enhanced CA.

rgbSymCAAttestationBlob

Pointer containing the encrypted credential from the enhanced CA.

pulCredentialLength

Size of the decrypted Credential

rgbCredential

Pointer containing the Credential received from the enhanced CA.

4.3.4.32.8 Tspicb_DAA_Sign

Start of informative comment:

This method is called when the function Tspi_TPM_DAA_Sign is used and the callback mechanism is set in the assigned DAA object. The intent is to allow application writers to efficiently proof additional assertions about the DAA Credential. For instance, to proof that a certified attribute is greater than certain value.

End of informative comment.

Definition:

```
TSS_RESULT Tspicb_DAA_Sign
(
   PVOID                         lpAppData,                     // in
   TSS_HDAA_DATA                 daaPublicKey,                  // in
   UINT32                        gammasLength,                  // in
   BYTE**                        gammas,                        // in
   UINT32                        attributesLength,              // in
   BYTE**                        attributes,                    // in
   UINT32                        randomAttributesLength,        // in
   BYTE**                        randomAttributes,              // in
   UINT32                        attributeCommitmentsLength,// in
   TSS_DAA_ATTRIB_COMMIT*        attributeCommitments,          // in
   TSS_DAA_ATTRIB_COMMIT*       attributeCommitmentsProof,      // in
   TSS_DAA_PSEUDONYM_PLAIN*      pseudonym,                     // in
   TSS_DAA_PSEUDONYM_PLAIN*      pseudonymTilde,                // in
   TSS_DAA_PSEUDONYM_ENCRYPTED* pseudonymEncrypted,             // in
   TSS_DAA_PSEUDONYM_ENCRYPTED* pseudonymEncProof,              // in
   TSS_DAA_SIGN_CALLBACK**       additionalProof                // out
);
```

Parameters

lpAppData

Pointer to application provided data as provided on registration of callback function.

daaPublicKey

DAA Issuer public key. This handle represents a TSS_DAA_PK structure.

gammasLength

Length of the array of gammas, which is determined by the DAA Issuer public key ($l_h + l_i$). The length of a single element is $\frac{l_\Gamma}{8}$.

gammas

Array of gammas.

attributesLength

Length of the array of attributes, which is determined by the DAA Issuer public key ($l_h + l_i$). The length of a single attribute is $\frac{l_f}{8}$.

attributes

Array of attributes that the owner of the DAA Credential is committing to.

randomAttributesLength

Length of the array of randomAttributes, which is determined by the DAA Issuer public key ($l_h + l_i$). The length of a single randomAttribute is $\frac{l_f}{8}$.

randomAttributes

Array of random values (attributes). The length of such a value is equal the length of the attributes.

attributeCommitmentsLength

Length l_C of array of attributeCommitments and attributeCommitmentsProof

attributeCommitments

Array of commitments to the attributes that were selected to be committed to.

attributeCommitmentsProof

Array of correctness proofs of the attributeCommitments

pseudonym

Pseudonym

pseudonymTilde

Correctness proof of the pseudonym

pseudonymEncrypted

Encrypted pseudonym

pseudonymEncProof

Correctness proof of the encrypted pseudonym

additionalProof

Additional proof data generated by this callback function

Return Values

TSS_SUCCESS
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

4.3.4.32.9 Tspicb_DAA_VerifySignature

Start of informative comment:

This method is called when the function Tspi_DAA_VerifySignature is used and the callback mechanism is set in the assigned DAA object. The intent is to allow application writers to efficiently proof additional assertions about the DAA Credential. For instance, to proof that a certified attribute is greater than certain value.

This is an optional function and does not require a TPM or a TCS.

End of informative comment.

Definition:

```
TSS_RESULT Tspicb_DAA_VerifySignature
(
   PVOID                      lpAppData,                     // in
   UINT32                     challengeLength,               // in
   BYTE*                      challenge,                     // in
   TSS_DAA_SIGN_CALLBACK*     additionalProof,               // in
   TSS_HDAA_DATA              daaPublicKey,                  // in
   UINT32                     gammasLength,                  // in
   BYTE**                     gammas,                        // in
   UINT32                     sAttributesLength,             // in
   BYTE**                     sAttributes,                   // in
   UINT32                     attributeCommitmentsLength,// in
   TSS_DAA_ATTRIB_COMMIT*     attributeCommitments,          // in
   TSS_DAA_ATTRIB_COMMIT*     attributeCommitmentsProof,     // in
   UINT32                     zetaLength,                    // in
   BYTE*                      zeta,                          // in
   UINT32                     sFLength,                      // in
   BYTE*                      sF,                            // in
   TSS_DAA_PSEUDONYM*         pseudonym,                     // in
   TSS_DAA_PSEUDONYM*         pseudonymProof,                // in
   TSS_BOOL*                  isCorrect                      // out
);
```

Parameters

lpAppData

Pointer to application provided data as provided on registration of callback function.

challengeLength

Length of challenge

challenge

Challenge

additionalProof

Additional proof data generated by this callback function *daaPublicKey*

DAA Issuer public key. This handle represents a TSS_DAA_PK structure.

gammasLength

Length of the array of gammas, which is determined by the DAA Issuer public key (l_h). The length of a single element is $\frac{l_\Gamma}{8}$.

gammas

Array of gammas.

sAttributesLength

Length of the array of sAttributes, which is determined by the DAA Issuer public key ($l_h + l_i$). The length of a single sAttribute is $\frac{l_f}{8}$.

sAttributes

Array of attributes that the owner of the DAA Credential is committing to.

attributeCommitmentsLength

Length l_c of array of attributeCommitments and attributeCommitmentsProof

attributeCommitments

Array of commitments to the attributes that were selected to be committed to.

attributeCommitmentsProof

Array of correctness proofs of the attributeCommitments

zetaLength

Length of zeta ($\frac{l_\Gamma}{8}$)

zeta

zeta

sFLength

Length of sF ($\frac{2l_f}{8}$)

sF

sF

pseudonym

pseudonym

pseudonymProof

Correctness proof of the pseudonym

isCorrect

Boolean value that determines if the additional Proof was indeed correct.

Return Values

TSS_SUCCESS
TSS_E_BAD_PARAMETER
TSS_E_INTERNAL_ERROR

THIS PAGE LEFT INTENTIONALLY BLANK

TCG Software Stack (TSS) Specification

Part2: TCS, TDDL and Administrative functions

TABLE OF CONTENTS

List of Figures

5. TCG Core Services (TCS)

TCS Architecture and Interface Description

5.1.1 TCS Memory Manager

Start of informative comment:

The TCS interfaces use the standard cross-process memory management. The memory allocation model is supported by using the RPC data marshalling available on the platform. The memory buffers are arranged and allocate depending on the IDL parameter type:

[in] parameters – Caller-allocated memory management. The data is copied from the Caller-to- Callee. The Caller must allocate the memory before the function call and de-allocate the memory when finished with the data (typically immediately after function return).

[out] parameters - Callee-allocated memory management. The data is copied from the Callee-to-Caller. The Callee must allocate the memory before returning from the function call and the Caller must de-allocate the memory when it is finished referencing the data (typically sometime after function return). For memory pointers, a pointer-to-pointer declaration is used.

[in,out] parameters – The data is copied in both directions, Caller-to-Callee and Callee-to-Caller. The Caller and Callee are responsible for allocating and de-allocating any respective allocated memory as stated in the [in] and [out] convention above.

End of informative comment.

5.1.2 TCS Data Marshalling

Start of informative comment:

Data marshalling is typically performed using the RPC mechanism available on the specific platform. This mechanism is used to perform the inter-process communication between a TCG-enabled application and TCS platform service. Data is transferred across the process boundaries via the data marshalling proxy and stub routines. Most platforms provide support for default proxy and stub generation, but also allow for custom data marshalling code to be designed. The specifics of each platform data marshalling technique and options are outside the scope of this document.

End of informative comment.

5.1.3 TCS Interface Dynamics

Start of informative comment:

The TPM parameter block generator functions are incorporated as part of a TCS platform system service, which is typically implemented as an out-of-process server. The TCS interface (Tcsi) and TPM parameter block generator API do not define callback function entry points and interface dynamics are left up to the implementation. Either synchronous or asynchronous dynamics can be implemented depending on the type of underlying RPC support available on the platform

End of informative comment.

5.2 TCS-specific Return Code Defines

With the following table the error codes common to all Tcsi functions are listed. In addition to these error codes, the TSS_E_* error code may also be returned with the layer set to the value for the TCS.

In addition each Tcsi function will list in its description the error return codes specific to the function.

Type	Definition
TCS_SUCCESS	Successful completion
TCS_E_FAIL	General failure.
TCS_E_KEY_MISMATCH	Key addressed by the application key handle does not match the key addressed by the given UUID.
TCS_E_KM_LOADFAILED	Key addressed by Key's UUID cannot be loaded because one of the required parent keys needs authorization.
TCS_E_KEY_CONTEXT_RELOAD	The Key Cache Manager could not reload the key into the TPM.
TCS_E_INVALID_CONTEXTHANDLE	The context handle supplied is invalid.
TCS_E_INVALID_KEYHANDLE	The key handle supplied is invalid.
TCS_E_INVALID_AUTHHANDLE	The authorization session handle supplied is invalid.
TCS_E_INVALID_AUTHSESSION	The auth session has been closed by the TPM itself. E.g. due to a command failure like authorization failure
TCS_E_INVALID_KEY	The key has been unloaded by the TPM itself. E.g. due to a OwnerCLear command
TCS_E_KEY_ALREADY_REGISTERED	Key is already registered
TCS_E_KEY_NOT_REGISTERED	Key isn't registered
TCS_E_KEY_CONTEXT_RELOAD	Need to reload the key context
TCS_E_BAD_INDEX	Bad memory index
TCS_E_BAD_PARAMETER	Bad parameter
TCS_E_OUTOFMEMORY	TPM out of memory
TCS_E_SIZE	PCR size wrong
TCS_E_NOTIMPL	Command not implemented
TCS_E_INTERNAL_ERROR	TPM internal error
TCS_E_VERIFICATION_FAILED	Field upgrade verification error
TCS_E_MAXNVWRITES	TPM NVRAM has its max writes
TCS_E_BAD_DELEGATE	Delegation authorization failed
TCS_E_INVALID_COUNTER_HANDLE	Counter handle not valid

5.3 TSPI-specific Return code Rules

Functions or methods within this layer MAY return common errors defined in section: Common Return Code Defines 2.4.2 Any of above return codes may returned by any of the functions in this section.

5.4 Structures and Definitions

This document utilizes structures in the function definitions that are defined in the TCG 1.1b Main Specification and in the TSP section of this specification. In addition, the following structures are defined as follows:

5.4.1 Data Types of the Tcsi

Start of informative comment:

This section describes data type declarations especially required at the Tcsi.

For all parameters providing a buffer length a size of 32 bit should be sufficient; the same applies for the flags parameter indicating the attribute type of an object.

Handles are used as unsigned integer values to address any instantiated object.

End of informative comment.

Type	Definition	Usage
TCS_AUTHHANDLE	UINT32	Handle addressing a authorization session
TCS_CONTEXT_HANDLE	UINT32	Basic context handle
TCS_KEY_HANDLE	UINT32	Basic key handle

5.4.2 TCS_LOADKEY_INFO

Start of informative comment:

TCS_LOADKEY_INFO provides information to enable the TSS CS Key Manager Service to load a registered key if a required parent key needs authorization.

End of informative comment.

Definition:

```
typedef struct tdTCS_LOADKEY_INFO
   {
   TSS_UUID           keyUUID;
   TSS_UUID           parentKeyUUID;
   TCPA_DIGEST        paramDigest; //   SHA1   digest   of   the
                                   TPM_LoadKey
         // Command input parameters
         // As defined in TCG 1.1b Main Specification
   TPM_AUTH           authData;  // Data regarding a valid auth
                                   // Session including the
                                   // HMAC digest
   } TCS_LOADKEY_INFO;
```

5.5 TCS Context Manager

5.5.1 TCS Context Manager Functions and Operations

All resources a calling application can work with are assigned to a certain context.

If the TCS has to allocate memory, which has to be provided to the calling application (important for variable sized output data blocks), this kind of resource must also be assigned to a certain context.

Resource Relationship:

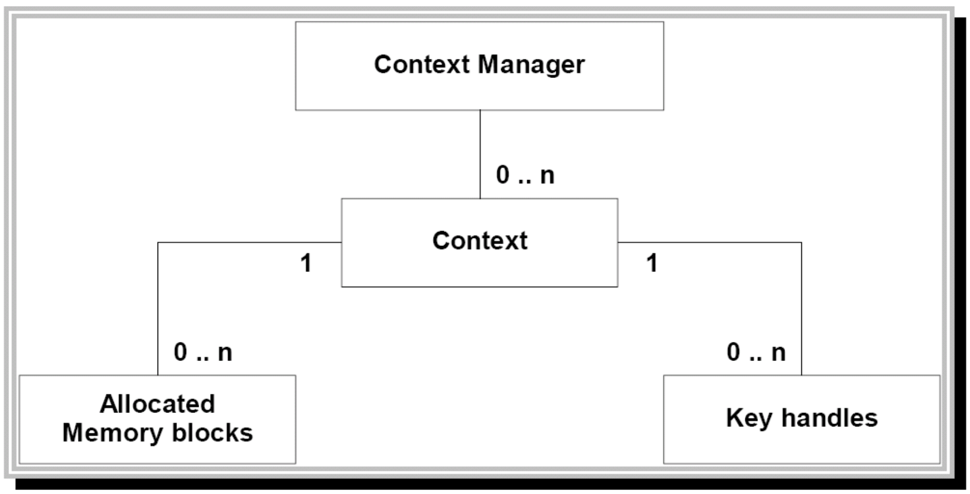


Figure 5-13 TCS Context Manager and Operations

5.5.2 TCS Context Manager Interface

5.5.2.1 Tcsi_OpenContext

Start of informative comment:

Tcsi_OpenContext is used to obtain a handle to a new context.

The context handle is used in various functions to assign resources to it. An application (i.e., TSP or application directly utilizing the TCS) may require more than one context open.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsi_OpenContext
(
   TCS_CONTEXT_HANDLE*hContext// out
);
```

IDL-Definition:

```
[helpstring("method Tcsi_OpenContext")]
TSS_RESULT Tcsi_OpenContext
(
   [out] TCS_CONTEXT_HANDLE*  hContext
);
```

Type	Label	Description
TCS_CONTEXT_HANDLE*	hContext	Return a handle to an established TSS CS context. This handle can now be supplied to other functions attempting to do work within this context.

5.5.2.2 Tcsi_CloseContext

Start of informative comment:

Tcsi_CloseContext releases all resources assigned to the given context and the context itself.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsi_CloseContext
(
   TCS_CONTEXT_HANDLE hContext// in
);
```

IDL-Definition:

```
[helpstring("method Tcsi_CloseContext")]
TSS_RESULT Tcsi_CloseContext
(
   [in] TCS_CONTEXT_HANDLE  hContext
);
```

Type	Label	Description
TCS_CONTEXT_HANDLE	Hcontext	Context handle to be released.

5.5.2.3 Tcsi_FreeMemory

Start of informative comment:

Tcsi_FreeMemory frees memory allocated by TSS CS on a context base. If pMemory equals NULL all allocated memory blocks will be freed.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsi_FreeMemory
(
   TCS_CONTEXT_HANDLE hContext,  // in
   BYTE*              pMemory    // in
);
```

IDL-Definition:

```
[helpstring("method Tcsi_FreeMemory")]
TSS_RESULT Tcsi_FreeMemory
(
   [in] TCS_CONTEXT_HANDLE  hContext,
   [in] BYTE*               pMemory
}
```

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle to established context.
BYTE*	pMemory	Pointer addressing memory to be freed.

5.5.2.4 Tcsi_GetCapability

Start of informative comment:

Tcsi_GetCapability provides the capabilities of the TCS.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsi_GetCapability
(
   TCS_CONTEXT_HANDLE    hContext,  // in
   TCPA_CAPABILITY_AREA  capArea,   // in
   UINT32                subCapSize,   // in
   BYTE*                 subCap,    // in
   UINT32*               respSize,  // out
   BYTE**                resp       // out
);
```

IDL-Definition:

```
[helpstring("method Tcsi_GetCapability")]
TSS_RESULT Tcsi_GetCapability
(
   [in] TCS_CONTEXT_HANDLE          hContext,
   [in] TCPA_CAPBILITY_AREA         capArea,
   [in] UINT32                      subCapSize,
   [in, size_is(subCapSize)] BYTE*  subCap,
   [out] UINT32*                   respSize,
   [out, size_is(*respSize)] BYTE**    resp
);
```

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCPA_CAPABILITY_AREA	capArea	Partition of capabilities to be interrogated
UINT32	subCapSize	Size of subCap parameter
BYTE*	subCap	Further definition of information
UINT32*	respSize	The length of the returned capability response
BYTE**	resp	The capability response

Defined Attributes

Capability Area	SubCap Area	Response
TSS_TCSCAP_ALG	TSS_ALG_XX	Queries whether an algorithm is supported.
	TSS_ALG_DEFAULT	Returns the default public key algorithm, A value of TSS Algorithm ID as defined in 2.3.2.19

	TSS_ALG_DEFAULT_SIZE	Returns the default public key length, a UINT32 value
TSS_TCSCAP_VERSION		Queries the current TCS version.
TSS_TCSCAP_MANUFACTURER	TSS_TCSCAP_PROP_MANUFACTURER_ID	Returns the manufacturer or implementer of the TCS. Return SHALL be a UINT32 using the same identity system used in the main specification for: Cap: TCPA_CAP_PROPERTY Subcap: TCPA_CAP_PROP_MANUFACTURER
TSS_TCSCAP_MANUFACTURER	TSS_TCSCAP_PROP_MANUFACTURER_STR	Returns an TSS_UNICODE string of the TCS manufacturer. The contents of this string is determined by the manufacturer and is subject to change in subsequent releases of the TCS.
TSS_TCSCAP_CACHING		Queries the support of key and authorization caching.
TSS_TCSCAP_CACHING	TSS_TCSCAP_PROP_KEYCACHE	TSS_BOOL value. Indicates support of key caching
TSS_TCSCAP_CACHING	TSS_TCSCAP_PROP_AUTHCACHE	TSS_BOOL value. Indicates support of authorization session caching
TSS_TCSCAP_PERSSTORAGE		Queries the support of a persistent storage
TSS_TCSCAP_PLATFORM_CLASS	TSS_TCSCAP_PROP_HOST_PLATFORM	Returns a single TSS_PLATFORM_CLASS structure containing the definition of only the host platform's class
	TSS_TCSCAP_PROP_ALL_PLATFORMS	Returns an array of TSS_PLATFORM_CLASS structures which enumerates all the TCG_defined platforms associated with the HOST Platform. The Host Platform MUST NOT be returned as one of these platform classes. There is no relationship required in the order of the platforms listed.
TSS_TCSCAP_PLATFORM_INFO	TSS_TCSCAP_PLATFORM_TYPE	TSS_UINT32 value. Indicates the platform type (e.g. PC, PDA,…)

TSS_TCSCAP_PLATFORM_INFO	TSS_TCSCAP_PLATFORM_VERSION	TSS_VERSION value. Queries for the current available platform version.
TSS_TCSCAP_TRANSPORT	0	Queries the support of transport features. (bool)
TSS_TCSCAP_TRANSPORT	TSS_TCSCAP_PROP_TRANS_EXCLUSIVE	Queries whether the exclusive transport mode is supported. (bool)

Comment:

This command differs from the Tcsi_GetCapability (note the P missing from this command) in that it retrieves the capabilities of the Core Services not the TPM. This command is not directly sent to the TDDL but may invoke or make inquires to it in order to service this request.

For the PLATFORM_TYPE the header file includes the definitions for the different platforms that can be queried.

5.6 TCS Key and Credential Manager

5.6.1 TCS Key & Credential Manager Functions and Operations

5.6.1.1 TCS Key Manager

Start of informative comment:

The TCS Key Manager Services allow definition of a persistent key hierarchy. The persistent key hierarchy consists of storage keys that make up the base storage key structure that will exist before any user may attempt to load a key. Additionally the persistent key hierarchy may contain system specific leaf keys as for instance identity keys.

The TCS Key Manager Services interface was designed to allow a flexible key structure such that an instance like an IT department of an enterprise may define a deep key hierarchy, a shallow hierarchy, roaming keys, migration base keys, etc.

All keys, which should be internally managed by the Key Manager Services of TCS must be registered in the persistent storage database of TCS. Each key registered in that database will be referenced by its UUID and called a persistent key from this specification's point of view.

Some registered keys have a defined fixed UUID by which they can be referenced on all systems providing the same registered key hierarchy. These UUIDs do not provide any information to identify the system the key is registered on.

An application can also load keys not registered in the TCS database. These keys are loaded utilizing the Tcsi by providing a key blob as defined by TCPA_KEY. These keys are called temporary keys from this specification's point of view.

After a key was loaded either by using a UUID or by using a key blob, this key will be addressed on further calls utilizing the key by the application key handle, which was returned from TCS on a load key command.

Using the Key Manager Services provided by TCS will simplify the whole mechanism of loading a key into the TPM from a calling context's point of view. The application must only address a key to be loaded by its well known UUID and the Key Manager Services will do all the required loading of the underlying parent keys depending on the registered key hierarchy, which may be totally hidden from the application's scope.

The key hierarchy can be defined by some instance like for example the IT department of an enterprise and the TCG-aware applications may not need to know this key hierarchy at all.

Keys once registered in PS will stay registered in PS until they are unregistered. The PS will stay valid across boots.

Application key handles obtained from a load key command are valid as long as the TCS is not restarted or the key is not evicted from the Key Cache Manager Service. Application key handles will not stay valid across boots.

End of informative comment.

Keys once registered in PS will stay registered in PS until they are unregistered. The PS will stay valid across boots.

5.6.1.2 TCS Key Cache Manager

Start of informative comment:

The TCS Key Cache Manager Service (KCM) allows caching keys to manage the restricted resources of a TPM. The KCM is responsible to manage the restricted resources of the TPM and to hide these restrictions from the calling applications. An application can load a key in to the TPM by utilizing the KCM functionality and can assume that this key is available for further use. The KCM is responsible to ensure that a key, which has already been loaded by an application, is available in the TPM, when the application requires that key for a certain command. If all TPM resources are in use, the KCM has to free resources in order to load a key or to get the required key back in to the TPM.

An application must load a key into the TPM utilizing the KCM. The KCM returns an application key handle to the caller and manages a mapping mechanism between the returned application key handle and the actual TPM key handle. The actual TPM key handle will change whenever a key has to be unloaded from the TPM by the KCM in order to free resources since another key has to be loaded and the KCM reloads the key into the TPM again. The application key handle returned to the calling application remains constant as long as the key is not reloaded by the application itself.

End of informative comment.

The key cache mechanism can be implemented by using the TPM commands:

- LoadKey and EvictKey or
- LoadKeyContext and SaveKeyContext

If TPM_LoadKey / TPM_EvictKey has to be used only, because the TPM does not provide TPM_LoadKeyContext / SaveKeyContext, the Key Cache Manager can transparently reload keys only if the required parent key(s) needs no authorization. There must not be any caching of secrets required for authorization.

If TPM_LoadKeyContext / TPM_SaveKeyContext is provided by the TPM, the reloading of keys can be done totally transparent for the calling applications no matter if the required parent keys needs authorization or not.

An application key handle returned to a caller must remain unchanged as long as the caller evicts the key by itself.

The internally managed TPM key handles returned by the appropriate TPM commands to reload keys may change all the time.

The Key Cache Management Service will map the returned stable application key handles with the unstable internal TPM key handles.

A caller must address a loaded key only by the application key handle got from the Key Cache Manager.

Each returned application key handle must be bound to a certain context.

On initial loading a key for the first time the KCM utilizes the TPM command TPM_LoadKey in order to load the given key blob into the TPM. Depending on available authorization data, the TPM command is built with or without authorization data.

The returned application key handles are valid:

- As long as a caller does not evict the key providing the application key handle.
- As long as the context the application key handle is bound to is not closed.
- As long as the Key Cache Manager is not stopped.

If a key can not be reloaded, since authorization is required or since some other failure, the Tcsi command addressing that key will return an error: TCS_E_KEY_CONTEXT_RELOAD.

5.6.1.3 TCS Credential Management

Start of informative comment:

The TCS manages the endorsement credential, the platform credential and the conformance credential. The registration and management of these credentials is TCS vendor specific.

It is not possible to register any credentials or certificates by an application utilizing the Tcsi. This would simplify the credential management for applications. But retrieving credentials without any access control provided by TCS may cause privacy concerns.

The TCS provides the endorsement credential, the platform credential and the conformance credential for a calling application only if the TCS could proof an owner authorization. This TCS Credential Service is provided for making TPM identities only and to fulfill any privacy concerns.

Example of Tcsip_MakeIdentity(…):

The TSS CS first creates a new identity key by processing the TPM command TPM_MakeIdentity().

The TPM command to create an identity key is owner authorized. If this command was successfully processed an owner could be proofed and the TCS will additionally return the endorsement credential, the platform credential and the conformance credential to the caller. This should be an atomic function to start the process of making a new TPM identity.

End of informative comment.

Registration and management of the endorsement credential, the platform credential and the conformance credential is TCS vendor specific. No other credential or certificate can be registered in the TCS Credential Management Services by utilizing the Tcsi. The endorsement credential, platform credential and conformance credential are provided by TCS Credential Manager Services utilizing Tcsip_MakeIdentity only if an owner authorization can be proven.

5.6.2 TCS Key and Credential Manager Interface

5.6.2.1 Interfaces

5.6.2.2 Key Registration

5.6.2.2.1 Tcsi_RegisterKey

Start of informative comment:

Tcsi_RegisterKey allows registering a key in the TCS Persistent Storage (PS). Only system specific keys (keys definitely bound to a certain system) should be registered in TCS PS.

A key can be registered in TCS PS by providing:

- A UUID for that key,
- A UUID for its wrapping parent key and
- The key blob itself.

If the same UUID is used to register a key on different systems this key can be addressed on different systems by the same UUID. This may be done for a basic roaming key, which will wrap all user storage keys in the appropriate key hierarchy.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsi_RegisterKey
(
    TCS_CONTEXT_HANDLE hContext,          // in
    TSS_UUID           WrappingKeyUUID,   // in
    TSS_UUID           KeyUUID,           // in
    UINT32             cKeySize,          // in
    BYTE*              rgbKey,            // in
    UINT32             cVendorDataSize,   // in
    BYTE*              rgbVendorData      // in
);
```

IDL-Definition:

```
[helpstring("method Tcsi_RegisterKey")]
TSS_RESULT Tcsi_RegisterKey
(
   [in] TCS_CONTEXT_HANDLE       hContext,
   [in] TSS_UUID                 WrappingKeyUUID,
   [in] TSS_UUID                 KeyUUID,
   [in] UINT32                   cKeySize,
   [in, size_is(cKeySize)] BYTE* rgbKey,
   [in, defaultvalue(0)] UINT32  cVendorData,
   [in, size_is(cVendorData), defaultvalue(NULL)] BYTE*
                                   rgbVendorData
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle to established context.
TSS_UUID	WrappingKeyUUID	UUID of the already registered wrapping parent key.
UINT32	cKeySize	Size of the provided keyblob in bytes.
BYTE*	rgbKey	Byte stream containing the key blob of the key to be registered.
UINT32	cVendorData	Size of vendor specific data blob in bytes; may be 0.
BYTE*	rgbVendorData	Vendor specific data blob; may be NULL.

Comment:

If a key has already been registered under the provided keyUUID, the function will fail with the error TCS_E_KEY_ALREADY_REGISTERED. The application may then unregister the key, which is already assigned to that UUID. Now the new key can be registered.

The key is stored in the persistent storage without any information about the context used for doing the registration.

Return Value:

TCS_SUCCESS
TCS_E_KEY_ALREADY_REGISTERED
TCS_E_KEY_NOT_REGISTERED
TCS_E_FAIL

5.6.2.2.2 Tcsip_UnregisterKey

Start of informative comment:

A key once registered in the TCS PS can be unregistered from the PS, if that key is not required any longer.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_UnregisterKey
(
    TCS_CONTEXT_HANDLE hContext,  // in
    TSS_UUID           KeyUUID    // in
);
```

IDL-Definition:

```
[helpstring("method Tcsip_UnregisterKey")]
TSS_RESULT Tcsip_UnregisterKey
(
    [in] TCS_CONTEXT_HANDLE  hContext,
    [in] TSS_UUID            KeyUUID,
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	HContext	Handle to established context.
TSS_UUID	KeyUUID	UUID by which the key is registered.

Comment:

If a key has not been registered under the provided KeyUUID, the function will fail with the error TCS_E_KEY_NOT_REGISTERED.

Return Value:

TCS_SUCCESS
TCS_E_KEY_NOT_REGISTERED
TCS_E_KEY_MISMATCH
TCS_E_INVALID_CONTEXTHANDLE
TCS_E_FAIL

5.6.2.2.3 Tcsip_KeyControlOwner

Start of informative comment:

Tcsip_KeyControlOwner controls attributes of a loaded key. This command requires owner authorization.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_KeyControlOwner
(
    TCS_CONTEXT_HANDLE  hContext,           // in
    TCS_KEY_HANDLE      hKey,               // in
    UINT32               ulPublicInfoLength,// in
    BYTE*                rgbPublicInfo,     // in
    UINT32              attribName,         // in
    TSS_BOOL            attribValue,        // in
    TPM_AUTH*           pOwnerAuth,         // in, out
    TSS_UUID*           pUuidData           // out
);
```

IDL-Definition:

```
[helpstring("method Tcsip_KeyControlOwner")]
TSS_RESULT Tcsip_KeyControlOwner
(
    [in] TCS_CONTEXT_HANDLE     hContext,           // in
    [in] TCS_KEY_HANDLE         hKey,               // in
    [in] UINT32                 ulPublicInfoLength //in
    [in] BYTE*                  rgbPublicInfo       //in
    [in] UINT32                 attribName,         // in
    [in] TSS_BOOL               attribValue,        // in
    [in, out] TPM_AUTH*         pOwnerAuth,         // in, out
    [out] TSS_UUID*             pUuidData           // out

);
```


Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	Hcontext	Handle to establsihed context
TCS_KEY_HANDLE	Hkey	Application key handle
UINT32	attribName	Attribute Name
TSS_BOOL	attribValue	Attribute Value
TPM_AUTH	pOwnerA	Owner authorization session data including the HMAC digest. If NULL, no authorization is required
TSS_UUID*	pUuidData	The UUID the key was registered as a TPM resident key. See comments section for more details

Comment:

TCS processes the TPM_KeyControlOwner command, which requires an owner authorization.

Once the key is made resident in the TPM by this command, it is given one of the reserved UUIDs. This UUID is used to obtain a handle to this key at a later time. The UUID is volatile – associated with the key by the TSS at boot, so once used to obtain a handle, the key referenced should be checked against a public key to make certain it is the correct key the user wants to use. Alternatively an application can loop through the possible predefined set of UUIDs using GetKeybyUUID and comparing the results against a public key to determine the correct handle. In most cases the UUID will not change, but if one of the keys resident in the TPM is removed, a reordering of the UUIDs could take place for other keys resident in the TPM.

Return Value:

TCS_SUCCESS

TCS_E_INVALID_CONTEXTHANDLE

TCS_E_FAIL

5.6.2.3 TCS Get Key Hierarchy Information

Start of informative comment:

The Key Management Services of TCS provide information about the registered key hierarchy.

The Key Management Services will provide an array of information or only one entry of that array based on a certain key.

The returned information contains data like the following:

- The UUID of the key,
- The UUID of the wrapping parent key,
- Authorization required and
- Key already loaded.

The application can use this information to improve the strategy of how a key may be loaded into the TPM, if for instance one or more storage keys require authorization.

End of informative comment.

5.6.2.3.1 Tcsi_EnumRegisteredKeys

Start of informative comment:

Tcsi_EnumRegisteredKeys allows obtaining an array of TSS_KM_KEYINFO structures. This information reflects the registered key hierarchy. The caller will receive information of the whole key hierarchy. The keys stored in the persistent storage are totally independent from either the context provided in the function call or the context, which was provided while processing the key registration.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsi_EnumRegisteredKeys
   (
   TCS_CONTEXT_HANDLE hContext,          // in
   TSS_UUID*          pKeyUUID,          // in
   UINT32*            pcKeyHierarchySize,// out
   TSS_KM_KEYINFO**   ppKeyHierarchy     // out
   );
```

IDL-Definition:

```
[helpstring("method Tcsi_EnumRegisteredKeys")]
TSS_RESULT Tcsi_EnumRegisteredKeys
(
   [in] TCS_CONTEXT_HANDLE                    hContext,
   [in] TSS_UUID*                             pKeyUUID,
   [out] UINT32*                              pcKeyHierarchySize,
   [out,      size_is(*pcKeyHierarchySize)]      TSS_KM_KEYINFO**
                                  ppKeyHierarchy
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle to established context.
TSS_UUID*	pKeyUUID	UUID of key the key hierarchy should be returned of. If NULL, the whole key hierarchy will be returned.
UINT32*	pcKeyHierarchySize	Return number of array entries.
TSS_KM_KEYINFO**	ppKeyHierarchy	Return pointer to memory containing array of structures reflecting the registered key hierarchy.

Comment:

The TCS allocates memory, assigns the memory resource to the given context and returns an array of TSS_KM_KEYINFO structures.

This array will return information of the whole registered key hierarchy independent from any context.

If a certain key UUID is provided, the returned array of TSS_KM_KEYINFO structures only contains data reflecting the path of the key hierarchy regarding that key. The first array entry is the key addressed by the given UUID followed by its parent key up to the root key.

If no key UUID is provided (pKeyUUID == NULL), the returned array TSS_KM_KEYINFO structures contains data reflecting the whole key hierarchy starting with root key.

If no keys have been registered *pcKeyHierarchySize = 0 and *ppkeyHierarchy = NULL and the function returns with TCS_SUCCESS.

Return Value:

TCS_SUCCESS
TCS_E_FAIL

5.6.2.3.2 Tcsi_GetRegisteredKey

Start of informative comment:

Tcsi_GetRegisteredKey allows obtaining a TSS_KM_KEYINFO structure containing information about the registered key.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsi_GetRegisteredKey
   (
   TCS_CONTEXT_HANDLE hContext,  // in
   TSS_UUID           KeyUUID,   // in
   TSS_KM_KEYINFO**   ppKeyInfo  // out
   );
```

IDL-Definition:

```
[helpstring("method Tcsi_GetRegisteredKey")]
TSS_RESULT Tcsi_GetRegisteredKey
(
   [in] TCS_CONTEXT_HANDLE      hContext,
   [in] TSS_UUID                KeyUUID,
   [out] TSS_KM_KEYINFO**       ppKeyInfo
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	Hcontext	Handle to established context.
TSS_UUID	KeyUUID	UUID of the key information is required.
TSS_KM_KEYINFO**	PpKeyInfo	Return pointer to memory containing information about the key of interest.

Comment:

The TCS allocates memory, assigns the memory resource to the given context and returns a TSS_KM_KEYINFO structure.

If a key has not been registered under the provided KeyUUID, the function fails with the error TCS_E_KEY_NOT_REGISTERED and returns a NULL pointer.

Return Value:

TCS_SUCCESS
TCS_E_KEY_NOT_REGISTERED
TCS_E_FAIL

5.6.2.3.3 Tcsi_GetRegisteredKeyBlob

Start of informative comment:

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsi_GetRegisteredKeyBlob
   (
   TCS_CONTEXT_HANDLE hContext,  // in
   TSS_UUID           KeyUUID,   // in
   UINT32*            pcKeySize, // out
   BYTE**             prgbKey // out
   );
```

IDL-Definition:

```
[helpstring("method Tcsi_GetRegisteredKeyBlob")]
TSS_RESULT Tcsi_GetRegisteredKeyBlob
(
   [in] TCS_CONTEXT_HANDLE               hContext,
   [in] TSS_UUID                         KeyUUID,
   [out] UINT32*                         pcKeySize,
   [out, size_is(, *pcKeySize)] BYTE**  prgbKey
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle to established context.
TSS_UUID	KeyUUID	UUID of the key information is required.
UINT32*	pcKeySize	Size of the returned keyblob in bytes.
BYTE**	prgbKey	Returned pointer to a byte stream containing the key blob of interest.

Comment:

The TCS allocates memory, assigns the memory resource to the given context and returns a registered key blob as defined by TCPA_KEY structure.

If a key has not been registered under the provided KeyUUID, the function fails with the error TCS_E_KEY_NOT_REGISTERED and returns a NULL pointer and *pcKeySize = 0.

Return Value:

TCS_SUCCESS

TCS_E_KEY_NOT_REGISTERED
TCS_E_FAIL

5.6.2.3.4 Tcsip_GetRegisteredKeyByPublicInfo

Start of informative comment:

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_GetRegisteredKeyByPublicInfo
   (
   TCS_CONTEXT_HANDLE hContext,
   TSS_ALGORITHM_ID   algID,              // in
   UINT32             ulPublicInfoLength,// in
   BYTE*              rgbPublicInfo,     // in
   UINT32*            keySize,           //out
   BYTE**             keyBlob            //out
   )
```

IDL-Definition:

```
[helpstring("method Tcsip_GetRegisteredKeyByPublicInfo ")]
TSS_RESULT Tcsip_GetRegisteredKeyByPublicInfo
(
   [in] TCS_CONTEXT_HANDLE      hContext,
   [in] TSS_ALGORITHM_ID        algID,
   [in] UINT32                  ulPublicInfoLength,
   [in] BYTE*                   rgbPublicInfo,
   [out] UINT32*                keySize,
   [out, size_is(, *keySize)] BYTE** keyBlob
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle to established context.
TSS_ALGORITHM_ID	AlgID	Algorithm ID for public key
UINT32	ulPublicInfoLength	Size of buffer rgbPublicInfo
BYTE*	rgbPublicInfo	Buffer containing the public key
UINT32*	pcKeySize	Size of the returned keyblob in bytes.
BYTE**	prgbKey	Returned pointer to a byte stream containing the key blob of interest.

Comment:

The TCS allocates memory, assigns the memory resource to the given context and returns a registered key blob as defined by TCPA_KEY structure.

If a key has not been registered with the specified public key, the function fails with the error TCS_E_KEY_NOT_REGISTERED and returns a NULL pointer and *pcKeySize = 0.

Return Value:

TSS_SUCCESS
TCS_E_KEY_NOT_REGISTERED

5.6.2.4 TCS Loading a Key

Start of informative comment:

After an application had loaded a key by utilizing the TSS Core Services (TCS), the application will receive an application key handle associated with that loaded key. If the application wants to use the loaded key later (ex. to sign or for a quote), the key will be addressed by that application key handle.

End of informative comment.

For the entity type of SRK the associated application key handle (TCS_KEY_HANDLE) MUST be 0x40000000.

5.6.2.4.1 Tcsip_LoadKeyByBlob

Start of informative comment:

A key can be loaded by providing a key blob as defined in the TCPA_KEY structure. The key defined by the key blob gets unwrapped by the already loaded parent key associated with the given application parent key handle. After the key is loaded an appropriate application key handle is returned by which the key can be addressed for further use. Depending on the parent key this can be done with or without required authorization.

This is a low level mechanism and the calling application must manage the required key blobs on its own but give the caller as much flexibility as possible.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_LoadKeyByBlob
   (
   TCS_CONTEXT_HANDLE hContext,             // in
   TCS_KEY_HANDLE     hUnwrappingKey,       // in
   UINT32             cWrappedKeyBlobSize,  // in
   BYTE*              rgbWrappedKeyBlob,    // in
   TPM_AUTH*          pAuth,                // in, out
   TCS_KEY_HANDLE*    phKeyTCSI,            // out
   TCS_KEY_HANDLE*    phKeyHMAC             // out
   );
```

IDL-Definition:

```
[helpstring("method Tcsip_LoadKeyByBlob")]
TSS_RESULT Tcsip_LoadKeyByBlob
(
   [in] TCS_CONTEXT_HANDLE      hContext,
   [in] TCS_KEY_HANDLE          hUnwrappingKey,
   [in] UINT32                  cWrappedKeyBlobSize,
   [in, size_is(cWrappedKeyBlobSize)] BYTE* rgbWrappedKeyBlob,
   [in, out] TPM_AUTH*          pAuth,
   [out] TCS_KEY_HANDLE*        phKeyTCSI,
   [out] TCS_KEY_HANDLE*        phKeyHMAC
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	HContext	Handle to established context.
TCS_KEY_HANDLE	hUnwrappingKey	Application key handle of the already loaded parent key.
UINT32	cWrappedKeyBlobSize	Size of the provided keyblob in bytes.
BYTE*	rgbWrappedKeyBlob	Key blob of the key to be loaded.
TCS_KEY_HANDLE*	phKeyTCSI	Return application key handle the loaded key can be addressed on further use.
TCS_KEY_HANDLE*	phKeyHMAC	Return TPM key handle required to evaluate the the returned HMAC digist. This TPM key handle can not be used to address the key on on further use.
TPM_AUTH*	pAuth	Authorization session data including the HMAC

		digest for using the unwrapping key.If NULL, no authorization is required.

Comment:

Tcsip_LoadKeyByBlob initially loads the key utilizing the Key Cache Manager Services (KCM) and returns a new created application key handle by which the key can be addressed on further use. The returned application key handle MUST be bound to the context provided by hContext.

After this command the key is managed by the Key Cache Management Services.

If pAuth == NULL, no authorization is required.

Loading a key MUST utilize the Key Cache Manager Service.

Return Value:

TCS_SUCCESS
TCS_E_FAIL

5.6.2.4.2 Tcsip_LoadKeyByUUID

Start of informative comment:

A key registerd in the TCS Persistent Storage Database (PS) can only be loaded by referring a UUID.

TCS Key Management Services will internally provide all the information stored in the PS that is required to load a key when provided with the appropriate UUID. The TCS will implicitly load all the required wrapping parent keys to get the key loaded addressed by the given UUID.

If no authorization is required, the additional required load of one or more parent keys is completely hidden from the calling application. If authorization is required for one or more parent keys the application can provide the authorization data in intermediate steps (see 5.6.3.1).

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_LoadKeyByUUID
   (
   TCS_CONTEXT_HANDLE hContext,     // in
   TSS_UUID           KeyUUID,      // in
   TCS_LOADKEY_INFO*  pLoadKeyInfo, // in, out
   TCS_KEY_HANDLE*    phKeyTCSI     // out
   );
```

IDL-Definition:

```
[helpstring("method Tcsip_LoadKeyByUUID")]
TSS_RESULT Tcsip_LoadKeyByUUID
(
   [in] TCS_CONTEXT_HANDLE      hContext,
   [in] TSS_UUID                KeyUUID,
   [in, out] TCS_LOADKEY_INFO*  pLoadKeyInfo
   [out] TCS_KEY_HANDLE*        phKeyTCSI
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	Hcontext	Handle to established context.
TSS_UUID	KeyUUID	UUID of the key to be loaded.
TCS_KEY_HANDLE*	PhKeyTCSI	Return application key handle the loaded key can be addressed on further use.
TCS_LOADKEY_INFO*	PloadKeyInfo	Information required to load a key if authorization is needed.

Comment:

The TCS Key Management Service utilizes the information stored in the system persistent storage. The service loads all required wrapping parent keys important to load that parent key which wraps the key addressed by KeyUUID.

When this last parent key is loaded, the key addressed by KeyUUID is initially loaded utilizing the Key Cache Manager Service and a new created application key handle is returned. Using that application key handle the key can be addressed on further use. The returned application key handle must be bound to the context given by hContext.

After that command the key is managed by the Key Cache Management Services.

If one of the required parent keys needs authorization, the load key information structure is filled with:

Type	Description
TCS_LOADKEY_INFO->hContext	Context as provided by Tcsip_LoadKeyByUUID call.
TCS_LOADKEY_INFO->keyUUID	UUID of key to be loaded next up the key hierarchy.
TCS_LOADKEY_INFO->parentKeyUUID	UUID of the parent key wrapping the key addressed by TCS_LOADKEY_INFO->keyUUID. This key requires authorization.
TCS_LOADKEY_INFO->paramDigest	Digest of the TPM_LoadKey command input parameters as defined in the TCG 1.1b Main Specification.

The function call will return with the error TCS_E_KM_LOADFAILED.

The caller may use the returned information to:

- Start an OIAP authorization session
- Calculate the HMAC digest using TCS_LOADKEY_INFO->paramDigest and the started OIAP session required for loading the key addressed by TCS_LOADKEY_INFO->keyUUID and wrapped with the parent key addressed by TCS_LOADKEY_INFO->parentKeyUUID which requires authorization.
- Fill in the information in TCS_LOADKEY_INFO->authData
- Recall Tcsip_LoadKeyByUUID with the same input parameters and the completely filled TCS_LOADKEY_INFO structure

The Key Management Services now utilizes the provided TCS_LOADKEY_INFO->authData to call the TPM command TPM_LoadKey providing the appropriate authorization data to get that key loaded which is addressed by TCS_LOADKEY_INFO->keyUUID.

Loading a key must utilize the Key Cache Manager Service.

Return Value:

TCS_SUCCESS

TCS_E_KM_LOADFAILED
TCS_E_FAIL

5.6.2.4.3 Tcsip_EvictKey

Start of informative comment:

Tcsip_EvictKey allows flushing a key from the cache managed by the Key Cache Manager Services.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_EvictKey
(
   TCS_CONTEXT_HANDLE    hContext, // in
   TCS_KEY_HANDLE        hKey      // in
);
```

IDL-Definition:

```
[helpstring("method Tcsip_EvictKey")]
TSS_RESULT Tcsip_EvictKey
(
   [in] TCS_CONTEXT_HANDLE  hContext,
   [in] TCS_KEY_HANDLE      hKey
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle to established context.
TCS_KEY_HANDLE	hKey	Application key handle to be evicted.

Comment:

Tcsip_EvictKey flushes the key addressed by hKey from the key cache managed by the TSS CS Key Management Services.

If key object addressed with hKey is not assigned to context addressed with hContext the function fails with the error TCS_E_INVALID_CONTEXTHANDLE

All resources bound to the application key handle must be released and the application key handle is not longer valid on return.

Return Value:

TCS_SUCCESS
TCS_E_INVALID_CONTEXTHANDLE
TCS_E_FAIL

5.6.2.5 TCS Creating a Key

5.6.2.5.1 Tcsip_CreateWrapKey

Start of informative comment:

Tcsip_CreateWrapKey allows creating a new key, which is wrapped by the already loaded wrapping key addressed by hWrappingKey handle.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_CreateWrapKey
(
   TCS_CONTEXT_HANDLE hContext,           // in
   TCS_KEY_HANDLE     hWrappingKey,       // in
   TCPA_ENCAUTH       KeyUsageAuth,       // in
   TCPA_ENCAUTH       KeyMigrationAuth,   // in
   UINT32             keyInfoSize,        // in
   BYTE*              keyInfo,            // in
   TPM_AUTH*          pAuth,              // in, out
   UINT32*            keyDataSize,        // out
   BYTE**             keyData,            // out
);
```

IDL-Definition:

```
[helpstring("method Tcsip_CreateWrapKey")]
TSS_RESULT Tcsip_CreateWrapKey
(
   [in]TCS_CONTEXT_HANDLE   hContext,              // in
   [in]TCS_KEY_HANDLE    hWrappingKey,             // in
   [in]TCPA_ENCAUTH      KeyUsageAuth,             // in
   [in]TCPA_ENCAUTH      KeyMigrationAuth,         // in
   [in]UINT32            keyInfoSize,              // in
   [in, size_is( keyInfoSize )]BYTE*   keyInfo,    // in
   [in, out]TPM_AUTH*    pAuth,                    // in, out
   [out]UINT32*          keyDataSize,              // out
   [out, size_is(, *keyDataSize )]BYTE** keyData //out
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle to established context.
TCS_KEY_HANDLE	hWrappingKey	Application key handle of the already loaded wrapping parent key.
TCPA_ENCAUTH	KeyUsageAuth	Encrypted usage authorization data for

		the key to be created.
TCPA_ENCAUTH	KeyMigrationAuth	Encrypted migration authorization data for the key to be created.
UINT32*	pcKeySize	Size of the provided/returned byte stream in bytes.
BYTE**	prgbKey	IN: Information about key to be created, pubkey.keyLength and pKey->encSize elements are 0. OUT: The key blob as defined in TCPA_KEY structure which includes the public and encrypted private key.
TPM_AUTH*	pAuth	Authorization session data including the HMAC digest for using the wrapping key. If NULL, no authorization is required.

Comment:

Tcsip_CreateWrapKey creates a new key as defined by the parameters provided by pKey. The new key gets a usage authorization secret and a migration authorization secret as given by the input parameters KeyUsageAuth and KeyMigrationAuth. Both secrets are encrypted as defined in the TCG 1.1b Main Specification.

The TCS must utilize the TPM command TPM_CreateWrapKey to create and wrap the new key.

Return a key blob wrapped with the key addressed by the application key handle of the wrapping key, which must already be loaded.

All required memory resources to return the public key and the encrypted private key data must be allocated by TCS. The appropriate memory resources must be bound to the context provided by hContext.

If pAuth == NULL, no authorization for using the wrapping key is required.

Return Value:

TCS_SUCCESS
TCS_E_KM_LOADFAILED
TCS_E_FAIL

5.6.2.6 TCS Working with Keys

5.6.2.6.1 Tcsip_GetPubKey

Start of informative comment:

Tcsip_GetPubKey allows obtaining the public key data of a key loaded in the TPM. This information may have privacy concerns so the command must have authorization from the key owner.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_GetPubKey
(
   TCS_CONTEXT_HANDLE hContext,     // in
   TCS_KEY_HANDLE     hKey,         // in
   TPM_AUTH*          pAuth,        // in, out
   UINT32*            pcPubKeySize, // out
   BYTE**             prgbPubKey    // out
);
```

IDL-Definition:

```
[helpstring("method Tcsip_GetPubKey")]
TSS_RESULT Tcsip_GetPubKey
(
   [in] TCS_CONTEXT_HANDLE  hContext,
   [in] TCS_KEY_HANDLE      hKey,
   [in, out] TPM_AUTH*      pAuth,
   [out] UINT32*            pcPubKeySize,
   [out, size_is(, *pcPubKeySize)] BYTE** prgbPubKey
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle to established context.
TCS_KEY_HANDLE	hKey	Application key handle of the loaded key.
UINT32*	pcPubKeySize	Size of the returned byte stream in bytes.
BYTE**	prgbPubKey	Returned pointer to a byte stream containing information about the public key of interest.
TPM_AUTH*	pAuth	Authorization session data including the HMAC digest for authorizing the key. If NULL, no authorization is

		required.

Comment:

Tcsip_GetPubKey obtains the public key value of a key loaded in the TPM and addressed by the application key handle. The TPM command TPM_GetPubKey must be utilized.

Return a blob containing the public key data to the caller.

All required memory resources to return the public key data, must be allocated by TSS CS. The appropriate memory resources must be bound to the context assigned to the application key handle provided by hKey.

If pAuth == NULL, no authorization for using the wrapping key is required.

If key object addressed with hKey or auth session addressed with pAuth is not assigned to context addressed with hContext the function fails with the error TCS_E_INVALID_CONTEXTHANDLE.

Return Value:

TCS_SUCCESS
TCS_E_KEY_CONTEXT_RELOAD
TCS_E_INVALID_CONTEXTHANDLE
TCS_E_FAIL

5.6.2.6.2 Tcsip_OwnerReadInternalPub

Start of informative comment:

Tcsip_OwnerReadInternalPub allows the TPM owner to read the public SRK key or the internal public EK key.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_OwnerReadInternalPub
(
   TCS_CONTEXT_HANDLE hContext,        // in
   TCS_KEY_HANDLE     hKey,            // in
   TPM_AUTH*          pOwnerAuth,      // in, out
   UINT32*            punPubKeySize,   // out
   BYTE**             ppbPubKeyData    // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_OwnerReadInternalPub")]
TSS_RESULT Tcsip_OwnerReadInternalPub
(
   [in] TCS_CONTEXT_HANDLE    hContext,
   [in] TCS_KEY_HANDLE        hKey,
   [AUTH, in, out] TPM_AUTH*  pOwnerAuth,
   [AUTH, out] UINT32*        punPubSRKKeySize,
   [AUTH, out, size_is(, *punPubKeySize)] BYTE**
                              ppbPubKey
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_KEY_HANDLE	HKey	Either SRK or EK handle
TPM_AUTH*	pOwnerAuth	Pointer to Owner's authorization
UINT32*	punPubKeySize	Size of (ppbPubKey)
BYTE**	ppbPubKey	The public key (EK or SRK)

Comment:

TPM command – TPM_OwnerReadInternalPub

TPM ordinal – TPM_OwnerReadInternalPub

Return Value:

TCS_SUCCESS
TCS_E_INVALID_CONTEXTHANDLE

TCS_E_FAIL

5.6.2.7 TCS Credential Management

5.6.2.7.1 Tcsip_MakeIdentity

Start of informative comment:

Tcsip_MakeIdentity allows creating a TPM identity and additionally returns the endorsement credential, the platform credential and the conformance credential.

These three credentials are stored TCS vendor specific and are provided by Tcsip_MakeIdentity only. This simplifies the management of these credentials because each calling application can get this platform and system specific information from the appropriate system the information belongs to and ensures that the credentials are only provided if an owner authorization could be proofed.

Only the Owner of the TPM has the privilege of creating a TPM identity. This ensures that the credentials are provided to the caller only after a TPM owner authorization could be proofed.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_MakeIdentity
(
   TCS_CONTEXT_HANDLE hContext,                     // in
   TCPA_ENCAUTH       identityAuth,                 // in
   TCPA_CHOSENID_HASH IDLabel_PrivCAHash,           // in
   UINT32             idIdentityKeyInfoSize,        // in
   BYTE*              idIdentityKeyInfo,            // in
   TPM_AUTH*          pSrkAuth,                     // in, out
   TPM_AUTH*          pOwnerAuth,                   // in, out
   UINT32*            idIdentityKeySize,            // out
   BYTE**             idIdentityKey,                // out
   UINT32*            pcIdentityBindingSize,        // out
   BYTE**             prgbIdentityBinding,          // out
   UINT32*            pcEndorsementCredentialSize,  // out
   BYTE**             prgbEndorsementCredential,    // out
   UINT32*            pcPlatformCredentialSize,     // out
   BYTE**             prgbPlatformCredential,       // out
   UINT32*            pcConformanceCredentialSize,  // out
   BYTE**             prgbConformanceCredential     // out
);
```

IDL-Definition:

```
[helpstring("method Tcsip_MakeIdentity")]
TSS_RESULT Tcsip_MakeIdentity
(
   [in]TCS_CONTEXT_HANDLE   hContext,                      // in
   [in]TCPA_ENCAUTH         identityAuth,                  // in
   [in]TCPA_CHOSENID_HASH   IDLabel_PrivCAHash,            // in
   [in]UINT32               idIdentityKeyInfoSize,         // in
   [in, size_is( idIdentityKeyInfoSize )]BYTE*
                            idIdentityKeyInfo,             // in
   [in, out]TPM_AUTH*       pSrkAuth,                      //    in,
                                   out
   [in, out]TPM_AUTH*       pOwnerAuth,                    //    in,
                                   out
   [out]UINT32*             idIdentityKeySize,             // out
   [out, size_is(, *idIdentityKeySize )]BYTE**
                            idIdentityKey,                 // out
   [out]UINT32*             pcIdentityBindingSize,         // out
   [out, size_is(, *pcIdentityBindingSize )]BYTE**
                            prgbIdentityBinding,           // out
   [out]UINT32*             pcEndorsementCredentialSize,   // out
   [out, size_is(, *pcEndorsementCredentialSize )]BYTE**
                            prgbEndorsementCredential,     // out
   [out]UINT32*             pcPlatformCredentialSize,      // out
   [out, size_is(, *pcPlatformCredentialSize)]BYTE**
                            prgbPlatformCredential,        // out
   [out]UINT32*             pcConformanceCredentialSize,   // out
   [out, size_is(, *pcConformanceCredentialSize )]BYTE**
                            prgbConformanceCredential      // out
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle to established context.
TCPA_ENCAUTH	identityAuth	Encrypted usage authorization data for new identity.
TCPA_CHOSENID_HASH	IDLabel_PrivCAHash	The digest of the identity label and privacy CA chosen for the new TPM identity.
UINT32	pcIdentityKeySize	Size of the provided/returned byte stream in bytes.
BYTE**	prgbIdentityKey	IN: byte stream containing all parameters defining how to create the new identity key.

		OUT: new created identity key wrapped by SRK.
TPM_AUTH*	pSrkAuth	SRK authorization session data including the HMAC digest.If NULL, no authorization is required.
TPM_AUTH*	pOwnerAuth	Owner authorization session data including the HMAC digest. If NULL, no authorization is required.
UINT32*	pcIdentityBindingSize	Returned size of output for identity binding information in bytes.
BYTE**	prgbIdentityBinding	Return pointer to memory containing signature of TCPA_IDENTITY_CONTENTS using the private key of the new created identity key.
UINT32*	pcEndorsementCredentialSize	Returned size of output for endorsement credential information in bytes.
BYTE**	prgbEndorsementCredential	Return pointer to memory containing the endorsement credential.

Comment:

TCS processes the TPM_MakeIdentity command, which requires an owner authorization.

The TCS Credential Management Service MUST return the endorsement credential, the platform credential and the conformance credential only after a TPM identity was successfully created.

The TCS allocates memory resources for each of these credentials, the identity binding information, the public key and the encrypted private key data of the identity and binds the memory resources to the provided hContext.

Return Value:

TCS_SUCCESS
TCS_E_FAIL

5.6.2.7.2 Tcsip_MakeIdentity2

Start of informative comment:

This function performs the TPM operations necessary to create an identity key. It is identical to Tcsip_MakeIdentity except that it does not return the associated credentials. This can be used in conjunction with Tcsip_GetCredentials to duplicate the functionality of Tcsip_MakeIdentity.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_MakeIdentity2
   (
   TCS_CONTEXT_HANDLE   hContext,                   // in
   TCPA_ENCAUTH         identityAuth,               // in
   TCPA_CHOSENID_HASH   IDLabel_PrivCAHash,         // in
   UINT32               idIdentityKeyInfoSize,      // in
   BYTE*                idIdentityKeyInfo,          // in
   TPM_AUTH*            pSrkAuth,                   // in, out
   TPM_AUTH*            pOwnerAuth,                 // in, out
   UINT32*              idIdentityKeySize,          // out
   BYTE**               idIdentityKey,              // out
   UINT32*              pcIdentityBindingSize,      // out
   BYTE**               prgbIdentityBinding         // out
   );
```

IDL-Definition:

```
[helpstring("method Tcsip_MakeIdentity2")]
TSS_RESULT Tcsip_MakeIdentity2
   (
   [in]TCS_CONTEXT_HANDLE  hContext,                // in
   [in]TCPA_ENCAUTH        identityAuth,            // in
   [in]TCPA_CHOSENID_HASH  IDLabel_PrivCAHash,      // in
   [in]UINT32              idIdentityKeyInfoSize,   // in
   [in, size_is( idIdentityKeyInfoSize )]BYTE*
                                 idIdentityKeyInfo, // in
   [in, out]TPM_AUTH*      pSrkAuth,                // in, out
   [in, out]TPM_AUTH*      pOwnerAuth,              // in, out
   [out]UINT32*            idIdentityKeySize,       // out
   [out, size_is(, *idIdentityKeySize )]BYTE**
                           idIdentityKey,           // out
   [out]UINT32*            pcIdentityBindingSize,   // out
   [out, size_is(, *pcIdentityBindingSize )]BYTE**
                                 prgbIdentityBinding  // out
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle to established context.
TCPA_ENCAUTH	identityAuth	Encrypted usage authorization data for new identity.
TCPA_CHOSENID_HASH	IDLabel_PrivCAHash	The digest of the identity label and privacy CA chosen for the new TPM identity.
UINT32	pcIdentityKeySize	Size of the provided/returned byte stream in bytes.
BYTE**	prgbIdentityKey	IN: byte stream containing all parameters defining how to create the new identity key. OUT: new created identity key wrapped by SRK.
TPM_AUTH*	pSrkAuth	SRK authorization session data including the HMAC digest. If NULL, no authorization is required
TPM_AUTH*	pOwnerAuth	Owner authorization session data including the HMAC digest. If NULL, no authorization data is required.
UINT32*	pcIdentityBindingSize	Returned size of output for identity binding informaiton in bytes
BYTE**	prgbIdentityBinding	Return pointer to memory contianing signature of TCPA_IDENTITY_CONTENTS using the private key of the newly created identiy key.

Comment:

TCS processes the TPM_MakeIdentity command, which requires an owner authorization.

The TCS allocates memory resources for the identity binding information, the public key and the encrypted private key data of the identity and binds the memory resources to the provided hContext.

Return Value:

TCS_SUCCESS
TCS_E_FAIL

5.6.2.7.3 Tcsi_GetCredentials

Start of informative comment:

Tcsi_GetCredentials returns the endorsement, platform, and conformance credentials for a platform. These are the same credentials returned by Tcsip_MakeIdentity; however this function *only* returns the credentials, it does not create an identity key.

This function is intended to allow the TSP to retrieve the credentials when an identity key is created by a method other than Tcsip_MakeIdentity (for example, when Tcsip_ExecuteTransport is used to wrap a TPM_MakeIdentity command).

The credentials contain information that is unique to the the platform. Since this function does not require any authorization, administrators who make this function available to remote platforms (see section 1.9.5 “Remote Procedure Calls”) should be aware that they are making the machine identifiable to remote hosts.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsi_GetCredentials
   (
   TCS_CONTEXT_HANDLE  hContext,                        // in
   UINT32*              pcEndorsementCredentialSize,  // out
   BYTE**               prgbEndorsementCredential,    // out
   UINT32*              pcPlatformCredentialSize,     // out
   BYTE**               prgbPlatformCredential,       // out
   UINT32*              pcConformanceCredentialSize,  // out
   BYTE**               prgbConformanceCredential     // out
   );
```

IDL-Definition:

```
[helpstring("method Tcsi_GetCredentials")]
TSS_RESULT Tcsi_GetCredentials
   (
   [in]TCS_CONTEXT_HANDLE  hContext,                          // in
   [out]UINT32*            pcEndorsementCredentialSize,      // out
   [out, size_is(, *pcEndorsementCredentialSize )]BYTE**
                              prgbEndorsementCredential,    // out
   [out]UINT32*               pcPlatformCredentialSize,     // out
   [out, size_is(, *pcPlatformCredentialSize
                           )]BYTE**prgbPlatformCredential,  // out
   [out]UINT32*               pcConformanceCredentialSize,  // out
   [out, size_is(, *pcConformanceCredentialSize )]BYTE**
                           prgbConformanceCredential        // out
   );
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle to established context.
UINT32	pcEndorsementCredentialSize	Returned size of output for endorsement credential information in bytes.
BYTE**	prgbEndorsementCredential	Return pointer to memory containing the endorsement credential
UINT32*	pcPlatformCredentialSize	Returned size of output for platform credential information in bytes
TPM_AUTH*	prgbPlatformCredential	Return pointer ot memory containing the platform credential
UINT32*	PcConformanceCredentialSize	Returned size of output for conformance credential informaiotn in bytes
BYTE**	prgbConformanceCredential	Return pointer to memory containig the conformance credential

Comment:

The TCS Credential Management Service must return the endorsement credential, the platform credential and the conformance credential only after a TPM identity was successfully created.

The TCS allocates memory resources for each of these credentials and binds the memory resources to the provided hContext.

Return Value:

TCS_SUCCESS
TCS_E_FAIL

5.6.3 TCS Use Models

5.6.3.1 TCS Load Key by UUID

Assumptions:

- Key Hierarchy: all keys registered in Persistent Storage

```
SRK                               No authorization required
      Key1                        No authorization required
            Key2                  Auth required
                  Key3            No authorization required
                        Key4      No authorization required
```

- No key is loaded in TPM
- TCS knows that loading Key3 will fail since Key2 requires authorization.

Application Pseudo Code:

```
TCS_LOADKEY_INFO LoadKeyInfo
TSS_UUID keyUUID;
TCS_KEY_HANDLE tcsiKeyHandle;

// . . .

      // initialze TSS_UUID structure with UUID of key4
InitUuid (keyUUID, key4);
      // load key4 by UUID
while (TCS_E_KM_LOADFAILED == Tcsip_LoadKeyByUuid(hContext,
                                          keyUUID,
                                          &tcsiKeyHandle,
                                          &LoadInfo))
{
    // 1) initialize an OIAP session
    // 2) calculate the authorization HMAC digest
    //     using the secret of key2 and LoadInfo.paramDigest
    // 3) fill in the auth session data and HMAC digest of step
#2
    //     in LoadInfo.authData
}

// get public key of already loaded key4
// addressed by the application key handle tcsiKeyHandle,
// no authorization is required, since key4 requires no
authorization
```

```
TCPA_PUBKEY* pPubKey = NULL;
Tcsip_GetPubKey(tcsiKeyHandle,
              NULL,
              &pPubKey);

// evict key4 from Key Cache Manager and
// free allocated resources assigned to tcsiKeyHandle
Tcsi_EvictKey(tcsiKeyHandle);

// . . .
```

Comments:

TCS actions on first Tcsip_LoadKeyByUuid() call:

- load Key1, no authorization is required for SRK
- load Key2, no authorization is required for key1
- return from function call with a failure providing the TCS_LOADKEY_INFO blob (KeyUUID = Key3, parentKeyUUID = Key2).

TCS actions on second Tcsip_LoadKeyByUuid() call:

- load Key3 using the provided LOADKEY_INFO data (including the authortization data: authData)
- load Key4, no authorization is required for key3
- return from function call with TCS_SUCCESS

5.7 TCS Event Manager

5.7.1 TCS Event Manager Functions and Operations

Start of informative comment:

The TCS Event Log Services maintains the TCG Event Log. The TCS Event Log Services allow TPM extend PCR events to be logged, and allow challengers interested in extend PCR information contained in these logs, to access it.

End of informative comment.

TCS Event Manager structures and functions use the idioms: PcrEvent or TSS_PCR_EVENT (and not just “event”) in order to distinguish them from the TCS Audit Manager structures and functions, which also use the word event.

5.7.2 TCS Event Manager Interface

5.7.2.1 TCS Event Manager Interface Structures and Definitions

5.7.2.1.1 TCS The Event Log

Start of informative comment:

The TCS Event Log Services maintain a database of events called the Event Log. Conceptually, this log will consist of an array of events in which each entry is in the format of TSS_PCR_EVENT (defined below).

TCG defines certain event-type information (for instance, validation certificates). Other application-specific types may be added using the naming convention described.

The Event Log need not be held in TCG-shielded locations, and the logging and retrieval operations need not be TCG-protected capabilities. This is because servers or other software can detect tampering with the log.

The TCS is free to reallocate Event Log storage as it sees fit. The TCS also is free to maintain additional data structures that permit fast random access to events.

End of informative comment.

5.7.2.2 TCS Event Manager Interface Functions

Start of informative comment:

The Tcsi_LogPcrEvent operation adds a new event to the end of the array associated with the named PCR.

Logged information is retrieved using the Tcsi_GetPcrEvent, Tcsi_GetPcrEventsByPcr and Tcsi_GetPcrEventLog calls.

In Tcsi_GetPcrEvent, events are accessed by PCR index and number. Tcsi_GetPcrEventsByPcr returns a pointer to a data structure describing events related with a single PCR. Tcsi_GetPcrEventLog returns a pointer to a data structure that describes the entire log.

End of informative comment.

5.7.2.2.1 Tcsi_LogPcrEvent

Start of informative comment:

The Tcsi_LogPcrEvent operation adds a new event to the end of the array associated with the named PCR.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsi_LogPcrEvent
(
   TCS_CONTEXT_HANDLE              hContext,  // in
   TSS_PCR_EVENT                   Event,     // in
   UINT32*                         pNumber    // out
);
```

IDL-Definition:

```
[helpstring("method Tcsi_LogPcrEvent")]
TSS_RESULT Tcsi_LogPcrEvent
(
   [in] TCS_CONTEXT_HANDLE         hContext,
   [in] TSS_PCR_EVENT              Event
   [out] UINT32*                   pNumber
);
```

Parameter Description:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle to established context.
TSS_PCR_EVENT	Event	Details of the event being logged.
UNIT32*	pNumber	The number of the event just logged is returned in this variable. The TCS number events for each PCR

		monotonically from 0.

Action:

The Tcsi_LogPcrEvent operation MUST add supporting information for the named TPM_Extend event to the end of the Event Log. The TCS MUST maintain an array of event-supporting data with events identified by the register to which they belong and the order in which the events occurred.

The log need not be in a TCG-shielded location, and the Tcsi_LogPcrEvent action need not be a TCG-protected capability. The TCS MUST NOT impose arbitrary size limitations on the size of the event log. The event log size should be limited by physical memory, memory accessible in the given operating mode, or memory allocated to the log by system firmware or other software.

TSS_PCR_EVENT→PCRValue should be the actual digest-sized event passed to TPM_Extend.

Return Value:

TCS_SUCCESS
TCS_E_BAD_INDEX
TCS_E_BAD_PARAMETER
TCS_E_OUTOFMEMORY
TCS_E_FAIL

5.7.2.2.2 Tcsi_GetPcrEvent

Start of informative comment:

Tcsi_GetPcrEvent is used to retrieve events logged with Tcsi_LogPcrEvent .

Tcsi_GetPcrEvent need not be a TCG-protected capability, and the log events retrieved need not be in TCG-shielded locations. Tcsi_GetPcrEvent returns all the data stored in TSS_PCR_EVENT.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsi_GetPcrEvent
(
    TCS_CONTEXT_HANDLE              hContext,  // in
    UINT32                          PcrIndex,  // in
    UINT32*                         pNumber,   // in, out
    TSS_PCR_EVENT**                 ppEvent    // out
);
```

IDL-Definition:

```
[helpstring("method Tcsi_GetPcrEvent")]
TSS_RESULT Tcsi_GetPcrEvent
(
    [in] TCS_CONTEXT_HANDLE         hContext,
    [in] UINT32                     PcrIndex,
    [in, out] UINT32*               pNumber,
    [out] TSS_PCR_EVENT**           ppEvent
);
```

Parameter Description:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle to established context.
UINT32	PcrIndex	The index of the PCR.
UINT32*	pNumber	Number of event required. Events are numbered from 0 to the number of events logged on the named PCR.
TSS_PCR_EVENT**	ppEvent	Pointer to the returned event.

Actions:

The Tcsi_GetPcrEvent operation retrieves events previously logged using Tcsi_LogPcrEvent. The format of the data returned is identical to that previously logged. This operation interface retrieves log entries by PCR index and event number. On TCS initialization the event log for each PCR is empty. Then, for each PCR, the first event logged is numbered 0; the next is numbered 1, and so on. Attempts to receive log items beyond the end of the log return an error.

The Tcsi_GetPcrEvent allocates memory for the event and returns a pointer to the event. The caller can free this memory by calling the CS function: FreeMemory.

If ppEvent == NULL, Tcsi_GetPcrEvent returns the number of actually logged events in pNumber.

Note that that the event log is required to be accessible in the form of an array. TCS implementation MAY choose to provide supplemental data structures to make random array access through Tcsi_GetPcrEvent more efficient.

Return Value:

TCS_SUCCESS
TCS_E_BAD_INDEX
TCS_E_SIZE
TCS_E_FAIL

5.7.2.2.3 Tcsi_GetPcrEventsByPcr

Start of informative comment:

Tcsi_GetPcrEventsByPcr returns an event log bound to a single PCR. The event log is returned as an ordered sequence of TSS_PCR_EVENT structures.

The caller can limit the size of the returned array using EventCount. The caller can also specify the number of the first event on the returned event log using FirstEvent. These controls allow the caller to retrieve the event log step by step, or to retrieve a partial event log when required.

The array elements are of variable size, and the TSS_PCR_EVENT structure defines the size of the current event and the register with which it is associated. This data structure is not required to be thread-safe, so upper-level software should ensure that it is not modified during parsing. If the event log is kept in a TCG-shielded location, then a copy must be made in an unprotected area that can be traversed by non-TPM protected calling code.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsi_GetPcrEventsByPcr
(
   TCS_CONTEXT_HANDLE              hContext,        // in
   UINT32                          PcrIndex,        // in
   UINT32                          FirstEvent,      // in
   UINT32*                         pEventCount,     // in,out
   TSS_PCR_EVENT**                 ppEvents         // out
);
```

IDL-Definition:

```
[helpstring("method Tcsi_GetPcrEventsByPcr")]
TSS_RESULT Tcsi_GetPcrEventsByPcr
(
   [in] TCS_CONTEXT_HANDLE                          hContext,
   [in] UINT32                                      PcrIndex,
   [in] UINT32                                      FirstEvent,
   [in, out] UINT32*                                pEventCount,
   [out, size_is(, *pEventCount)] TSS_PCR_EVENT**   ppEvents
);
```

Parameter Description:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle to established context.
UINT32	PcrIndex	The index of the PCR.
UINT32	FirstEvent	The number of the first event in the returned array.
UINT32*	pEventCount	In: max number of events to

		be returned. Out: actual number of events in the returned array.
TSS_PCR_EVENT**	ppEvents	Pointer to the event log returned as an ordered sequence of TSS_PCR_EVENT structures.

Actions:

This command returns a pointer to a “Partial PCR Event Log”, which is an array reflecting a list of events bound to a single PCR, starting with event number: FirstEvent. The first event of the PCR event log is indexed with FirstEvent=0.

The size of the “Partial PCR Event Log” is determined by the number of events bound to that PCR, and the input value of EventCount:

If EventCount is set to –1, or if EventCount is greater than the actual number of events (related to that PCR and numbered above FirstEvent), the “Partial PCR Event Log” will consist all events related to that PCR and numbered above FirstEvent. In this case the command sets EventCount to that actual size.

If EventCount is smaller than the number of events (related to that PCR and numbered above FirstEvent) the command will return the first “EventCount” number of events starting with the event numbered: FirstEvent. In this case the returned EventCount is the same as the input EventCount.

If FirstEvent points to an event number that does not exist for that PCR, the command fails with the error code TCS_E_BAD_PARAMETER.

Return Value:

TCS_SUCCESS
TCS_E_BAD_INDEX
TCS_E_BAD_PARAMETER
TCS_E_SIZE
TCS_E_FAIL

5.7.2.2.4 Tcsi_GetPcrEventLog

Start of informative comment:

Tcsi_GetPcrEventLog returns the event log of all events since the TPM was initialized. The event log is returned as an ordered sequence of TSS_PCR_EVENT structures in the following order: all events bound to PCR 0 (in the order they have arrived), all events bound to PCR 1 (in the order they have arrived), etc.

The array elements are of variable size, and the TSS_PCR_EVENT structure defines the size of the current event and the register with which it is associated. This data structure is not required to be thread-safe, so upper-level software should ensure that it is not modified during parsing. If the event log is kept in a TCG-shielded location, then a copy must be made in an unprotected area that can be traversed by non-TPM protected calling code.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsi_GetPcrEventLog
(
    TCS_CONTEXT_HANDLE              hContext,       // in
    UINT32*                         pEventCount,    // out
    TSS_PCR_EVENT**                 ppEvents        // out
);
```

IDL-Definition:

```
[helpstring("method Tcsi_GetPcrEventLog")]
TSS_RESULT Tcsi_GetPcrEventLog
(
    [in] TCS_CONTEXT_HANDLE                        hContext,
    [out] UINT32*                                  pEventCount,
    [out, size_is(, *pEventCount)] TSS_PCR_EVENT**  ppEvents
);
```

Parameter Description:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle to established context.
UINT32*	pEventCount	Number of entries in the entire Event Log is returned in this variable.
TSS_PCR_EVENT**	ppEvents	Pointer to the head of the Event Log data structures.

Action:

This command returns to the caller the complete Event Log. When the Event Log is empty, Tcsi_GetPcrEventLog will return TCS_SUCCESS with EventCount set to 0.

The returned Event Log MUST consists of pointers to all events, in the following order (N indicated the number of PCRs in a particular TPM):

- All events bound to PCR 0, in increasing order starting from event number 0 of PCR 0.
- All events bound to PCR 1, in increasing order starting from event number 0 of PCR 1.
- All events bound to PCR N-1, in increasing order starting from event number 0 of PCR N-1.

Return Value:

TCS_SUCCESS
TCS_E_SIZE
TCS_E_FAIL

5.7.2.2.5 Tcsi_GetCredential

Start of informative comment:

Tcsi_GetCredential returns the addressed credential data (e.g. Endorsement, platform, or conformance) for a platform. These are the same credentials returned by Tcsip_MakeIdentity; however this function *only* returns the credential data.

This function is intended to allow the TSP to retrieve the credentials when an identity key is created by a method other than Tcsip_MakeIdentity (for example, when Tcsip_ExecuteTransport is used to wrap a TPM_MakeIdentity command).

The credentials contain information that is unique to the platform. Since this function does not require any authorization, administrators who make this function available to remote platforms (see section 19.9.6 “Remote Procedure Calls”) should be aware that they are making the machine identifiable to remote hosts.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsi_GetCredential
(
TCS_CONTEXT_HANDLE     hContext,                  // in
UINT32                 ulCredentialType,          // in
UINT32                 ulCredentialAccessMode,    // in
UINT32*                pulCredentialSize,         // out
TSS_PCR_EVENT**        prgbCredentialData         // out
);
```

IDL-Definition:

```
[helpstring("method Tcsi_GetCredential")]
TSS_RESULT Tcsi_GetCredential
(
[in]  TCS_CONTEXT_HANDLE   hContext,
[in]  UINT32               ulCredentialType,
[in]  UINT32               ulCredentialAccessMode,
[out] UINT32*              pulCredentialSize,
[out, size_is(, *pEventCount)] TSS_PCR_EVENT**   prgbCredentialData
);
```

Parameter Description:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle to established context
UINT32	ulCredentialType	Addressed credential data set
UINT32	ulCredentialAccessMode	Credential data access mode selection
UINT32*	pulCredentialSize	Size of the output credential, in bytes
TSS_PCR_EVENT**	prgbCredentialData	Pointer to memory containing the credential data

Comment:

The TCS Credential Management Service must return the addressed credential data (e.g. Endorsement, platform or conformance).

Return Value:

TCS_SUCCESS
TCS_E_FAIL

5.8 TCS TPM Parameter Block Generator

5.8.1 TCS TPM Parameter Block Generator Functions and Operations

Start of informative comment:

The TPM parameter block generator is a functional block within TSS Core Services (TCS). External access to this block is via the TSS Core Services Interface (TCSI). Direct access to the TPM device from applications is not permitted. The TPM parameter block generator is responsible for serializing, synchronizing, and processing TPM commands. This block builds byte streams to input to the TPM and converts byte streams output from the TPM. This block provides access to the TPM commands for TCG applications via the TCS platform service.

The TPM parameter block generator also contains internal interfaces to the Key and Credential Manager, Audit Manager, and Event Manager functional blocks. Interaction with these TCS blocks is required to support TPM data management, in and out of the TPM device. These blocks require knowledge of the data that is passing in/out of the TPM. Since these interfaces are internal to the TCS platform service, they are not discussed in this document.

End of informative comment.

5.8.2 TCS TPM Parameter Block Generator Interface

5.8.2.1 Functions

Start of informative comment:

The TPM parameter block generator exposes protected TPM commands through a set of interfaces. TPM functionality is exposed through the Tcsi via the TPM parameter block generator functional block. All TPM commands must pass through the TPM parameter block generator. This function set consists of TPM functions that TCG-enabled and TPM management applications require. The TPM parameter block generator communicates with the TPM using the low-level TPM diver via the TPM DDL. The function declarations, C-style and IDL, are defined in this document, while the TPM parameter block definitions are documented in the TCG 1.1b Main Specification.

Interface Types:

Every function must be defined in C, IDL, and TSS Parameter Block format. Individual computing platforms do not need to expose every interface type. A default interface type must be decided for each platform. For instance, the default Tcsi interface for a PC may be a C-style interface. Any additional interfaces could be implemented by adding encoder/decoder modules that plug into the C-style interface. The possibility of all interfaces must exist to support multiple platform types.

Authorization:

The TPM parameter block generator does not have knowledge of any authorization secrets for protected services. Therefore, all authorization data associated with the TPM function must be generated outside of the TCS. This includes nonce generation, HMAC (Auth Data) generation, and TPM response validation.

Context Control:

Before performing any TPM command, an application must first call Tcsi_OpenContext to obtain a context for a TPM command session. The application controls the memory resources during the session using Tcsi_FreeMemory and must close the context using Tcsi_CloseContext when it is finished.

End of informative comment.

For all functions in this section TSS_RESULT is the result from the TPM command. For the command ordinal and return values see the TCG 1.1b Main Specification.

5.8.2.2 TPM Ownership, Authorization, and Identity

5.8.2.2.1 Tcsip_SetOwnerInstall

Start of informative comment:

Tcsip_SetOwnerInstall determines if the TPM has a current owner. The TPM validates the assertion of physical access and then sets the value of TCPA_PERSISTENT_FLAGS -> ownership to the value in state.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_SetOwnerInstall
(
   TCS_CONTEXT_HANDLE hContext,  // in
   TSS_BOOL           state      // in
);
```

IDL Definition:

```
[helpstring("method Tcsip_SetOwnerInstall")]
TSS_RESULT Tcsip_SetOwnerInstall
(
   [in] TCS_CONTEXT_HANDLE  hContext,
   [in] TSS_BOOL            state
);
```

Parameters:

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TSS_BOOL	state	New disable flag state.

Comment:

TPM command – TPM_SetOwnerInstall
TPM ordinal – TPM_ORD_SetOwnerInstall

5.8.2.2.2 Tcsip_TakeOwnership

Start of informative comment:

Tcsip_TakeOwnership inserts the Owner-authorization data and creates a new Storage Root Key (SRK). This function fails if there is already an TPM Owner set.

After inserting the authorization data, this function creates the SRK.

To validate that the operation completes successfully, the TPM HMACs the response to the Tcsip_TakeOwnership function.

During Tcsip_TakeOwnership the TCS must add the SRK data to the persistent store. The information must not contain the RSA modulus n and should contain all other information.

If ownership is cleared the TCS should invalidate any registry info for the SRK. If ownership is cleared while the TCS is not running the TCS should detect this at startup and update the registry then.

If ownership is taken while the TCS is not running the TCS does not need to detect this and add or update the SRK registry entry. In this case it is the responsibility of the new owner to update the key registry.

During Tspi_TPM_TakeOwnership the hKeySRK is updated from the new SRK blob returned by the TPM, including the new RSA modulus field.

The TCS must allow the application to unregister the SRK altogether so that no information about the SRK is in the persistent store. (This means an application will not be able to determine the PCR restrictions or the keyFlags for the SRK unless they have been stored outside the TSS; there is not way to retrieve those fields from the TPM).

If the TPM has an owner the TCS must allow the application to register the SRK in the TCS persistent store. If the key blob being registered contains the RSA modulus n, this data will be stored in the persistent store and will be available to all applications via Tcsi_GetRegisteredKeyBlob. However just like any other key, if there is already information about the SRK in the persistent store the application must first unregister the existing blob before inserting the new data.

Whether the TCS persistent store has no, partial, or full information about the SRK blob, Tspi_Context_GetRegisteredKeysByUUID will always be able to fill in a correct TSS_KM_KEYINFO structure for the SRK if the TPM has an owner. The TCS must ensure that this information is correct.

Whether the TCS persistent store has no, partial, or full information about the SRK blob, Tspi_Context_GetKeyByUUID(TSS_UUID_SRK) will always succeed if the TPM has an owner. The function will return a TSS_HKEY associated with the SRK. Some or all GetAttrib calls may fail with TSS_E_BAD_PARAMETER if the SRK information is not stored in the persistent store.

The bAuthDataUsage field of the TSS_KM_KEYINFO structure remains a bi-valued field and its value indicates only whether authorization is required to load children. Both TSS_KEYAUTH_AUTH_ALWAYS and TSS_KEYAUTH_AUTH_PRIV_USE_ONLY are mapped to the value 0x01.

Whether the TCS persistent store has no, partial, or full information about the SRK blob, Tcsi_GetRegisteredKeyBlob(TSS_UUID_SRK) will succeed if the TPM has an owner. If the registry has no key information the TCS should return a 0-length blob.

The RSA modulus n is removed from a TPM_KEY or TPM_KEY12 by setting the pubKey.keyLength to 0 and pubKey.key to NULL.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_TakeOwnership
(
   TCS_CONTEXT_HANDLE hContext,          // in
   UINT16             protocolID,        // in
   UINT32             encOwnerAuthSize,  // in
   BYTE*              encOwnerAuth,      // in
   UINT32             encSrkAuthSize,    // in
   BYTE*              encSrkAuth,        // in
   UINT32             srkKeyInfoSize,    // in
   BYTE*              srkKeyInfo,        // in
   TPM_AUTH*          ownerAuth,         // in, out
   UINT32*            srkKeyDataSize,    // out
   BYTE**             srkKeyData         // out
);
```

IDL-Definition:

```
[helpstring("method Tcsip_TakeOwnership")]
TSS_RESULT Tcsip_TakeOwnership
(
   [in]TCS_CONTEXT_HANDLE                   hContext,          //
                                      in
   [in]UINT16                               protocolID,        //
                                      in
   [in]UINT32                               encOwnerAuthSize,  //
                                      in
   [in, size_is( encOwnerAuthSize )]BYTE*   encOwnerAuth,      //
                                      in
   [in]UINT32                               encSrkAuthSize,    //
                                      in
   [in, size_is( encSrkAuthSize )]BYTE*     encSrkAuth,        //
                                      in
   [in]UINT32                               srkKeyInfoSize,    //
                                      in
   [in, size_is( srkKeyInfoSize )]BYTE*     srkKeyInfo,        //
                                      in
   [in, out]TPM_AUTH*                       ownerAuth          //
                                      in, out
   [out]UINT32*                             srkKeyDataSize,    //
                                      out
   [out, size_is(,*srkKeyDataSize )]BYTE**  srkKeyData         //
                                      out
);
```

Parameters:

Type	Name	Description

TCS_CONTEXT_HANDLE	HContext	Handle of established context.
UINT32	ProtocolID	The ownership protocol in use.
UINT32	encOwnerAuthSize	The size of the encrypted owner authorization data.
BYTE*	encOwnerAuth	The encrypted owner authorization data.
UINT32	encSrkAuthSize	The size of the encrypted SRK authorization data.
BYTE*	EncSrkAuth	The encrypted Storage Root Key (SRK) authorization data.
UINT32*	SrkSize	The size of the TCPA_KEY byte stream
BYTE**	Srk	TCPA_KEY byte stream of the storage root key blob
TPM_AUTH*	OwnerAuth	The authorization from the TPM Owner. There is no validation of in parameters, just validation on the return that the proper authorization data was used. HMAC Key: the new ownerAuth value.

Comment:

TPM command – TPM_TakeOwnership
TPM ordinal – TPM_ORD_TakeOwnership

5.8.2.2.3 Tcsip_OIAP

Start of informative comment:

Tcsip_OIAP allows the creation of an authorization handle and the tracking of the handle by the TPM. The TPM generates the handle and nonce.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_OIAP
(
   TCS_CONTEXT_HANDLE hContext,    // in
   TCS_AUTHHANDLE*    authHandle,  // out
   TCPA_NONCE*        nonce0       // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_OIAP")]
TSS_RESULT Tcsip_OIAP
(
   [in] TCS_CONTEXT_HANDLE  hContext,
   [out] TCS_AUTHHANDLE*    authHandle,
   [out] TCPA_NONCE*        nonce0
);
```

Parameters:

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_AUTHHANDLE*	authHandle	Handle that TPM creates that points to the authorization state. The value is TPM specific and has no meaning except to identity the session.
TCPA_NONCE*	nonce0	Nonce generated by TPM and associated with session.

Comment:

TPM command – TPM_OIAP
TPM ordinal – TPM_ORD_OIAP

5.8.2.2.4 Tcsip_OSAP

Start of informative comment:

TPM_OSAP creates the authorization handle, the shared secret and generates nonceEven and nonceEvenOSAP. Note to TSS implementers: 1.2 TPMs may support a number of symmetric encryption algorithms (including XOR, AES, and 3DES) for use in OSAP. NIST recommends not using XOR, so if a TPM supports either AES or 3DES, it is recommended that one of those be used instead.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_OSAP
(
   TCS_CONTEXT_HANDLE hContext,     // in
   TCPA_ENTITY_TYPE   entityType,   // in
   UINT32             entityValue,  // in
   TCPA_NONCE         nonceOddOSAP, // in
   TCS_AUTHHANDLE*    authHandle,   // out
   TCPA_NONCE*        nonceEven,    // out
   TCPA_NONCE*        nonceEvenOSAP // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_OSAP")]
TSS_RESULT Tcsip_OSAP
(
   [in]  TCS_CONTEXT_HANDLE hContext,
   [in]  TCPA_ENTITY_TYPE   entityType,
   [in]  UINT32             entityValue,
   [in]  TCPA_NONCE         nonceOddOSAP,
   [out] TCS_AUTHHANDLE*    authHandle,
   [out] TCPA_NONCE*        nonceEven,
   [out] TCPA_NONCE*        nonceEvenOSAP
);
```

Parameters:

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCPA_ENTITY_TYPE	entityType	The type entity in use.
UINT32	entityValue	The selection value based on entity type.
TCPA_NONCE	nonceOddOSP	The nonce generated by the caller associated with shared secret.
TCS_AUTHHANDLE*	authHandle	Handle which points to an authorization state.
TCPA_NONCE*	nonceEven	Nonce generated by TPM and associated with a session.
TCPA_NONCE*	nonceEvenOSAP	Nonce generated by TPM and

		associated with shared secret.

Comment:

TPM command – TPM_OSAP
TPM ordinal – TPM_ORD_OSAP

5.8.2.2.5 Tcsip_ChangeAuth

Start of informative comment:

Tcsip_ChangeAuth allows the owner of an entity to change the authorization data for the entity.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_ChangeAuth
(
   TCS_CONTEXT_HANDLE hContext,       // in
   TCS_KEY_HANDLE     parentHandle,   // in
   TCPA_PROTOCOL_ID   protocolID,     // in
   TCPA_ENCAUTH       newAuth,        // in
   TCPA_ENTITY_TYPE   entityType,     // in
   UINT32             encDataSize,    // in
   BYTE*              encData,        // in
   TPM_AUTH*          ownerAuth,      // in, out
   TPM_AUTH*          entityAuth,     // in, out
   UINT32*            outDataSize,    // out
   BYTE**             outData         // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_ChangeAuth")]
TSS_RESULT Tcsip_ChangeAuth
(
   [in] TCS_CONTEXT_HANDLE       hContext,
   [in] TCS_KEY_HANDLE           parentHandle,
   [AUTH, in] TCPA_PROTOCOL_ID   protocolID,
   [AUTH, in] TCPA_ENCAUTH       newAuth,
   [AUTH, in] TCPA_ENTITY_TYPE   entityType,
   [AUTH, in] UINT32             encDataSize,
   [AUTH, in, size_is(endDataSize)] BYTE*   encData,
   [in, out] TPM_AUTH*           ownerAuth,
   [in, out] TPM_AUTH*           entityAuth,
   [AUTH, out] UINT32*           outDataSize,
   [AUTH, out, size_is(, *outDataSize)] BYTE** outData
);
```

Parameters:

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_KEY_HANDLE	parentHandle	Handle of the parent key to the entity.
TCPA_PROTOCOL_ID	protocolID	The ownership protocol in use.
TCPA_ENCAUTH	newAuth	The encrypted new authorization data for the entity. The encryption key is the shared secret from the OS-AP protocol.
TCPA_ENTITY_TYPE	entityType	The type of entity to be modified.
UINT32	encDataSize	The size of the encrypted entity.
BYTE	encData	The encrypted entity that is to be modified.
UINT32*	outDataSize	The size of the modified encrypted entity.
BYTE**	outData	The modified encrypted entity.
TPM_AUTH*	ownerAuth	The authorization and inputs from the TPM owner. There is no validation of in parameters, just validation on the return that the proper authorization data was used. HMAC key: parentKey.usageAuth.
TPM_AUTH*	entityAuth	The authorization and inputs from the encrypted entity. There is no validation of in parameters, just validation on the return that the proper authorization data was used. HMAC key: entity.usageAuth.

Comment:

TPM command – TPM_ChangeAuth
TPM ordinal – TPM_ORD_ChangeAuth

If the entity to be changed is a key the Key Manager SHALL replace key blobs associated with this key with *new key blobs that contain the new authorization value.*

Note: One method for finding the associated key would be to compare the value (or hash of the value) of encData with values stored in the Key Manager.

5.8.2.2.6 Tcsip_ChangeAuthOwner

Start of informative comment:

Tcsip_ChangeAuthOwner allows the owner of an entity to change the authorization data for the TPM Owner or the SRK.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_ChangeAuthOwner
(
   TCS_CONTEXT_HANDLE hContext,    // in
   TCPA_PROTOCOL_ID   protocolID,  // in
   TCPA_ENCAUTH       newAuth,     // in
   TCPA_ENTITY_TYPE   entityType,  // in
   TPM_AUTH*          ownerAuth    // in, out
);
```

IDL Definition:

```
[helpstring("method Tcsip_ChangeAuthOwner")]
TSS_RESULT Tcsip_ChangeAuthOwner
(
   [in] TCS_CONTEXT_HANDLE       hContext,
   [AUTH, in] TCPA_PROTOCOL_ID   protocolID,
   [AUTH, in] TCPA_ENCAUTH       newAuth,
   [AUTH, in] TCPA_ENTITY_TYPE   entityType,
   [in, out] TPM_AUTH*           ownerAuth
);
```

Parameters:

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCPA_PROTOCOL_ID	protocolID	The ownership protocol in use.
TCPA_ENCAUTH	newAuth	The encrypted new authorization data for the entity. The encryption key is the shared secret from the OS-AP protocol.
TCPA_ENTITY_TYPE	entityType	The type of entity to be modified.
TPM_AUTH*	ownerAuth	The authorization and inputs for OwnerHandle. There is no validation of in parameters, just validation on the return that the proper authorization data was used. HMAC key: tpmOwnerAuth. This is the new tpmOwnerAuth value if this command changed that value.

Comment:

TPM command – TPM_ChangeAuthOwner
TPM ordinal – TPM_ORD_ChangeAuthOwner

5.8.2.2.7 Tcsip_ChangeAuthAsymStart

Start of informative comment:

Tcsip_ChangeAuthAsymStart starts the process of changing authorization for an entity. It sets up an OI-AP session that must be retained for use by its twin Tcsip_ChangeAuthAsymFinish command.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_ChangeAuthAsymStart
(
   TCS_CONTEXT_HANDLE hContext,        // in
   TCS_KEY_HANDLE     idHandle,        // in
   TCPA_NONCE         antiReplay,      // in
   UINT32             TempKeyInfoSize, // in
   BYTE*              TempKeyInfoData, // in
   TPM_AUTH*          pAuth,           // in, out
   UINT32*            TempKeySize,     // out
   BYTE**             TempKeyData,     // out
   UINT32*            CertifyInfoSize, // out
   BYTE**             CertifyInfo,     // out
   UINT32*            sigSize,         // out
   BYTE**             sig,             // out
   TCS_KEY_HANDLE*    ephHandle        // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_ChangeAuthAsymStart")]
TSS_RESULT Tcsip_ChangeAuthAsymStart
(
   [in]TCS_CONTEXT_HANDLE                  hContext,        // in
   [in]TCS_KEY_HANDLE                      idHandle,        // in
   [in]TCPA_NONCE                          antiReplay,      // in
   [in]UINT32                              TempKeyInfoSize, // in
   [in, size_is( TempKeyInfoSize )]BYTE*   TempKeyInfoData, // in
   [in]TPM_AUTH*                           pAuth,           //    in,
                                 out
   [out]UINT32*                            TempKeySize,     // out
   [out, size_is(, *TempKeySize )]BYTE**   TempKeyData,     // out
   [out]UINT32*                            CertifyInfoSize, // out
   [out, size_is(, *CertifyInfoSize )]BYTE** CertifyInfo, // out
   [out]UINT32*                            sigSize,         // out
   [out, size_is(, *sigSize )]BYTE**       sig,             // out
   [out]TCS_KEY_HANDLE*                    ephHandle        // out
);
```

Parameters:

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_KEY_HANDLE	idHandle	Handle to identify loaded identity ID key.
TCPA_NONCE	antiReplay	Nonce to be inserted in the certifyInfo structure.
UINT32*	TempKeySize	Size of TCPA_KEY byte stream of ephemeral key
BYTE**	TempKey	TCPA_KEY byte stream of ephemeral key
UINT32*	CertifyInfoSize	Size of TCPA_CERTIFY_INFO byte stream
BYTE**	CertifyInfo	TCPA_CERTIFY_INFO byte stream that was signed
UINT32*	sigSize	Used size of the output area for the signature.
BYTE**	sig	Signature of the certify info parameter.
TCS_KEY_HANDLE*	ephHandle	keyHandle identifier to be used by ChangeAuthAsymFinish for the ephemeral.
TPM_AUTH*	pAuth	The authorization and inputs from the TPM owner. There is no validation of in parameters, just validation on the return that the proper

		authorization data was used. HMAC key: parentKey.usageAuth.

Comment:

TPM command – TPM_ChangeAuthAsymStart
TPM ordinal – TPM_ORD_ChangeAuthAsymStart

5.8.2.2.8 Tcsip_ChangeAuthAsymFinish

Start of informative comment:

TPM_ChangeAuthAsymFInish completes the process of changing authorization for an entity. The owner uses tempKkey to encrypt the desired new authorization data and inserts that encrypted data in a TPM_ChangeAuthAsymFinish command, in the knowledge that only a TPM with a specific identity can interpret the new authorization data.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_ChangeAuthAsymFinish
(
   TCS_CONTEXT_HANDLE hContext,        // in
   TCS_KEY_HANDLE     parentHandle,    // in
   TCS_KEY_HANDLE     ephHandle,       // in
   TCPA_ENTITY_TYPE   entityType,      // in
   TCPA_HMAC          newAuthLink,     // in
   UINT32             newAuthSize,     // in
   BYTE*              encNewAuth,      // in
   UINT32             encDataSizeIn,   // in
   BYTE*              encDataIn,       // in
   TPM_AUTH*          ownerAuth,       // in, out
   UINT32*            encDataSizeOut,  // out
   BYTE**             encDataOut,      // out
   TCPA_NONCE*        saltNonce,       // out
   TCPA_DIGEST*       changeProof      // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_ChangeAuthAsymFinish")]
TSS_RESULT Tcsip_ChangeAuthAsymFinish
(
   [in]TCS_CONTEXT_HANDLE                        hContext,        // in
   [in]TCS_KEY_HANDLE                            parentHandle,    // in
   [in]TCS_KEY_HANDLE                            ephHandle,       // in
   [in]TCPA_ENTITY_TYPE                          entityType,      // in
   [in]TCPA_HMAC                                 newAuthLink,     // in
   [in]UINT32                                    newAuthSize,     // in
   [in, size_is( newAuthSize )]BYTE*             encNewAuth,      // in
   [in]UINT32                                    encDataSizeIn,   // in
   [in, size_is( encDataSizeIn )]BYTE*           encDataIn,       // in
   [in, out]TPM_AUTH*                            ownerAuth,    //     in,
                                         out
   [out]UINT32*                                  encDataSizeOut,  // out
   [out, size_is(, *encDataSizeOut )]BYTE**      encDataOut,      // out
   [out]TCPA_NONCE*                              saltNonce,       // out
   [out]TCPA_DIGEST*                             changeProof      // out
);
```

Parameters:

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_KEY_HANDLE	parentHandle	Key handle of the parent key for input data.
TCS_KEY_HANDLE	ephHandle	Key handle identifier for the ephemeral key.
TCPA_ENTITY_TYPE	entityType	Type of entity to be modified.
TCPA_HMAC	newAuthLink	HMAC calculation that links the new and old authorization values together.
UNINT32	newAuthSize	Size of new authorization data and ephemeral key.
BYTE*	encNewAuth	New authorization data and ephemeral key.
TCPA_NONCE*	saltNonce	Nonce value from TPM RNG to add entropy to the changeProof value.
TCPA_DIGEST*	changeProof	Proof that authorization data has changed.
UINT32*	encDataSize	Encrypted entity data size.
BYTE**	encData	Encrypted entity data – input -> modified output.
TPM_AUTH*	ownerAuth	The authorization and inputs from the TPM owner. There is no validation of in parameters, just validation on the return that the proper authorization data was used.

		HMAC key: parentKey.usageAuth.

Comment:

TPM command – TPM_ChangeAuthAsymFinish
TPM ordinal – TPM_ORD_ChangeAuthAsymFinish

5.8.2.2.9 Tcsip_TerminateHandle

Start of informative comment:

Tcsip_TerminateHandle allows the TPM driver to clear out information in an authorization handle.

The TPM may maintain the authorization session even though a key attached to it has been unloaded or the authorization session itself has been unloaded in some way.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_TerminateHandle
(
   TCS_CONTEXT_HANDLE hContext,  // in
   TCS_AUTHHANDLE        handle  // in
);
```

IDL Definition:

```
[helpstring("method Tcsip_TerminateHandle")]
TSS_RESULT Tcsip_TerminateHandle
(
   [in] TCS_CONTEXT_HANDLE  hContext,
   [in] TCS_AUTHHANDLE   handle
);
```

Parameters:

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_AUTHHANDLE	handle	The handle to terminate.

Comment:

TPM command – TPM_Terminate_Handle
TPM ordinal – TPM_ORD_Terminate_Handle

5.8.2.2.10 Tcsip_ActivateTPMIdentity

Start of informative comment:

Tcsip_ActivateTPMIdentity purpose is twofold. The first purpose is to obtain assurance that the credential in the TCPA_SYM_CA_ATTESTATION is for this TPM. The second purpose is to obtain the session key used to encrypt the TCPA_IDENTITY_CREDENTIAL.

This function checks that the symmetric session key corresponds to a TPM-identity before releasing that session key.

Only the Owner of the TPM has the privilege of activating a TPM identity. The owner may authorize this function using either the TPM_OIAP or TPM_OSAP authorization protocols.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_ActivateTPMIdentity
(
   TCS_CONTEXT_HANDLE hContext,           // in
   TCS_KEY_HANDLE     idKey,              // in
   UINT32             blobSize,           // in
   BYTE*              blob,               // in
   TPM_AUTH*          idKeyAuth,          // in, out
   TPM_AUTH*          ownerAuth,          // in, out
   UINT32*            SymmetricKeySize,   // out
   BYTE**             SymmetricKey        // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_ActivateTPMIdentity")]
TSS_RESULT Tcsip_ActivateTPMIdentity
(
   [in] TCS_CONTEXT_HANDLE                hContext,
   [in] TCS_KEY_HANDLE                    idKey,
   [AUTH, in]                             blobSize,
   [AUTH, in, size_is(blobSize)] BYTE*    blob,
   [AUTH, in, out] TPM_AUTH*              idKeyAuth,
   [AUTH, in, out] TPM_AUTH*              ownerAuth,
   [AUTH, out] UINT32*                    SymmetricKeySize,
   [AUTH, out, size_is(, *SymmetricKeySize)] BYTE** SymmetricKey
);
```

Parameters:

Type	Name	Description
TCS_CONTEXT_HANDLE	HContext	Handle of established context.
TCS_KEY_HANDLE	IdKey	Identity key to be activated.
UINT32	BlobSize	Size of encrypted blob from CA.
BYTE*	Blob	Encrypted ASYM_CA_CONTENTS structure.

UINT32	SymmetricKeySize	Size of decrypted TCPA_SYMMETRIC_KEY byte stream
BYTE**	SymmetricKey	decrypted TCPA_SYMMETRIC_KEY byte stream
TPM_AUTH*	IdKeyAuth	The authorization and inputs from the TPM owner. There is no validation of in parameters, just validation on the return that the proper authorization data was used. HMAC key: idKey.usageAuth.
TPM_AUTH*	OwnerAuth	The authorization and inputs from the TPM owner. There is no validation of in parameters, just validation on the return that the proper authorization data was used. HMAC key: ownerAuth.

Comment:

TPM command – TPM_ActivateTPMIdentity
TPM ordinal – TPM_ORD_ActivateTPMIdentity

5.8.2.3 TCS Context

Start of informative comment:

The TCS Context provides dynamic handles that allow for efficient usage of the service provider's and TCS's resources and information.

End of informative comment.

5.8.2.4 Transport Protection

5.8.2.4.1 Tcsip_EstablishTransport

Start of informative comment:

Tcsip_EstablishTransport launches a transport session. Depending on the attributes specified for the session this may establish shared secrets, encryption keys, and session logs. The session will be in use for by the Tcsip_ExecuteTransport command.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_EstablishTransport
   (
     TCS_CONTEXT_HANDLE hContext,                // in
     UINT32             ulTransControlFlags,     // in
     TCS_KEY_HANDLE     hEncKey,                 // in
     UINT32             ulTransSessionInfoSize,  // in
     BYTE*              rgbTransSessionInfo,     // in
     UINT32             ulSecretSize,            // in
     BYTE*              rgbSecret,               // in
     TPM_AUTH*          pEncKeyAuth,             // in, out
     TCPA_LOCALITY_MOD* pbLocality,              // out
     TCS_HANDLE*        hTransSession,           // out
     UINT32*            ulCurrentTicks,          // out
     BYTE**             prgbCurrentTicks,        // out
     TCPA_NONCE*        pTransNonce              // out
   );
```

IDL Definition:

```
[helpstring("method Tcsip_EstablishTransport")]
   TSS_RESULT Tcsip_EstablishTransport(

   [in] TCS_CONTEXT_HANDLE                  hContext,
   [in] UINT32                              ulTransControlFlags,
   [in] TCS_KEY_HANDLE                      hEncKey,

   [in] UINT32                              ulTransSessionInfoSize,

   [in, size_is(ulTransSessionInfoSize)] BYTE*    rgbTransSessionInfo,
```

```
[in] UINT32                             ulSecretSize,
[in, size_is(ulSecretSize)] BYTE*       rgbSecret,
[in, out, ptr] TPM_AUTH*                pEncKeyAuth,
[out] TCPA_LOCALITY_MOD*                pbLocality,
[out] TCS_HANDLE*                       hTransSession,
[out] UINT32*                           ulCurrentTicks,
[out, size_is(, *ulCurrentTicks)] BYTE** prgbCurrentTicks,
[out] TCPA_NONCE*                        pTransNonce);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
UINT32	ulTransControlFlags	Flags to control the transport session handling in the TCS (e.g. Exclusive-Mode)
TCS_KEY_HANDLE	hEncKey	Handle to the key that encrypts the secret blob.
UINT32	ulTransSessionInfoSize	Size of the public info area.
BYTE*	rgbTransSessionInfo	The public info describing the transport session.(i.e. TPM_TRANSPORT_PUBLIC)
UINT32	ulSecretSize	The size of the secret area.
BYTE*	rgbSecret	The encrypted session secret data blob.
TPM_AUTH*	pEncKeyAuth	Authorization session data including the HMAC digest for authorizing the key. If NULL, no authorization is required.
TCPA_LOCALITY_MOD*	pbLocality	The locality that called this command.
TCS_HANDLE*	hTransSession	The handle for the transport session.
UINT32*	ulCurrentTicks	Size of the current tick count data blob.
BYTE**	prgbCurrentTicks	The current tick count data.
TCPA_NONCE*	pTransNonce	Even nonce for subsequent execute transport calls.

Comment:

TPM command – TPM_EstablishTransport
TPM ordinal – TPM_EstablishTransport

The parameter "ulTransControlFlags" is used to control the behavior of the established transport session:

```
// TCS flag for transport protection
#define TSS_TCSATTRIB_TRANSPORT_DEFAULT           0x00000000
#define TSS_TCSATTRIB_TRANSPORT_EXCLUSIVE         0x00000001
```

5.8.2.4.2 Tcsip_ExecuteTransport

Start of informative comment:

Tcsip_ExecuteTransport delivers a (e.g. by TSP) wrapped TPM command to the TPM device.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_ExecuteTransport
   (
     TCS_CONTEXT_HANDLE hContext,                // in
     TPM_COMMAND_CODE   unWrappedCommandOrdinal, // in
     UINT32             ulWrappedCmdDataInSize,  // in
     BYTE*              rgbWrappedCmdDataIn,     // in
     UINT32*            pulHandleListSize,       // in, out
     TCS_HANDLE**       rghHandles,              // in, out
     TPM_AUTH*          pWrappedCmdAuth1,        // in, out
     TPM_AUTH*          pWrappedCmdAuth2,        // in, out
     TPM_AUTH*          pTransAuth,              // in, out
     UINT64*            punCurrentTicks,         // out
     TCPA_LOCALITY_MOD* pbLocality,              // out
     TPM_RESULT*        pulWrappedCmdReturnCode, // out
     UINT32*            ulWrappedCmdDataOutSize,// out
     BYTE**             rgbWrappedCmdDataOut     // out
   );
```

IDL Definition:

```
[helpstring("method Tcsip_ExecuteTransport")]
   TSS_RESULT Tcsip_ExecuteTransport(

   [in] TCS_CONTEXT_HANDLE                        hContext,
   [in] TPM_COMMAND_CODE                          unWrappedCommandOrdinal,

   [in] UINT32                                    ulWrappedCmdDataInSize,

   [in, size_is(ulWrappedCmdDataInSize)] BYTE*    rgbWrappedCmdDataIn,

   [in, out] UINT32*                              pulHandleListSize,

   [in, out, ptr, size_is(, *pulHandleListSize)] TCS_HANDLE** rghHandles,

   [in, out, ptr] TPM_AUTH*                       pWrappedCmdAuth1,

   [in, out, ptr] TPM_AUTH*                       pWrappedCmdAuth2,

   [in, out, ptr] TPM_AUTH*                       pTransAuth,

   [out] UINT64*                                  punCurrentTicks,

   [out] TCPA_LOCALITY_MOD*                       pbLocality,

   [out] TPM_RESULT*                              pulWrappedCmdReturnCode,

   [out] UINT32*                                  pulWrappedCmdDataOutSize,

   [out, size_is(, *pulWrappedCmdDataOutSize)] BYTE**   prgbWrappedCmdDataOut);
```

Parameters:

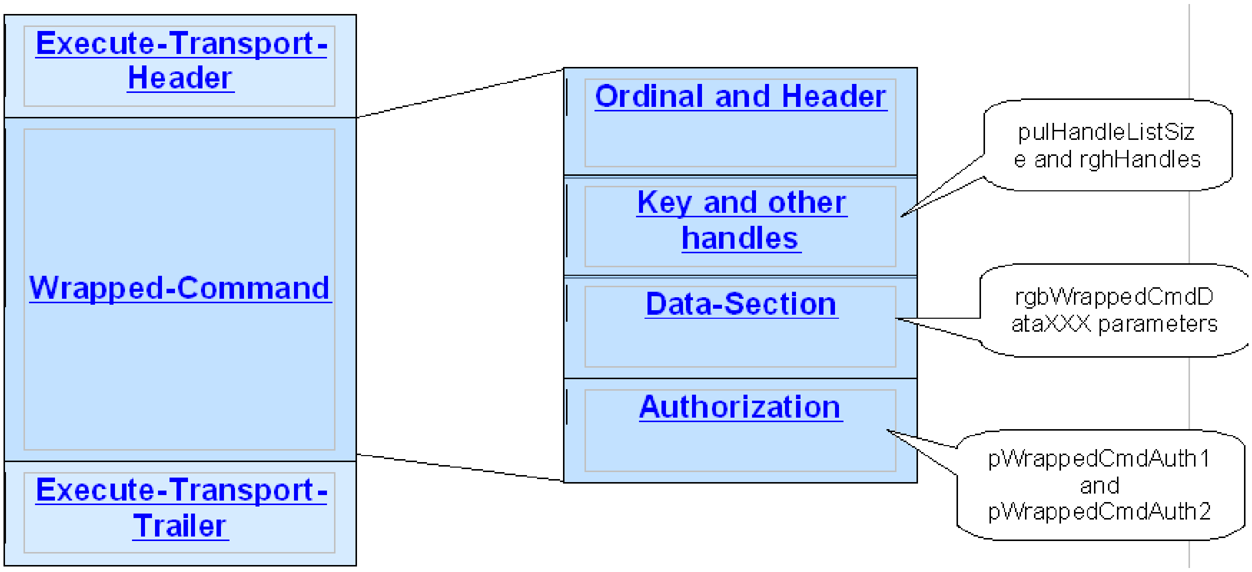
Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TPM_COMMAND_CODE	unWrappedCommandOrdinal	Hint for the TCS about the wrapped TPM operation.
UINT32	ulWrappedCmdDataInSize	Size of the data section of the wrapped command.
BYTE*	rgbWrappedCmdDataIn	The wrapped command data section blob.
UINT32*	pulHandleListSize	Size (number of handles) of the following handle list.
TCS_HANDLE**	rghHandles	List of TSS handles (e.g. Keys) used in the wrapped command
TPM_AUTH*	pWrappedCmdAuth1	First authorization session data for the wrapped command including the HMAC digest for authorizing e.g. key. If NULL, no authorization is required.
TPM_AUTH*	pWrappedCmdAuth2	Second authorization session data for the wrapped command including the HMAC digest for authorizing e.g. key. If NULL, no authorization is required.
TPM_AUTH*	pTransAuth	Authorization session data including the HMAC digest for authorizing the transport session.
UINT64*	punCurrentTicks	The current ticks when the command was executed.
TCPA_LOCALITY_MOD*	pbLocality	The locality that called this command.
TPM_RESULT*	pulWrappedCmdReturnCode	The return code of wrapped TPM operation.
UINT32*	ulWrappedCmdDataOutSize	Size of the data section for wrapped command return.
BYTE**	rgbWrappedCmdDataOut	The wrapped command return data seciton blob (output).

Comment:

TPM command – TPM_ExecuteTransport
TPM ordinal – TPM_ExecuteTransport

Remarks

Execute transport operation format representation overview:



5.8.2.4.3 Tcsip_ReleaseTransportSigned

Start of informative comment:

Tcsip_ReleaseTransportSigned completes the transport session. This command uses two authorizations, the key that will sign the log and the authorization from the session.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_ReleaseTransportSigned
   (
      TCS_CONTEXT_HANDLE hContext,           // in
      TCS_KEY_HANDLE     hSignatureKey,      // in
      TCPA_NONCE         AntiReplayNonce,    // in
      TPM_AUTH*          pKeyAuth,           // in, out
      TPM_AUTH*          pTransAuth,         // in, out
      TCPA_LOCALITY_MOD* pbLocality,         // out
      UINT32*            pulCurrentTicks,    // out
      BYTE**             prgbCurrentTicks,   // out
      UINT32*            pulSignatureSize,   // out
      BYTE**             prgbSignature       // out
   );
```

IDL Definition:

```
[helpstring("method Tcsip_ ReleaseTransportSigned")]
   TSS_RESULT Tcsip_ReleaseTransportSigned(
   [in] TCS_CONTEXT_HANDLE                          hContext,
   [in] TCS_KEY_HANDLE                              hSignatureKey,
   [in] TCPA_NONCE                                  AntiReplayNonce,
   [in, out, ptr] TPM_AUTH*                         pKeyAuth,
   [in, out, ptr] TPM_AUTH*                         pTransAuth,
   [out] TCPA_LOCALITY_MOD*                         pbLocality,
   [out] UINT32*                                    pulCurrentTicks,
   [out, size_is(, *pulCurrentTicks)] BYTE**        prgbCurrentTicks,
   [out] UINT32*                                    pulSignatureSize,
   [out, size_is(, *pulSignatureSize)] BYTE**       prgbSignature);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_KEY_HANDLE	hSignatureKey	Key that will perform the signing.
TCPA_NONCE	AntiReplayNonce	Value for anti-replay protection.
TPM_AUTH*	pKeyAuth	Authorization session data for the command including the HMAC digest

		for authorizing the key.
TPM_AUTH*	pTransAuth	Authorization session data for the ommand including the HMAC digest for authorizing transport session.
TCPA_LOCALITY_MOD*	bLocality	The locality that called this command.
UINT32*	pulCurrentTicks	Size of the current tick count data blob.
BYTE**	prgbCurrentTicks	The current tick count data.
UINT32*	pulSignatureSize	The size of the signature area.
BYTE**	prgbSignature	The signature of the digest.

Comment:

TPM command – TPM_ReleaseTransportSigned
TPM ordinal – TPM_ReleaseTransportSigned

5.8.2.5 TPM Mandatory

5.8.2.5.1 Tcsip_Extend

Start of informative comment:

Tcsip_Extend causes the modification of a specific PCR register.

End of informative comment.

C-Definition:

TSS_RESULT Tcsip_Extend

```
(
   TCS_CONTEXT_HANDLE hContext,  // in
   TCPA_PCRINDEX      pcrNum,    // in
   TCPA_DIGEST        inDigest,  // in
   TCPA_PCRVALUE*     outDigest  // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_Extend")]
TSS_RESULT Tcsip_Extend
(
   [in] TCS_CONTEXT_HANDLE  hContext,
   [in] TCPA_PCRINDEX       pcrNum,
   [in] TCPA_DIGEST         inDigest,
   [out] TCPA_PCRVALUE*     outDigest
);
```

Parameters:

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCPA_PCRINDEX	pcrNum	Index of the PCR to be modified.
TCPA_DIGEST	inDigest	160-bit value representing the event to be recorded.
TCPA_PCRVALUE*	outDigest	Pointer to a DIGEST-sized memory location that is updated by the TPM_Extend operation to be the contents of the named PCR when internal processing is complete. If this parameter is NULL, no value is returned. If the TPM is disabled, NULL is returned.

Comment:

TPM command – TPM_Extend
TPM ordinal – TPM_Extend

5.8.2.5.2 Tcsip_PcrRead

Start of informative comment:

Tcsip_PcrRead provides non-cryptographic reporting of the contents of a named PCR.

End of informative comment

C-Definition:

```
TSS_RESULT Tcsip_PcrRead
(
   TCS_CONTEXT_HANDLE hContext,  // in
   TCPA_PCRINDEX      pcrNum,    // in
   TCPA_PCRVALUE*     outDigest  // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_PcrRead")]
TSS_RESULT Tcsip_PcrRead
(
   [in] TCS_CONTEXT_HANDLE  hContext,
   [in] TCPA_PCRINDEX    pcrNum,
   [out] TCPA_PCRVALUE*     outDigest
);
```

Parameters:

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCPA_PCRINDEX	pcrNum	Index of the PCR to be read.
TCPA_PCRVALUE *	outDigest	Pointer to the current contents of the named PCR.

Comment:

TPM command – TPM_PcrRead

TPM ordinal – TPM_PcrRead

5.8.2.5.3 Tcsip_Quote

Start of informative comment:

Tcsip_Quote provides cryptographic reporting of PCR values. A loaded key is required for operation. Tcsip_Quote uses a key to sign a statement that names the current value of a chosen PCR and externally supplied data (which may be a nonce supplied by a Challenger).

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_Quote
(
   TCS_CONTEXT_HANDLE hContext,        // in
   TCS_KEY_HANDLE     keyHandle,       // in
   TCPA_NONCE         antiReplay,      // in
   UINT32             pcrTargetSize,   // in
   BYTE*              pcrTarget,       // in
   TPM_AUTH*          privAuth,        // in, out
   UINT32*            pcrDataSize,     // out
   BYTE**             pcrData,         // out
   UINT32*            sigSize,         // out
   BYTE**             sig              // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_Quote")]
TSS_RESULT Tcsip_Quote
(
   [in]TCS_CONTEXT_HANDLE                  hContext,        // in
   [in]TCS_KEY_HANDLE                      keyHandle,       // in
   [in]TCPA_NONCE                          antiReplay,      // in
   [in]UINT32                              pcrTargetSize,   // in
   [in, size_is( pcrTargetSize )]BYTE*     pcrTarget,       // in
   [in, out]TPM_AUTH*                      privAuth,        //    in,
                                     out
   [out]UINT32*                            pcrDataSize,     // out
   [out, size_is(, *pcrDataSize )]BYTE**   pcrData,         // out
   [out]UINT32*                            sigSize,         // out
   [out, size_is(, *sigSize )]BYTE**       sig              // out
);
```

Parameters:

Return Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_KEY_HANDLE	keyHandle	Handle associated with Key used to provide the Quote.
TCPA_NONCE	antiReplay	Nonce provided to fight

		replay attacks.
UINT32*	PcrDataSize	Size of TCPA_PCR_COMPOSITE byte stream getting signed
BYTE**	PcrData	TCPA_PCR_COMPOSITE byte stream getting signed
UINT32*	pcrTargetSize	Size of pcrTarget
BYTE*	pcrTarget	PCRs to be quoted.
TPM_AUTH*	privAuth	Authorization digest for keyHandle and input/returned parameters. HMAC key: Key -> usageAuth.
UINT32	sigSize	The used size of the output area for the signature.
BYTE**	sig	The signature

Request:

SHA (TPM_ORD_Quote, antiReplay, TargetPCR)

HMAC (SHA, authLastNonceEven, nonceOdd, continueAuthSession)

Response:

SHA (returnCode, TPM_ORD_Quote, pcrData, sigSize, sig)

HMAC (SHA, nonceEven, nonceOdd, continueAuthSession)

Comment:

TPM command – TPM_Quote

TPM ordinal – TPM_Quote

5.8.2.5.4 Tcsip_Quote2

Start of informative comment:

Tcsip_Quote2 provides cryptographic reporting of PCR values. A loaded key is required for operation. Tcsip_Quote2 uses a key to sign a statement that names the current value of a chosen PCR and externally supplied data (which may be a nonce supplied by a Challenger).

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_Quote2
(
    TCS_CONTEXT_HANDLE hContext,        // in
    TCS_KEY_HANDLE     keyHandle,       // in
    TCPA_NONCE         antiReplay,      // in
    UINT32             pcrTargetSize,   // in
    BYTE*              pcrTarget,       // in
    TSS_BOOL           addVersion,      // in
    TPM_AUTH*          privAuth,        // in, out
    UINT32*            pcrDataSize,     // out
    BYTE**             pcrData,         // out
    UINT32*            versionInfoSize,// out
    BYTE**             versionInfo,     // out
    UINT32*            sigSize,         // out
    BYTE**             sig              // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_Quote2")]
TSS_RESULT Tcsip_Quote2
(
   [in]TCS_CONTEXT_HANDLE                          hContext,        //
                                  in
   [in]TCS_KEY_HANDLE                              keyHandle,       //
                                  in
   [in]TCPA_NONCE                                  antiReplay,      //
                                  in
   [in]UINT32                                      pcrTargetSize,   //
                                  in
   [in, size_is( pcrTargetSize )]BYTE*             pcrTarget,       //
                                  in
   [in]TSS_BOOL                                    addVersion,      //
                                  in
   [in, out]TPM_AUTH*                              privAuth,        //
                                  in, out
   [out]UINT32*                                    pcrDataSize,     //
                                  out
   [out, size_is(, *pcrDataSize )]BYTE**           pcrData,         //
                                  out
   {out}UINT32*                                    versionInfoSize, //
                                  out
   [out, size is (, *versionInforSize)]BYTE**      versionInfo,     //
                                  out
   [out]UINT32*                                    sigSize,         //
                                  out
   [out, size_is(, *sigSize )]BYTE**               sig              //
                                  out
);
```

Parameters:

Return Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_KEY_HANDLE	keyHandle	Handle associated with Key used to provide the Quote.
TCPA_NONCE	antiReplay	Nonce provided to fight replay attacks.
UINT32	pcrTargetSize	Size of pcrTarget.
BYTE*	pcrTarget	PCRs to be quoted.
TSS_BOOL	addVersion	Add TPM version to output.
TPM_AUTH*	privAuth	Authorization digest for keyHandle and input/returned parameters. HMAC key: Key -> usageAuth.
UINT32*	PcrDataSize	Size of TSS_PCR_INFO_SHORT

		byte stream getting signed
BYTE**	PcrData	TSS_PCR_INFO_SHORT byte stream getting signed
UINT328	versionInfoSize	The used size of the output area for the versionInfo. It is zero if the addVersion flag is set FALSE.
BYTE**	versionInfo	The byte stream which returns the value of TSS_CAP_VERSION_INFO if the addVersion flag is set TRUE (otherwise NULL)
UINT32*	sigSize	The used size of the output area for the signature.
BYTE**	Sig	The signature

Request:

SHA (TPM_ORD_Quote2, externalData, targetPCR, addVersion)

HMAC (SHA, authLastNonceEven, nonceOdd, continueAuthSession)

Response:

SHA (returnCode, TPM_ORD_Quote2, pcrData, versionInfoSize, versionInfo, sigSize, sig)

HMAC (SHA, nonceEven, nonceOdd, continueAuthSession)

Comment:

TPM command – TPM_Quote2

TPM ordinal – TPM_Quote2

5.8.2.5.5 Tcsip_DirWriteAuth

Start of informative comment:

Tcsip_DirWriteAuth provides write access to the Data Integrity Registers.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_DirWriteAuth
(
   TCS_CONTEXT_HANDLE hContext,    // in
   TCPA_DIRINDEX      dirIndex,    // in
   TCPA_DIRVALUE      newContents, // in
   TPM_AUTH*          ownerAuth    // in, out
);
```

IDL Definition:

```
[helpstring("method Tcsip_DirWriteAuth")]
TSS_RESULT Tcsip_DirWriteAuth
(
   [in] TCS_CONTEXT_HANDLE    hContext,
   [AUTH, in] TCPA_DIRINDEX   dirIndex,
   [AUTH, in] TCPA_DIRVALUE   newContents,
   [AUTH, in, out] TPM_AUTH*  ownerAuth
);
```

Parameters:

Return Type	Label	Description
TCS_CONTEXT_HANDLE	HContext	Handle of established context.
TCPA_DIRINDEX	DirIndex	Index of the DIR.
TCPA_DIRVALUE	newContents	New value to be stored in the named DIR.
TPM_AUTH*	OwnerAuth	Authorization digest for the inputs and returned parameters. HMAC key: Key -> ownerAuth.

Comment:

TPM command – TPM_DirWriteAuth

TPM ordinal – TPM_DirWriteAuth

5.8.2.6 Tcsip_DirRead

Start of informative comment:

Tcsip_DirRead provides read access to the DIRs.

End of informative comment

C-Definition:

```
TSS_RESULT Tcsip_DirRead
(
   TCS_CONTEXT_HANDLE hContext,    // in
   TCPA_DIRINDEX      dirIndex,    // in
   TCPA_DIRVALUE*     dirValue     // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_DirRead")]
TSS_RESULT Tcsip_DirRead
(
   [in] TCS_CONTEXT_HANDLE  hContext,
   [in] TCPA_DIRINDEX       dirIndex,
   [out] TCPA_DIRVALUE*     dirValue
);
```

Parameters:

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCPA_DIRINDEX	dirIndex	Index of the DIR to be read.
TCPA_DIRVALUE *	dirValue	Pointer to the current contents of the named DIR.

Comment:

TPM command – TPM_DirRead
TPM ordinal – TPM_DirRead

5.8.2.6.1 Tcsip_Seal

Start of informative comment:

Tcsip_Seal allows software to explicitly state the future “trusted” configuration that the platform must be in for the secret to be revealed. The SEAL operation also implicitly includes the relevant platform configuration (PCR-values) when the SEAL operation was performed. The SEAL operation uses the tpmProof value to BIND the blob to an individual.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_Seal
(
   TCS_CONTEXT_HANDLE hContext,        // in
   TCS_KEY_HANDLE     keyHandle,       // in
   TCPA_ENCAUTH       encAuth,         // in
   UINT32             pcrInfoSize,     // in
   BYTE*              PcrInfo,         // in
   UINT32             inDataSize,      // in
   BYTE*              inData,          // in
   TPM_AUTH*          pubAuth,         // in, out
   UINT32*            SealedDataSize,  // out
   BYTE**             SealedData       // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_Seal")]
TSS_RESULT Tcsip_Seal
(
   [in] TCS_CONTEXT_HANDLE                            hContext,
   [in] TCS_KEY_HANDLE                                keyHandle,
   [AUTH, in] TCPA_ENCAUTH                            encAuth,
   [AUTH, in] UINT32                                  pcrInfoSize,
   [AUTH, in, size_is(pcrInfoSize)] BYTE*             PcrInfo,
   [AUTH, in] UINT32                                  inDataSize,
   [AUTH, in, size_is(inDataSize)] BYTE*              inData,
   [AUTH, in, out] TPM_AUTH*                          pubAuth,
   [AUTH, out]   UINT32*                              SealedDataSize,
   [AUTH, out, size_is(, *SealedDataSize)] BYTE**     SealedData
);
```

Parameters:

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_KEY_HANDLE	keyHandle	Application key handle of the loaded key.
TCPA_ENCAUTH	encAuth	The encrypted authorization data for the sealed data. The

		encryption key is the shared secret from the OS-AP protocol.
UINT32	pcrInfoSize	The size of the pcrInfo parameter. If 0 there are no PCR registers in use
BYTE*	PcrInfo	The PCR selection information
UINT32	inDataSize	The size of the inData parameter
BYTE*	inData	The data to be sealed to the platform and any specified PCRs
TPM_AUTH*	pubAuth	Authorization digest for the inputs and returned parameters. HMAC key: entity.usageAuth
UINT32*	SealedDataSize	Size of sealed data
BYTE**	SealedData	Encrypted, integrity-protected data object that is the result of the TPM_Seal operation.

Comment:

TPM command – TPM_Seal

TPM ordinal – TPM_Seal

5.8.2.6.2 Tcsip_Unseal

Start of informative comment:

Tcsip_Unseal will reveal TPM_Sealed data only if it was encrypted on this platform and the current configuration (as defined by the named PCR contents) is the one named as qualified to decrypt it. Internally, Tcsip_Unseal accepts a data blob generated by a Tcsip_Seal operation. Tcsip_Unseal decrypts the structure internally, checks the integrity of the resulting data, *and* checks that the PCR named has the value named during Tcsip_Seal. Additionally, the caller must supply appropriate authorization data for blob and for the key that was used to seal that data.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_Unseal
(
   TCS_CONTEXT_HANDLE hContext,        // in
   TCS_KEY_HANDLE     keyHandle,       // in
   UINT32             SealedDataSize,  // in
   BYTE*              SealedData,      // in
   TPM_AUTH*          keyAuth,         // in
   TPM_AUTH*          dataAuth,        // in, out
   UINT32*            DataSize,        // out
   BYTE**             Data             // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_Unseal")]
TSS_RESULT Tcsip_Unseal
(
   [in] TCS_CONTEXT_HANDLE                       hContext,
   [in] TCS_KEY_HANDLE                           parentHandle,
   [AUTH, in] UINT32                             SealedDataSize,
   [AUTH, in, size_is(SealedDataSize)] BYTE*     SealedData,
   [AUTH, in, out] TPM_AUTH*                     parentAuth,
   [AUTH, in, out] TPM_AUTH*                     dataAuth,
   [AUTH, out] UINT32*                           DataSize,
   [AUTH, out, size_is(, *DataSize)] BYTE**      Data
);
```

Parameters:

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_KEY_HANDLE	keyHandle	Handle of the key that can decrypt the encData.
UINT32	SealedDataSize	Size of sealed data
BYTE*	SealedData	Encrypted, integrity-protected data object that is the result of the TPM_Seal operation.

TPM_AUTH*	keyAuth	Authorization digest for the key usage and input/ returned parameters. HMAC key: Key. Usaage Auth.
TPM_AUTH*	dataAuth	The encrypted authorization data for the sealed data. The decryption key is the shared secret from the OS-AP protocol.
UINT32*	DataSize	The size of the Data parameter
BYTE**	Data	The data that was unsealed.

Comment:

TPM command – TPM_Unseal

TPM ordinal – TPM_Unseal

5.8.2.6.3 Tcsip_UnBind

Start of informative comment:

Tcsip_UnBind takes the data blob that is the result of a Bind command and decrypts it for export to the User. The caller must authorize the use of the key that will decrypt the incoming blob.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_UnBind
(
   TCS_CONTEXT_HANDLE hContext,    // in
   TCS_KEY_HANDLE     keyHandle,   // in
   UINT32             inDataSize,  // in
   BYTE*              inData,      // in
   TPM_AUTH*          privAuth,    // in, out
   UINT32*            outDataSize, // out
   BYTE**             outData      // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_UnBind")]
TSS_RESULT Tcsip_UnBind
(
   [in] TCS_CONTEXT_HANDLE                         hContext,
   [in] TCS_KEY_HANDLE                             keyHandle,
   [AUTH, in] UINT32                               inDataSize,
   [AUTH, in, size_is(inDataSize)] BYTE*           inData,
   [AUTH, in, out] TPM_AUTH*                       privAuth,
   [AUTH, out] UINT32*                             outDataSize,
   [AUTH, out, size_is(, *outDataSize)] BYTE** outData
);
```

Parameters:

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_KEY_HANDLE	keyHandle	Handle of the key that can decrypt the inData.
UINT32	inDataSize	Size of encrypted data
BYTE*	inData	Encrypted data object that is the result of the Tcsi_Bind operation.
TPM_AUTH*	privAuth	The authorization digest that authorizes the inputs and use of keyHandle. HMAC key: key.usageAuth.
UINT32*	outDataSize	The length of the returned decrypted data

BYTE**	outData	The resulting decrypted data.

Comment:

TPM command – TPM_UnBind

TPM ordinal – TPM_UnBind

5.8.2.6.4 Tcsip_Sealx

Start of informative comment:

Tcsip_Seal and Tcsip_Sealx allow software to explicitly state the future "trusted" configuration that the platform must be in for the secret to be revealed. The SEAL/SEALX operation also implicitly includes the relevant platform configuration (PCR-values) when the SEAL/SEALX operation was performed. The SEAL/SEALX operation uses the tpmProof value to BIND the blob to an individual.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_Sealx
(
TCS_CONTEXT_HANDLE hContext,          // in
TCS_KEY_HANDLE     keyHandle,         // in
TCPA_ENCAUTH       encAuth,           // in
UINT32             pcrInfoSize,       // in
BYTE*              PcrInfo,           // in
UINT32             inDataSize,        // in
BYTE*              inData,            // in
TPM_AUTH*          pubAuth,           // in, out
UINT32*            SealedDataSize,    // out
BYTE**             SealedData         // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_Sealx")]
TSS_RESULT Tcsip_Sealx
(
[in] TCS_CONTEXT_HANDLE                  hContext,
[in] TCS_KEY_HANDLE                      keyHandle,
[in] TCPA_ENCAUTH                        encAuth,
[in] UINT32                              pcrInfoSize,
[in, size_is(pcrInfoSize)] BYTE*         PcrInfo,
[in] UINT32                              inDataSize,
[in, size_is(inDataSize)] BYTE*          inData,
[in, out] TPM_AUTH*                      pubAuth,
[out] UINT32*                            SealedDataSize,
[out, size_is(, *SealedDataSize)] BYTE** SealedData
);
```

Parameters:

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_KEY_HANDLE	keyHandle	Application key handle of the loaded key.
TCPA_ENCAUTH	encAuth	The encrypted authorization data for the sealed data. The

		encryption key is the shared secret from the OS-AP protocol.
UINT32	pcrInfoSize	The size of the pcrInfo parameter. If 0 there are no PCR registers in use
BYTE*	PcrInfo	The PCR selection information
UINT32	inDataSize	The size of the inData parameter
BYTE*	inData	The data to be sealed to the platform and any specified PCRs
TPM_AUTH*	pubAuth	Authorization digest for the inputs and returned parameters. HMAC key: entity.usageAuth
UINT32*	SealedDataSize	Size of sealed data
BYTE**	SealedData	Encrypted, integrity-protected data object that is the result of the TPM_Sealx operation.

Comment:

TPM command – TPM_Sealx

TPM ordinal – TPM_Sealx

5.8.2.6.5 Tcsip_LoadKey2ByBlob

Start of informative comment:

A key can be loaded by providing a key blob as defined in the TCPA_KEYxx complex structure. The key defined by the key blob gets unwrapped by the already loaded parent key associated with the given application parent key handle. After the key is loaded an appropriate application key handle is returned by which the key can be addressed for further use. Depending on the parent key this can be done with or without required authorization.

This is a low level mechanism and the calling application must manage the required key blobs on its own but give the caller as much flexibility as possible.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_LoadKey2ByBlob
(
   TCS_CONTEXT_HANDLE hContext,              // in
   TCS_KEY_HANDLE     hUnwrappingKey,        // in
   UINT32             cWrappedKeyBlobSize,   // in
   BYTE*              rgbWrappedKeyBlob,     // in
   TPM_AUTH*          pAuth,                 // in, out
   TCS_KEY_HANDLE*    phKeyTCSI              // out
);
```

IDL-Definition:

```
[helpstring("method Tcsip_LoadKey2ByBlob")]
TSS_RESULT Tcsip_LoadKey2ByBlob
(
   [in] TCS_CONTEXT_HANDLE                   hContext,
   [in] TCS_KEY_HANDLE                       hUnwrappingKey,
   [in] UINT32                               cWrappedKeyBlobSize,
   [in, size_is(cWrappedKeyBlobSize)] BYTE*    rgbWrappedKeyBlob,
   [in, out] TPM_AUTH*                       pAuth,
   [out] TCS_KEY_HANDLE*                     phKeyTCSI
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	Hcontext	Handle to established context.
TCS_KEY_HANDLE	hUnwrappingKey	Application key handle of the already loaded parent key.
UINT32	cWrappedKeyBlobSize	Size of the provided keyblob in bytes.

BYTE*	rgbWrappedKeyBlob	Key blob of the key to be loaded.
TPM_AUTH*	pAuth	Authorization session data including the HMAC digest for using the unwrapping key.If NULL, no authorization is required.
TCS_KEY_HANDLE*	phKeyTCSI	Return application key handle the loaded key can be addressed on further use.

Comment:

Tcsip_LoadKey2ByBlob initially loads the key utilizing the Key Cache Manager Services (KCM) and returns a newly created application key handle by which the key can be addressed on further use. The returned application key handle must be bound to the context provided by hContext.

After this command the key is managed by the Key Cache Management Services.

If pAuth == NULL, no authorization is required.

Loading a key MUST utilize the Key Cache Manager Service.

Return Value:

TCS_SUCCESS
TCS_E_FAIL

TPM Ordinal - TPM_LoadKey2

5.8.2.6.6 Tcsip_CreateMigrationBlob

Start of informative comment:

Tcsip_CreateMigrationBlob implements the first step in the process of moving a migratable key to a new parent key or platform. Execution of this command requires knowledge of the migrationAuth field of the key to be migrated.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_CreateMigrationBlob
(
   TCS_CONTEXT_HANDLE    hContext,              // in
   TCS_KEY_HANDLE        parentHandle,          // in
   TSS_MIGRATE_SCHEME    migrationType,         // in
   UINT32                MigrationKeyAuthSize,  // in
   BYTE*                 MigrationKeyAuth,      // in
   UINT32                encDataSize,           // in
   BYTE*                 encData,               // in
   TPM_AUTH*             parentAuth,            // in, out
   TPM_AUTH*             entityAuth,            // in, out
   UINT32*               randomSize,            // out
   BYTE**                random,                // out
   UINT32*               outDataSize,           // out
   BYTE**                outData                // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_CreateMigrationBlob")]
TSS_RESULT Tcsip_CreateMigrationBlob
(
   [in] TCS_CONTEXT_HANDLE                              hContext,
   [in] TCS_KEY_HANDLE                                  parentHandle,
   [AUTH, in] TSS_MIGRATE_SCHEME                        migrationType,
   [AUTH, in] UINT32
                                 MigrationKeyAuthSize,
   [AUTH,      in,      size_is(MigrationKeyAuthSize)]      BYTE*
                                 MigrationKeyAuth,
   [AUTH, in] UINT32                                    encDataSize,
   [AUTH, in, size_is(encDataSize)] BYTE*               encData,
   [AUTH, in, out] TPM_AUTH*                            parentAuth,
   [AUTH, in, out] TPM_AUTH*                            entityAuth,
   [AUTH, out] UINT32*                                  randomSize,
   [AUTH, out, size_is(, *randomSize)] BYTE**           random,
   [AUTH, out] UINT32*                                  outDataSize,
   [AUTH, out, size_is(, *outDataSize)] BYTE**          outData
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_KEY_HANDLE	parentHandle	Handle of the parent key that can decrypt the encData.
TSS_MIGRATE_SCHEME	migrationType	Migration type, either MIGRATE or REWRAP.
UINT32	MigrationKeyAuthSize	Size of TCPA_MIGRATONKEYAUTH byte stream with public key and authorization digest
BYTE*	MigrationKeyAuth	TCPA_MIGRATONKEYAUTH byte stream with public key and authorization digest
UINT32	encDataSize	Size of encData.
BYTE*	encData	Encrypted entity to be modified.
UINT32*	randomDataSize	Used size of the output area for randomData.
BYTE**	randomData	String used for XOR encryption.
UNIT32*	outDataSize	Used size of the output area for outData.
BYTE**	outData	Modified encrypted entity.
TPM_AUTH*	parentAuth	Authorization digest for the owner and input/ returned parameters. HMAC key: parentKey. Usaage Auth.
TPM_AUTH*	entityAuth	Authorization digest for the owner and nput/ returned parameters. HMAC Key: entityKey.migrationAuth.

Comment:

TPM command – TPM_CreateMigrationBlob
TPM ordinal – TPM_CreateMigrationBlob

5.8.2.6.7 Tcsip_ConvertMigrationBlob

Start of informative comment:

Tcsip_ConvertMigrationBlob takes a migration blob and creates a normal wrapped blob. The migrated blob must be loaded into the TPM using the normal TPM_LoadKey function.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_ConvertMigrationBlob
(
   TCS_CONTEXT_HANDLE hContext,     // in
   TCS_KEY_HANDLE     parentHandle, // in
   UINT32             inDataSize,   // in
   BYTE*              inData,       // in
   UINT32             randomSize,   // in
   BYTE*              random,       // in
   TPM_AUTH*          parentAuth,   // in, out
   UINT32*            outDataSize,  // out
   BYTE**             outData       // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_ConvertMigrationBlob")]
TSS_RESULT Tcsip_ConvertMigrationBlob
(
   [in] TCS_CONTEXT_HANDLE                       hContext,
   [in] TCS_KEY_HANDLE                           parentHandle,
   [AUTH, in] UINT32                             inDataSize,
   [AUTH, in, size_is(inDataSize)] BYTE*         inData,
   [AUTH, in] UINT32                             randomSize,
   [AUTH, in, size_is(randomSize)] BYTE*         random,
   [AUTH, in, out] TPM_AUTH*                     parentAuth,
   [AUTH, out] UINT32*                           outDataSize,
   [AUTH, out, size_is(, *outDataSize)] BYTE** outData
);
```

Parameters*:*

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_KEY_HANDLE	parentHandle	Handle of the parent key that can decrypt the encData.
UINT32	inDataSize	Size of inData.
BYTE*	inData	XOR'd and encrypted key.
UINT32	randomDataSize	Size of randomData.
BYTE*	randomData	Random value used to hide the key data.
UNIT32*	outDataSize	Used size of the output area

		for outData.
BYTE**	outData	The encrypted private key that can be loaded with TPM_LoadKey.
TPM_AUTH*	parentAuth	Authorization digest for the owner and input/returned parameters. HMAC key: parentKey.usageAuth.

Comment:

TPM command – TPM_ConvertMigrationBlob

TPM ordinal – TPM_ConvertMigrationBlob

5.8.2.6.8 Tcsip_AuthorizeMigrationKey

Start of informative comment:

Tcsip_AuthorizeMigrationKey creates an authorization blob, to allow the TPM owner to specify which migration facility they will use and allow users to migrate information without further involvement with the TPM owner.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_AuthorizeMigrationKey
(
   TCS_CONTEXT_HANDLE hContext,              // in
   TSS_MIGRATE_SCHEME migrateScheme,         // in
   UINT32             MigrationKeySize,      // in
   BYTE*              MigrationKey,          // in
   TPM_AUTH*          ownerAuth,             // in, out
   UINT32*            MigrationKeyAuthSize, // out
   BYTE**             MigrationKeyAuth      // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_AuthorizeMigrationKey")]
TSS_RESULT Tcsip_AuthorizeMigrationKey
(
   [in] TCS_CONTEXT_HANDLE                           hContext,
   [AUTH, in] TSS_MIGRATE_SCHEME                     migrateScheme,
   [AUTH, in] UINT32                                 MigrationKeySize,
   [AUTH, in, size_is(MigrationKeySize)] BYTE* MigrationKey,
   [AUTH, in, out] TPM_AUTH*                         ownerAuth,
   [AUTH, out] UINT32*
                                    MigrationKeyAuthSize,
   [AUTH,    out,    size_is(,    *MigrationKeyAuthSize)]    BYTE**
                                    MigrationKeyAuth
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TSS_MIGRATE_SCHEME	migrateScheme	Type of migration operation that is to be permitted for this key.
UINT32	MigrationKeySize	Size of TCPA_PUBKEY byte stream with Public key to be authorized
BYTE*	MigrationKey	TCPA_PUBKEY byte stream with Public key to be authorized
TPM_AUTH*	ownerAuth	Authorization digest for the

		owner and input/returned parameters. HMAC key: ownerAuth.
UINT32*	MigrationKeyAuth Size	Size of TCPA_MIGRATIONKEYAUTH byte stream with public key and authorization digest
BYTE**	MigrationKeyAuth	TCPA_MIGRATIONKEYAUTH byte stream with public key and authorization digest

Comment:

TPM command – TPM_AuthorizeMigrationKey

TPM ordinal – TPM_AuthorizeMigrationKey

5.8.2.6.9 Tcsip_SetOperatorAuth

Start of informative comment:

Sets the operator authorization value for the platform.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_SetOperatorAuth
(
   TCS_CONTEXT_HANDLE   hContext,     // in
   TCPA_SECRET          operatorAuth  // in
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCPA_SECRET	operatorAuth	The new operator authorization value, in plaintext.

Return Values

TSS_SUCCESS
TSS_E_VERSION_MISMATCH

Comment:

TPM ordinal – TPM_SetOperatorAuth

5.8.2.7 TPM Cryptographic Capabilities

5.8.2.7.1 Tcsip_CertifyKey

Start of informative comment:

Tcsip_CertifyKey allows a key to certify the public portion of certain storage and signing keys.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_CertifyKey
(
   TCS_CONTEXT_HANDLE hContext,        // in
   TCS_KEY_HANDLE     certHandle,      // in
   TCS_KEY_HANDLE     keyHandle,       // in
   TCPA_NONCE         antiReplay,      // in
   TPM_AUTH*          certAuth,        // in, out
   TPM_AUTH*          keyAuth,         // in, out
   UINT32*            CertifyInfoSize, // out
   BYTE**             CertifyInfo,     // out
   UINT32*            outDataSize,     // out
   BYTE**             outData          // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_CertifyKey")]
TSS_RESULT Tcsip_CertifyKey
(
   [in] TCS_CONTEXT_HANDLE         hContext,
   [in] TCS_KEY_HANDLE             certHandle,
   [in] TCS_KEY_HANDLE             keyHandle,
   [AUTH, in] TCPA_NONCE           antiReplay,
   [AUTH, in, out] TPM_AUTH*       certAuth,
   [AUTH, in, out] TPM_AUTH*       keyAuth,
   [AUTH, out] UINT32*             CertifyInfoSize,
   [AUTH, out, size_is(, *CertifyInfoSize)] BYTE**  CertifyInfo,
   [AUTH, out] UINT32*             outDataSize,
   [AUTH, out, size_is(, *outDataSize)] BYTE** outData
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_KEY_HANDLE	certHandle	Handle of the key to be used to certify the key.
TCS_KEY_HANDLE	keyHandle	Handle of the key to be certified.

TCPA_NONCE	antiReplay	Nonce to be inserted in the certifyInfo structure.
TPM_AUTH*	certAuth	The authorization handle used for certHandle.
TPM_AUTH*	keyAuth	The authorization handle used for the key to be signed.
UINT32*	CertifyInfoSize	Size of the CertifyInfo
BYTE**	CertifyInfo	The certifyInfo structure that corresponds to the signed key.
UINT32*	outDataSize	The used size of the output area for outData
BYTE**	outData	The signed public key.

Comment:

TPM command – TPM_CertifyKey

TPM ordinal – TPM_CertifyKey

5.8.2.7.2 Tcsip_CertifyKey2

Start of informative comment:

Tcsip_CertifyKey allows a key to certify the public portion of certifiable migratable storage and signing keys.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_CertifyKey2
(
   TCS_CONTEXT_HANDLE hContext,        // in
   TCS_KEY_HANDLE     certHandle,      // in
   TCS_KEY_HANDLE     keyHandle,       // in
   BYTE*              MSAdigest,       // in
   TCPA_NONCE         antiReplay,      // in
   TPM_AUTH*          certAuth,        // in, out
   TPM_AUTH*          keyAuth,         // in, out
   UINT32*            CertifyInfoSize, // out
   BYTE**             CertifyInfo,     // out
   UINT32*            outDataSize,     // out
   BYTE**             outData          // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_CertifyKey2")]
TSS_RESULT Tcsip_CertifyKey2
(
   [in] TCS_CONTEXT_HANDLE                               hContext,
   [in] TCS_KEY_HANDLE                                   certHandle,
   [in] TCS_KEY_HANDLE                                   keyHandle,
   [in] BYTE*                                            MSAdigest,
   [AUTH, in] TCPA_NONCE                                 antiReplay,
   [AUTH, in, out] TPM_AUTH*                             certAuth,
   [AUTH, in, out] TPM_AUTH*                             keyAuth,
   [AUTH, out] UINT32*
                                        CertifyInfoSize,
   [AUTH, out, size_is(, *CertifyInfoSize)] BYTE**       CertifyInfo,
   [AUTH, out] UINT32*                                   outDataSize,
   [AUTH, out, size_is(, *outDataSize)] BYTE**           outData
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_KEY_HANDLE	certHandle	Handle of the key to be used to certify the key.
TCS_KEY_HANDLE	keyHandle	Handle of the key to be certified.

BYTE*	MSAdigest	The digest of a TPM_MSA_COMPOSITE strucutre, containing at least one public key of a Migration Authority
TCPA_NONCE	antiReplay	Nonce to be inserted in the certifyInfo structure.
TPM_AUTH*	certAuth	The authorization handle used for certHandle.
TPM_AUTH*	keyAuth	The authorization handle used for the key to be signed.
UINT32*	CertifyInfoSize	Size of the CertifyInfo
BYTE**	CertifyInfo	The certifyInfo structure that corresponds to the signed key.
UINT32*	outDataSize	The used size of the output area for outData
BYTE**	outData	The signed public key.

Comment:

TPM command – TPM_CertifyKey2

TPM ordinal – TPM_CertifyKey2

5.8.2.7.3 Tcsip_Sign

Start of informative comment:

Tcsip_Sign signs a digest and returns the resulting digital signature. This command uses a properly authorized signature key.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_Sign
(
   TCS_CONTEXT_HANDLE hContext,        // in
   TCS_KEY_HANDLE     keyHandle,       // in
   UINT32             areaToSignSize,  // in
   BYTE*              areaToSign,      // in
   TPM_AUTH*          privAuth,        // in, out
   UINT32*            sigSize,         // out
   BYTE**             sig              // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_Sign")]
TSS_RESULT Tcsip_Sign
(
   [in] TCS_CONTEXT_HANDLE                        hContext,
   [in] TCS_KEY_HANDLE                            keyHandle,
   [AUTH, in] UINT32                              areaToSignSize,
   [AUTH, in size_is(areaToSignSize] BYTE*        areaToSign,
   [AUTH, in, out] TPM_AUTH*                      privAuth,
   [AUTH, out] UINT32*                            sigSize,
   [AUTH, out, size_is(, *sigSize)] BYTE**        sig
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_KEY_HANDLE	keyHandle	The keyHandle identifier of a loaded key that can perform digital signatures.
UINT32	areaToSignSize	The size of the areaToSign parameter
BYTE*	areaToSign	The value to sign
TPM_AUTH*	privAuth	The authorization digest that authorizes the use of keyHandle. HMAC key: key.usageAuth
UINT32*	sigSize	The length of the returned digital signature
BYTE**	sig	The resulting digital signature.

Comment:

TPM command – TPM__Sign

TPM ordinal – TPM__Sign

5.8.2.7.4 Tcsip_GetRandom

Start of informative comment:

Tcsip_GetRandom returns the next bytesRequested bytes from the random number generator to the caller.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_GetRandom
(
   TCS_CONTEXT_HANDLE hContext,        // in
   UINT32*            bytesRequested,  // in, out
   BYTE**             randomBytes      // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_GetRandom")]
TSS_RESULT Tcsip_GetRandom
(
   [in] TCS_CONTEXT_HANDLE                     hContext,
   [in, out] UINT32*                           bytesRequested,
   [out, size_is(, *bytesRequested)] BYTE** randomBytes
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
UINT32*	bytesRequested	Number of bytes to return
BYTE**	randomBytes	The returned bytes

Comment:

TPM command – TPM_GetRandom

TPM ordinal – TPM_GetRandom

5.8.2.7.5 Tcsip_StirRandom

Start of informative comment:

Tcsip_StirRandom adds entropy to the RNG state.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_StirRandom
(
   TCS_CONTEXT_HANDLE hContext,    // in
   UINT32             inDataSize,  // in
   BYTE*              inData       // in
);
```

IDL Definition:

```
[helpstring("method Tcsip_StirRandom")]
TSS_RESULT Tcsip_StirRandom
(
   [in] TCS_CONTEXT_HANDLE          hContext,
   [in] UINT32                      inDataSize,
   [in, size_is(*inDataSize)] BYTE* inData
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
UINT32	inDataSize	Number of bytes of input (<256)
BYTE*	inData	Data to add entropy to RNG state

Comment:

TPM command – TPM_StirRandom

TPM ordinal – TPM_StirRandom

5.8.2.7.6 Tcsip_GetCapability

Start of informative comment:

Tcsip_GetCapability allows the TPM to report back to the requestor what type of TPM it is dealing with.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_GetCapability
(
   TCS_CONTEXT_HANDLE    hContext,     // in
   TCPA_CAPABILITY_AREA  capArea,      // in
   UINT32                subCapSize,   // in
   BYTE*                 subCap,       // in
   UINT32*               respSize,     // out
   BYTE**                resp          // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_GetCapability")]
TSS_RESULT Tcsip_GetCapability
(
   [in] TCS_CONTEXT_HANDLE          hContext,
   [in] TCPA_CAPBILITY_AREA         capArea,
   [in] UINT32                      subCapSize,
   [in, size_is(subCapSize)] BYTE*  subCap,
   [out] UINT32*                    respSize,
   [out, size_is(,*respSize)] BYTE**resp
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCPA_CAPABILITY_AREA	capArea	Partition of capabilities to be interrogated
UINT32	subCapSize	Size of subCap parameter
BYTE*	subCap	Further definition of information
UINT32*	respSize	The length of the returned capability response
BYTE**	resp	The capability response

Comment:

TPM command – TPM_GetCapability

TPM ordinal – TPM_GetCapability

Information about capArea and rgbSubCap is transmitted to the TPM without any interpretation by TCS. The TPM will return an appropriate error on wrong values.

5.8.2.7.7 Tcsip_GetCapabilitySigned

Start of informative comment:

NOTE: The TPM function TPM_GetCapabilitySigned that actually performs this functions was found to contain a vulnerability that makes its security questionable therefore its use unadvised. Since the final TPM specification contained this function and products have shipped with this function it is exposed at the TPM layer. However, the TSS Working Group has decided that TSS should not require the implementation of this function for any TSS. However, if a TSS provider should decided to include this function the TSS WG recommends the implementation contained here.

Tcsip_GetCapabilitySigned is almost the same as Tcsip_GetCapability. The differences are that the input includes a challenge (a nonce) and the response includes a digital signature to vouch for the source of the answer.

C-Definition:

```
TSS_RESULT Tcsip_GetCapabilitySigned
   (
   TCS_CONTEXT_HANDLE    hContext,     // in
   TCS_KEY_HANDLE        keyHandle,    // in
   TCPA_NONCE            antiReplay,   // in
   TCPA_CAPABILITY_AREA  capArea,      // in
   UINT32                subCapSize,   // in
   BYTE*                 subCap,       // in
   TPM_AUTH*             privAuth,     // in, out
   TCPA_VERSION*         Version,      // out
   UINT32*               respSize,     // out
   BYTE**                resp,         // out
   UINT32*               sigSize,      // out
   BYTE**                sig           // out
   );
```

IDL Definition:

```
[helpstring("method Tcsip_GetCapabilitySigned")]
TSS_RESULT Tcsip_GetCapabilitySigned
   (
   [in] TCS_CONTEXT_HANDLE                       hContext,
   [in] TCS_KEY_HANDLE                           keyHandle,
   [AUTH, in] TCPA_NONCE                         antiReplay,
   [AUTH, in] TCPA_CAPBILITY_AREA                capArea,
   [AUTH, in] UINT32                             subCapSize,
   [AUTH, in, size_is(subCapSize)] BYTE*         subCap,
   [AUTH, in, out] TPM_AUTH*                     privAuth,
   [AUTH, out] TCPA_VERSION*                     Version,
   [AUTH, out] UINT32*                           respSize,
   [AUTH, out, size_is(, *respSize)] BYTE**      resp,
   [AUTH, out] UINT32*                           sigSize,
   [AUTH, out, size_is(, *sigSize)] BYTE**       sig,
   );
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_KEY_HANDLE	keyHandle	The handle of a loaded key that can perform digital signatures.
TCPA_NONCE	antiReplay	Nonce to be inserted in the certifyInfo structure.
TCPA_CAPABILITY_AREA	capArea	Partition of capabilities to be interrogated
UINT32	subCapSize	Size of subCap parameter
BYTE*	subCap	Further definition of information
TPM_AUTH*	privAuth	The authorization digest that authorizes the use of keyHandle. HMAC key: key.usageAuth
TCPA_VERSION*	Version	A properly filled out version

		structure.
UINT32*	respSize	The length of the returned capability response
BYTE**	resp	The capability response
UINT32*	sigSize	The length of the returned digital signature
BYTE**	sig	The resulting digital signature.

Comment:

TPM command – TPM_GetCapabilitySigned

TPM ordinal – TPM_GetCapabilitySigned

Information about capArea and rgbSubCap is transmitted to the TPM without any interpretation by TCS. The TPM will return an appropriate error on wrong values.

End of informative comment.

5.8.2.7.8 Tcsip_GetCapabilityOwner

Start of informative comment:

Tcsip_GetCapabilityOwner enables the TPM Owner to retrieve information belonging to the TPM Owner.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_GetCapabilityOwner
(
   TCS_CONTEXT_HANDLE    hContext,           // in
   TPM_AUTH*             pOwnerAuth,         // in out
   TCPA_VERSION*         pVersion,           // out
   UINT32*               pNonVolatileFlags,  // out
   UINT32*               pVolatileFlags      // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_GetCapabilityOwner")]
TSS_RESULT Tcsip_GetCapabilityOwner
(
   [in] TCS_CONTEXT_HANDLE        hContext,
   [in, out, ptr] TPM_AUTH*       pOwnerAuth,
   [out] TCPA_VERSION*            pVersion,
   [out] UINT32*                  pNonVolatileFlags,
   [out] UINT32*                  pVolatileFlags
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TPM_AUTH*	pOwnerAuth	Owner authorization
TCPA_VERSION*	pVersion	A properly filled out version structure.
UINT32*	pNonVolatileFlags	The current state of the non-volatile flags.
UINT32*	pVolatileFlags	The current state of the volatile flags.

Comment:

TPM command – TPM_GetCapabilityOwner

TPM ordinal – TPM_GetCapabilityOwner

Information about capArea and rgbSubCap is transmitted to the TPM without any interpretation by TCS. The TPM will return an appropriate error on wrong values.

TPM Endorsement Credentials

5.8.2.7.9 Tcsip_CreateEndorsementKeyPair

Start of informative comment:

Tcsip_CreateEndorsementKeyPair generates the endorsement key pair.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_CreateEndorsementKeyPair
(
   TCS_CONTEXT_HANDLE hContext,               // in
   TCPA_NONCE         antiReplay,             // in
   UINT32             endorsementKeyInfoSize, // in
   BYTE*              endorsementKeyInfo,     // in
   UINT32*            endorsementKeySize,     // out
   BYTE**             endorsementKey,         // out
   TCPA_DIGEST*       checksum                // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_CreateEndorsementKeyPair")]
TSS_RESULT Tcsip_CreateEndorsementKeyPair
(
   [in]TCS_CONTEXT_HANDLE   hContext,               // in
   [in]TCPA_NONCE           antiReplay,             // in
   [in]UINT32               endorsementKeyInfoSize, // in
   [in, size_is( endorsementKeyInfoSize )]BYTE*
                            endorsementKeyInfo,     // in
   [out]UINT32*             endorsementKeySize,     // out
   [out, size_is(, *endorsementKeySize )]BYTE**
                            endorsementKey,         // out
   [out]TCPA_DIGEST*        checksum                // out
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCPA_NONCE	antiReplay	Nonce to be inserted in the certifyInfo structure.
UINT32	endorsementKeyInfoSize	Endorsement key info size
BYTE*	endorsementKeyInfo	Endorsement key info
UINT32*	endorsementKeySize	Size of the endorsement key
BYTE**	endorsementKey	The public endorsement key
TCPA_DIGEST*	Checksum	Hash of pubEndorsementKey and antiReplay

Comment:

TPM command – TPM_CreateEndorsementKeyPair

TPM ordinal – TPM_CreateEndorsementKeyPair

5.8.2.7.10 Tcsip_ReadPubek

Start of informative comment:

Tcsip_ReadPubek returns the public portion of the endorsement key.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_ReadPubek
(
  TCS_CONTEXT_HANDLE hContext,                 // in
  TCPA_NONCE         antiReplay,               // in
  UINT32*            pubEndorsementKeySize,    // out
  BYTE**             pubEndorsementKey,        // out
  TCPA_DIGEST*       checksum                  // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_ReadPubek")]
TSS_RESULT Tcsip_ReadPubek
(
  [in] TCS_CONTEXT_HANDLE  hContext,
  [in] TCPA_NONCE          antiReplay,
  [out] UINT32*            pubEndorsementKeySize,
  [out, size_is(, *pubEndorsementKeySize)] BYTE**
                           pubEndorsementKey,
  [out] TCPA_DIGEST*       checksum
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCPA_NONCE	antiReplay	Nonce to be inserted in the certifyInfo structure.
UINT32*	pubEndorsementKeySize	Size of puEndorsementKey
BYTE**	pubEndorsementKey	The public endorsement key
TCPA_DIGEST*	checksum	Hash of pubEndorsementKey and antiReplay

Comment:

TPM command – TPM_ReadPubek

TPM ordinal – TPM_ReadPubek

5.8.2.7.11 Tcsip_DisablePubekRead

Start of informative comment:

Tcsip_DisablePubekRead allows the TPM owner to prevent any entity from reading the public portion of the endorsement key.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_DisablePubekRead
(
   TCS_CONTEXT_HANDLE hContext,  // in
   TPM_AUTH*          ownerAuth  // in, out
);
```

IDL Definition:

```
[helpstring("method Tcsip_DisablePubekRead")]
TSS_RESULT Tcsip_DisablePubekRead
(
   [in] TCS_CONTEXT_HANDLE    hContext,
   [AUTH, in, out] TPM_AUTH*  ownerAuth
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TPM_AUTH*	ownerAuth	Owner’s authorization

Comment:

TPM command – TPM_DisablePubekRead

TPM ordinal – TPM_DisablePubekRead

5.8.2.7.12 Tcsip_OwnerReadPubek

Start of informative comment:

Tcsip_OwnerReadPubek allows the TPM owner to read the public endorsement key.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_OwnerReadPubek
(
   TCS_CONTEXT_HANDLE hContext,                 // in
   TPM_AUTH*          ownerAuth,                // in, out
   UINT32*            pubEndorsementKeySize,    // out
   BYTE**             pubEndorsementKey         // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_OwnerReadPubek")]
TSS_RESULT Tcsip_OwnerReadPubek
(
   [in] TCS_CONTEXT_HANDLE     hContext,
   [AUTH, in, out] TPM_AUTH*   ownerAuth,
   [AUTH, out] UINT32*         pubEndorsementKeySize,
   [AUTH, out, size_is(, *pubEndorsementKeySize)] BYTE**
                               pubEndorsementKey
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TPM_AUTH*	ownerAuth	Owner's authorization
UINT32*	pubEndorsementKey Size	Size of puEndorsementKey
BYTE**	pubEndorsementKey	The public endorsement key

Comment:

TPM command – TPM_OwnerReadPubek

TPM ordinal – TPM_OwnerReadPubek

5.8.2.8 TPM Self-Test and Management

5.8.2.8.1 Tcsip_SelfTestFull

Start of informative comment:

Tcsip_SelfTestFull test all of the TPM protected capabilities.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_SelfTestFull
(
   TCS_CONTEXT_HANDLE hContext   // in
);
```

IDL Definition:

```
[helpstring("method Tcsip_SelfTestFull")]
TSS_RESULT Tcsip_SelfTestFull
(
   [in] TCS_CONTEXT_HANDLE  hContext
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.

Comment:

TPM command – TPM_OwnerReadPubek
TPM ordinal – TPM_OwnerReadPubek

5.8.2.8.2 Tcsip_CertifySelfTest

Start of informative comment:

Tcsip_CertifySelfTest performs a full TPM self-test and returns an authenticated value if the test passes.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_CertifySelfTest
(
   TCS_CONTEXT_HANDLE hContext,    // in
   TCS_KEY_HANDLE     keyHandle,   // in
   TCPA_NONCE         antiReplay,  // in
   TPM_AUTH*          privAuth,    // in, out
   UINT32*            sigSize,     // out
   BYTE**             sig          // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_CertifySelfTest")]
TSS_RESULT Tcsip_CertifySelfTest
(
   [in] TCS_CONTEXT_HANDLE    hContext,
   [in] TCS_KEY_HANDLE        keyHandle,
   [AUTH, in] TCPA_NONCE      antiReplay,
   [AUTH, in, out] TPM_AUTH*  privAuth,
   [AUTH, out] UINT32*        sigSize,
   [AUTH, out, size_is(, *sigSize)] BYTE**  sig
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_KEY_HANDLE	keyHandle	The keyHandle identifier of a loaded key that can perform digital signatures.
TCPA_NONCE	antiReplay	Nonce to be inserted in the certifyInfo structure.
TPM_AUTH*	privAuth	The authorization digest that authorizes the inputs and use of keyHandle. HMAC key: key.usageAuth
UINT32*	sigSize	The length of the returned digital signature
BYTE**	sig	The resulting digital signature.

Comment:

TPM command – TPM_CertifySelfTest

TPM ordinal – TPM_CertifySelfTest

5.8.2.8.3 Tcsip_ContinueSelfTest

Start of informative comment:

CotinueSelfTest informs the TPM that it may complete the self test of all TPM functions.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_ContinueSelfTest
(
   TCS_CONTEXT_HANDLE hContext   // in
);
```

IDL Definition:

```
[helpstring("method Tcsip_ContinueSelfTest")]
TSS_RESULT Tcsip_ContinueSelfTest
(
   [in] TCS_CONTEXT_HANDLE  hContext
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.

Comment:

TPM command – TPM_ContinueSelfTest

TPM ordinal – TPM_ContinueSelfTest

5.8.2.8.4 Tcsip_GetTestResult

Start of informative comment:

Tcsip_GetTestResult provides manufacturer specific information regarding the results of the self-test. This command will work when the TPM is in self-test failure mode.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_GetTestResult
(
   TCS_CONTEXT_HANDLE hContext,    // in
   UINT32*            outDataSize, // out
   BYTE**             outData      // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_GetTestResult")]
TSS_RESULT Tcsip_GetTestResult
(
   [in] TCS_CONTEXT_HANDLE  hContext,
   [AUTH, out] UINT32*      outDataSize,
   [AUTH, out, size_is(, *outDataSize)] BYTE** outData
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
UINT32*	outDataSize	The size of the outData area
BYTE**	outData	The outData this is manufacturer specific

Comment:

TPM command – TPM_GetTestResult

TPM ordinal – TPM_GetTestResult

5.8.2.8.5 *Tcsip_OwnerSetDisable*

Start of informative comment:

Tcsip_OwnerSetDisable is used to change the status of the TCPA_PERSISTENT_DISABLE flag.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_OwnerSetDisable
(
   TCS_CONTEXT_HANDLE hContext,     // in
   TSS_BOOL           disableState, // in
   TPM_AUTH*          ownerAuth     // in, out
);
```

IDL Definition:

```
[helpstring("method Tcsip_OwnerSetDisable")]
TSS_RESULT Tcsip_OwnerSetDisable
(
   [in] TCS_CONTEXT_HANDLE    hContext,
   [AUTH, in] TSS_BOOL        disableState,
   [AUTH, in, out] TPM_AUTH*  ownerAuth
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TSS_BOOL	disableState	Value for disable state – enable if TRUE
TPM_AUTH*	ownerAuth	Owner authorization

Comment:

TPM command – TPM_OwnerSetDisable
TPM ordinal – TPM_OwnerSetDisable

5.8.2.8.6 Tcsip_OwnerClear

Start of informative comment:

Tcsip_OwnerClear performs the clear operation under TPM owner authorization.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_OwnerClear
(
   TCS_CONTEXT_HANDLE hContext,     // in
   TPM_AUTH*          ownerAuth     // in, out
);
```

IDL Definition:

```
[helpstring("method Tcsip_OwnerClear")]
TSS_RESULT Tcsip_OwnerClear
(
   [in] TCS_CONTEXT_HANDLE    hContext,
   [AUTH, in, out] TPM_AUTH*  ownerAuth
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TPM_AUTH*	ownerAuth	Owner authorization

Comment:

TPM command – TPM_OwnerClear

TPM ordinal – TPM_OwnerClear

5.8.2.8.7 Tcsip_DisableOwnerClear

Start of informative comment:

Tcsip_DisableOwnerClear disables the ability to execute the TPM_OwnerClear command permanently.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_DisableOwnerClear
(
   TCS_CONTEXT_HANDLE hContext,  // in
   TPM_AUTH*          ownerAuth  // in, out
);
```

IDL Definition:

```
[helpstring("method Tcsip_DisableOwnerClear")]
TSS_RESULT Tcsip_DisableOwnerClear
(
   [in] TCS_CONTEXT_HANDLE    hContext,
   [AUTH, in, out] TPM_AUTH*  ownerAuth
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TPM_AUTH*	ownerAuth	Owner authorization

Comment:

TPM command – TPM_DisableOwnerClear

TPM ordinal – TPM_DisableOwnerClear

5.8.2.8.8 Tcsip_ForceClear

Start of informative comment:

Tcsip_ForceClear performs the Clear operation under physical access.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_ForceClear
(
   TCS_CONTEXT_HANDLE hContext// in
);
```

IDL Definition:

```
[helpstring("method Tcsip_ForceClear")]
TSS_RESULT Tcsip_ForceClear
(
   [in] TCS_CONTEXT_HANDLE  hContext
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.

Comment:

TPM command – TPM_ForceClear

TPM ordinal – TPM_ForceClear

5.8.2.8.9 Tcsip_DisableForceClear

Start of informative comment:

Tcsip_DisableForceClear disables the execution of the ForceClear command until the next startup cycle.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_DisableForceClear
(
   TCS_CONTEXT_HANDLE hContext// in
);
```

IDL Definition:

```
[helpstring("method Tcsip_DisableForceClear")]
TSS_RESULT Tcsip_DisableForceClear
(
   [in] TCS_CONTEXT_HANDLE  hContext
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.

Comment:

TPM command – TPM_DisableForceClear

TPM ordinal – TPM_DisableForceClear

TCG Software Stack (TSS) Specification

5.8.2.8.10 Tcsip_PhysicalDisable

Start of informative comment:

Tcsip_PhysicalDisable disables TPM physical presence.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_PhysicalDisable
(
   TCS_CONTEXT_HANDLE hContext   //in
);
```

IDL Definition:

```
[helpstring("method Tcsip_PhysicalDisable")]
TSS_RESULT Tcsip_PhysicalDisable
(
   [in] TCS_CONTEXT_HANDLE  hContext
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.

Comment:

TPM command – TPM_PhysicalDisable

TPM ordinal – TPM_PhysicalDisable

5.8.2.8.11 Tcsip_PhysicalEnable

Start of informative comment:

Tcsip_PhysicaEnable enables TPM physical presence.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_PhysicalEnable
(
   TCS_CONTEXT_HANDLE hContext// in
);
```

IDL Definition:

```
[helpstring("method Tcsip_PhysicalEnable")]
TSS_RESULT Tcsip_PhysicalEnable
(
   [in] TCS_CONTEXT_HANDLE  hContext
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.

Comment:

TPM command – TPM_PhysicalEnable

TPM ordinal – TPM_PhysicalEnable

5.8.2.8.12 Tcsip_PhysicalSetDeactivated

Start of informative comment:

Sets the TCPA_PERSISTENT_FLAGS.deactivated flag to the value in the state parameter

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_PhysicalSetDeactivated
(
   TCS_CONTEXT_HANDLE hContext,  // in
   TSS_BOOL           state      // in
);
```

IDL Definition:

```
[helpstring("method Tcsip_PhysicalSetDeactivated")]
TSS_RESULT Tcsip_PhysicalSetDeactivated
(
   [in] TCS_CONTEXT_HANDLE  hContext,
   [in] TSS_BOOL          state
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TSS_BOOL	state	State to which deactivated flag is to be set.

Comment:

TPM command – TPM_PhysicalSetDeactivated

TPM ordinal – TPM_PhysicalSetDeactivated

5.8.2.8.13 Tcsip_SetTempDeactivated

Start of informative comment:

Sets the flag TCPA_VOLATILE_FLAGS.deactivated to the value TRUE which temporality deactivate the TPM.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_SetTempDeactivated
(
  TCS_CONTEXT_HANDLE hContext   // in
);
```

IDL Definition:

```
[helpstring("method Tcsip_SetTempDeactivated")]
TSS_RESULT Tcsip_SetTempDeactivated
(
  [in] TCS_CONTEXT_HANDLE  hContext
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.

Comment:

TPM command – TPM_SetTempDeactivated

TPM ordinal – TPM_SetTempDeactivated

5.8.2.8.14 Tcsip_SetTempDeactivated2

Start of informative comment:

This function is identical to the Tcsip_SetTempDeactivated function except that it accepts an optional authorization parameter so it can be used with 1.2 TPMs. Sets the flag TCPA_VOLATILE_FLAGS.deactivated to the value TRUE which temporality deactivates the TPM.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_SetTempDeactivated2
(
   TCS_CONTEXT_HANDLE   hContext,     // in
   TPM_AUTH*            pOperatorAuth // in, out
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TPM_AUTH*	pOperatorAuth	Operator authorization.

Return Values

TSS_SUCCESS
TSS_E_VERSION_MISMATCH

Comment:

When connected to a 1.1b TPM the caller should supply a NULL pOperatorAuth parameter, in which case the TCS will call the unauthorized 1.1b TPM_SetTempDeactivated command.

When connected to a 1.2 TPM the caller should either supply a NULL pOperatorAuth parameter (in which case the command requires physical presence to be asserted) or a valid TPM_AUTH structure providing operator authentication.

TPM ordinal – TPM_SetTempDeactivated2

5.8.2.8.15 Tcsip_PhysicalPresence

Start of informative comment:

This method sets the physical presence flags.**End of informative comment.**

C-Definition:

```
   TSS_RESULT Tcsip_PhysicalPresence
(
      TCS_CONTEXT_HANDLE      hContext,           // in
      TCPA_PHYSICAL_PRESENCE  fPhysicalPresence   // in
   );
```

IDL Definition:

```
    [helpstring("method Tcsip_ PhysicalPresence")]
   TSS_RESULT Tcsip_PhysicalPresence
(
      [in] TCS_CONTEXT_HANDLE       hContext,
      [in] TCPA_PHYSICAL_PRESENCE   fPhysicalPresence
   );
```

Parameters

fPhysicalPresence

Value of the physical presence flag.

Return Values

TCS_SUCCESS
TCS_E_NOTIMPL

Remarks

The TSC_PhysicalPresence command is only available on platforms that provide the command method for indicating physical presence of the operator. This is determined by the nature and design of the platform. Further, execution of this command, if implemented, requires the platform be in a predetermined state. This state is usually, but not required, to be pre-OS. Because of these restrictions, this command will likely not be available and will return TSS_E_NOTIMPL. It is included here for the benefit of platforms which will execute the TSS in a “restricted” environment.

5.8.2.8.16 Tcsip_FieldUpgrade

Start of informative comment:

The TPM needs a mechanism to allow for updating the protected capabilities once a TPM is in the field. Given the varied nature of TPM implementations there will be numerous methods of performing an upgrade of the protected capabilities. This command, when implemented, provides a manufacturer hardware specific method of performing the upgrade.

The manufacturer can determine, within the listed requirements, how to implement this command from the TPM perspective. The command may be more than one command and actually a series of commands.

The IDL definition is to create an ordinal for the command, however the remaining parameters are manufacturer specific.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_FieldUpgrade
(
   TCS_CONTEXT_HANDLE hContext,    // in
   UINT32             dataInSize,  // in
   BYTE*              dataIn,      // in
   TPM_AUTH*          ownerAuth,   // in, out
   UINT32*            dataOutSize,// out
   BYTE**             dataOut      // out
);
```

Parameters:

hContext

Handle of the context object

dataInSize

size of the update blob.

ownerAuth

TPM owner authorization.

dataIn

Protected capabilites update blob, includes RevMajor, and RevMinor

dataOutSize

Size of the dataout

dataOut

manufacturer specific data used by the update process

Return Values

TCS_SUCCESS

TCS_E_INVALID_HANDLE
TCS_E_INTERNAL_ERROR
TCS_E_VERIFICATION_FAILED
TCS_E_INVALID_REVISION_NUMBER

IDL Definition:

```
[helpstring("method Tcsip_FieldUpgrade")]
TSS_RESULT Tcsip_FieldUpgrade
(
   [in]TCS_CONTEXT_HANDLE                    hContext,     // in
   [in]UINT32                                dataInSize,   // in
   [in]BYTE*                                 dataIn,       // in
   [in, out]TPM_AUTH*                        ownerAuth,    // in, out
   [out]UINT32*                              dataOutSize,  // out
   [out, size_is(, *dataOutSize )]BYTE**     dataOut       // out
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
UINT32	dataInSize	Size of field upgrade input data
BYTE*	dataIn	Field upgrade input data
TPM_AUTH*	ownerAuth	Owner authorization
UINT32*	dataOutSize	Size of field upgrade output data
BYTE**	dataOut	Field upgrade output data

Comment:

TPM command – TPM_FieldUpgrade

TPM ordinal – TPM_FieldUpgrade

Remarks:

dataIn is a blob that was signed by the private key of the manufacturer. TPMs that support field upgrade have the manufacturer's public key embedded in them to verify updates. This does not violate any of the constraints specified in the main spec. dataout could be a hash of the protected capabilities used to verify whether the update operation had no glitches. So Upon updating the protected caps, the TPM can hash the protected caps area and return that hash to **Tcsip_FieldUpgrade().** This hash along with the return code can be used to check the status of the update; this may need to be combined with a restart / TPM full self test operation.

5.8.2.8.17 Tcsip_ResetLockValue

Start of informative comment:

Resets the lock that gets set in a TPM after multiple false authorization attempts. This is used to prevent hammering attacks.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_ResetLockValue
(
   TCS_CONTEXT_HANDLE      hContext, // in
   TPM_AUTH*               ownerAuth // in, out
);
```

IDL Definition:

```
[helpstring("method Tcsip_ResetLockValue ")]TSS_RESULT Tcsip_ResetLockValue
    (
       [in]            TCS_CONTEXT_HANDLE hContext,
       [AUTH, in, out] TPM_AUTH*          ownerAuth
    );
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TPM_AUTH*	ownerAuth	The TPM owner authorization

Comment:

TPM command - TPM_ResetLockValue
TPM ordinal - TPM_ResetLockValue

5.8.2.8.18 Tcsip_SetRedirection

Start of informative comment:

'Redirected" keys enable the output of a TPM to be directed to non-TCG security functions in the platform, without exposing that output to non-security functions.

It is sometimes desirable to direct the TPM's output directly to specific platform functions without exposing that output to other platform functions. To enable this, the key in a leaf node of TCG Protected Storage can be tagged as a "redirect" key. Any plaintext output data secured by a redirected key is passed by the TPM directly to specific platform functions and is not interpreted by the TPM.

Since redirection can only affect leaf keys, redirection applies to: TPM_Unbind, TPM_Unseal, TPM_Quote, TPM_Sign.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_SetRedirection
(
   TCS_CONTEXT_HANDLE hContext,  // in
   TCS_KEY_HANDLE     keyHandle, // in
   UINT32             c1,        // in
   UINT32             c2,        // in
   TPM_AUTH*          privAuth   // in, out
);
```

IDL Definition:

```
[helpstring("method Tcsip_SetRedirection")]
TSS_RESULT Tcsip_SetRedirection
(
   [in] TCS_CONTEXT_HANDLE    hContext,
   [in] TCS_KEY_HANDLE        keyHandle,
   [AUTH, in] UINT32          c1,
   [AUTH, in] UINT32          c2,
   [AUTH, in, out] TPM_AUTH*  ownerAuth
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_KEY_HANDLE	keyHandle	The keyHandle identifier of a loaded key that can implement redirection.
UINT32	c1	Manufacturer parameter
UINT32	c2	Manufacturer parameter
TPM_AUTH*	privAuth	The authorization handle used for keyHandle authorization

Comment:

TPM command – TPM_SetRedirection

TPM ordinal – TPM_SetRedirection

5.8.2.9 Delegation

5.8.2.9.1 Tcsip_DSAP

Start of informative comment:

- Tcsip_DSAP opens a delegated authorization session.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_DSAP
(
   TCS_CONTEXT_HANDLE             hContext,        // in
   TPM_ENTITY_TYPE                entityType,           // in
   TCS_KEY_HANDLE                 keyHandle,       // in
   TPM_NONCE                      nonceOddDSAP,    // in
   UINT32                         entityValueSize, // in
   BYTE*                          entityValue,          // in
   TCS_AUTHHANDLE*                authHandle,           // out
   TPM_NONCE*                     nonceEven,       // out
   TPM_NONCE*                     nonceEvenDSAP    // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_DSAP")]
TSS_RESULT Tcsip_DSAP
(
   [in] TCS_CONTEXT_HANDLE                        hContext,
   [in] TPM_ENTITY_TYPE                           entityType,
   [in]  TCS_KEY_HANDLE                           keyHandle,
   [in]  TPM_NONCE                                nonceOddDSAP,
   [in]  UINT32                                   entityValueSize,
   [in, size_is(entityValueSize)]   BYTE*         entityValue,
   [out] TCS_AUTHHANDLE*                          authHandle,
   [out] TPM_NONCE*                               nonceEven,
   [out] TPM_NONCE*                               nonceEvenDSAP
);
```

Parameters:

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TPM_ENTITY_TYPE	entityType	The type of delegation, TPM_ET_DEL_*.
TCS_KEY_HANDLE	keyHandle	The handle of the key whose delegated authorization is being used. This parameter should be 0 if owner delegation is being used.
TPM_NONCE	nonceOddDSAP	The odd nonce for the DSAP session, provided by the caller.

UINT32	entityValueSize	The size of the entityValue buffer
BYTE*	entityValue	The delegation being used. This should either be a TPM_DELEGATE_KEY_BLOB, TPM_DELEGATE_OWNER_BLOB, or a UINT32 index into the TPM's owner delegation table. This must be consistent with the entityType parameter.
TCS_AUTHHANDLE*	authHandle	The session handle for the new auth session.
TPM_NONCE*	nonceEven	The nonceEven for the new DSAP session.
TPM_NONCE*	nonceEvenDSAP	The nonceEvenDSAP for the new DSAP session.

Return Values

TCS_SUCCESS
TCS_E_INVALID_HANDLE
TCS_E_BAD_PARAMETER

Remarks

Comment:

TPM command – TPM_DSAP

5.8.2.9.2 Tcsip_Delegate_Manage

Start of informative comment:

Tcsip_Delegate_Manage command is called by Tspi_Delegate_AddFamily, Tspi_Delegate_Invalidate_Family, SetAttributes,. It is authorized either by the TPM Owner or by physical presence. If no Owner is installed, Tcsip_Delegate_Manage requires no privilege to execute. The command uses the opFlag parameter with values

- TPM_FAMILY_CREATE to create a new family
- TPM_FAMILY_INVALIDATE to invalidate an existing family
- TPM_FAMILY_ENABLE to enable/disable use of a family and all the rows that belong to that family
- TPM_FAMILY_ADMIN to lock or unlock a family against further modification by TPM_Delegate_Manage. If a family is locked while there is no owner it cannot be unlocked until after ownership is established.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_Delegate_Manage
(
  TCS_CONTEXT_HANDLE             hContext,        // in
  TPM_FAMILY_ID                  familyID,        // in
  TPM_FAMILY_OPERATION           opFlag,          // in
  UINT32                         opDataSize,      // in
  BYTE*                          opData,          // in
  TPM_AUTH*                      ownerAuth,       // in, out
  UINT32*                        retDataSize,     // out
  BYTE**                         retData          // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_Delegate_Manage")]
TSS_RESULT Tcsip_Delegate_Manage
(
  [in] TCS_CONTEXT_HANDLE                        hContext,
  [AUTH, in] TPM_FAMILY_ID                       familyID,
  [AUTH, in] TPM_FAMILY_OPERATION                opFlag,
  [AUTH, in] UINT32                              opDataSize,
  [AUTH, in, size_is(opDataSize)] BYTE*          opData,
  [AUTH, in,out] TPM_AUTH*                       ownerAuth,
  [AUTH, out] UINT32*                            retDataSize,
  [AUTH, out, size_is(,*retDataSize)] BYTE**     retData
);
```

Parameters:

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.

TPM_FAMILY_ID	familyID	The family ID of the delegation family. The set of valid family ids can be determined from the output of Tcsip_Delegate_ReadTable.
TPM_FAMILY_OPERATION	opFlag	TPM_FAMILY_ENABLE, TPM_FAMILY_ADMIN, TPM_FAMILY_CREATE, TPM_FAMILY_INVALIDATE.
UINT32	opDataSize	The size of the opData buffer
BYTE*	opData	The data to drive the operation. For ENABLE, CREATE, and ADMIN, this is a boolean indicating the desired state. It should be NULL for INVALIDATE.
TPM_AUTH*	ownerAuth	TPM Owner authorization.
UINT32	retDataSize	The size of the output buffer.
BYTE**	retData	The output data. For CREATE, this is the new family id.

Return Values

TCS_SUCCESS
TCS_E_INVALID_HANDLE
TCS_E_BAD_PARAMETER

Remarks

If an owner is not installed, this TPM operation does not require authorization.

Comment:

TPM command – TPM_Delegate_Manage

5.8.2.9.3 Tcsip_Delegate_CreateKeyDelegation

Start of informative comment:

Tcsip_Delegate_CreateKeyDelegation is used to delegate the privilege to use a key by creating a blob that can be used by TPM_DSAP. These blobs CANNOT be used as input data for loading owner delegation, because the internal TPM delegate table is used to store owner delegations only.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_Delegate_CreateKeyDelegation
(
   TCS_CONTEXT_HANDLE    hContext,         // in
   TCS_KEY_HANDLE        hKey,             // in
   UINT32                publicInfoSize,   // in
   BYTE*                 publicInfo,       // in
   TPM_ENCAUTH           encDelAuth,       // in
   TPM_AUTH*             keyAuth,          // in, out
   UINT32*               blobSize,         // out
   BYTE**                blob              // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_Delegate_CreateKeyDelegation")]
TSS_RESULT Tcsip_Delegate_CreateKeyDelegation
(
   [in]        TCS_CONTEXT_HANDLE                  hContext,
   [in]        TCS_KEY_HANDLE,                     hKey,
   [AUTH, in] UINT32,                              publicInfoSize,
   [AUTH, in, size_is(publicInfoSize)] BYTE*,      publicInfo,
   [AUTH, in] TPM_ENCAUTH,                         encDelAuth,
   [in, out]  TPM_AUTH*,                           keyAuth,
   [AUTH, out]   UINT32*,                          blobSize,
   [AUTH, out, size_is(,*blobSize)] BYTE**         blob
);
```

Parameters:

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_KEY_HANDLE	hKey	The key whose use is being delegated.
UINT32	publicInfoSize	The size of the TPM_DELEGATE_PUBLIC.
BYTE*	publicInfo	The TPM_DELEGATE_PUBLIC describing the properties of the new key delegation.
TPM_ENCAUTH	encDelAuth	The new delegated authorization value, encrypted as appropriate for the authorization session.
TPM_AUTH*	keyAuth	Key usage authorization.
UINT32*	blobSize	The size of the output delegation.

BYTE**	blob	The output TPM_DELEGATE_KEY_BLOB.

Return Values

TCS_SUCCESS
TCS_E_INVALID_HANDLE
TCS_E_BAD_PARAMETER

Remarks

Comment:

TPM command – TPM_Delegate_CreateKeyDelegation

5.8.2.9.4 Tcsip_Delegate_CreateOwnerDelegation

Start of informative comment:

Tcsip_Delegate_CreateOwnerDelegation is used to delegate owner privileges to use a set of command ordinals by creating a blob. This blob can in turn be used as input data for TPM_DSAP or TPM_Delegate_LoadOwnerDelegation to provide proof of privilege. TPM_Delegate_CreateKeyDelegation must be used to delegate privilege to use a key.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_Delegate_CreateOwnerDelegation
(
   TCS_CONTEXT_HANDLE    hContext,        // in
   TSS_BOOL              increment,       // in
   UINT32                publicInfoSize,  // in
   BYTE*                 publicInfo,      // in
   TPM_ENCAUTH           encDelAuth,      // in
   TPM_AUTH*             ownerAuth,       // in, out
   UINT32*               blobSize,        // out
   BYTE**                blob             // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_Delegate_CreateOwnerDelegation")]
TSS_RESULT Tcsip_Delegate_CreateOwnerDelegation
(
   [in]       TCS_CONTEXT_HANDLE                  hContext,
   [AUTH, in] UINT32                              publicInfoSize,
   [AUTH, in, size_is(publicInfoSize)] BYTE*      publicInfo,
   [AUTH, in] TPM_ENCAUTH                         encDelAuth,
   [AUTH, in, out]  TPM_AUTH*                     ownerAuth,
   [AUTH, out]   UINT32*                          blobSize,
   [AUTH, out, size_is(,*blobSize)] BYTE**        blob
);
```

Parameters:

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
UINT32	publicInfo Size	The size of the TPM_DELEGATE_PUBLIC.
BYTE*	publicInfo	The TPM_DELEGATE_PUBLIC describing the featured of the new owner blob.
TPM_ENCAUTH	encDelAuth	The delegated authorization value, encrypted as appropriate for the auth session.
TPM_AUTH*	ownerAuth	TPM Owner authorization.
UINT32*	blobSize	The size of the output TPM_DELEGATE_OWNER_BLOB.

BYTE**	blob	The output TPM_DELEGATE_OWNER_BLOB.

Return Values

TCS_SUCCESS
TCS_E_INVALID_HANDLE
TCS_E_BAD_PARAMETER

Remarks

Comment:

TPM command – TPM_Delegate_CreateOwnerDelegation

5.8.2.9.5 Tcsip_Delegate_LoadOwnerDelegation

Start of informative comment:

Tcsip_Delegate_LoadOwnerDelegation is used to load an owner delegation blob into the TPM non-volatile delegation table. If an owner is installed the owner blob must be created with Tcsip_Delegate_CreateOwnerDelegation. If an owner is not installed the owner blob may be created outside the TPM and its TPM_DELEGATE_SENSITIVE component must be left unencrypted.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_Delegate_LoadOwnerDelegation
(
   TCS_CONTEXT_HANDLE    hContext,       // in
   TPM_DELEGATE_INDEX    index,          // in
   UINT32                blobSize,       // in
   BYTE*                 blob,           // in
   TPM_AUTH*             ownerAuth       // in, out
);
```

IDL Definition:

```
[helpstring("method Tcsip_Delegate_LoadOwnerDelegation")]
TSS_RESULT Tcsip_Delegate_LoadOwnerDelegation
(
   [in]       TCS_CONTEXT_HANDLE      hContext,
   [AUTH, in] TPM_DELEGATE_INDEX      index,
   [AUTH, in] UINT32                  blobSize,
   [AUTH, in, size_is(blobSize)] BYTE* blob,
   [AUTH, in, out]  TPM_AUTH*         ownerAuth,
);
```

Parameters:

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TPM_DELEGATE_INDEX	index	The target index in the delegation table.
UINT32	blobSize	The size of the TPM_DELEGATE_OWNER_BLOB
BYTE*	blob	The TPM_DELEGATE_OWNER_BLOB.
TPM_AUTH	ownerAuth	TPM Owner authorization.

Return Values

TCS_SUCCESS
TCS_E_INVALID_HANDLE
TCS_E_BAD_PARAMETER

Remarks

Comment:

TPM command – TPM_Delegate_LoadOwnerDelegation

5.8.2.9.6 Tcsip_Delegate_UpdateVerificationCount

Start of informative comment:

Tcsip_Delegate_UpdateVerificationCount sets the verificationCount in an entity (a blob or a delegation row) to the current family value, in order that the delegations represented by that entity will continue to be accepted by the TPM.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_Delegate_UpdateVerificationCount
(

   TCS_CONTEXT_HANDLE    hContext,     // in
   UINT32                inputSize,    // in
   BYTE*                 input,        // in
   TPM_AUTH*             ownerAuth,    // in, out
   UINT32*               outputSize,   // out
   BYTE**                output        // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_Delegate_UpdateVerificationCount")]
TSS_RESULT Tcsip_Delegate_UpdateVerificationCount
(
   [in]          TCS_CONTEXT_HANDLE            hContext,
   [AUTH, in]    UINT32                        inputSize,
   [AUTH, in, size_is(inputSize)] BYTE*        input,
   [AUTH, in, out]    TPM_AUTH*                ownerAuth,
   [AUTH, out]   UINT32*                       outputSize,
   [AUTH, out, size_is(,*outputSize)] BYTE**   output
);
```

Parameters:

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
UINT32	inputSize	The size of the input buffer
BYTE*	input	The input buffer, either a TPM_DELEGATE_KEY_BLOB, TPM_DELEGATE_OWNER_BLOB, or an index into the delegation table.
TPM_AUTH	ownerAuth	owner authorization.
UINT32*	outputSize	The size of the output buffer.
BYTE**	output	The output buffer. If the input was a TPM_DELEGATE_KEY_BLOB or TPM_DELEGATE_OWNER_BLOB, this contains the updated blob.

Return Values

TCS_SUCCESS
TPM_E_BAD_INDEX
TCS_E_INVALID_HANDLE
TCS_E_BAD_PARAMETER

Remarks

Comment:

TPM command – TPM_Delegate_UpdateVerfication

5.8.2.9.7 Tcsip_Delegate_VerifyDelegation

Start of informative comment:

Tcsip_Delegate_VerifyDelegation interprets a delegate blob and returns success or failure, depending on whether the blob is currently valid.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_Delegate_VerifyDelegation
(
   TCS_CONTEXT_HANDLE    hContext,     // in
   UINT32                delegateSize, // in
   BYTE*                 delegate      // in
);
```

IDL Definition:

```
[helpstring("method Tcsip_Delegate_VerifyDelegation")]
TSS_RESULT Tcsip_Delegate_VerifyDelegation
(
   [in]  TCS_CONTEXT_HANDLE          hContext,
   [in]  UINT32                      delegateSize,
   [in, size_is(delegateSize)] BYTE* delegate
);
```

Parameters:

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
UINT32	delegateSize	The size of the delegation blob
BYTE*	delegate	The delegation, either a TPM_DELEGATE_OWNER_BLOB or TPM_DELEGATE_KEY_BLOB

Return Values

TCS_SUCCESS
TPM_E_BAD_DELGATE
TCS_E_BAD_PARAMETER

Remarks

Comment:

TPM command – TPM_Delegate_VerfyDelegation

5.8.2.9.8 Tcsip_Delegate_ReadTable

Start of informative comment:

This command is used to read from the TPM the public contents of the family and delegate tables that are stored on the TPM. Such data is required during external verification of tables.

There are no restrictions on the execution of this command; anyone can read this information regardless of the state of the PCRs, regardless of whether they know any specific authorization value and regardless of whether or not the enable and admin bits are set one way or the other.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_Delegate_ReadTable
(
   TCS_CONTEXT_HANDLE    hContext,               // in
   UINT32*               pulFamilyTableSize,     // out
   BYTE**                ppFamilyTable,          // out
   UINT32*               pulDelegateTableSize,   // out
   BYTE**                ppDelegateTable         // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_Delegate_ReadTable")]
TSS_RESULT Tcsip_Delegate_ReadTable
(
   [in]  TCS_CONTEXT_HANDLE                       hContext,
   [out] UINT32*                                  pulFamilyTableSize,
   [out, size_is(,*pulFamilyTableSize)]   BYTE* ppFamilyTable,
   [out] UINT32*                                pulDelegateTableSize,
   [out, size_is(,*pulDelegateTableSize)]BYTE**ppDelegateTable
);
```

Parameters:

Type	Name	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
UINT32*	pulFamilyTableSize	Size of Family Table
BYTE**	ppFamilyTable	Family Table
UINT32*	pulDelegateTableSize	Size of the Delegation table
BYTE**	ppDelegateTable	Delegation table

Return Values

TCS_SUCCESS

TCS_E_INVALID_HANDLE
TCS_E_BAD_PARAMETER

Remarks

Comment:

TPM command – TPM_Delegate_ReadTable

5.8.2.10 NVRAM

5.8.2.10.1 Tcsip_NV_DefineOrReleaseSpace

Start of informative comment:

This command sets aside space in the TPM NVRAM and defines the access requirements necessary to read and write that space.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_NV_DefineOrReleaseSpace
(
   TCS_CONTEXT_HANDLE hContext,     // in
   TPM_UINT32         cPubInfoSize, // in
   BYTE*              pPubInfo,     // in
   TCPA_ENCAUTH       encAuth,      // in
   TPM_AUTH*          pAuth         // in, out
  );
```

IDL-Definition:

```
[helpstring("method Tcsip_NV_DefineOrReleaseSpace")]
TSS_RESULT Tcsip_NV_DefineOrReleaseSpace
(
   [in]TCS_CONTEXT_HANDLE                hContext,     // in
   [in]TPM_UINT32                        cPubInfoSize, // in
   [in, size_is(cPubInfoSize)]BYTE*      pPubInfo,     // in
   [in]TCPA_ENCAUTH                      encAuth,      // in
   [in, out]TPM_AUTH*                    pAuth         // in, out
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle to established context.
TPM_UINT32	cPubInfoSize	Size of the public info data.
BYTE*	pPubInfo	The public parameters of the requested NV area.
TCPA_ENCAUTH	encAuth	If XOR encryption is indicated this is used to decrypt AuthData secrets.
TPM_AUTH*	pAuth	Authorization session data including the HMAC digest for using the wrapping key. If NULL, no authorization is required.

Comment:

The attributes of the NV space requested, with the exception of read and/or write authorization data, (including PCRs necessary to either read or write to the space) are included in the public parameters of the requested NV area. The authorization data which will be used for either read/write or both is provided (encrypted) in encAuth. If this function is called twice, the first time it will create the space and the second time delete it.

Return Value:

TCS_SUCCESS
TCS_E_FAIL

5.8.2.10.2 Tcsip_NV_WriteValue

Start of informative comment:

This command writes the value to a defined area. The write can be TPM Owner authorized or unauthorized and protected by other attributes and will work when no TPM owner is present.

End of informative comment.

Definition:

```
TSS_RESULT Tcsip_NV_WriteValue
(
   TCS_CONTEXT_HANDLE hContext,        // in
   TSS_NV_INDEX       hNVStore,        // in
   UINT32             offset,          // in
   UINT32             ulDataLength,    // in
   BYTE*              rgbDataToWrite,  // in
   TPM_AUTH*          privAuth         // in, out
);
```

IDL Definition:

```
[helpstring("method Tcsip_NV_WriteValue")]
TSS_RESULT Tcsip_NV_WriteValue
(
   [in] TCS_CONTEXT_HANDLE                       hContext,
   [AUTH, in] TSS_NV_INDEX                       hNVStore,
   [AUTH, in} UINT32                             offset,
   [AUTH, in] UINT32                             ulDataLength,
   [AUTH, in, size_is(ulDataLength)] BYTE*       rgbDataToWrite
   [AUTH, in, out] TPM_AUTH*                     privAuth
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TSS_NV_INDEX	hNVStore	Index of NV
UINT32	offset	Offset into the memory to begin writing
UINT32	ulDataLength	Size of the data to write
UINT32	rgbDataToWrite	Data to write.
TPM_AUTH*	privAuth	If NULL, this command uses no owner authorization

Return Values

TCS_SUCCESS
TCS_E_BAD_PARAMETER
TCS_E_INTERNAL_ERROR
TPM_E_BAD_INDEX

TPM_MAXNVWRITE
TPM_AUTH_CONFLICT
TPM_AUTHFAIL
TPM_AREA_LOCKED
TPM_BAD_LOCALITY
TPM_BAD_PRESENCE
TPM_DISABLED_CMD
TPM_NOSPACE
TPM_NOT_FULLWRITE
TPM_WRONGPCRVALUE

Remarks

If a policy object is assigned to this object, the authData within the policy object will be used to authorize this operation. If there is no policy object associated with this object, an unauthenticated write will be performed.

5.8.2.10.3

5.8.2.10.4 Tcsip_NV_WriteValueAuth

Start of informative comment:

This command writes to a previously defined area. The area must require authorization to write. This command is for using when authorization other than the owner authorization is to be used.

End of informative comment.

Definition:

```
TSS_RESULT Tcsip_NV_WriteValueAuth
(
   TCS_CONTEXT_HANDLE hContext,        // in
   TSS_NV_INDEX       hNVStore,        // in
   UINT32             offset,          // in
   UINT32             ulDataLength,    // in
   BYTE*              rgbDataToWrite,  // in
   TPM_AUTH*          NVAuth           // in, out
);
```

IDL Definition:

```
[helpstring("method Tcsip_NV_WriteValueAuth")]
TSS_RESULT Tcsip_NV_WriteValueAuth
(
   [in] TCS_CONTEXT_HANDLE                    hContext,
   [AUTH, in] TSS_NV_INDEX                    hNVStore,
   [AUTH, in} UINT32                          offset,
   [AUTH, in] UINT32                          ulDataLength,
   [AUTH, in, size_is(ulDataLength)] BYTE*    rgbDataToWrite,
   [AUTH, in, out] TPM_AUTH*                  NVAuth
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TSS_NV_INDEX	hNVStore	Index of NV
UINT32	offset	Offset into the memory to begin writing
UINT32	ulDataLength	Size of the data to write
UINT32	rgbDataToWrite	Data to write.
TPM_AUTH*	NVAuth	NV element authorization data. This command uses no owner authorization.

Return Values

TCS_SUCCESS

TCS_E_BAD_PARAMETER
TCS_E_INTERNAL_ERROR
TPM_BAD_INDEX
TPM_MAXNVWRITE
TPM_AUTH_CONFLICT
TPM_AUTHFAIL
TPM_AREA_LOCKED
TPM_BAD_LOCALITY
TPM_BAD_PRESENCE
TPM_DISABLED_CMD
TPM_NOSPACE
TPM_NOT_FULLWRITE
TPM_WRONGPCRVALUE

Remarks

5.8.2.10.5 Tcsip_NV_ReadValue

Start of informative comment:

Read a value from the NV store. This command uses optional owner authorization.

End of informative comment.

Definition:

```
TSS_RESULT Tcsip_NV_ReadValue
(
   TCS_CONTEXT_HANDLE hContext,     // in
   TSS_NV_INDEX       hNVStore,     // in
   UINT32             offset,       // in
   UINT32*            pulDataLength,// in, out
   TPM_AUTH*          privAuth,     // in, out
   BYTE**             rgbDataRead   // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_NV_ReadValue")]
TSS_RESULT Tcsip_NV_ReadValue
(
   [in] TCS_CONTEXT_HANDLE                          hContext,
   [AUTH, in] TSS_NV_INDEX                          hNVStore,
   [AUTH, in] UINT32                                offset,
   [AUTH, in, out] UINT32*                          pulDataLength,
   [AUTH, in, out] TPM_AUTH*                        privAuth,
   [AUTH, out, size_is(ulDataLength)] BYTE**        rgbDataRead
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TSS_NV_INDEX	hNVStore	Index of NV Storage area
UINT32	offset	Offset into the memory to begin reading
UINT32*	pulDataLength	Size of the data to read
TPM_AUTH*	privAuth	Owner authorization data that authorizes the reading of the NV storage area (if needed)
BYTE**	RgbDataRead	Where to place the data read

Return Values

TCS_SUCCESS
TCS_E_INVALID_HANDLE
TCS_E_BAD_PARAMETER
TCS_E_INTERNAL_ERROR
TPM_BAD_INDEX

TPM_AUTH_CONFLICT
TPM_AUTHFAIL
TPM_BAD_LOCALITY
TPM_BAD_PRESENCE
TPM_DISABLED_CMD
TPM_NOSPACE
TPM_WRONGPCRVALUE

Remarks

If a policy object is assigned to this object, the authData within the policy object will be used to authorize this operation. If there is no policy object associated with this object, an unauthenticated read will be performed.

5.8.2.10.6 Tcsip_NV_ReadValueAuth

Start of informative comment:

Read a value from the NV store. This command uses optional owner authentication.

End of informative comment.

Definition:

```
TSS_RESULT Tcsip_NV_ReadValueAuth
(
  TCS_CONTEXT_HANDLE hContext,     // in
  TSS_NV_INDEX       hNVStore,     // in
  UINT32             offset,       // in
  UINT32*            pulDataLength,// in
  TPM_AUTH*          NVAuth,       // in, out
  BYTE**             rgbDataRead   // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_NV_ReadValueAuth")]
TSS_RESULT Tcsip_NV_ReadValueAuth
(
  [in] TCS_CONTEXT_HANDLE                       hContext,
  [AUTH, in] TSS_NV_INDEX                       hNVStore,
  [AUTH, in] UINT32                             offset,
  [AUTH, in, out] UINT32*                       pulDataLength,
  [AUTH, in, out] TPM_AUTH*                     NVAuth,
  [AUTH, out, size_is(ulDataLength)] BYTE**     rgbDataRead
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TSS_NV_INDEX	hNVStore	Index of NV Storage area
UINT32	offset	Offset into the memory to begin reading
UINT32*	pulDataLength	Size of the data to read
TPM_AUTH*	NVAuth	NV element authorization data that authorize the reading of the NV storage area.
BYTE**	rgbDataRead	Where to place the data read

Return Values

TCS_SUCCESS
TCS_E_INVALID_HANDLE
TCS_E_BAD_PARAMETER
TCS_E_INTERNAL_ERROR
TPM_BAD_INDEX

TPM_AUTH_CONFLICT
TPM_AUTHFAIL
TPM_BAD_LOCALITY
TPM_BAD_PRESENCE
TPM_DISABLED_CMD
TPM_NOSPACE
TPM_WRONGPCRVALUE

Remarks

5.8.2.11 TPM Optional

5.8.2.11.1 Tcsip_CreateMaintenanceArchive

Start of informative comment:

Tcsip_CreateMaintenanceArchive creates a TPM maintenance archive.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_CreateMaintenanceArchive
(
   TCS_CONTEXT_HANDLE hContext,        // in
   TSS_BOOL           generateRandom,  // in
   TPM_AUTH*          ownerAuth,       // in, out
   UINT32*            randomSize,      // out
   BYTE**             random,          // out
   UINT32*            archiveSize,     // out
   BYTE**             archive          // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_CreateMaintenanceArchive")]
TSS_RESULT Tcsip_CreateMaintenanceArchive
(
   [in] TCS_CONTEXT_HANDLE         hContext,
   [AUTH, in] TSS_BOOL             generateRandom,
   [AUTH, in, out] TPM_AUTH*       ownerAuth,
   [AUTH, out] UINT32*             randomSize,
   [AUTH, out, size_is(, *randomSize)] BYTE**  random,
   [AUTH, out] UINT32*             archiveSize,
   [AUTH, out, size_is (, *archiveSize)] BYTE**  archive
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TSS_BOOL	generateRandom	Use RNG or Owner auth to generate 'random'.
TPM_AUTH*	ownerAuth	The authorization handle used for owner authorization.
UINT32*	randomSize	Size of the returned random data. Will be 0 if generateRandom is FALSE.
BYTE**	random	Random data to XOR with result.
UINT32*	archiveSize	Size of the encrypted archive
BYTE**	archive	Encrypted key archive.

Comment:

TPM command – TPM_CreateMaintenanceArchive
TPM ordinal – TPM_CreateMaintenanceArchive

5.8.2.11.2 Tcsip_LoadMaintenanceArchive

Start of informative comment:

Tcsip_LoadMaintenanceArchive loads a TPM maintenance archive that has been massaged by the manufacturer to load into another TPM.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_LoadMaintenanceArchive
(
   TCS_CONTEXT_HANDLE hContext,     // in
   UINT32             dataInSize,   // in
   BYTE*              dataIn,       // in
   TPM_AUTH*          ownerAuth,    // in, out
   UINT32*            dataOutSize,  // out
   BYTE**             dataOut       // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_LoadMaintenanceArchive")]
TSS_RESULT Tcsip_LoadMaintenanceArchive
(
   [in]TCS_CONTEXT_HANDLE                 hContext,     // in
   [in]UINT32                             dataInSize,   // in
   [in]BYTE*                              dataIn,       // in
   [in]TPM_AUTH*                          ownerAuth,    // in, out
   [out]UINT32*                           dataOutSize,  // out
   [out, size_is(, *dataOutSize )]BYTE** dataOut       // out
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
UINT32	dataInSize	Size of vendor-specific data
BYTE*	dataIn	Vendor specific data
TPM_AUTH*	ownerAuth	Owner authorization
UINT32*	dataOutSize	Size of Vendor specific data
BYTE**	dataOut	Vendor specific data

Comment:

TPM command – TPM_LoadMaintenanceArchive

TPM ordinal – TPM_LoadMaintenanceArchive

5.8.2.11.3 Tcsip_KillMaintenanceArchive

Start of informative comment:

Tcsip_KillMaintenanceFeature is a permanent action that prevents ANYONE from creating a TPM maintenance archive until a new TPM owner is set.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_KillMaintenanceFeature
(
   TCS_CONTEXT_HANDLE hContext,  // in
   TPM_AUTH*          ownerAuth  // in, out
);
```

IDL Definition:

```
[helpstring("method Tcsip_KillMaintenaceFeature")]
TSS_RESULT Tcsip_KillMaintenanceFeature
(
   [in] TCS_CONTEXT_HANDLE    hContext,
   [AUTH, in, out] TPM_AUTH*  ownerAuth
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TPM_AUTH*	ownerAuth	Owner authorization

Comment:

TPM command – TPM_KillMaintenanceFeature

TPM ordinal – TPM_KillMaintenanceFeature

5.8.2.11.4 Tcsip_LoadManuMaintPub

Start of informative comment:

Tcsip_LoadManuMainPub loads the TPM manufacture's public key for use in the maintenance process.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_LoadManuMaintPub
(
   TCS_CONTEXT_HANDLE hContext,  // in
   TCPA_NONCE         antiReplay, // in
   UINT32             PubKeySize, // in
   BYTE*              PubKey,    // in
   TCPA_DIGEST*       checksum   // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_LoadManuMaintPub")]
TSS_RESULT Tcsip_LoadManuMaintPub
(
   [in] TCS_CONTEXT_HANDLE         hContext,
   [in] TCPA_NONCE                 antiReplay,
   [in] UINT32                     PubKeySize,
   [in, size_is(PubKeySize)] BYTE* PubKey,
   [out] TCPA_DIGEST*              checksum
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCPA_NONCE	antiReplay	Nonce to be inserted in the certifyInfo structure.
UINT32	PubKeySize	Size of the public key
BYTE*	PubKey	The public key of the manufacturer to be in use for maintenance
TCPA_DIGEST*	checksum	Digest of pubKey and antiReplay

Comment:

TPM command – TPM_LoadManuMaintPub

TPM ordinal – TPM_LoadManuMaintPub

5.8.2.11.5 Tcsip_ReadManuMaintPub

Start of informative comment:

Tcsip_ReadManuMainPub is used to check whether the manufacturer's public maintenance key in a TPM has the expected value.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_ReadManuMaintPub
(
   TCS_CONTEXT_HANDLE hContext,    // in
   TCPA_NONCE         antiReplay,  // in
   TCPA_DIGEST*       checksum     // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_ReadManuMaintPub")]
TSS_RESULT Tcsip_ReadManuMaintPub
(
   [in] TCS_CONTEXT_HANDLE  hContext,
   [in] TCPA_NONCE          antiReplay,
   [out] TCPA_DIGEST*       checksum
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCPA_NONCE	antiReplay	Nonce to be inserted in the certifyInfo structure.
TCPA_DIGEST*	checksum	Digest of pubKey and antiReplay

Comment:

TPM command – TPM_ReadManuMaintPub

TPM ordinal – TPM_ReadManuMaintPub

5.8.2.12 New EK Commands

5.8.2.12.1 Tcsip_CreateRevocableEndorsementKeyPair

Start of informative comment:

Tcsip_CreateRevocableEndorsementKeyPair generates the revocable endorsement key pair.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_CreateRevocableEndorsementKeyPair
(
   TCS_CONTEXT_HANDLE hContext,                // in
   TCPA_NONCE         antiReplay,              // in
   UINT32             endorsementKeyInfoSize,  // in
   BYTE*              endorsementKeyInfo,      // in
   TSS_BOOL           GenResetAuth,            // in
   TCPA_DIGEST*       EKResetAuth,             // in, out
   UINT32*            endorsementKeySize,      // out
   BYTE**             endorsementKey,          // out
   TCPA_DIGEST*       checksum                 // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_CreateRevocableEndorsementKeyPair")]
TSS_RESULT Tcsip_CreateRevocableEndorsementKeyPair
(
   [in]TCS_CONTEXT_HANDLE   hContext,                // in
   [in]TCPA_NONCE           antiReplay,              // in
   [in]UINT32               endorsementKeyInfoSize,  // in
   [in, size_is( endorsementKeyInfoSize )]BYTE*
                            endorsementKeyInfo,      // in
   [in] TSS_BOOL            GenResetAuth,            // in
   [in, out, ptr] TCPA_DIGEST* EKResetAuth,          // in, out
   [out]UINT32*             endorsementKeySize,      // out
   [out, size_is(, *endorsementKeySize )]BYTE**
                            endorsementKey,          // out
   [out]TCPA_DIGEST*        checksum                 // out
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCPA_NONCE	antiReplay	Nonce to be inserted in the certifyInfo structure.
UINT32	endorsementKeyInfoSize	Endorsement key info size
BYTE*	endorsementKeyInfo	Endorsement key info
TSS_BOOL	GenResetAuth	If set the TPM generates EK reset value, otherwise the passed EKResetAuth value is used.

TCPA_DIGEST*	EKResetAuth	The authorization value to revoke the TPM EK.
UINT32*	endorsementKeySize	Size of the endorsement key
BYTE**	endorsementKey	The public endorsement key
TCPA_DIGEST*	Checksum	Hash of pubEndorsementKey and antiReplay

Comment:

TPM command – TPM_CreateRevocableEK

TPM ordinal – TPM_CreateRevocableEK

5.8.2.12.2 Tcsip_RevokeEndorsementKeyPair

Start of informative comment:

Tcsip_RevokeEndorsementKeyPair clears the TPM revocable endorsement key pair.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_RevokeEndorsementKeyPair
(
   TCS_CONTEXT_HANDLE hContext,           // in
   TCPA_DIGEST          EKResetAuth      // in
);
```

IDL Definition:

```
[helpstring("method Tcsip_RevokeEndorsementKeyPair")]
TSS_RESULT Tcsip_RevokeEndorsementKeyPair
(
   [in] TCS_CONTEXT_HANDLE  hContext,           // in
   [in] TCPA_DIGEST         EKResetAuth         // in
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCPA_DIGEST	EKResetAuth	The authorization value to revoke the TPM EK.

Comment:

TPM command – TPM_RevokeTrust

TPM ordinal – TPM_RevokeTrust

5.8.2.13 Section 2: New PCR commands:

5.8.2.13.1 Tcsip_PcrReset

Start of informative comment:

This method resets a PCR register. Whether or not it succeeds may depend on the locality executing the command. PCRs can be defined in a platform specific specification to allow reset of certain PCRs only for certain localities. The one exception to this is PCR 15, which can always be reset in a 1.2 implementation. (This is to allow for software testing).

End of informative comment.

Definition:

```
TSS_RESULT Tcsip_PcrReset
(
    TCS_CONTEXT_HANDLE    hContext,        // in
    UINT32                pcrTargetSize,   // in
    BYTE*                 pcrTarget        // in
);
```

```
IDL Definition:
[helpstring("method Tcsip_PcrReset")]
TSS_RESULT Tcsip_PcrReset
(
    [in]TCS_CONTEXT_HANDLE                  hContext,        // in
    [in]UINT32                              pcrTargetSize,   // in
    [in, size_is( pcrTargetSize )]BYTE*     pcrTarget        // in

);
```

Parameters

pcrTarget

The PCRs to reset (as defined in TPM spec)

Return Values

TCS_SUCCESS
TCS_E_INVALID_HANDLE
TCS_TPM_NOT_RESETABLE
TCS_WRONG_LOCALITY
TCS_E_INTERNAL_ERROR

Remarks

The Tcsip_PcrReset will either reset ALL of the PCRs selected in pcrTarget or NONE of them.

5.8.2.14 Monotonic Counter TCS functions

Start of informative comment:

The monotonic counter is likely to be used to tag data as corresponding to a particular counter value. It is important that the counter itself not be incremented except as authorized by the owner of the platform, as otherwise it could really mess up software that counts on knowing every time the counter is incremented. Therefore a number of the commands that speak directly to the counter will have to have restricted access. Implementations on how to control this restricted access will no doubt vary from OS to OS. Specifically the create, increment, and release counter arguments need to be controlled.

End of informative comment.

5.8.2.14.1 Tcsip_ReadCounter

Start of informative comment:

Tcsip_ReadCounter reads the current value of a counter register.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_ReadCounter
(
   TCS_CONTEXT_HANDLE hContext,   // in
   TSS_COUNTER_ID     idCounter,  // in
   TPM_COUNTER_VALUE* counterValue// out
);
```

IDL Definition:

```
[helpstring("method Tcsip_ReadCounter")]
TSS_RESULT Tcsip_ReadCounter
(
   [in]TCS_CONTEXT_HANDLE  hContext,      // in
   [in]TSS_COUNTER_ID      idCounter,     // in
   [out]TPM_COUNTER_VALUE* counterValue   // out
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TSS_COUNTER_ID	idCounter	Handle of the counter to read.
TPM_COUNTER_VALUE*	counterValue	Current value of the counter.

Comment

TPM command – TPM_ReadCounter

TPM ordinal - TPM_ReadCounter

5.8.2.14.2 Tcsip_CreateCounter

Start of informative comment:

This method creates a new counter in the TPM. It does NOT select that counter. Counter creation assigns an authorization value to the counter and sets the counter's original start value to be one more than the internal base counter. The pLable size is 4.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_CreateCounter
   (
     TCS_CONTEXT_HANDLE  hContext,    // in
     UINT32              LabelSize,   // in (=4)
     BYTE *              pLabel,      // in
     TCPA_ENCAUTH        CounterAuth, // in
     TPM_AUTH *          pOwnerAuth,  // in, out
     TSS_COUNTER_ID *    idCounter,   // out
     TPM_COUNTER_VALUE * counterValue // out
   );
```

IDL Definition:

```
[helpstring("method Tcsip_CreateCounter")]
TSS_RESULT Tcsip_CreateCounter
   (
   [in]        TCS_CONTEXT_HANDLE   hContext,       // in
   [in]        UINT32               LabelSize,      // in (always=4)
   [in]        BYTE *               pLabel,         // in
   [in]        TCPA_ENCAUTH         CounterAuth,    // in
   [in, out]   TPM_AUTH *           pOwnerAuth,     // in, out
   [out]       TSS_COUNTER_ID *     idCounter,      // out
   [out]       TPM_COUNTER_VALUE*   counterValue    // out
   );
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
UINT32	LabelSize	Always equals 4
BYTE*	pLabel	Label to assign to this counter, up to 4 characters.
TCPA_ENCAUTH	CounterAuth	Encrypted authorization data for subsequent access to this counter.
TPM_AUTH*	pOwnerAuth	TPM owner authorization.
TSS_COUNTER_ID	idCounter	Handle used for subsequent access to this counter.
TPM_COUNTER_VALUE*	counterValue	Current value of the counter.

Comment

TPM command – TPM_CreateCounter

TPM ordinal - TPM_CreateCounter

5.8.2.14.3 Tcsip_IncrementCounter

Start of informative comment:

This method selects a counter if one has not yet been selected, and increments that counter register. If a counter has already been selected and it is different from the one requested, the increment counter will fail. To change the selected counter, the TPM must go through a startup cycle.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_IncrementCounter
   (
     TCS_CONTEXT_HANDLE  hContext,       // in
     TSS_COUNTER_ID      idCounter,      // in
     TPM_AUTH *          pCounterAuth,   // in, out
     TPM_COUNTER_VALUE * counterValue    // out
   );
```

IDL Definition:

```
[helpstring("method Tcsip_IncrementCounter")]
TSS_RESULT Tcsip_IncrementCounter
   (
   [in]       TCS_CONTEXT_HANDLE hContext,
   [in]       TSS_COUNTER_ID     idCounter,
   [in, out]  TPM_AUTH *         pCounterAuth,
   [out]      TPM_COUNTER_VALUE* counterValue
   );
```

Parameters

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TSS_COUNTER_ID	idCounter	Handle of the counter to increment.
TPM_AUTH*	pCounterAuth	Authorization for this counter.
TPM_COUNTER_VALUE*	counterValue	Current value of the counter.

Comment

TPM command – TPM_IncrementCounter

TPM ordinal - TPM_IncrementCounter

5.8.2.14.4 Tcsip_ReleaseCounter

Start of informative comment:

This method releases a counter so that no reads or increments of the indicated counter will succeed. It invalidates all information regarding that counter, including the counter handle.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_ReleaseCounter
(
   TCS_CONTEXT_HANDLE  hContext,   // in
   TSS_COUNTER_ID      idCounter,  // in
   TPM_AUTH *          pCounterAuth// in, out
);
```

IDL Definition

```
[helpstring("method Tcsip_ReleaseCounter")]
TSS_RESULT Tcsip_ReleaseCounter
(
   [in]TCS_CONTEXT_HANDLE  hContext,  // in
   [in]TSS_COUNTER_ID      idCounter,  // in
   [in, out]TPM_AUTH *     pCounterAuth // in, out
);
```

Parameters

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TSS_COUNTER_ID	idCounter	Handle of the counter to release.
TPM_AUTH*	pCounterAuth	Authorization for this counter.

Comment

TPM command – TPM_ReleaseCounter

TPM ordinal - TPM_ReleaseCounter

5.8.2.14.5 Tcsip_ReleaseCounterOwner

Start of informative comment:

This method releases a counter so that no reads or increments of the indicated counter will succeed. It invalidates all information regarding that counter, including the counter handle. It differs from Tcsip_ReleaseCounter in that it requires the TPM owner authorization instead of the authorization for the counter.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_ReleaseCounterOwner
(
   TCS_CONTEXT_HANDLE  hContext,   // in
   TSS_COUNTER_ID      idCounter,  // in
   TPM_AUTH *          pOwnerAuth  // in, out
);
```

IDL Definition

```
[helpstring("method Tcsip_ReleaseCounterOwner")]
TSS_RESULT Tcsip_ReleaseCounterOwner
   (
   [in]       TCS_CONTEXT_HANDLE  hContext,   // in
   [in]       TSS_COUNTER_ID      idCounter,  // in
   [in, out]  TPM_AUTH *          pOwnerAuth  // in, out
   );
```

Parameters

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TSS_COUNTER_ID	idCounter	Handle of the counter to release.
TPM_AUTH*	pOwnerAuth	TPM Owner authorization.

Comment

TPM command – TPM_ReleaseCounterOwner

TPM ordinal - TPM_ReleaseCounterOwner

5.8.2.15 Time Stamping Function Definitions

5.8.2.15.1 Tcsip_ReadCurrentTicks

Start of informative comment:

This method reads the current tick out of the TPM.

End of informative comment.

Definition:

```
TSS_RESULT Tcsip_ReadCurrentTicks
(
   TCS_CONTEXT_HANDLE    hContext,        // in
   UINT32*               pulCurrentTime,  // out
   BYTE**                prgbCurrentTime  // out
);
```

IDL-Definition:

```
[helpstring("method Tcsip_ReadCurrentTicks")]
TSS_RESULT Tcsip_ReadCurrentTicks
(
   [in]  TCS_CONTEXT_HANDLE                   hContext,
   [out] UINT32*                              pulCurrentTime,
   [out, size_is(, *ulCurrentTime)] BYTE**    prgbCurrentTime
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle to established context.
UINT32*	pulCurrentTime	Size of the current time count data blob.
BYTE**	prgbCurrentTime	The current time count data held in the TPM (TPM_CURRENT_TICKS struct).

Comment:

The Tcsip_ReadCurrentTicks method reads the current value of the time counter in the TPM using the internal TPM command TPM_GetTicks.

Return Value:

TCS_SUCCESS
TCS_E_FAIL
TCS_E_INTERNAL_ERROR

Remarks

The Tcsi_TPM_ReadCurrentTicks method reads the current value of the tick counter in the TPM using the internal TPM command TPM_GetTicks.

5.8.2.15.2 Tcsip_TickStampBlob

Start of informative comment:

This method is similar to a time stamp: it associates a tickvalue with a blob, indicating that the blob existed at some point earlier than the time corresponding to the tickvalue.

End of informative comment.

Definition:

```
TSS_RESULT Tcsip_TickStampBlob
(
    TCS_CONTEXT_HANDLE hContext,           // in
    TSS_HKEY           hKey,               // in
    TPM_NONCE          antiReplay,         // in
    TPM_DIGEST         digestToStamp,      // in
    TPM_AUTH*          privAuth,           // in, out
    UINT32*            pulSignatureLength,// out
    BYTE**             prgbSignature,      // out
    UINT32*            pulTickCountLength,      // out
    BYTE**             prgbTickCount       // out
);
```

Parameters

hContext

The handle of 20 byte hash of blob to be tickstamped

hKey

The key used to perform the signature operation.

antiReplay

An application-supplied nonce to ensure freshness of the signature.

digestToStamp

The value being signed

privAuth

The authorization digests that authorizes the use of hKey.

pulSignatureLength

Length of resultant signed tickstamp

prgbSignature

On successful completion this parameter points to the signature data which makes up the tickstamp

pulTickCountLength

Length of the resulting tick count

prgbTickCount

On successful completion, this parameter is the current tick value used in the signature.

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TSS_HKEY	hKey	The key used to perform the signature operation.
TPM_NONCE	AntiReplay	An application supplied nonce to ensure freshness of the signature
TPM_DIGEST	digestToStamp	The value being signed
TPM_AUTH*	privAuth	The authorization digests that aughorizes the use of hKey
UINT32*	pulSignatureLength	Length of resultant signed tickstamp
BYTE**	prgbSignature	Points to the signature data which makes up the tickstamp.
UINT32*	pulTickCountLength	Size of the current time count data blob.
BYTE**	prgbTickCount	The current time count data held in the TPM (TPM_CURRENT_TICKS struct).

Return Values

TCS_SUCCESS
TCS_E_INVALID_HANDLE
TCS_E_INTERNAL_ERROR

Remarks

Tcsip_TickStampBlob can be used to link an external timestamp to the tick / ticknonce inside the TPM

5.8.2.16 DAA Commands

5.8.2.16.1 Tcsip_TPM_DAA_Join

Start of informative comment:

Executes the TPM DAA Join command

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_TPM_DAA_Join
(
   TCS_CONTEXT_HANDLE hContext,     // in
   TPM_HANDLE         handle,       // in
   BYTE               stage,        // in
   UINT32             inputSize0,   // in
   BYTE*              inputData0,   // in
   UINT32             inputSize1,   // in
   BYTE*              inputData1,   // in
   TPM_AUTH*          ownerAuth,    // in/out
   UINT32*            outputSize,   // out
   BYTE**             outputData    // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_TPM_DAA_Join")]
TSS_RESULT Tcsip_TPM_DAA_Join
(
   [in]TCS_CONTEXT_HANDLE                 hContext,
   [in]TPM_HANDLE                         handle,
   [in]BYTE                               stage,
   [in]UINT32*                            inputSize0,
   [in, size_is(*inputSize0)]UINT32*      inputData0,
   [in]UINT32*                            inputSize1,
   [in, size_is(*inputSize1)]UINT32*      inputData1
   [in,out]TPM_AUTH*                      ownerAuth,
   [out]UINT32**                          outputSize0,
   [out, size_is(*ordSize)]UINT32**       outputData0
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TPM_HANDLE	handle	Handle to the TPM object
BYTE	Stage	Stage of the DAA Join command
UNIT32	inputSize0	Size of the input data 0

UINT32*	inputData0	The input data 0
UINT32	inputSize1	Size of the input data 1
UINT32	InputData1	The input data 1
TPM_AUTH*	ownerAuth	The authorization handle used for owner authorization.
UNIT32*	outputSize	Size of the output data
UINT32**	outputData	The output data

Comment:

TPM command – TPM_DAA_Join
TPM ordinal – TPM_ORD_DAA_JOIN

5.8.2.16.2 Tcsip_TPM_DAA_Sign

Start of informative comment:

Executes the TPM DAA Sign command

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_TPM_DAA_Sign
(
    TCS_CONTEXT_HANDLE hContext,     // in
    TPM_HANDLE         handle,       // in
    BYTE               stage,        // in
    UINT32             inputSize0,   // in
    BYTE*              inputData0,   // in
    UINT32             inputSize1,   // in
    BYTE*              inputData1,   // in
    TPM_AUTH*          ownerAuth,    // in/out
    UINT32*            outputSize,   // out
    BYTE**             outputData    // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_TPM_DAA_Sign")]
TSS_RESULT Tcsip_TPM_DAA_Sign
(
    [in]TCS_CONTEXT_HANDLE                  hContext,
    [in]TPM_HANDLE                          handle,
    [in]BYTE                                stage,
    [in]UINT32*                             inputSize0,
    [in, size_is(*inputSize0)]UINT32*       inputData0
    [in]UINT32*                             inputSize1,
    [in, size_is(*inputSize1)]UINT32*       inputData1
    [in,out]TPM_AUTH*                       ownerAuth,
    [out]UINT32**                           outputSize,
    [out, size_is(*ordSize)]UINT32**        outputData
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TPM_HANDLE	handle	Handle to the TPM object
BYTE	Stage	Stage of the DAA Sign command
UNIT32	inputSize0	Size of the input data 0
UINT32*	inputData0	The input data 0
UNIT32	inputSize1	Size of the input data 1
UINT32*	inputData1	The input data 1

TPM_AUTH*	ownerAuth	The authorization handle used for owner authorization.
UINT32*	outputSize	Size of the output data
UINT32**	outputData	The output data

TSS_TSPATTRIB_CMK_INFO	TSS_TSPATTRIB_CMK_INFO_MA_APPROVAL	HMAC of the migration authority approval
	TSS_TSPATTRIB_CMK_INFO_MA_DIGEST	Migration authority digest data

Comment:

TPM command – TPM_DAA_Sign
TPM ordinal – TPM_ORD_DAA_SIGN

5.8.2.17 CMK commands:

5.8.2.17.1 Tcsip_MigrateKey

Start of informative comment:

The Tcsip_MigrateKey command performs the function of a migration authority. This command is used to permit a TPM enabled system to be a migration authority. To prevent the execution of this command using any other key as a parent key, this TPM operation works only if the keyUsage for the "hMaKey" is TPM_KEY_MIGRATE.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_MigrateKey
(
   TCS_CONTEXT_HANDLE hContext,          // in
   TCS_KEY_HANDLE     hMaKey,            // in
   UINT32             PublicKeySize,     // in
   BYTE*              PublicKey,         // in
   UINT32             inDataSize,        // in
   BYTE*              inData,            // in
   TPM_AUTH*          ownerAuth,         // in, out
   UINT32*            outDataSize,       // out
   BYTE**             outData            // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_MigrateKey")]
TSS_RESULT Tcsip_MigrateKey
(
   [in] TCS_CONTEXT_HANDLE                   hContext,
   [in] TCS_KEY_HANDLE                       hMaKey,
   [in] UINT32                               PublicKeySize,
   [in, size_is(PublicKeySize)] BYTE*        PublicKey,
   [in] UINT32                               inDataSize,
   [in, size_is(inDataSize)] BYTE*           inData,
   [in, out] TPM_AUTH*                       ownerAuth,
   [out] UINT32*                             outDataSize,
   [out, size_is(, *outDataSize)] BYTE**     outData
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_KEY_HANDLE	hMaKey	Handle of the key to be used to migrate the key.
UINT32	PublicKeySize	Public key info size

BYTE*	PublicKey	Public key info to be migrated
UINT32	inDataSize	Input data size
BYTE*	inData	Input data blob.
TPM_AUTH	ownerAuth	Authorization digest for the owner and input/returned parameters. HMAC key: ownerAuth.
UINT32*	outDataSize	Output data size
BYTE**	outData	The re-encrypted blob.

Comment:

TPM command – TPM_MigrateKey

TPM ordinal – TPM_MigrateKey

5.8.2.17.2 Tcsip_CMK_SetRestrictions

Start of informative comment:

This command is used by the owner to order the usage of a CMK with delegated authorization.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_CMK_SetRestrictions
(
    TCS_CONTEXT_HANDLE     hContext,          // in
    TSS_CMK_DELEGATE       Restriction,       // in
    TPM_AUTH*              ownerAuth          // in, out
);
```

IDL Definition:

```
[helpstring("method Tcsip_CMK_SetRestrictions")]
TSS_RESULT Tcsip_CMK_SetRestrictions
(
    [in] TCS_CONTEXT_HANDLE        hContext,
    [in] TSS_CMK_DELEGATE          Restriction,
    [in, out] TPM_AUTH*            ownerAuth
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TSS_CMK_DELEGATE	Restriction	Bit mask how to set the restriction on CMK keys.
TPM_AUTH*	ownerAuth	Authorization digest for the owner and input/ returned parameters. HMAC key: ownerAuth.

Comment:

TPM command – TPM_CMK_SetRestrictions

TPM ordinal – TPM_CMK_SetRestrictions

5.8.2.17.3 Tcsip_CMK_ApproveMA

Start of informative comment:

This command is used to create an authorization ticket, to allow the TPM owner to specify/select one or more Migration-Authorities they approve and allow users to generate CMK's without further involvement of the owner.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_CMK_ApproveMA
(
    TCS_CONTEXT_HANDLE    hContext,              // in
    TCPA_DIGEST           migAuthorityDigest,    // in
    TPM_AUTH*             ownerAuth,             // in, out
    TCPA_HMAC*            HmacMigAuthDigest      // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_CMK_ApproveMA")]
TSS_RESULT Tcsip_CMK_ApproveMA
(
    [in]  TCS_CONTEXT_HANDLE    hContext,
    [in]  TCPA_DIGEST           migAuthorityDigest,
    [in, out] TPM_AUTH*         ownerAuth,
    [out] TCPA_HMAC*            HmacMigAuthDigest
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCPA_DIGEST	migAuthorityDigest	The digest of a TCPA_MSA_COMPOSITE structure containing the selecte MA's.
TPM_AUTH*	ownerAuth	Authorization digest for the owner and input/ returned parameters. HMAC key: ownerAuth.
TCPA_HMAC	HmacMigAuthDigest	HMAC of the migAuthorityDigest.

Comment:

TPM command – TPM_CMK_ApproveMA

TPM ordinal – TPM_CMK_ApproveMA

5.8.2.17.4 Tcsip_CMK_CreateKey

Start of informative comment:

The TPM_CMK_CreateKey command both generates and creates a secure storage bundle for asymmetric keys whose migration is controlled/restricted by a migration authority. Only this command can be used to create these kind of keys.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_CMK_CreateKey
(
   TCS_CONTEXT_HANDLE hContext,           // in
   TCS_KEY_HANDLE     hWrappingKey,       // in
   TCPA_ENCAUTH       KeyUsageAuth,       // in
   TCPA_HMAC          MigAuthApproval,    // in
   TCPA_DIGEST        MigAuthorityDigest,// in
   UINT32*            keyDataSize,        // in, out
   BYTE**             prgbKeyData,        // in, out
   TPM_AUTH*          pAuth               // in, out
);
```

IDL-Definition:

```
[helpstring("method Tcsip_CMK_CreateKey")]
TSS_RESULT Tcsip_CMK_CreateKey
(
   [in] TCS_CONTEXT_HANDLE    hContext,
   [in] TCS_KEY_HANDLE        hWrappingKey,
   [in] TCPA_ENCAUTH          KeyUsageAuth,
   [in] TCPA_HMAC             MigAuthApproval,
   [in] TCPA_DIGEST           MigAuthorityDigest,
   [in, out] TCPA_UINT32*     keyDataSize,
   [in, out, size_is(, *keyDataSize)] BYTE** prgbKeyData,
   [in, out, ptr] TCPA_AUTH*  pAuth
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle to established context.
TCS_KEY_HANDLE	hWrappingKey	Application key handle of the already loaded wrapping parent key.
TCPA_ENCAUTH	KeyUsageAuth	Encrypted usage authorization data for the key to be created.
TCPA_HMAC	MigAuthApproval	A ticket, created by the TPM owner by using TPM_CMK_ApproveMA.

TCPA_DIGEST	MigAuthorityDigest	The digest of a TPM_MSA_COMPOSITE structure.
TCPA_UINT32*	keyDataSize	Size of the provided/returned key structure byte stream in bytes.
BYTE**	prgbKeyData	IN: Information about key to be created, pubkey.keyLength and pKey->encSize elements are 0. OUT: The key blob as defined in TCPA_KEY12 structure which includes the public and encrypted private key.
TPM_AUTH*	pAuth	Authorization session data including the HMAC digest for using the wrapping key. If NULL, no authorization is required.

Comment:

Tcsip_CMKCreateKey creates a new key as defined by the parameters provided by prgbKeyData. The new key gets a usage authorization secret and a migration authority selection/secret as given by the input parameters MigAuthApproval and MigAuthorityDigest. The content of both parameters are described in the TCG v1.2 Main Specification.

Return a key blob wrapped with the key addressed by the application key handle of the wrapping key, which must already be loaded.

All required memory resources to return the public key and the encrypted private key data must be allocated by TCS. The appropriate memory resources must be bound to the context provided by hContext.

If pAuth == NULL, no authorization for using the wrapping key is required.

Return Value:

TCS_SUCCESS
TCS_E_KM_LOADFAILED
TCS_E_FAIL

5.8.2.17.5 Tcsip_CMK_CreateTicket

Start of informative comment:

The owner controlled command TPM_CMK_CreateTicket uses a public key to verify the signature over a digest.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_CMK_CreateTicket
(
   TCS_CONTEXT_HANDLE hContext,             // in
   UINT32             PublicVerifyKeySize,  // in
   BYTE*              PublicVerifyKey,      // in
   TCPA_DIGEST        SignedData,           // in
   UINT32             SigValueSize,         // in
   BYTE*              SigValue,             // in
   TPM_AUTH*          pOwnerAuth,           // in, out
   TCPA_HMAC*         SigTicket             // out
);
```

IDL-Definition:

```
[helpstring("method Tcsip_CMK_CreateTicket")]
TSS_RESULT Tcsip_CMK_CreateTicket
(
   [in] TCS_CONTEXT_HANDLE                   hContext,
   [in] UINT32                               PublicVerifyKeySize,
   [in, size_is(PublicVerifyKeySize)] BYTE*  PublicVerifyKey,
   [in] TCPA_DIGEST                          SignedData,
   [in] UINT32                               SigValueSize,
   [in, size_is(SigValueSize)] BYTE*         SigValue,
   [in, out, ptr] TCPA_AUTH*                 pOwnertAuth,
   [out] TCPA_HMAC*                          SigTicket
   );
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle to established context.
UINT32	PublicVerifyKeySize	Public key size info.
BYTE*	PublicVerifyKey	Public key to be used to check the signature value.
TCPA_DIGEST	SignedData	The data to be signed.

UINT32	SigValueSize	Size of the signature value.
BYTE*	SigValue	The signature to be verified.
TPM_AUTH*	pOwnerAuth	Authorization digest for the owner and input/ returned parameters. HMAC key: ownerAuth.
TCPA_HMAC*	SigTicket	Ticket that proves digest created on this TPM.

Comment:

TPM command – TPM_CMK_CreateTicket

TPM ordinal – TPM_CMK_ CreateTicket

5.8.2.17.6 Tcsip_CMK_CreateBlob

Start of informative comment:

The TPM_CMK_CreateBlob command is very similar to TPM_CreateMigrationBlob, except that it uses migration authority data whose migration data are independent from tpmProof. It is possible for the parameter restrictTicket to be NULL.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_CMK_CreateBlob
(
    TCS_CONTEXT_HANDLE    hContext,              // in
    TCS_KEY_HANDLE        parentHandle,          // in
    TSS_MIGRATE_SCHEME    migrationType,         // in
    UINT32                MigrationKeyAuthSize,  // in
    BYTE*                 MigrationKeyAuth,      // in
    TCPA_DIGEST           PubSourceKeyDigest,    // in
    UINT32                msaListSize,           // in
    BYTE*                 msaList,               // in
    UINT32                restrictTicketSize,    // in
    BYTE*                 restrictTicket,        // in
    UINT32                sigTicketSize,         // in
    BYTE*                 sigTicket,             // in
    UINT32                encDataSize,           // in
    BYTE*                 encData,               // in
    TPM_AUTH*             parentAuth,            // in, out
    UINT32*               randomSize,            // out
    BYTE**                random,                // out
    UINT32*               outDataSize,           // out
    BYTE**                outData                // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_CMK_CreateBlob")]
TSS_RESULT Tcsip_CMK_CreateBlob
(
   [in] TCS_CONTEXT_HANDLE                         hContext,
   [in] TCS_KEY_HANDLE                             parentHandle,
   [in] TSS_MIGRATE_SCHEME                         migrationType,
   [in] UINT32
                                       MigrationKeyAuthSize,
   [in, size_is(MigrationKeyAuthSize)] BYTE*   MigrationKeyAuth,
   [in] TCPA_DIGEST                                PubSourceKeyDigest,
   [in] UINT32                                     msaListSize,
   [in, size_is(msaListSize)] BYTE*            msaList,
   [in] UINT32                                     restrictTicketSize,
   [in, size_is(restrictTicketSize)]   BYTE*   restrictTicket,
   [in] UINT32                                     sigTicketSize,
   [in, size_is(sigTicketSize)]  BYTE*         sigTicket,
   [in] UINT32                                     encDataSize,
   [in, size_is(encDataSize)] BYTE*            encData,
   [in, out] TPM_AUTH*                             parentAuth,
   [out] UINT32*                                   randomSize,
   [out, size_is(, *randomSize)] BYTE**        random,
   [out] UINT32*                                   outDataSize,
   [out, size_is(, *outDataSize)] BYTE**       outData
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_KEY_HANDLE	parentHandle	Handle of the parent key that can decrypt the encData.
TSS_MIGRATE_SCHEME	migrationType	Migration type, either MIGRATE or REWRAP.
UINT32	MigrationKeyAuthSize	Size of TCPA_MIGRATONKEYAUTH byte stream with public key and authorization digest
BYTE*	MigrationKeyAuth	TCPA_MIGRATONKEYAUTH byte stream with public key and authorization digest
TCPA_DIGEST	PubSourceKeyDigest	The digest of the TPM_PUBKEY of the entity to be migrated.
UINT32	msaListSize	The size of the msalist parameter.
BYTE*	msaList	One or more digests of

		the public keys beloging to migration authority.
UINT32	restrictTicketSize	The size of the restrictTicket paremeter in mode RESTRICT_APPROVE_DOUBLE or NULL.
BYTE*	restrictTicket	Can be NULL or containig the digest of migration authority as the destination parent.
UINT32	sigTicketSize	The size of the sigTicket paremeter in mode RESTRICT_APPROVE_DOUBLE or NULL.
BYTE*	sigTicket	Can be NULL or containing the HMAC signature ticket.
UINT32	encDataSize	Size of encData.
BYTE*	encData	Encrypted entity to be modified.
TPM_AUTH*	parentAuth	Authorization digest for the owner and input/ returned parameters. HMAC key: parentKey. Usaage Auth.
UINT32*	randomDataSize	Used size of the output area for radndomData.
BYTE**	randomData	String used for XOR encryption.
UNIT32*	outDataSize	Used size of the output area for outData.
BYTE**	outData	Modified encrypted entity.

Comment:

TPM command – TPM_CMK_CreateBlob
TPM ordinal – TPM_CMK_CreateBlob

5.8.2.17.7 Tcsip_CMK_ConvertMigration

Start of informative comment:

This command is used as the final step to finish migrating a key to a new TPM.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_CMK_ConvertMigration
(
   TCS_CONTEXT_HANDLE    hContext,        // in
   TCS_KEY_HANDLE        parentHandle,    // in
   TCPA_CMK_AUTH         restrictTicket,  // in
   TCPA_HMAC             sigTicket,       // in
   TCPA_UINT32           keyDataSize,     // in
    BYTE*                prgbKeyData,     // in
   UINT32                msaListSize,     // in
   BYTE*                 msaList,         // in
   UINT32                randomSize,      // in
   BYTE*                 random,          // in
   TPM_AUTH*             parentAuth,      // in, out
   UINT32*               outDataSize,     // out
   BYTE**                outData          // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_CMK_ConvertMigration")]
TSS_RESULT Tcsip_CMK_ConvertMigration
(
   [in] TCS_CONTEXT_HANDLE                 hContext,
   [in] TCS_KEY_HANDLE                     parentHandle,
   [in] TCPA_CMK_AUTH                      restrictTicket,
   [in] TCPA_HMAC                          sigTicket,
   [in] UINT32                             keyDataSize,
   [in, size_is(keyDataSize)] BYTE*        prgbKeyData,
   [in] UINT32                             msaListSize,
   [in, size_is(msaListSize)]  BYTE*       msaList,
   [in] UINT32*                            randomSize,
   [in, size_is(randomSize)] BYTE*         random,
   [in, out] TPM_AUTH*                     parentAuth,
   [out] UINT32*                           outDataSize,
   [out, size_is(, *outDataSize)] BYTE**   outData
);
```

Parameters*:*

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_KEY_HANDLE	parentHandle	Handle of the parent key that can decrypt the encData.

TCPA_CMK_AUTH	restrictTicket	Digest of the combination of Migration authority, destination parent key and the key to be migrated.
TCPA_HMAC	sigTicket	A signed ticket generated by the TPM device.
UINT32	keyDataSize	Size of the prgbKeyData parameter.
BYTE*	prgbKeyData	The public key of the key to be migrated. The private portion must be the XOR'd TPM_MIGRATE_ASYMKEY.
UINT32	msaListSize	Size of the msaList parameter.
BYTE*	msaList	One or more digests of public keys beloging to migration authorities.
UINT32	randomDataSize	Size of randomData.
BYTE*	randomData	Random value used to hide the key data.
TPM_AUTH*	parentAuth	Authorization digest for the owner and input/returned parameters. HMAC key: parentKey.usageAuth.
UNIT32*	outDataSize	Used size of the output area for outData.
BYTE**	outData	The encrypted private key that can be loaded with TPM_LoadKey/TPM_LoadKey2.

Comment:

TPM command – TPM_CMK_ConvertMigration
TPM ordinal – TPM_CMK_ConvertMigration

Note that the related TPM command migrates private keys only. The migration of the associated public keys is not specified by the TPM.
The application (i.e. in context of TCG it is the TSP) must generate a TPM_KEYxx complex structure before the migrated key can be used by the target TPM in a LoadKeyX command.

5.8.2.18 TPM Set/Get Capability Commands

5.8.2.18.1 Tcsip_SetCapability

Start of informative comment:

Tcsip_SetCapability allows the caller to set values in the TPM.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_SetCapability
(
   TCS_CONTEXT_HANDLE    hContext,  // in
   TCPA_CAPABILITY_AREA  capArea,   // in
   UINT32                subCapSize,   // in
   BYTE*                 subCap,    // in
   UINT32                valueSize,    // in
   BYTE*                 value,     // in
   TPM_AUTH*             ownerAuth  // in out
);
```

IDL Definition:

```
[helpstring("method Tcsip_SetCapability")]
TSS_RESULT Tcsip_SetCapability
(
   [in] TCS_CONTEXT_HANDLE          hContext,
   [in] TCPA_CAPBILITY_AREA         capArea,
   [in] UINT32                      subCapSize,
   [in, size_is(subCapSize)] BYTE*  subCap,
   [in] UINT32                      valueSize,
   [in, size_is(valueSize)] BYTE*   value,
   [in, out] TPM_AUTH*              ownerAuth
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCPA_CAPABILITY_AREA	capArea	Partition of capabilities to be set
UINT32	subCapSize	Size of subCap parameter
BYTE*	subCap	Further definition of information
UINT32	valueSize	The length of the value to be set
BYTE*	value	The value to be set
TPM_AUTH*	ownerAuth	Owner authorization

Comment:

TPM command – TPM_SetCapability
TPM ordinal – TPM_SetCapability

Information about capArea and subCap is transmitted to the TPM without any interpretation by TCS. The TPM will return an appropriate error on wrong values.

ownerAuth may be NULL if owner authorization is not required.

5.8.2.19 Audit Commands:

5.8.2.19.1 Tcsip _GetAuditDigest

Start of informative comment:

Tcsip_GetAuditDigest gets the Digest of audited originals

End of informative comment.

```
TSS_RESULT Tcsip_GetAuditDigest
(
    TCS_CONTEXT_HANDLE hContext,          // in
    UINT32             startOrdinal,      // in
    TPM_DIGEST*        auditDigest,       // out
    UINT32*            counterValueSize,  // out
    BYTE**             counterValue,      // out
    TSS_BOOL*          more,              // out
    UINT32*            ordSize,           // out
    UINT32**           ordList            // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_GetAuditDigest")]
TSS_RESULT Tcsip_GetAuditDigest
(
    [in]  TCS_CONTEXT_HANDLE                          hContext,
    [in]  UINT32                                      startOrdinal,
    [out] TPM_DIGEST*                                 auditDigest,
    [out] UINT32*                                   counterValueSize,
    [out, size_is(,*counterValueSize)]  BYTE**        counterValue,
    [out] TSS_BOOL*                                   more,
    [out] UINT32*                                     ordSize
    [out, size_is(,*ordSize)]UINT32**                 ordList
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
UINT32	startOrdinal	The starting ordinal for the list of audited ordinals to get.
TPM_DIGEST*	auditDigest	Digest of audited events
UINT32	counterValueSize	Size of counterValue buffer
BYTE**	counterValue	Byte-encoding of the TPM audit counter.
UNIT32*	ordSize	Number of ordinals in the audited ordinal list
UINT32**	ordList	The audited ordinal list

5.8.2.19.2 Tcsip _GetAuditDigestSigned

Start of informative comment:

Tcsip_GetAuditDigestSigned gets the signed Digest of audited originals

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_GetAuditDigestSigned
(
  TCS_CONTEXT_HANDLE hContext,          // in
  TCS_KEY_HANDLE     keyHandle,         // in
  TSS_BOOL           closeAudit,        // in
  TPM_NONCE          antiReplay,        // in
  TPM_AUTH*          privAuth,          // in, out
  UINT32*            counterValueSize,  // out
  BYTE**             counterValue,      // out
  TPM_DIGEST*        auditDigest,       // out
  TPM_DIGEST*        ordinalDigest,     // out
  UINT32*            sigSize,           // out
  BYTE**             sig                // out
);
```

IDL Definition:

```
[helpstring("method Tcsip_GetAuditDigestSigned")]
TSS_RESULT Tcsip_GetAuditDigestSigned
(
  [in]          TCS_CONTEXT_HANDLE   hContext,
  [in]          TCS_KEYHANDLE        keyHandle,
  [AUTH, in]    TSS_BOOL             closeAudit,
  [AUTH, in]    TPM_NONCE            antiReplay,
  [AUTH, in, out]TPM_AUTH*           privAuth,
  [AUTH, out]   UINT32*              counterValueSize,
  [AUTH, out,size_is(,*counterValueSize)] BYTE**counterValue,
  [AUTH, out]   TPM_DIGEST*          auditDigest,
  [AUTH, out]   TPM_DIGEST*          ordinalDigest,
  [AUTH, out]   UINT32*              sigSize,
  [AUTH, out, size_is(, *sigSize)] BYTE**  sig
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TCS_KEYHANDLE	keyHandle	The keyHandle identifier of a loaded key that can perform digital signatures.
TSS_BOOL	closeAudit	Indication of whether to close audit session
TPM_NONCE	antiReplay	The anti-replay nonce for the signature operation.
UINT32	counterValueSize	Size of counterValue buffer
BYTE**	counterValue	Byte-encoding of the TPM audit counter.

TPM_AUTH*	privAuth	The authorization digest that authorizes the use of keyHandle. HMAC key: key.usageAuth
TPM_DIGEST*	auditDigest	The TPM audit digest
TPM_DIGEST*	ordinalDigest	The digest of the audited ordinal list.
UNIT32*	sigSize	The length of the returned digital signature
BYTE**	Sig	The resulting digital signature.

Comment:

TPM command – TPM_GetAuditDigestSigned

TPM ordinal – TPM_GetAuditDigestSigned

5.8.2.19.3 Tcsip _SetOrdinalAuditStatus

Start of informative comment:

This command set the audit flag for a given ordinal. This command requires Owner authorization.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsip_SetOrdinalAuditStatus
(
   TCS_CONTEXT_HANDLE hContext,        // in
   TPM_AUTH*          ownerAuth,       // in, out
   UINT32             ordinalToAudit,  // in
   TSS_BOOL           auditState       // in
);
```

IDL Definition:

```
[helpstring("method Tcsip_SetOrdinalAuditStatus")]
TSS_RESULT Tcsip_SetOrdinalAuditStatus
(
   [in]TCS_CONTEXT_HANDLE     hContext,
   [AUTH, in, out] TPM_AUTH*  ownerAuth,
   [AUTH, in]UINT32*          ordinalToAudit,
   [AUTH, in]TSS_BOOL         auditState
);
```

Parameters:

Type	Label	Description
TCS_CONTEXT_HANDLE	hContext	Handle of established context.
TPM_AUTH*	ownerAuth	The authorization handle used for owner authorization.
UINT32	ordinalToAudit	The ordinal whose audit flag is to be set.
TSS_BOOL	auditState	The state value for the audit flag

Comment:

TPM command – TPM_SetOrdinalAuditStatus

TPM ordinal – TPM_ SetOrdinalAuditStatus

6. TCG Device Driver Library (TDDL)

6.1 TDDL Architecture

Start of informative comment:

The intent of this document is to describe an interface between the TCG Software Stack (TSS) and Trusted Platform Module (TPM) in a TCG-enabled Trusted Platform. This interface is called the TPM Device Driver Library Interface (TPM DDLI). The TPM device driver library (TPM DDL) is a module that exists between TSS and the low-level TPM device driver (TPM DD). The TPM DDL is implemented in user-mode and performs processing in the calling application context (i.e. TSS core system service). The TPM DDL is designed to be single-threaded, single-instance, and assumes that TPM command serialization has been performed by the calling application. The TPM DDLI is of a synchronous nature. The TPM vendor is responsible for defining the interface between this library and the actual TPM device. The TPM vendor can choose the communication and resource allocation mechanisms between this library and any kernel-mode TPM driver or software TPM simulator.

In most platform implementations, the TPM DLL is loaded when a TSS application (i.e. TCS) initializes. On most platforms, this will occur during operating system startup. To guarantee access to the TPM DDL by any TSS application module, a strict library naming convention must be followed for each operating system implementation.

End of informative comment.

6.2 Memory Management

Start of informative comment:

The "classical" memory allocation approach is used, by the TPM DDL, where the calling application allocates memory for the in and out parameters associated with each interface call. Symmetrically, the calling application is responsible for de-allocating the memory associated with any call to the TPM DDL.

Retrieving the required parameter size from the callee accepting to call the callee twice or always checking for an error return code "TPMDDL_INSUFFICIENT_BUFFER" is not supported since not every TPM command can be repeated getting the same results (e.g. TPM_Extend).

In the TPM DDLI described in this document, parameters are documented as follows:

in parameters with the comment "// in"

out parameters with the comment "// out"

End of informative comment.

6.3 TDDL Error Code Defines

With the following table the error codes common to all TDDL functions are listed. In addition to these error codes, the TSS_E_* error codes out of the range of common errors may also be returned with the layer set to the value for the TDDL.

In addition each Tddli function will list in its description the error return codes specific to the function.

Type	Definition
TDDL_SUCCESS	Successful completion of the operation.
TDDL_E_FAIL	The operation failed.
TDDL_E_BAD_PARAMETER	Same as TSS_E_BAD_PARAMETER
TDDL_E_OUTOFMEMORY	Same as TSS_E_OUTOFMEMORY
TDDL_E_COMPONENT_NOT_FOUND	TPM device driver is not running
TDDL_E_ALREADY_OPENED	TPM device driver opened.
TDDL_E_BADTAG	The capability or sub capability code is not correct or not supported.
TDDL_E_TIMEOUT	The operation has timed out.
TDDL_E_INSUFFICIENT_BUFFER	The receive buffer is too small.
TDDL_E_COMMAND_COMPLETED	The command has already completed.
TDDL_E_ALREADY_CLOSED	TPM device driver closed.
TDDL_E_IOERROR	An IO error occurred transmitting information to the TPM.
TDLL_E_COMMAND_ABORTED	TPM aborted processing of command.

6.4 TDDL-specific Return code Rules

Only return codes specified within each function MAY be returned for each function.

6.5 TDDL Interface

6.5.1 Tddli_Open

Start of informative comment:

This function establishes a connection with the TPM device driver. Following a successful response to this function, the TPM device driver must be prepared to process TPM command requests from the calling application. The application utilizing the TPM DDL is guaranteed to have exclusive access to the TPM device. If this call fails, it may be an indication that the TPM device driver is not loaded, started, or the TPM cannot support any protected requests.

This function must be called before calling Tddli_GetStatus, Tddli_GetCapability, Tddli_SetCapability, or Tddli_TransmitData.

End of informative comment.

Definition:

```
TSS_RESULT Tddli_Open( );
```

Parameters:

None.

Return Value:

```
TDDL_SUCCESS
TDDL_E_COMPONENT_NOT_FOUND
TDDL_E_ALREADY_OPENED
TDDL_E_FAIL
```


6.5.2 Tddli_Close

Start of informative comment:

This function closes a connection with the TPM device driver. Following a successful response to this function, the TPM device driver can clean up any resources used to maintain a connection with the TPM device driver library. If this call fails, it may provide an indication that the TPM device driver cannot clean up or may need to be restarted or reloaded.

End of informative comment.

Definition:

```
TSS_RESULT Tddli_Close( );
```

Parameters:

None.

Return Value:

TDDL_SUCCESS
TDDL_E_ALREADY_CLOSED
TDDL_E_FAIL

6.5.3 Tddli_Cancel

Start of informative comment:

This function cancels an outstanding TPM command. An application can call this function, in a separate context, to interrupt a TPM command that has not completed. The previous TPM command must be the result of a call to the Tddli_TransmitData function. The TPM device driver must acknowledge this function if it has not returned from a previous TPM command and return TDDL_COMMAND_ABORTED for the call in process.

End of informative comment.

Definition:

```
TSS_RESULT Tddli_Cancel( );
```

Parameters:

None.

Return Value:

TDDL_SUCCESS
TDDL_COMMAND_COMPLETED
TDDL_E_FAIL

6.5.4 Tddli_GetCapability

Start of informative comment:

This function queries the TPM hardware, firmware and device driver attributes such as firmware version, driver version, etc.

End of informative comment.

Definition:

```
TSS_RESULT Tddli_GetCapability
(
    UINT32  CapArea,    // in
    UINT32  SubCap,     // in
    BYTE*   pCapBuf,    // out
    UINT32* pCapBufLen  // in, out
);
```

Parameters:

Type	Name	Description
UINT32	CapArea	Partition of capabilities to be interrogated.
UINT32	SubCap	Subcode of the requested capabilities.
BYTE*	pCapBuf	Pointer to a buffer containing the received attribute data.
UINT32*	pCapBufLen	[in] Size of the receive buffer in bytes [out] Number of written bytes.

Return Values:

TDDL_SUCCESS
TDDL_E_BAD_PARAMETER
TDDL_E_OUTOFMEMORY
TDDL_E_BADTAG
TDDL_E_FAIL

Defined Capability Areas	**Defined Capability Sub-Codes**	**Response**
TCPA_CAP_VERSION	TSS_CAP_PROP_DRV	Returns the version of the TPM device driver. The version is coded in the TPM_VERSION format.
TCPA_CAP_VERSION	TSS_CAP_PROP_FW	Returns the version of the current TPM firmware. The version is coded in the TPM_VERSION format.
TCPA_CAP_VERSION	TSS_CAP_PROP_FW_DATE	Returns the release date of the firmware. The date is coded in three bytes mm/dd/yy (mm=month,

		dd=day, yy=year).
TCPA_CAP_PROPERTY	TCPA_CAP_PROP_MANUFACTURER	Returns the name of the device vendor. The returned data is coded in an ASCII string without the trailing null.
TCPA_CAP_PROPERTY	TSS_CAP_PROP_MODULE_TYPE	Returns the vendor specific designation type of the device. The returned data is coded in an ASCII string without the trailing null.
TCPA_CAP_PROPERTY	TSS_CAP_PROP_GLOBAL_STATE	Returns the global state of the module, (e.g. initialized or personalized).
TCPA_CAP_VENDOR	TCPA_CAP_VENDOR_XXX	Returns the vendor specific capabilities of the TPM.

6.5.5 Tddli_SetCapability

Start of informative comment:

This function sets parameters in the TPM hardware, firmware and device driver attributes. An application can set TPM device driver and operating parameters that may be defined by the TPM vendor. For now, the parameter definitions are vendor-defined.

End of informative comment.

Definition:

```
TSS_RESULT Tddli_SetCapability
(
    UINT32  CapArea,      // in
    UINT32  SubCap,       // in
    BYTE*   pSetCapBuf,   // in
    UINT32  SetCapBufLen  // in
);
```

Parameters:

Type	Name	Description
UINT32	CapArea	Partition of capabilities to be set.
UINT32	SubCap	Subcode of the capabilities to be set.
BYTE*	pSetCapBuf	Pointer to a buffer containing the capability data to be sent.
UINT32	SetCapBufLen	[in] Size of the request buffer in bytes.

Return Values:

TDDL_SUCCESS
TDDL_E_OUTOFMEMORY
TDDL_E_BAD_PARAMETER

TDDL_E_BADTAG
TDDL_E_FAIL

6.5.6 Tddli_GetStatus

Start of informative comment:

This function queries the status the TPM driver and device. An application can determine the health of the TPM subsystem by utilizing this function.

End of informative comment.

Definition:

```
TSS_RESULT Tddli_GetStatus
(
   UINT32  ReqStatusType,   // in
   UINT32* pStatus,         // out
);
```

Parameters:

Type	Name	Description
UINT32	ReqStatusType	Requested type of status information, driver or device.
UINT32*	punStatus	[out] Requested status.

Return Values:

TDDL_SUCCESS
TDDL_E_BAD_PARAMETER
TDDL_E_INSUFFICIENT_BUFFER
TDDL_E_FAIL

Defined Status Type	Defined Response Code-Code	Description
TDDL_DRIVER_STATUS	TDDL _DRIVER_OK	TPM driver is functionaing okay.
TDDL_DRIVER_STATUS	TDDL _DRIVER_FAILED	TPM driver is not functioning.
TDDL_DRIVER_STATUS	TDDL _DRIVER_NOT_OPENED	Device was found, but the corresponding driver could not be opened.
TDDL_DEVICE_STATUS	TDDL _DEVICE_OK	TPM device is functioning okay.
TDDL_DEVICE_STATUS	TDDL_DEVICE_UNRECOVERABLE	TPM device contains an unrecoverable error.
TDDL_DEVICE_STATUS	TDDL _DEVICE_RECOVERABLE	TPM device contains a recoverable error.
TDDL_DEVICE_STATUS	TDDL _DEVICE_NOT_FOUND	TPM device is not found.

6.5.7 Tddli_TransmitData

Start of informative comment:

The function sends a TPM command directly to a TPM device driver, causing the TPM to perform the corresponding operation. This function provides a pass through for the TPM parameter block definitions are defined in the TCG 1.1b Main Specification.

End of informative comment.

Definition:

```
TSS_RESULT Tddli_TransmitData
(
    BYTE*   pTransmitBuf,       // in
    UINT32  TransmitBufLen,     // in
    BYTE*   pRececeiveBuf,      // out
    UINT32* pRececeiveBufLen    // in, out
);
```

Parameters:

Type	Name	Description
BYTE*	pTransmitBuf	Pointer to a buffer containing TPM transmit data.
UINT32	TransmitBufLen	Size of TPM transmit data in bytes.
BYTE*	pRececeiveBuf	Pointer to a buffer containing TPM receive data
UINT32 *	pRececeiveBufLen	[in] Size of TPM receive buffer in bytes [out] Number of written bytes.

Return Value:

TDDL_SUCCESS
TDDL_E_INSUFFICIENT_BUFFER
TDDL_E_IOERROR
TDDL_E_FAIL

6.5.8 Tddli_PowerManagement

Start of informative comment:

Terminology Note: In the following discussion, the term "Higher" indicates a lower Dx value. E.g. D0 is a higher power state than D1.

This function sets and queries the TPM's power states.

Note: There is no corresponding TSPI interface for this function as power management is considered to be the purview of the system, not individual applications.

End of informative comment.

Definition:

```
TSS_RESULT Tddli_SetPowerManagement
(
   BOOLEAN SendSaveStateCommand,      // in
   UINT32* pQuerySetNewTPMPowerState  // in/out
);
```

Parameters:

Type	Name	Description
BOOLEAN	SendSaveStateCommand	TRUE = Instructs driver to send the TPM_SaveState command to the TPM. Other parameters in this function MUST be ignored. Caller MUST set the other "in" parameters to 0. Driver MUST set the "out" parameters to 0. FALSE = Do not send the TPM_SaveState, rather perform actions indicated in the other parameters.

UINT32	QuerySetNewTPMPowerState	**On input, if set to -1,** instructs the driver to return the TPM's current ACPI device state and take no further action. **On input, if *not* set to -1,** instructs the TPM to enter the ACPI device power state per the value of this parameter as follows: 0 = Request to Enter D0 power state 1 = Request to Enter D1 power state 2 = Request to Enter D2 power state 3 = Request to Enter D3 power state **On output:** Value of TPM's current power management state: 0 = TPM current in D0 power state 1 = TPM current in D1 power state 2 = TPM current in D2 power state 3 = TPM current in D3 power state

Comments:

This function MUST return TDDL_E_FAIL if control of power management is set to the driver using the Tddli_PowerManagementControl or if the Tddli_PowerManagementControl returns or would have returned parameter DriverManagesPowerStates Bit 0 = 0.

If the parameter SendSaveStateCommand == FALSE and the TPM cannot maintain its internal state in the requested power state (i.e., without the TPM_SaveState command being issued), this function MUST fail with return value = TDDL_E_BAD_PARAMETER.

If the TPM does not support the requested power state, this function MUST return one of the following two failures:

- TDDL_E_FAIL: Output parameter QuerySetNewTPMPowerState **is not** valid
- TDDL_E_BAD_PARAMETER: Output parameter QuerySetNewTPMPowerState **is** valid

Return Values:

TDDL_SUCCESS
TDDL_E_BAD_PARAMETER
TDDL_E_FAIL

6.5.9 Tddli_PowerManagementControl

Start of informative comment:

This command determines and sets which component, TCS or the Driver, receives and handles the platform's OS power state management signals.

This function should be called only during TSS initialization and is not intended to be used to toggle or change the handler of the platform's OS power state management signals.

During TSS installation and subsequently during TSS initialization the TCS should query the driver. During TSS installation, the state of the returned value may be saved for later TSS initialization without performing the query, thus, making the assumption the driver doesn't change. However if there is a chance the driver may change the handling of the platform's OS power state management signals post TSS installation, upon each TSS initialization, the TCS should query the driver.

Note: There is no corresponding TSPI interface for this function as power management is considered to be the purview of the system, not individual applications.

End of informative comment.

Definition:

```
TSS_RESULT Tddli_SetPowerManagement
(
  UINT32  SetPowerManager,           // in
  UINT32* pDriverManagesPowerStates  // out
);
```

Parameters:

Type	Name	Description
UINT32	SetPowerManager	Queries the driver or sets the driver to expect either the TCS or the driver to handle the TPM's power management signals. NOTE, these are bit maps, not values. **Bit 0 = 0:** No change to handler. Used to query the driver if it can/must handle the platform's OS power management signals. Output parameter DriverManagesPowerStates is valid. All other bits are ignored and MUST be zero. **Bit 0 = 1:** indicate change of platform's OS power management handler per bits 1 – 31. Output parameter DriverManagesPowerStates is not valid and must be ignored. **Bit 1 = 0:** Sets the driver to not handle the platform's OS power management signal (i.e., TPM's power state management will be handle by the TCS using the Tddli_PowerManagement function.) If the driver requires that it must handle the TPM's power state, this function MUST perform no action and return error TDDL_E_BAD_PARAMETER. **Bit 1 = 1:** Sets the driver to handle and manage power management signal (i.e., TPM's power state management will not be handle by the TCS using the Tddli_PowerManagement function.) If the driver cannot handle the power managment signals, this function MUST return TDDL_E_BAD_PARAMETER **Bit 2 - 31:** undefined and MUST be 0.

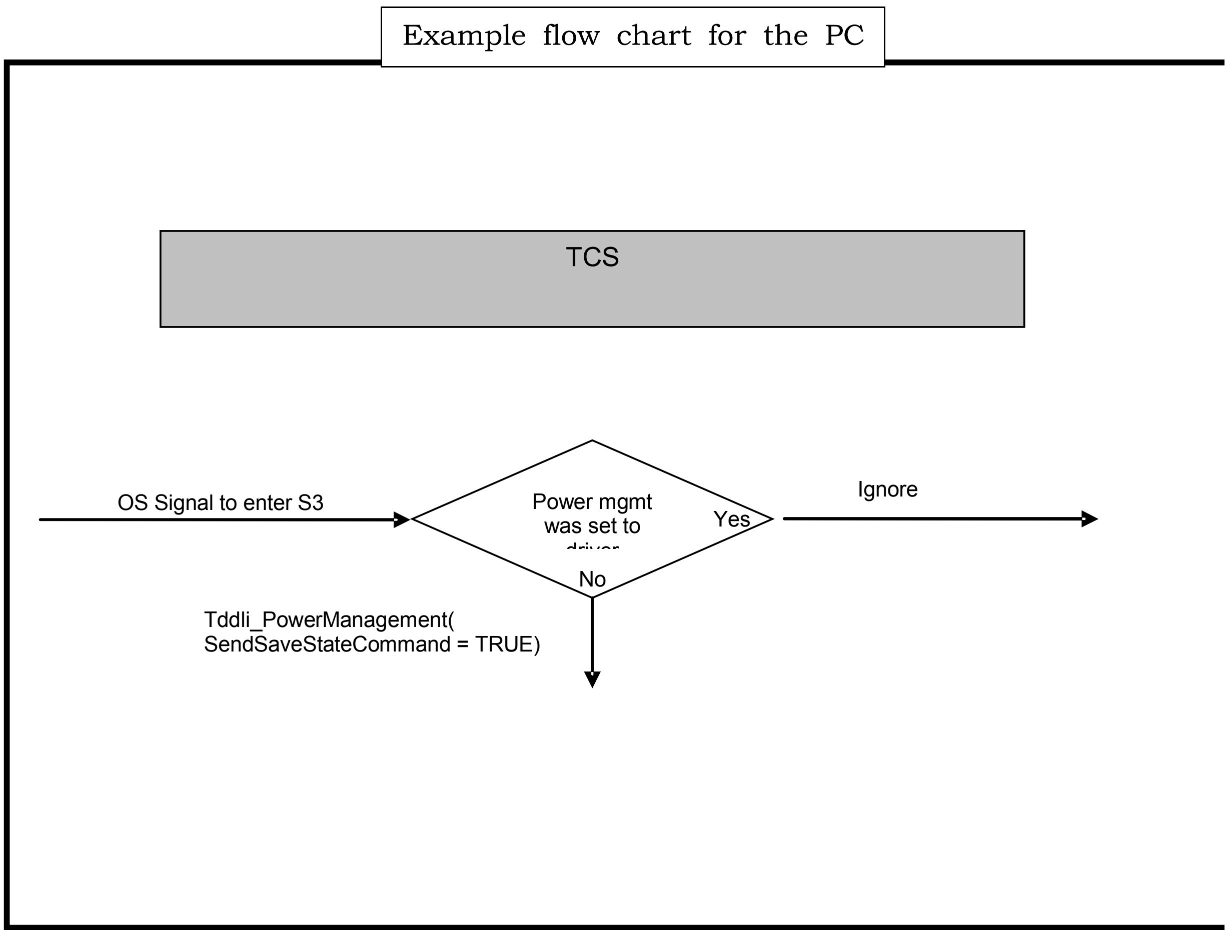
UINT32	DriverManagesPowerStates	**Bit 0 = 0:** driver will handle all platform OS power state management signals and will return an error for calls to Tddli_PowerManagement. **Bit 0 = 1:** driver expects platform's power management signals to be managed by TCS using the Tddli_PowerManagement function. **Bit 1 – 4:** If bit 0 == 1; is a bit map of the supported ACPI device power states that can be contolled by Tddli_PowerManagement QuerySetNewTPMPowerState function. Bit 1 = D0 Bit 2 = D1 Bit 3 = D2 Bit 4 = D3 Bit 5-31 reserved

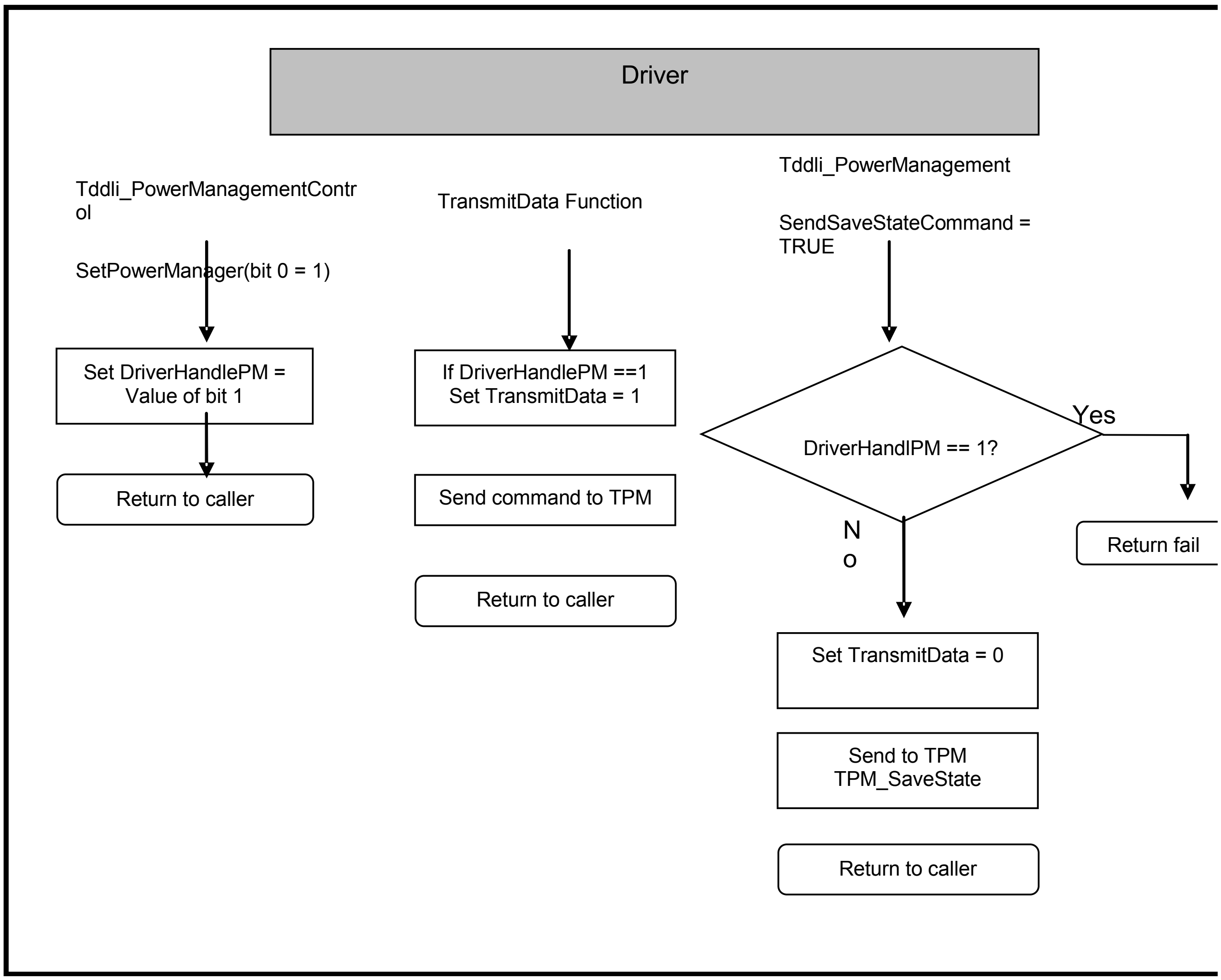
Comments:

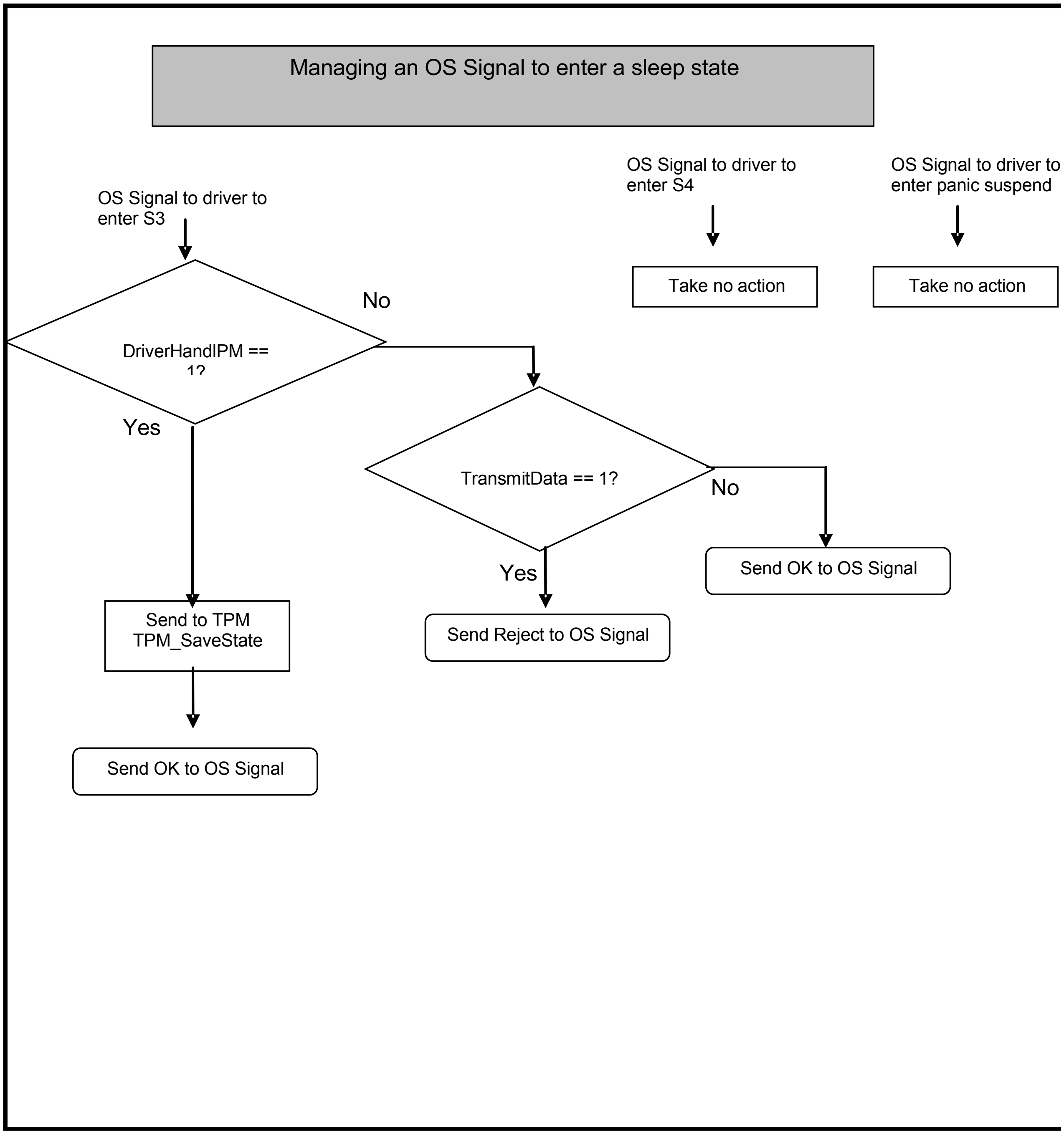
Return Values:

TDDL_SUCCESS
TDDL_E_BAD_PARAMETER
TDDL_E_FAIL

7. Flow Chart


Example flow chart for the PC
TCS
OS Signal to enter S3
Power mgmt
was set to
Yes
Ignore
No
Tddli_PowerManagement(
SendSaveStateCommand = TRUE)


Driver
Tddli_PowerManagementControl
SetPowerManager(bit 0 = 1)
Set DriverHandlePM = Value of bit 1
Return to caller
TransmitData Function
If DriverHandlePM ==1 Set TransmitData = 1
Send command to TPM
Return to caller
Tddli_PowerManagement
SendSaveStateCommand = TRUE
DriverHandlPM == 1?
Yes
Return fail
No
Set TransmitData = 0
Send to TPM TPM_SaveState
Return to caller


Managing an OS Signal to enter a sleep state
OS Signal to driver to enter S3
OS Signal to driver to enter S4
OS Signal to driver to enter panic suspend
Take no action
Take no action
DriverHandlPM == 1?
No
Yes
TransmitData == 1?
No
Yes
Send to TPM TPM_SaveState
Send Reject to OS Signal
Send OK to OS Signal
Send OK to OS Signal

8. Administration Functions

These functions are used to manage resources of the TSS / TPM that need to be shared between / among localities. These include such things as the amount of time a locality can “hog” the TPM for, the number of sessions that each locality is allowed to have, and handling of the counter resources. The counter resource versions of these commands already exist in the Monotonic counter section of this specification.

8.1 Locality Administration

These functions are used for coordinating among different TSSs talking to the same TPM in different localities

8.1.1 Tcsi_Admin_TSS_SessionsPerLocality

Start of informative comment:

This method specifies how many sessions to allocate to each locality. The maximum number of sessions that can be allocated by any call to this method is equal to the number of sessions supported by the TPM minus the number already allocated to other sessions. Since all sessions are initially allocated to locality 0, this method must be used to reduce the number of sessions allocated to locality 0 before any sessions can be allocated to other localities.

The values set using this method are stored in a non-volatile memory region with a reserved index that is writeable only by the TPM owner.

End of informative comment.

C-Definition:

```
TSS_RESULT Tcsi_Admin_TSS_SessionsPerLocality
(
   TCS_CONTEXT_HANDLE  hContext,   // in
   UINT32              ulLocality, // in
   UINT32              ulSessions, // in
   TPM_AUTH *          pOwnerAuth  // in, out
);
```

IDL Definition:

```
[helpstring("method Tcsi_Admin_TSS_SessionsPerLocality")]
TSS_RESULT Tcsi_Admin_TSS_SessionsPerLocality
(
   [in]UINT32               ulLocality,  // in
   [in]UINT32               ulSessions,  // in
   [in, out]TPM_AUTH *      pOwnerAuth   // in, out
);
```

Parameters:

Type	Label	Description
UINT32	ulLocality	Index of locality for which to allocate sessions.
UINT32	ulSessions	Number of sessions to allocate to the specified locality.
TPM_AUTH*	pOwnerAuth	TPM owner authorization.

Comment

Initially all sessions are allocated to locality 0. If ulSessions exceeds the number of currently unallocated sessions or ulLocality exceeds the number of localities, TSS_E_BAD_PARAMETER is returned.

The values set by this command are stored in an area of non-volatile memory allocated by the TSS using the reserved index TPM_NV_INDEX_Sessions. This memory region is created with permissions set to TPM_NV_PER_OWNERWRITE and contains one UINT32 value per locality.

8.1.2 Tcsi_Admin_TSS_MaxTimePerLocality

This command sets the maximum time that a locality will have to wait to get the TPM after it is requested.

9. References

- M. Bellare, J. A. Garay, T. Rabin, Fast batch verification for modular exponentialtion and digital signatures. In K. Nyberg, editor, Advances in Cryptology – EUROCRYPT '98, volume 1403 of LNCS, pages 236-250. Slpringer Verlag, 1998
- E. Brickell, J. Camenisch, L. Chen, *Direct Anonymous Attestation*, 2004.
- J. Camenisch, *Better Privacy for Trusted Computing Platforms*, draft 2004.
- J. Camenisch, V. Shoup, Practical Verifiable Encryption and Discrete Logarithms, Crypto 2003
- Swox AB, GMP (GNU Multiple Precision Arithmetic) Library, http://www.swox.com/gmp/
- Trusted Computing Group, TPM Specifications v1.2, October 2003.
- Trusted Computing Gropu, TPM Specifications v 1.1b
- Victor Shoup, NTL, A Library for doing Number Theory version 5.3.2, http://shoup.net/ntl/

10. APPENDIX 2. TSP Function Authorization Usage

TSS Function	Object	Authorization required	Comment
Tspi_TPM_GetStatus	hTPM	Yes	Owner authorization required for all TPM status queries
Tspi_Context_UnregisterKey	phkey	No	This is an output of the function. Key usage authorization is NOT required to unregister the key from the TSS key store
Tspi_SetAttribUint32	hObject	varies	
Tspi_GetAttribUint32	hObject	varies	
Tspi_SetAttribData	hObject	varies	
Tspi_GetAttribData	hObject	varies	
Tspi_Context_Create	hContext	n	
Tspi_Context_Close	hContext	n	
Tspi_Context_Connect	hContext	n	
Tspi_Context_FreeMemory	hContext	n	
Tspi_Context_GetDefaultPolicy	hContext	n	
Tspi_Context_CreateObject	hContext	varies	TPM objects may require authorization
Tspi_Context_CloseObject	hContext	n	
Tspi_Context_GetCapability	hContext	n	
Tspi_Context_GetTPMObject	hContext	n	
Tspi_Context_SetTransEncryptionKey	hContext	n	
Tspi_Context_CloseSignTransport	hContext	y	Parent Key auth required
Tspi_Context_LoadKeyByBlob	hContext	y	Parent Key auth required
Tspi_Context_LoadKeyByUUID	hContext	y	Parent Key auth required
Tspi_Context_RegisterKey	hContext	n	

Tspi_Context_UnregisterKey	hContext		
Tspi_Context_GetKeyByUUID	hContext	n	
Tspi_Context_GetKeyByPublicInfo	hContext	n	
Tspi_Context_GetRegisteredKeysByUUID	hContext	n	
Tspi_Context_GetRegisteredKeysByUUID2	hContext	n	
Tspi_SetOperator_Auth	hTPM?	n	
Tspi_Policy_SetSecret	hPolicy	n	
Tspi_Policy_FlushSecret	hPolicy	n	
Tspi_Policy_AssignToObject	hPolicy	n	
Tspi_TPM_CreateEndorsementKey	hTPM	y	
Tspi_TPM_GetPubEndorsementKey	hTPM	y	
Tspi_TPM_CollateIdentityRequest	hTPM	n	
Tspi_TPM_ActivateIdentity	hTPM	y	
Tspi_TPM_CreateRevocableEndorsementKey	hTPM	y	
Tspi_TPM_RevokeEndorsementKey	hTPM	y	
Tspi_TPM_TakeOwnership	hTPM	n	
Tspi_TPM_ClearOwner	hTPM	y	
Tspi_TPM_CreateMaintenanceArchive	hTPM	y	
Tspi_TPM_KillMaintenanceFeature	hTPM	y	
Tspi_TPM_LoadMaintenancePubKey	hTPM	y	
Tspi_TPM_CheckMaintenancePolicy	hTPM	n	
Tspi_TPM_SetStatus	hTPM	varies	
Tspi_TPM_GetStatus	hTPM	varies	
Tspi_TPM_GetCapability	hTPM	n	
Tspi_TPM_SelfTestFull	hTPM	n	
Tspi_TPM_CertifySelfTest	hTPM	y	
Tspi_TPM_GetTestResult	hTPM	n	
Tspi_TPM_GetRandom	hTPM	n	
Tspi_TPM_StirRandom	hTPM	n	
Tspi_TPM_GetEvent	hTPM	n	
Tspi_TPM_GetEventLog	hTPM	n	
Tspi_TPM_Quote	hTPM	y	
Tspi_TPM_PcrExtend	hTPM	n	

Tspi_TPM_PcrRead	hTPM	n	
Tspi_PCRComposite_SelectPcrIndex	hPcrComposite	n	
Tspi_PCRComposite_SetPcrValue	hPcrComposite	n	
Tspi_PCRComposite_GetPcrValue	hPcrComposite	n	
Tspi_TPM_PcrReset	hTPM	n	
Tspi_Data_Seal	hEncData	y	
Tspi_Data_SealX	hEncData	y	
Tspi_TPM_Quote2	hTPM	y	
Tspi_PCRComposite_SetPcrLocality	hPcrComposite	n	
Tspi_PCRComposite_GetPcrLocality	hPcrComposite	n	
Tspi_PCRComposite_GetCompositeHash	hPcrComposite	n	
Tspi_PCRComposite_SelectPcrIndexEx	hPcrComposite	n	
Tspi_ChangeAuth	hObject	y	
Tspi_GetPolicyObject	hObject	n	
Tspi_Key_LoadKey	hKey	y	
Tspi_Key_UnloadKey	hKey	n	
Tspi_Key_GetPubKey	hKey	varies	
Tspi_Key_CertifyKey	hKey	y	
Tspi_Key_CreateKey	hKey	n	
Tspi_Key_WrapKey	hKey	y	
Tspi_TPM_AuthorizeMigrationTicket	hTPM	y	
Tspi_Key_CreateMigrationBlob	hKey	y	
Tspi_Key_ConvertMigrationBlob	hKey	y	
Tspi_ChangeAuthAsym	HObject	y	
Tspi_TPM_SetRestrictions	hTPM	y	
Tspi_TPM_CMKApproveMA	hTPM	y	
Tspi_TPM_CMKCreateTicket	hTPM	y	
Tspi_Key_MigrateKey	hKey	y	
Tspi_Key_CMKCreateBlob	hKey	y	
Tspi_Key_CMKConvertMigration	hKey	y	
Tspi_Hash_Sign	hHash	y	
Tspi_Hash_VerifySignature	hHash	n	

Tspi_Hash_SetHashValue	hHash	n	
Tspi_Hash_GetHashValue	hHash	n	
Tspi_Hash_UpdateHashValue	hHash	n	
Tspi_Data_Bind	hEncData	n	
Tspi_Data_Unbind	hEncData	y	
Tspi_Data_Unseal	hEncData	y	
Tspi_TPM_ReadCounter	hTPM	n	
Tspi_TPM_ReadCurrentTicks	hTPM	n	
Tspi_Hash_TickStampBlob	hHash	y	
Tspi_TPM_DirWrite	hTPM	n	
Tspi_TPM_DirRead	hTPM	n	
Tspi_NV_DefineSpace	hNVStore	n	
Tspi_NV_ReleaseSpace	hNVStore	n	
Tspi_NV_WriteValue	hNVStore	varies	
Tspi_NV_ReadValue	hNVStore	varies	
Tspi_TPM_Delegate_AddFamily	hTPM	y	
Tspi_TPM_Delegate_InvalidateFamily	hTPM	y	
Tspi_TPM_Delegate_CreateDelegation	hObject	y	
Tspi_TPM_Delegate_CacheOwnerDelegation	hTPM	y	
Tspi_TPM_Delegate_UpdateVerificationCount	hTPM	y	
Tspi_TPM_Delegate_VerifyDelegation	hPolicy	y	
Tspi_TPM_Delegate_ReadTables	hContext	n	
Tspi_TPM_DAA_JoinInit	hDaa	y	
Tspi_TPM_DAA_JoinCreateDaaPubKey	hDaa	y	
Tspi_TPM_DAA_JoinStoreCredential	hDaa	y	
Tspi_TPM_DAA_Sign	hDaa	y	
Tspi_TPM_DAA_IssuerKeyVerify	hDaa	y	
Tspi_TPM_DAA_Issuer_GenerateKey	hDaa	y	
Tspi_TPM_DAA_Issuer_InitCredential	hDaa	y	
Tspi_TPM_DAA_Issuer_IssueCredential	hDaa	y	
Tspi_TPM_DAA_VerifyInit	hDaa	y	
Tspi_TPM_DAA_VerifySignature	hDaa	n	
Tspi_TPM_DAA_ARA_GenerateKey	hDaa	y	
Tspi_TPM_DAA_ARA_RevokeAnonymity	hDaa	y	
