# TCG Software Stack Feature API

**Family "2.0"**

**Level 00 Revision .12**

**November 7, 2014**

Committee Draft

**Contact:** admin@trustedcomputinggroup.org
Please provide public review comments by Tuesday, January 6, 2015.

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

**TCG Public Review**

TCG

## Disclaimers, Notices, and License Terms

## Contents

# 1 Introduction:

The TSS 2.0 Feature API is meant to be a very high level API, aimed at having commands in it that will allow 80% of the programmers who write a program using the TPM to find everything they want in the specification. The remaining 20% of programmers will have to supplement this set of APIs with the Extended System API (ESAPI) or System API (SAPI).

This specification is meant to making programming with the TPM as simple as possible – but no simpler. The cognitive load for a new programmer using this API is meant to be kept as low as possible. Because of this, a number of decisions have been made including:

- A Profile is used by a programmer that makes many of the complicated decisions for the programmer. It decides such things as the default algorithm sets that are used when creating keys, where they are stored and found.
- Key template names have been created for the dozen or so keys that are expected to be used by most programmers
- Key names will be based on path descriptors, much as files are today.
- All entities used by the feature API will be authenticated by use of a policy. (The policy may point to an authorization done using the authorization data, however.) This means that no entity will be created with a NULL policy. It probably also means that bits will be set to disable use of the authorization data in objects.
- All authorizations done using authorization data will use salted HMAC sessions. Decrypt and encrypt sessions will also be used.
- Policy instances and forms are described in an XML representation which may be found in the Policy XML format document.
- PCR log files will be in the format described by the PC Client Specification
- Commands syntax looks like Tss2_<EntityName>_<Command> :
  - Tss2_Key_Sign
  - Tss2_Nv_Write
  - Tss2_Entity_ChangeAuth
  - Tss2_TPM_GetRandom
- The Feature API doesn't include two other things which are necessary to get it to work, which are expected to be needed, namely
  - A utility used to create a policy in the correct XML format

    For example:

    ```
    <PolicyAce type="PolicySigned">
       <Name>Company SmartCard</Name>
       <Driver>MySmartCard</Driver>
       <DriverInfo>DN=CompandSmartcard.com</DriverInfo>
       <etc>...</etc>
    </PolicyAce>
    ```
  - A device driver and library which instantiates the appropriate callback function

    In this case Tss2_PolicySignatureCallback
  - A means to register the driver and library itself, so that it is available for use
  - Callback functions will be used to obtain decisions from the user and interfaces related to policy commands that require input. The FAPI will read the policy associated with an entity when it is used, create a Policy session to satisfy it, and walk through the command necessary to satisfy the command. It will use the callback functions to determine
    - Which branches of an OR (or PolicyAuthorize) policy to follow
    - How to obtain passwords or signatures necessary to satisfy the policies.

- The default TPM this will work with is assumed to be the local one, but another one can be specified when a context is created.
- As much as possible, functions and parameters will be kept to a minimum. Sometimes these mandates are in conflict. When in conflict, the solution believed to be easiest for the programmer is chosen.

Thanks are given to the kind members of the group who gave of their time in order to make this specification come together. In alphabetical order, they are: Will Arthur, David Challener, Michael Cox, Paul England, Andreas Fuchs, Ken Goldman, James Nguyen, Lee Wilson.

# 2 Structures

To reduce the cognitive load on users, we are trying to restrict ourselves in the number of new structures a programmer will have to learn. There are a few that will make things easier rather than harder, though, both for the FAPI creator and for the programmer.

## 2.1 TSS2_CONTEXT

This data type describes an opaque structure that can be used by implementations to store state information. It is allocated during Tss2_Context_Initialize() and freed during Tss2_Context_Finalize(). Note: An application must ensure that there is only one FAPI call that uses a context at any given time. If an application desires to make concurrent function calls to the Feature-API, it needs to initialize multiple contexts. Even though most functions are globally atomic and therefore stateless, a few functions do store state information inside the context object, e.g. the policy callback setters.

## 2.2 TSS2_SIZED_BUFFER

One fundamental type of structure arising often in functions is a sized buffer. In order to reduce the number of parameters we have to write for a command, we have decided to call such a buffer a TSS2_SIZED_BUFFER

```
typedef struct { size_t    size;
                 uint8_t  *buffer;
                 } TSS2_SIZED_BUFFER;
```

NOTE: This is different from the TPM TPM_2B structure in that the buffer is a pointer, not a fixed size array.

## 2.3 Key Structure

Example of what the implementation might put in the key structure. This is never used in the specification, but is rather a help for the implementer.

```
typedef struct {
   char                 *keyPath;          /* key path */
   TSS2_SIZED_BUFFER    *private;          /* the private portion of the object */
   TSS2_SIZED_BUFFER    *public;           /* the wrapped public portion of the created object */
   TSS2_SIZED_BUFFER    *name;             /* TPM library Name */
   char                 *policyInstance;   /* Key's policy */
   TSS2_SIZED_BUFFER    *certificate;      /* Key's certificate (if any) */
   TSS2_SIZED_BUFFER    *description;      /* Human readable description of key */
} TSS2_KEY;
```

# 3 Definitions

## 3.1 Policies

### 3.1.1 Standard Policies and Authorizations (for values, see header file)

These policies don't have a path associated with them. They are represented by a fixed set of bytes, which only depends on the hash algorithm. As such, they have a fixed name which can be used when creating an entity, without first creating a policy instance. One authorization, TSS2_AUTHNULL, is defined to provide an easy way to refer to the trivial authorization, usually used (e.g.) by root keys.

| | |
|---|---|
| **TSS2_ AUTHNULL** | refers to the NULL password, in a TSS2_SIZED_BUFFER which can be trivially satisfied. |
| **TSS2_POLICY_NULL** | refers to the NULL policy (empty buffer) which can never be satisfied |
| **TSS2_POLICY_AUTHVALUE** | refers to a policy that merely points to the object's authorization data |
| **TSS2_POLICY_SECRET_EH** | refers to a policy that points to the endorsement hierarchy's authorization data |
| **TSS2_POLICY_SECRET_SH** | refers to a policy that points to the storage hierarchy's authorization data |
| **TSS2_POLICY_SECRET_DA** | refers to a policy that points to the dictionary attack handle's authorization data |
| **TSS2_POLICY_TRIVIAL** | refers to a policy with length equal to the hash algorithm output size, filled with zeros |

## 3.2 PCRs

When PCRs are called, if the TPM only has one bank of hash algorithms, that one is chosen (This is likely to be the case 100% of the time.) If more than one hash algorithm is available, and one corresponds to the default hash algorithm, that one is chosen. Otherwise an error is returned.

PCRs are passed in as a monotonic array terminated with a -1.

Example 1:

| | |
|---|---|
| Selecting PCRs 1,3 and 5: | {1,3,5,-1} |
| Selecting PCRs 0,1,2,3,4,5: | {0,1,2,3,4,5,-1} |
| Selecting PCRs 0-23: | {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,-1} |

## 3.3 Key Types

### 3.3.1 Named types of keys: (for values, see header file)

- **TSS2_ASYM_STORAGE_KEY**
  This is an asymmetric key used to store other keys/data
- **TSS2_SRK**
  This is a primary key at the top of a tree. It is never directly referenced by a path. The default SRK will correspond to the default Profile algorithm profile and be persistent. If any other algorithm profile is chosen, the FAPI will have to generate the appropriate primary SRK, which need not be primary.
- **TSS2_EK**
  This is an endorsement key which has a certificate used to prove that it and other keys belong to a genuine TPM. It resides in the endorsement hierarchy, and is described by a template pointed to in the Profile.
- **TSS2_ASYM_RESTRICTED_SIGNING_KEY**
  This is like the AIK of 1.2, but has further capabilities of being able to sign any data that does not claim to come from the TPM.
- **TSS2_HMAC_KEY**
  This is is for an HMAC key. It is NOT a restricted key and cannot be used to attest to data.

### 3.3.2 NV types

Hybrid NV is not supported in this revision of the FAPI specification.

| | |
|---|---|
| **TSS2_NV_MEMORY** | (normal NV memory) |
| **TSS2_NV_BITFIELD** | (a 64 bit bitfield) |
| **TSS2_NV_COUNTER** | (a 64bit counter) |
| **TSS2_NV_PCR** | (a NV_PCR using template hash algorithm) |
| **TSS2_NV_TEMP_READ_DISABLE** | (This memory object can be made not readable for a boot cycle) |

## 3.4 Standard Default Profiles

These profiles are referenced by the FAPI and used to determine the algorithms and settings used when creating entities. One of these is selected by the user for the FAPI to use as default. If the programmer doesn't want to use the default, he can specify a different one of these when creating or using entities.The programmer can create a Profile file different from the specified ones and import it into the FAPI library by placing it at the root of the path described in Section 4.

Profile Names (used in paths, see Section 4)

### 3.4.1 P_RSASHA1

This Profile uses RSA-2048, SHA-1, and AES128 for its algorithms.

### 3.4.2 P_RSASHA2

This Profile uses RSA-2048, SHA-256, and AES128 for its algorithms.

### 3.4.3 P_ECCP256

This Profile uses ECC P256, SHA-256, and AES128 for its algorithms.

### 3.4.4 P_ECCP384

This Profile uses ECC P384, SHA-384, and AES256for its algorithms.

### 3.4.5 P_ECCP521

This Profile uses ECC P521, SHA-512, and AES256 for its algorithms.

### 3.4.6 Profile Features

When defining or querying a FAPI (see Section 5.3.2) to determine characteristics of its default profile, the following are the vocabulary terms used.

| | |
|---|---|
| **TSS2_ASYM_ALG** | |
| **TSS2_SYM_ALG** | |
| **TSS2_HASH_ALG** | |
| **TSS2_ASYM_ALG_SIZE** | number of bits in the asymmetric algorithm |
| **TSS2_SYM_ALG_SIZE** | number of bits in the symmetric algorithm |
| **TSS2_HASH_ALG_SIZE** | number of bits in the hash algorithm |

## 3.5 Example Algorithm Profiles in XML

The algorithm profile includes a number of things for use as defaults when creating entities in the TPM. The main thing it includes is a list of algorithms that are used for creation of all the key types, and where their default storage is. There will be algorithm profiles of various sorts shipped with a Feature API which represent platform specifications of supported algorithms, however these three will always be included. Programs can ship with their own custom algorithm profiles as well, with the suggestion that end users make use of them. Spaces after > or before < are ignored.

### 3.5.1 Algorithm profile for RSASHA1: P_RSASHA1

```
<TSS2_Defaults>
 <Title> RSA2048 SHA1 AES128</Title>
 < ASYM_ALG> RSA</ASYM_ALG>
 <ASYM_ALG_SIZE> 2048</ASYM_ALG_SIZE>
 <ASYM_ALG_SIGNING_SCHEME> RSASSA </ASYM_ALG_SIGNING_SCHEME>
 <HASH_ALG> SHA1</HASH_ALG>
 <HASH_ALG_SIZE> 20 </HASH_ALG_SIZE>
 <SYM_ALG> AES </SYM_ALG>
 <SYM_ALG_SIZE> 128 </SYM_ALG_SIZE>
 <SYM_ALG_ENCRYPTION_SCHEME> CFB </SYM_ALG_ENCRYPTION_SCHEME>
 <SRK>
    <Description>  SRK </Description>
    <Persistent> YES </Persistent>
    <Handle>0x400000001</Handle>
    <Standard_template>SRK/template0</Standard_template>
 </SRK>
 <EK>
    <Description> EK </Description>
    <Persistent> No </Persistent>
    <Standard_template> EK/template0 </Standard_template>
     <CertIndex> 0x01c00000</CertIndex>
 </EK>
</TSS2_Defaults>
```

### 3.5.2 Algorithm profile for RSASHA2: P_RSASHA2

```
<TSS2_Defaults>
  <Title> RSA2048 SHA256 AES128</Title>
  <ASYM_ALG> RSA</ASYM_ALG>
  <ASYM_ALG_SIZE> 2048</ASYM_ALG_SIZE>
  <ASYM_ALG_SIGNING_SCHEME> ECDHA </ASYM_ALG_SIGNING_SCHEME>
  <HASH_ALG> SHA256</HASH_ALG>
  <HASH_ALG_SIZE> 32 </HASH_ALG_SIZE>
  <SYM_ALG> AES </SYM_ALG>
  <SYM_ALG_SIZE> 128 </SYM_ALG_SIZE>
  <SYM_ALG_ENCRYPTION_SCHEME> CFB </SYM_ALG_ENCRYPTION_SCHEME>
  <SRK>
      <Description>  SRK </Description>
      <Persistent> YES </Persistent>
      <Handle>400000001</Handle>
      <Standard_template>SRK/template0</Standard_template>
  </SRK>
  <EK>
      <Description> EK </Description>
      <Persistent> No </Persistent>
      <Standard_template> EK/template0 </Standard_template>
      <CertIndex> 0x01c00010</CertIndex>
  </EK>
</TSS2_Defaults>
```

### 3.5.3 Algorithm profile for ECCP256: P_ECCP256

```
<TSS2_Defaults>
  <Title> ECC using P256,  SHA256, and  AES128 </Title>
  <ASYM_ALG> ECC</ASYM_ALG>
  <ASYM_ALG_SIZE> 256</ASYM_ALG_SIZE>
  <ASYM_ALG_SIGNING_SCHEME> ECDHA </ASYM_ALG_SIGNING_SCHEME>
  <HASH_ALG> SHA256</HASH_ALG>
  <HASH_ALG_SIZE> 32 </HASH_ALG_SIZE>
  <SYM_ALG> AES </SYM_ALG>
  <SYM_ALG_SIZE> 128 </SYM_ALG_SIZE>
  <SYM_ALG_ENCRYPTION_SCHEME> CFB </SYM_ALG_ENCRYPTION_SCHEME>
  <SRK>
      <Description>  SRK </Description>
      <Persistent> YES </Persistent>
      <Handle>400000001</Handle>
      <Standard_template>SRK/template0</Standard_template>
  </SRK>
  <EK>
      <Description> EK </Description>
      <Persistent> No </Persistent>
      <Standard_template> EK/template0 </Standard_template>
      <CertIndex> 0x01c00020 </CertIndex>
  </EK>
</TSS2_Defaults>
```

# 4 Path Descriptors

Path descriptor are used to point to keys, NV, policy, etc. A policy looks like this:

**P_RSA2048SHA1/HS/SNK/Enterprise/PettyCashKey**

The **second term**, called the hierarchy, tells which hierarchy the items is stored under

The **third term** called a Defined Object ancestor defines the type of key

The **last term** references the specific key.

Path descriptors tell the FAPI where to find and store information about entities, including those which are stored as blobs (keys), NV indexes, hierarchies, and policy forms and policies. Path descriptors are NOT case sensitive.

## 4.1 The Profile term:

The Profile (e.g. P_RSA2048SHA1/) is the first term in a path, but may be implied. They always start with P_. If a Profile is not specified in a path, then the default Profile will be assumed (see 4.6 Implicit Paths). All keys under a particular profile will have the same algorithm set, as determined by the Profile.

There are some standard Profile names (referenced in part 3.4) which can be assumed by the programmer to exist. However the programmer can create new Profiles and install them as a root of a path addressed by the FAPI. Once installed, they can be referenced like any other Profile name.

For the first version of the specification, it is assumed that all keys below primary keys are stored in the Storage hierarchy.

## 4.2 Hierarchy terms

There are 4 possible hierarchy terms: HS (the storage hierarchy), HE (the endorsement hierarchy), HP (the platform hierarchy) and HN (the Null hierarchy). If the hierarchy term is not listed, then the storage hierarchy is assumed (See 4.6 Implicit Profiles).

## 4.3 Defined Object Ancestors

These keys are standard ancestors of various key hierarchies that should always be present in a system. By pointing to one of these ancestors when creating a key, you implicitly select if the key is duplicable or not, and if the key is available to the system, or only to the user. That is, a key ancestor tells whether a key is one of four possible kinds: A system key that is non-duplicable , a system key that is duplicable, a user key that is non-duplicable or a user key that is duplicable. One special key, the EK, has a symbol of its own. There may be many users on a system. If so, there will be many user object ancestors, but they will all be referenced by the same name. It is up to the operating system to make certain that when a user refers to UNK or UDK, the FAPI knows the correct tree for that user.

If the object is not a key, then the object ancestor will either by NV (for a non-volatile index) or Policy (for a policy instance).

### 4.3.1 HS/SNK

System NonDuplicatable Key: This refers to a non-duplicable key in the storage hierarchy. It will be stored under the SRK. It is the parent storage key of all non-primary keys under a Profile, and is the direct parent of all system-wide non-primary duplicable keys. ***If a key is created by the user under this key, it is automatically assumed to be non-duplicable.***

### 4.3.2 HS/SDK

System Duplicable Key: This refers to a duplicable storage key. (The TSS creates this key under the SNK, but the SNK is not referenced in its path). It is the parent of all system-wide duplicable keys. In order to move all system-wide duplicable keys to a new system, only this key needs to be duplicated. Its children and descendants can be merely copied to the new system***. If a key is created under this key, it is automatically assumed to be duplicable. Fixedparent is NOT set.***

### 4.3.3 HS/UNK

The User Non-Duplicable Key This refers to a non-duplicable restricted decryption key under the SRK, which is assigned to the user. The OS is responsible for only allowing the user to have access to these keys. All user non-duplicable keys are placed under this key***. If a key is created under this key, it is automatically assumed to be non-duplicable.***

### 4.3.4 HS/UDK

The User Duplicable Key: This refers to a duplicable restricted decryption key under the HS/UNK, which is assigned to a user. The OS is responsible for only allowing the user to have access to these keys. If a user moves to another system, only this key will need to be duplicated to the new system. Its children and descendants need only be copied to the new system. ***If a key is created under this key, it is automatically assumed to be duplicable. Fixedparent is not set.***

### 4.3.5 HE/EK

This refers to a restricted decryption key under the endorsement hierarchy. It may also be refered to as just the EK.

### 4.3.6 NV

This means that the entity referred to is an Non Volatile memory location

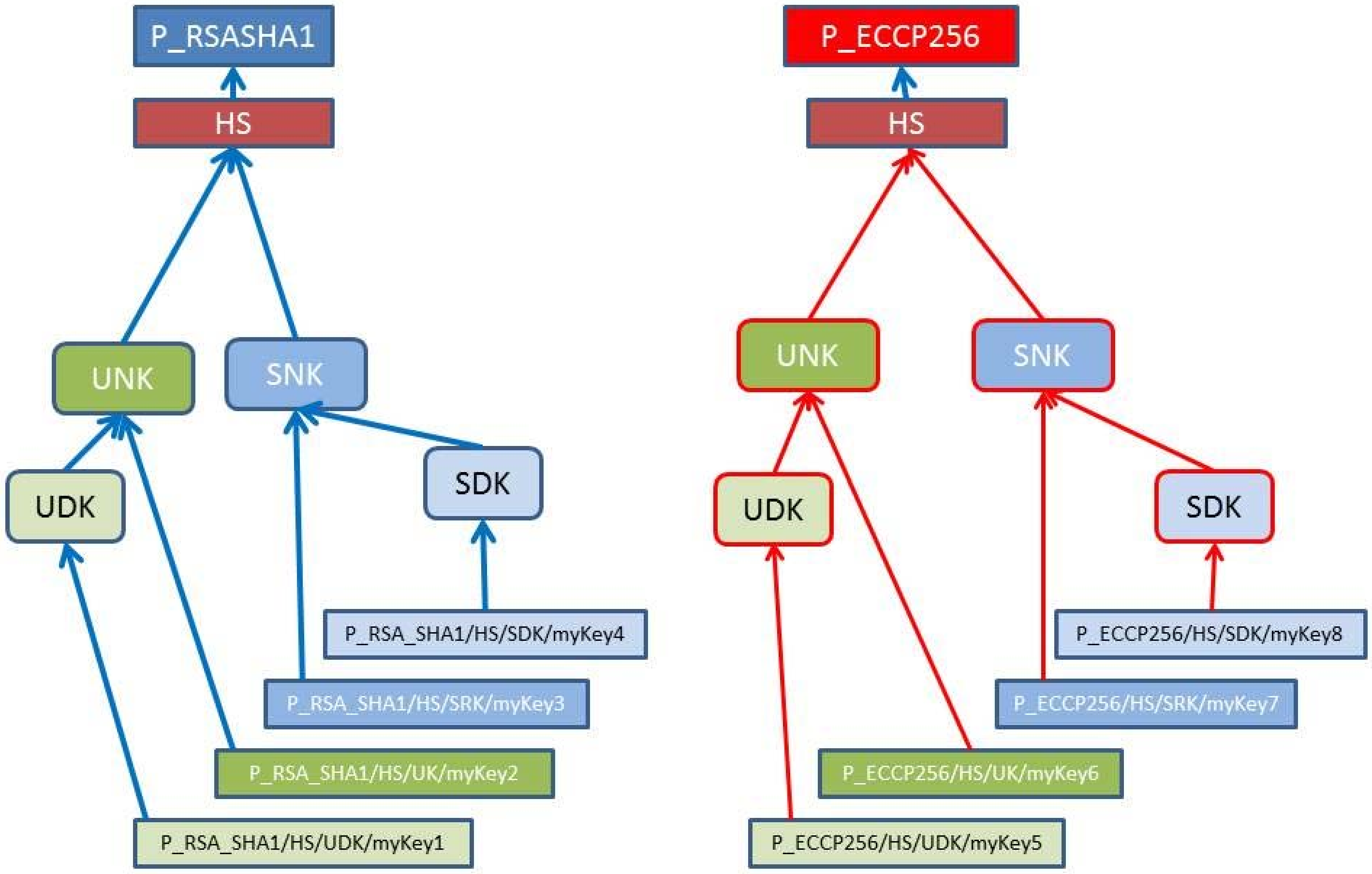
### 4.3.7 Policy

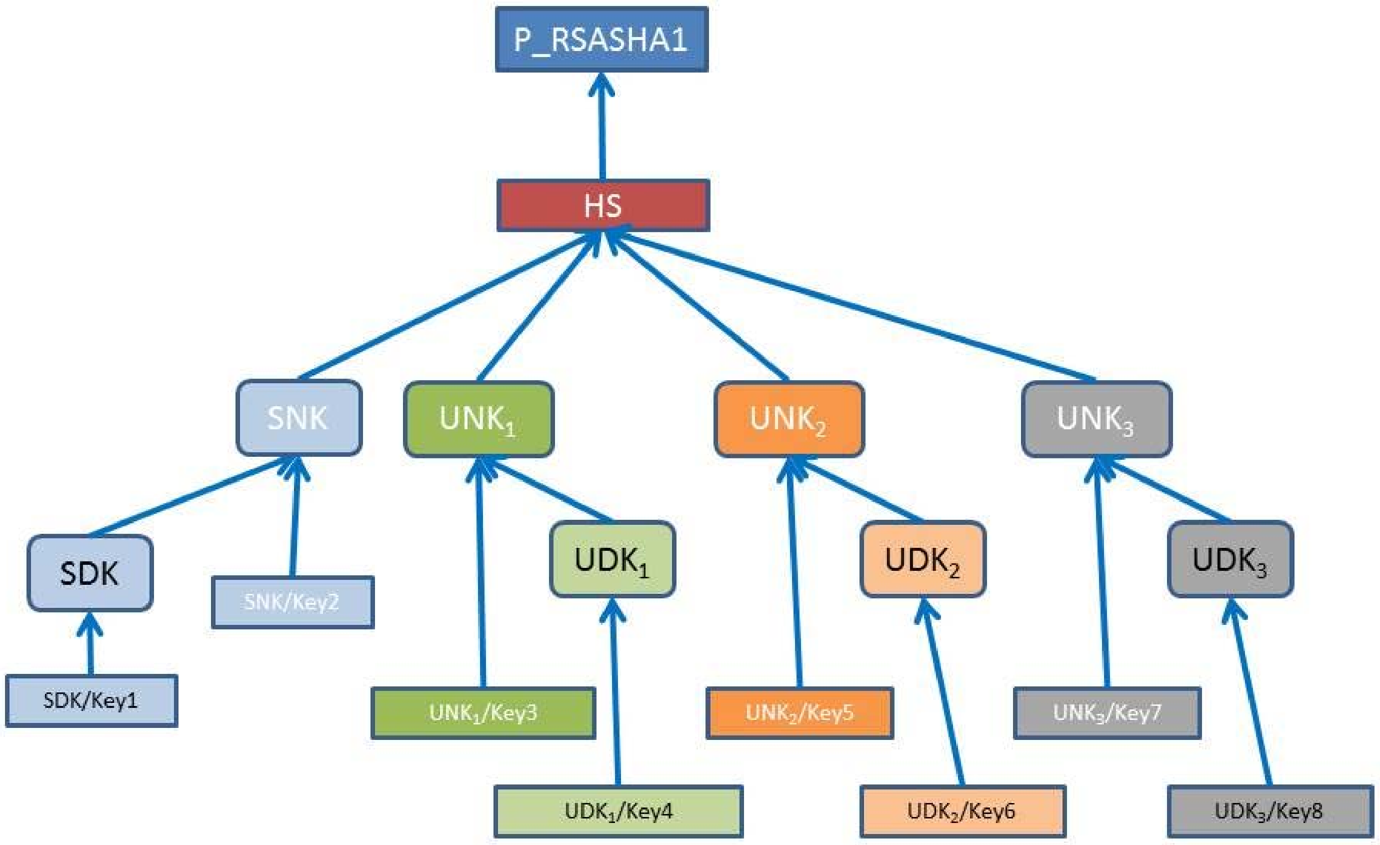
This means that the entity referred to is a policy

### 4.4 Example explicit paths

Explicit paths are used when referring to a key that was not created using the current Profile

- Profile /HS/SNK/keyname
- Profile /HS/SDK/keyname
- Profile /HS/UNK/keyname
- Profile /HS/UDK/keyname
- Profile /HE/EK
- Profile/Policy/
- Profile/HS/NV
- Profile/HE
- Profile/HS
- Profile/HP
- Profile/HN




P_RSASHA1
HS
SNK
UNK1
UNK2
UNK3
SDK
SNK/Key2
UDK1
UDK2
UDK3
SDK/Key1
UNK1/Key3
UNK2/Key5
UNK3/Key7
UDK1/Key4
UDK2/Key6
UDK3/Key8

### 4.5 Policies

Policies are stored under a Policy

#### 4.5.1 Examples

- Implicit: Policy/myLittlePolicy
- Explicit: P_ECCP256/HS/Policy/myLittlePolicy

### 4.6 Example implicit paths

Implicit paths are used when referring to a key that was created using the current Profile. If a hierarchy is not defined, then it is assumed to be HS except that EK is always assumed to be in the HE. (Here P_DefaultPolicy is whatever the user has selected for his default policy, which the FAPI will know. Examples are P_RSASHA256 or P_ECC256.)

| | | |
|---|---|---|
| • SNK/keyname | = HS/SNK/keyname | = P_DefaultPolicy/HS/SNK/keyname |
| • UDK/keyname | = HS/UDK/keyname | = P_DefaultPolicy/HS/UDK/keyname |
| • UNK/keyname | = HS/UNK/keyname | = P_DefaultPolicy/HS/UNK/keyname |
| • UDK/keyname | = HS/UDK/keyname | = P_DefaultPolicy/HS/UDK/keyname |
| • EK | = HE/EK | = P_DefaultPolicy/HE/EK |
| • Policy/myPolicy | | = P_DefaultPolicy/Policy/myPolicy |
| • NV/myNV_forStoringHashes | | = P_DefaultPolicy/NV/myNV_forStoringHashes |
| • HP/NV/EK_Certificate | | = P_DefaultPolicy/HP/NV/EK_Certificate |

### 4.7 Format

#### 4.7.1 Keys

##### 4.7.1.1 Systemwide non-duplicable keys

System-wide non-duplicable keys are referred to starting either with /SNK or Profile/SNK. An example might be a key used by a network to both provide access to a network, but also to verify the deviceID. As such, it is locked to a particular platform, so that the network knows what device is trying to connect.

###### 4.7.1.1.1 Examples

- Implicit: SNK/VPN1/HomeNetwork
- Explicit: P_ECCP256/HS/SNK/VPN2/WorkNetwork

##### 4.7.1.2 System-wide duplicable keys

Some keys may be tied to a physical location but not a particular system, and may potentially need to be moved to a new system when the hardware is upgraded without disruption caused by needing to create a new key. As such, they would be created and then later duplicated to a different system.

###### 4.7.1.2.1 Examples

- Implicit: SDK/Service1Key
- Explicit: P_RSASHA1/HS/SDK/Service1Key

#### 4.7.1.3 User non-duplicable keys

Users may also have specific keys that they use to access a service that are not meant to be shared between different users, and which are meant to be locked to a particular device.

##### 4.7.1.3.1 Examples

All of these refer to the same key, if the default Profile file is P_RSASHA256:

- Implicit: UNK/Finance/DepartmentPettyCashKey
- Partially implicit: HS/UNK/Finance/DepartmentPettyCashKey
- Explicit: P_RSASHA256/HS/UNK/Finance/DepartmentPettyCashKey

Note: here Finance refers to a storage key and DepartmentPettyCashKey is a key wrapped by the Finance key.

#### 4.7.1.4 User duplicable keys

A user may wish to share a key between different systems he owns, without tying them to a particular machine as well

##### 4.7.1.4.1 Examples

- Implicit: UDK/Finance/PayPalKey
- Explicit: ECC384/HS/UDK/Finance/PayPalKey

### 4.7.2 NV

NV indexes will be created under the NV label. Users will most likely create such indexes under the storage hierarchy.

**Examples**

- Implicit: NV/MyHashIndex
- Explicit: HP/NV/EKcertificate

### 4.7.3 Permanent Handles

There are 4 permanent handles that have data associated with them that needs to be stored elsewhere. Although the hierarchies may have Profiles which in turn are associated with a hash algorithm, there are not multiple copies of the policy, so these permanent handles are not associated with a Profile:

#### 4.7.3.1 HE is the endorsement hierarchy, which has a policy associated with it.

This hierarchy is controlled by a privacy administrator, but is unlikely to be used except by the EK.

#### 4.7.3.2 HS is the storage hierarchy, which has a policy associated with it

This hierarchy is the one most used by the end user.

#### 4.7.3.3 HP is the platform hierarchy, which may have a policy associated with it.

This hierarchy is unlikely to be ever usable by an end user.

#### 4.7.3.4 HEph is the ephemeral hierarchy, which disappears on each reboot, and always has NULL password and NULL policy.

This hierarchy mostly likely will be used for loading in public keys for verification of signatures.

#### 4.7.3.5 DA is the dictionary attack permanent handle which may have a policy associated with it.

## 4.8 Data kept at the path

When the FAPI creates entities or instantiates them, information about the entity is stored at a path location. The following describes that data.

### 4.8.1 Keys

- One of these three
  - If persistent, an index
  - If primary, Configuration (used for creation of primary keys, as data for non-primary)
  - If a blob, the blob
- Name
  - Hash
  - Data used to create name
  - Birth Certificate (if any)
- The policy data
  - Hash
  - Full policy data
- Short name
- The description of the key “My little key I used for my VPN”
- Certificate

### 4.8.2 NV indexes

- Index
- Name
  - Hash
  - Data used to create name
- The policy data
  - Hash
  - Full policy data
- Short name
- Description

### 4.8.3 Hierarchies

- The policy data
  - Hash
  - Full policy data
- ShortName “Storage”, “Hierarchy”, “Platform” or “NULL”
- Description “Storage”, “Hierarchy”, “Platform” or “NULL”

### 4.8.4 policyForms

- XML description of the policy. This may include placeholders, such as "Use the current PCR value stored in PCR 5", or "ask for the smartcard public key", or "ask for what the biometric signs to be used as the policyRef". It may be compound, PolicyOR, or anything and can be very complicated
- PORT for callback if a PolicySigned
- ShortName
- Description of PolicyForm

### 4.8.5 policyInstances

- Hash of policy when used as an object policy (this is used as a UUID)
- Data that is extended into a PolicyBuffer when this policy is satisfied
- Type: ACE, Compound (policyANDed of other policies), PolicyOR, or PolicyAuthorize
  - ACE
    - PolicyCommand
    - PolicyParameters
    - RegisteredCommand (if external hardware needed)
  - Compound
    - List of other PolicyInstances in order
  - PolicyOR
    - List of other PolicyInstances
  - PolicyAuthorize
    - List of authorized PolicyInstances
    - Ticket associated with those instances
    - Parameters necessary to satisfy command (public key)
- PORT for callback if PolicySigned
- Date of Creation
- Short Name
- Description of PolicyInstance

# 5 Commands

## 5.1 Context commands

### 5.1.1 Tss2_Context_Initialize

#### 5.1.1.1 Description

Tss2_Context_Initialize() initializes a context and establishes a connection to a TSS stack which ultimately leads to a TPM. The url parameter can be used to define which TSS to connect to. If url == NULL, then the local TPM is used. The memory necessary for the context object is allocated by the FAPI and freed during Tss2_Finalize(). If there are multiple local TPMs, NULL should not be used.

#### 5.1.1.2 Command

```
TSS2_RC Tss2_Context_Initialize(TSS2_CONTEXT **context, const char *url);
```

#### 5.1.1.3 Returns

TSS2_SUCCESS

TSS2_OUT_OF_MEMORY

#### 5.1.1.4 Example

```
TSS2_CONTEXT *context;
ret = Tss2_Context_Initialize(&context, NULL);
assert(ret == TSS2_SUCCESS);
ret = Tss2_Encrypt(context, "path/to/key", data_in, &data_out);
assert(ret == TSS2_SUCCESS);
Tss2_Context_Finalize(context);
```

### 5.1.2 Tss2_Context_Finalize

#### 5.1.2.1 Description

Tss2_Context_Finalize() finalizes a context by closing IPC/RPC connections and freeing its consumed memory.

#### 5.1.2.2 Command

```
void Tss2_Context_Finalize(TSS2_CONTEXT *context);
```

#### 5.1.2.3 Returns

No Return value

#### 5.1.2.4 Example

See 5.1.1.4

## 5.2 TPM commands

### 5.2.1 Tss2_TPM_GetRandom

#### 5.2.1.1 Description

This command is given the number of random bytes to return and Returns them in the array “data”.

Note: The memory for data is allocated by the caller.

#### 5.2.1.2 Command

```
TSS2_RC Tss2_TPM_GetRandom( TSS2_CONTEXT    *context,     /*input*/
                            size_t          numBytes,    /*input*/
                            uint8_t         *data)       /*output*/
```

#### 5.2.1.3 Returns

| | |
|---|---|
| **TSS2_SUCCESS** | Command Successful |
| **TSS2_TOO_BIG** | The number of bytes requests is larger than the default in the Profile |
| **TSS2_OUT_OF_MEMORY** | |
| **TSS2_FAIL** | Cannot provide the request number of bytes |

#### 5.2.1.4 Example

```
uint8_t randomData[25];

ret = Tss2_TPM_GetRandom(context, 25,randomData);
```

## 5.3 TSS Commands

### 5.3.1 Tss2_GetVersion

#### 5.3.1.1 Description

This command RETURNS the major, minor, and revision version of the specification that this TSS was written to.

- If version 2.00, rev .13 it will return major=2, minor=0, revision 13
- If version 2.01, rev .22, it will return major=2, minor=1, revision 22

#### 5.3.1.2 Command

```
TSS2_RC Tss2_GetVersion(TSS2_CONTEXT *context,     /*input */
                        uint16_t      *major,      /*output*/
                        uint16_t      *minor,      /*output*/
                        uint16_t      *revision)   /*output*/
```

#### 5.3.1.3 Returns

| | |
|---|---|
| **TSS2_SUCCESS** | Command Successful |
| **TSS2_FAILURE** | Command failed |

#### 5.3.1.4 Example

```
TSS2_CONTEXT  *context;
Tss2_Context_Intialize(context);
uint16_t      major, minor, revision;
ret = Tss2_GetVersion(context, &major, &minor, &revision);
assert(ret == TSS2_SUCCESS);
printf("TSS version %d.%d Revision %d\n",major, minor, revision);
Tss2_Context_Finalize(context);
```

### 5.3.2 Tss2_GetConfig

#### 5.3.2.1 Description

This command Returns to the program information about the default Profile file. This is useful to determine the size of the hash, for example. The command Returns both a numeric value for what is returned and a string identifier.

In all cases the last two bytes of an algorithm are identical to the TPM specification. However to prevent ECC BN from being confused with TPM_ALG_NULL, the BN functions return 0x10000010 and 0x10000011 for BN_P256 and BN_638 respectively instead of 0x00000010 and 0x00000011

Values that can be input for feature are:

- TSS2_ASYM_ALG 0x0000
  - Values returned:
    - 0x00000001 RSA
    - 0x00000018 ECC prime fields
    - 0x10000010 BN256
    - 0x10000011 BN638
- TSS2_ASYM_KEYSIZE 0x0001
  - Value returned is keysize
- TSS2_ASYM_SIGN_SCHEME 0x0002
  - Values returned
    - 0x00000014 RSASSA
    - 0x00000001 ECDHA
- TSS2_ASYM_ENCRYPT_SCHEME 0x0003
  - Values returned
    - 0x00000000 CFB
- TSS2_HASHALG 0x0010
  - Values returned
    - 0x00000004 SHA1
    - 0x0000000B SHA256
    - 0x0000000C SHA384
    - 0x0000000D SHA512
- TSS2_HASHSIZE 000011
  - Values returned is keysize
- TSS2_SYM_ALG 0x0020
  - Values returned
    - 0x00000006 AES
- TSS2_SYM_KEYSIZE 0x0021
  - Values returned is keysize
- TSS2_SYM_MODE 0x0022
  - Values returned
    - 0x00000041 CFB

- TSS2_SRK_HANDLE 0x0031
    - Value returned:
        - The value of the handle.
        - Otherwise  error
- TSS2_EK_HANDLE 0x0041
    - Value returned:
        - The value of the handle.
        - Otherwise error.
- TSS2_EK_CERT_INDEX 0x0042  // Do we already have a getter for cert.
    - Value returned:
        - If persistent, the value of the index.
        - Otherwise error
- TSS2_MAX_RANDOM 0x0050
    - Value returned is the maximum number of random bytes to be asked for in one command (as a uint32_t)

#### 5.3.2.2 Command

```
TSS2_RC Tss2_GetConfig( TSS2_CONTEXT    *context,        /*input */
                        uint16_t        feature,         /*input*/
                        uint32_t        *value)          /* output*/
```

#### 5.3.2.3 Returns:

| | |
|---|---|
| **TSS2_SUCCESS** | Command Successful |
| **TSS2_NO_PERSISTENT_HANDLE** | |

#### 5.3.2.4 Examples

```
uint32_t   hashSize, asymSize, symSize, hashAlg, asymAlg, symAlg;

char *ToString(uint32_t value)
{
    char *retString = NULL;
    switch (value) {
        /* Hash Algorithms */
        case 0x00000004:
            retString = "SHA1";
            break;
        case 0x0000000B:
            retString = "SHA256";
        case 0x0000000C:
            retString = "SHA384";
            case 0x000000D:
                retString = "SHA256";
            /* Assymmetric Key Algorithms */
            case 0x00000001:
                retString = "RSA";
                break;
            /* other cases */
            /* Symmetric Key Algorithms */
            case 0x000006:
                retString = "AES";
        default:
            break;
    }

    return retString;
}


TSS2_RC rc;
rc = Tss2_GetConfig(context, TSS2_HASHALG, &hashAlg);
assert(rc == TSS2_SUCCESS);
rc = Tss2_GetConfig(context, TSS2_HASHSIZE, &hashSize);
assert(rc == TSS2_SUCCESS);
printf("The hash’s algorithm is %s %d bits long\n", ToString(hashAlg), hashSize);

rc = Tss2_GetConfig(context, TSS2_ASYM_ALG, &asymAlg);
assert(rc == TSS2_SUCCESS);
rc = Tss2_GetConfig(context, TSS2_ASYM_KEYSIZE, &asymSize);
assert(rc == TSS2_SUCCESS);
printf("The asymmetric algorithm %s %d bits long \n", ToString(asymAlg),asymSize);

rc = Tss2_GetConfig(context, TSS2_SYM_ALG, &symAlg);
assert(rc == TSS2_SUCCESS);
rc = Tss2_GetConfig(context, TSS2_SYM_KEYSIZE, &symSize);
assert(rc == TSS2_SUCCESS);
printf("The symmetric algorithm algorithm is %s %d bits long\n", ToString(symAlg), symSize);
```

## 5.4 Entity

Some commands need to be available for all entities.  As such, we have general commands described.

### 5.4.1 Tss2_Entity_ChangeAuth

#### 5.4.1.1 Description

Changes the Authorization data of an entity found at keyPath

#### 5.4.1.2 Command

```
TSS2_RC Tss2_Entity_ChangeAuth(TSS2_CONTEXT                 *context,       /*input*/
                               const char                   *entityPath,    /*input*/
                               const TSS2_SIZED_BUFFER      *AuthValue);    /*input*/
```

#### 5.4.1.3 Returns:

| | |
|---|---|
| **TSS2_SUCCESS** | Command Successful |
| **TSS2_PATH_NOT_FOUND** | Path was not found |

#### 5.4.1.4 Example Code

```
char     password[]=”horse”;
TSS2_SIZED_BUFFER myPassword;
myPassword.buffer = malloc(strlen(password) + 1);
assert(myPassword.buffer);
myPassword.size=strlen(password)
memcpy(myPassword.buffer, password, strlen(password));

ret = Tss2_Entity_ChangeAuth(context, “/NV/MyNVindex”, &myPassword);
```

### 5.4.2 Tss2_Entities_List

#### 5.4.2.1 Description

Given a search Path, the list of entities that reside under this path shall be enumerated and returned to the application.

This will return a list of complete paths, such that the return paths can be directly used for another query;

Note: The caller needs to call free() on each of the string entries as well as on the returned list of pointers to the entries.

#### 5.4.2.2 Command

```
TSS2_RC Tss2_Entities_List(  TSS2_CONTEXT   *context,         /* input */
                             const char     *searchPath,      /*input*/
                             char           ***pathlist,      /*output*/
                             size_t         *num_elements);   /*output*/
```

#### 5.4.2.3 Returns:

| | |
|---|---|
| **TSS2_SUCCESS** | Command Successful |
| **TSS2_PATH_NOT_FOUND** | Path was not found |
| **TSS2_ERROR_NOT_A_STORAGE_KEY** | Path to be searched does not point to a storage key |

#### 5.4.2.4 Example usage:

```
/* List the first two levels of user-keys */
ret = tss2_feat_listKeys("/USK", &l1_paths, &num_l1_paths);
for (l1 = 0; l1 < num_l1_paths; l1++) {
    if ((ret = Tss2_Entities_List(l1_paths[l1], &l2_paths, &num_l2_paths)) == ERROR_NOT_A_STORAGE_KEY) {
        printf("Found Leaf Key %s\n", l1_paths[l1]);
    } else {
        printf("Found Key %s\n", l1_paths[l1]);
        for (l2 = 0; l2 < num_l2_paths; l2++)
            printf("Found Key %s\n", l2_paths[l2]);
    }
    free_string_list(l2_paths, num_l2_paths);
}
free_string_list(l1_paths, num_l1_paths);
```

### 5.4.3 Tss2_Entity_Delete

#### 5.4.3.1 Description

Tss2_Entity_Delete() does different things according to the entity type

If Entity is:

- Regular key:
  - delete entry from path
  - flush from the TPM (including any resource managers)
- Primary key:
  - Evict from TPM
  - Delete entry from path
- NV
  - Delete index from TPM
  - Delete entry from path
- Policy
  - Delete entry from path
- Policy form
  - Delete entry from path
- Hierarchy
  - Return error
- PCR
  - Return error
- If the key is anObjecdt Ancestors (e.g. SNK, UNK, UDK, or SDK), the command Returns an error.

#### 5.4.3.2 Command

```
TSS2_RC Tss2_Entity_Delete( TSS2_CONTEXT    *context,     /* input */
                            const char      *Path);       /*input*/
```

#### 5.4.3.3 Returns

| | |
|---|---|
| **TSS2_SUCCESS** | Command Successful |
| TSS2_PATH_NOT_FOUND | Path was not found |
| **TSS2_ENTITY_NOT_DELETABLE** | The entity is not deletable (e.g. a PCR or hierarchy) |

#### 5.4.3.4 Example usage

```
ret = Tss2_Entity_Delete(context, “USK/myVISAkey”);
```

## 5.5 Policy commands

### 5.5.1 Tss2_Policy_CreateInstance

#### 5.5.1.1 Description

Allows the creation of a Policy instance from Policy forms.

#### 5.5.1.2 Command

```
TSS2_RC Tss2_Policy_CreateInstance( TSS2_CONTEXT    *context,               /* input */
                                    const char      *PolicyForm,            /*input*/
                                    const char      *policyPathInstance);   /*input*/
```

Note: if the policyPath does not start with "policy/" then this will be prefixed automatically.

#### 5.5.1.3 Returns

| | |
|---|---|
| **TSS2_SUCCESS** | Command successful |
| **TSS2_POLICY_FORM_NOT_FOUND** | One of the Policy Forms was not found |

#### 5.5.1.4 Example usage:

```
policyDefinition1 = “insert XML description here";
policyDefinition2 = "{ \"policy' = {...}}\";

Tss2_Policy_CreateInstance (context, policyDefinition1,  "/policy/exampleInc/genericServiceKeyPolicy”);
/* New Policy */
Tss2_PathSetDescription(context, "/policy/exampleInc/genericServiceKeyPolicy", "mylittlePolicy");
Tss2_Key_Create(….,"/policy/exampleInc/genericServiceKeyPolicy”, …);
```

### 5.5.2 Tss2_Policy_Export

#### 5.5.2.1 Description

PathExport simply takes a particular path and copies the contents into a buffer in the correct format to be used by the Policy_CreateInstance command. It differs from Key_ExportTree, in that no encryption or decryption is done, and that a Key_ExportTree will combine all the keys beneath the selected parent key.

#### 5.5.2.2 Command

```
TSS2_RC Tss2_PathExport(   TSS2_CONTEXT       *context,   /*input*/
                           const char         *path,      /*input*/
                           TSS2_SIZED_BUFFER  *buffer     /*input*/
                         );
```

#### 5.5.2.3 Returns

| | |
|---|---|
| **TSS2_SUCCESS** | Command Successful |
| **TSS2_PATH_NOT_FOUND** | **Can't find the path** |

#### 5.5.2.4 Example

```
TSS2_SIZED_BUFFER *myUserPolicy;
Tss2_PathExport(context, “policy/myFingerprint”,myUserPolicy);
```

### 5.5.3 Tss2_Policy_PcrRestriction

#### 5.5.3.1 Description

Tss2_Policy_PcrRestriction takes *PCRs as values

Creates and registers a policy for those PCRs at the current values of those PCRs

Note that it is possible to do this same operation with a Tss2_Policy_CreateInstance command, but it would require too many potential PolicyForms in order for it to be practical. Hence this feature, which allows a programmer not to have to rely on forms being available.

#### 5.5.3.2 Command

```
TSS_RC Tss2_Policy_PcrRestriction(   TSS2_CONTEXT    *context,        /*input*/
                                     uint64_t        *PCRs,           /*input*/
                                     const char      *policyPath);   /*input*/
```

#### 5.5.3.3 Returns

| | |
|---|---|
| **TSS2_SUCCESS** | Command Successful |
| TSS2_PATH_NOT_FOUND | Path was not found |

#### 5.5.3.4 Example Usage

```
Tss2_Policy_PcrRestriction({0,1,2,3,4,5,-1}, “/policy/pcr0_5”);
Tss2_PatHSetDescription(“/POLICY/pcr0_5”,”Policy set to current values of PCRs 0,1,2,3,4 and 5”);
Tss2_Create(SEALED_DATA, data,32,”/Policy/pcr0_5”);
```

### 5.5.4 Tss2_Policy_AuthorizeNewPolicy

If a current policy happens to be a PolicyAuthorize, then for it to be used, the user must first satisfy a policy authorized by a having been signed (and made into a ticket) by an authorized party. In this case, the PolicyInstance will point to one or more other policyInstances along with a ticket that can be used with them. The key used for authorization is pointed to in the policyPathToUpdate.

#### 5.5.4.1 Description

| | |
|---|---|
| policyPathToUpdate: | A current PolicyAuthorize policy, to which authorized policies are to be added |
| policyPathToAddOrDelete: | A policy to be auhorized (added) or removed from the list of authorized policies |
| An integer ADD_DELETE | to indicate if policyPathToAddOrDelete is being added or deleted (1=Add, 0=Delete) |

If the integer is delete, it checks if the current policy is one registered with a ticket for the updatable policy. If it is, it deletes it. Otherwise it returns an error code.

If the flag is add, it looks up the correct key to use in the current PolicyAuthorize policy, and creates a ticket using the private portion of that signing key, and then registers the policyPathToAddOrDelete, with its ticket in the policyPathToUpdate.

#### 5.5.4.2 Command

```
Tss2_Policy_AuthorizeNewPolicy( TSS2_CONTEXT    *context,                 /*input*/
                                constant char   *policyPathToUpdate,      /*input*/
                                constant  char  *policyPathToAddOrDelete, /*input*/
                                int             ADD_DELETE);              /*input*/
```

#### 5.5.4.3 Returns

| | |
|---|---|
| **TSS2_SUCCESS** | command successful |
| **TSS2_KEY_NOT_FOUND** | can’t find the key necessary to quote |
| **TSS2_PATH_NOT_FOUND** | path to delete not found |

#### 5.5.4.4 Example Usage

```
Tss2_Policy_AuthorizeNewPolicy(context, “Path/MyExtendablePolicy”, “Path/BillsSmartCard”,ADD);
```

## 5.6 Keys

Most programs using a TPM will be using TPM keys of some sort, be they asymmetric or symmetric, for signing, encryption or HMACing.

### 5.6.1 Tss2_Key_Create

#### 5.6.1.1 Description

Creates a key based on the Key type, using the supplied policy and authvalue.
Stores the key based on the path, possibly caches the key.
Returns an error if the implementation doesn't know about the parent.
When FAPI is installed it will automatically create an SRK using the template. If it is not persistent, then when it will create the SRK on demand. Users must create their own SNK, SDK, UNK and UDK keys.

#### 5.6.1.2 Command

```
TSS2_RC Tss2_Key_Create(   TSS2_CONTEXT          *context,       /*input*/
                           const char            *keyPath,       /*input*/
                           const char            *KeyType,       /*input*/
                           const char            *policy,        /*input*/
                           TSS2_SIZED_BUFFER     *authvalue);    /*input*/
```

#### 5.6.1.3 Returns

| | |
|---|---|
| **TSS2_SUCCESS** | Command Successful |
| **TSS2_KEY_NOT_FOUND** | Can’t find the key necessary to quote |
| **TSS2_POLICY_NOT_FOUND** | Path to policy not found |

#### 5.6.1.4 Example Usage

##### 5.6.1.4.1 Create an AIK-like key

```
/* The following assigns a password of 12345 to myAIKPassword, then creates a restricted signing key under
   the SNK (which makes it system-wide and non-duplicable). It further assigns it a policy which requires
   entering the password to use the AIK. */
TSS2_SIZED_BUFFER myAIKPassword ;
    myAIKPassword.size=5;
    myAIKPassword.buffer=calloc(5,1);
    memcpy(myAIKPassword.buffer,”12345”,5);

Tss2_Key_Create(  context,
                  “SNK/MyAIK”,             /* The SNK root makes this non-duplicable */
                  ASYM_RESTRICTED_SIGNING_KEY,
                  POLICY_PASSWORD,         /* This uses a preset policy pointing to the password */
                  &myAIKPassword );        /* This is where you enter the password */

Free(myAIKPassword.buffer);
/* Note that if using P_RSASHA1 as a default, this key would be an RSA2048 bit key using SHA1.  If using
the T_ECCP256, it would be an ECC key using a 256 bit prime field and SHA256. */
```

#### 5.6.1.4.2 Create a duplicable HMAC key only usable for signing

```
Tss2_Key_Create(  context,
                  “HS/UDK/MyHmac”,         /* The SDK root makes this duplicable */
                  SYM_RESTRICTED_SIGNING_KEY,
                  POLICY_SECRET_EH,        /* This policy points to the EH password */
                  NULL_AUTH );             /* This password is largely irrelevant */
```

### 5.6.2 Tss2_Key_Sign

#### 5.6.2.1 Description

Uses a key, identified by its path, to sign a digest, and puts the result in a TPM2B byte stream

Note that the FAPI is responsible for allocating memory for the signature, publicKey and certificate buffers, but it is the responsibility of the programmer to free them.

The format for the public key is a PEM[1] data format

If certificate is not available, it Returns an empty buffer.

#### 5.6.2.2 Command

```
TSS2_RC Tss2_Key_Sign( TSS2_CONTEXT              *context,      /*input*/
                       const    char             *keyPath,      /*input*/
                       const TSS2_SIZED_BUFFER   *digest,       /*input*/
                       TSS2_SIZED_BUFFER         *signature,    /*output*/
                       TSS2_SIZED_BUFFER         *publicKey,    /*output*/
                       TSS2_SIZED_BUFFER         *certificate   /*output*/
                       );
```

#### 5.6.2.3 Returns

| | |
|---|---|
| **TSS2_SUCCESS** | command successful |
| **TSS2_KEY_NOT_FOUND** | Can't find the key necessary to sign |
| **TSS2_POLICY_NOT_FOUND** | Path to policy not found |

#### 5.6.2.4 Example

```
TSS2_SIZED_BUFFER myDigest;
/* code inserted here to set the myDigest to equal the thing you want signed */
TSS2_SIZED_BUFFER signature, publicKey, certificate;
Tss2_Key_Sign(context,
              "/SNK/MyAik"
              &myDigest,
              &signatureOut
              &publicKey,
              &certificate);
free(signatureOut.buffer);
free(publicKey.buffer);
free(certificate.buffer);
```

1 Defined in RFC's 1421 through 1424

### 5.6.3 Tss2_Key_Verify

#### 5.6.3.1 Description

Verifies a signature using a public key found in a keyPath. The publicKey is a PEM data format

#### 5.6.3.2 Command

```
TSS2_RC Tss2_Key_Verify(    TSS2_CONTEXT                 *context,       /*input*/
                            const TSS2_SIZED_BUFFER      *signature,     /*input*/
                            const TSS2_SIZED_BUFFER      *publicKey      /*input*/
                            const TSS2_SIZED_BUFFER      *digest);       /*input*/
```

#### 5.6.3.3 Returns

| | |
|---|---|
| **TSS2_SUCCESS** | command successful |
| **TSS2_SIGNATURE_FAIL** | Signature did not match |
| **TSS2_KEY_NOT_FOUND** | The key was not found at the path |
| **TSS2_BAD_PARAMETER** | |

#### 5.6.3.4 Example

```
TSS2_SIZED_BUFFER myDigest;
TSS2_SIZED_BUFFER publicKey;
TSS2_SIZED_BUFFER signatureIn;

/* code inserted here to set the myDigest to equal the thing supposedly signed, publicKey an signature */
/* code inserted here to read in the public key used to verify signature */
Tss2_Key_Verify(context, &signatureIn, &publicKey,  &myDigest);
```

### 5.6.4 Tss2_Key_VerifyQuote

#### 5.6.4.1 Description

Verifies that the data returned by a quote is valid. It does this by reproducing the hash from the data given.

If the hashSigned buffer is not NULL, it is compared against that value.  If there is a mismatch, an error is returned.

Next the signature is checked using the public key.  If there is a mismatch, an error is returned.

Next the hash in the signature is checked against a hash of the message. If there is a mismatch, an error is returned

If everything is correct, return TSS2_SUCCESS.

#### 5.6.4.2 Command format

```
TSS2_RC Tss2_Key_VerifyQuote(  TSS2_CONTEXT              *context,        /*input*/
                               TSS2_PCRSELECTION         PCRsChosen,      /*input*/
                               const uint8_t *           PCRvalues,       /*input*/
                               const TSS2_SIZED_BUFFER   *PCRlog,         /*input*/
                               const char                *keyPath,        /*input*/
                               TSS2_SIZED_BUFFER         *nonce,          /*input*/
                               TSS2_SIZED_BUFFER         *quote,          /*input*/
                               TSS2_SIZED_BUFFER         *hashSigned,     /*input*/
                               uint16_t                  hashAlg,         /*input*/
                               TSS2_SIZED_BUFFER      *TPMdataSigned,    /*input*/
                               const TSS2_SIZED_BUFFER   *publicKey)      /*input*/
```

#### 5.6.4.3 Returns

| | |
|---|---|
| **TSS2_SUCCESS** | Command successful |
| **TSS2_KEY_NOT_FOUND** | Can't find the key necessary to quote |
| **TSS2_SIGNATURE_FAIL** | Signature did not match |
| **TSS2_HASHMISMATCH** | Hash did not match |

#### 5.6.4.4 Example Usage

```
Tss2_Key_VerifyQuote( context,
                      {2,3,-1},
                      “SNK/aik”,
                      &nonce,
                      &quote,
                      &hashSigned,
                      SHA256,
                      PCRvalues,
                      &TPMdataSigned
                      &publicKey);
```

### 5.6.5 Tss2_Key_ExportTree

#### 5.6.5.1 Description

Given a key it will (if the key is a storage key) duplicate the key and package up the duplicated key and all keys below it into file ready to move to a new TPM. If it is not a storage key, it will only duplicate the key itself. The data that it exports will be a TSS2_SIZED_BUFFER.

The buffer will be a ExportTree structure:

Exported data is a sized buffer. Inside the buffer a set of TSS2_KEYs serialized and concatenated.

And the first of the keys contains the public data of the target parent, and its path on the target machine.

The buffer of the exported data will contain first the re-wrapped key pointed to by the pathOfKeyToDuplicate and then the serialized concatenated set of all keys underneath that storage key (if any).

Note that the FAPI is responsible for allocating memory for the exportedData buffer, but it is the responsibility of the programmer to free them.

#### 5.6.5.2 Command

```
TSS2_RC Tss2_KeyExportTree(   TSS2_CONTEXT         *context,                      /*input*/
                              const char           *pathOfKeyToDuplicate,         /*input*/
                              TSS2_SIZED_BUFFER    *publicKeyofNewParent,         /*input*/
                              TSS2_SIZED_BUFFER    *exportedData);                /*output*/
```

#### 5.6.5.3 Returns

| | |
|---|---|
| **TSS2_SUCCESS** | Command Successful |
| **TSS2_PATH_NOT_FOUND** | Path was not found |
| **TSS2_KEY_NOT_FOUND** | Path to new parent not found |
| **TSS2_PATH_NOT_DUPLICABLE** | The key is not a duplicable key |

#### 5.6.5.4 Example

```
/* In order to export a user's duplicable keys, he picks HS/UDK as his path to move.  In order to move, he
needs to have a public key to which he must move.  It will be loaded into the Ephemeral Hierarchy, and then
duplicated.  Rather than have the user have to load a public key in a separate step, this just uses a
TSS2_SIZED_BUFFER structure for the public key.*/
FILE   *inFile, *outFile;
inFile=fopen("myPublicKey.bin","rb");
outFile=fopen("myExportedTree.bin","wb");

TSS2_SIZED_BUFFER myPublicKey, outTree;
myPublicKey.size=2048;
myPublicKey.buffer=calloc(2048,1);
fread(myPublicKey.buffer,2048,1,inFile);

Tss2_KeyExportTree(context, "HEph/UDK",&myPublicKey, &outTree);
fwrite(outTree.buffer,outTree.size,1,outFile);

free myPublicKey.buffer;
fclose(inFile);
fclose(outFile);
```

### 5.6.6 Tss2_Key_ImportTree

#### 5.6.6.1 Description

Given a file created by a Tss2_Key_ExportTree command it will import the key and appropriately place the rest of the keys underneath it.

#### 5.6.6.2 Command

```
TSS2_RC Tss2_KeyImportTree(TSS2_CONTEXT        *context,          /*input*/
                           TSS2_SIZED_BUFFER *importedTree,     /*input*/
                           char                *newPathName);     /*input*/
```

#### 5.6.6.3 Returns

| | |
|---|---|
| **TSS2_SUCCESS** | Command successful |
| **TSS2_PATH_NOT_FOUND** | Path to parent key was not found |
| **TSS2_PATHALREADY_EXISTS** | Name of imported path key already exists. |

#### 5.6.6.4 Example

```
FILE   *inFile;
inFile=fopen(“myExportedTree.bin”,”rb”);

TSS2_SIZED_BUFFER myTree;
myPublicKey.size=2048;
myPublicKey.buffer=calloc(2048,1);
fread(myTree.buffer,2048,1,inFile);

Tss2_KeyImportTree(context, &myTree,”HS/UDK/OldUDK”);
/* Internal to the myTree structure, the first key listed contains the public data of the key being
duplicated. This includes its path name.  The second key is imported under this key using TPM2_Import
command, to the appropriate path.  The keys under it are then copied into the tree under this newly named
key”
```

### 5.6.7 Tss2_Key_SetCertificate

#### 5.6.7.1 Definition

Sets an x509 cert into the path of a key.

#### 5.6.7.2 Command

```
TSS2_RC Tss2_Key_SetCertificate(const char * path,                  /*input*/
                                TSS2_SIZED_BUFFER *x509cert);       /*output*/
```

#### 5.6.7.3 Returns

| | |
|---|---|
| **TSS2_SUCCESS** | Command successful |
| **TSS2_PATH_NOT_FOUND** | Path not found |

### 5.6.8 Tss2_Key_GetCertificate

#### 5.6.8.1 Definition

Gets an x.509 cert from the path of a key.

#### 5.6.8.2 Command

```
TSS2_RC Tss2_Key_GetCertificate(const char * path,                /*input*/
                                TSS2_SIZED_BUFFER *x509cert); /*output*/
```

#### 5.6.8.3 Returns

| | |
|---|---|
| **TSS2_SUCCESS** | Command successful |
| **TSS2_PATH_NOT_FOUND** | Path not found |

## 5.7 Data

### 5.7.1 Tss2_Data_Encrypt

#### 5.7.1.1 Description

Input pointer to data be encrypted via a key found in keyPath requiring a policy found in policyPath for the decryption, and a place to put the cipherText. This command does as follows:

Encrypts plaintext using a key found in keyPath and produces cipherText. It does this by

1) Checking if the key in keyPath is symmetric. If it is, it gets the symmetric key and uses it to encrypt the data. If not:
2) Checking if the plaintext is too big to be encrypted with the asymmetric key pointed to in keypath in one encryption.
    a. If so:
        i. It creates a new (ephemeral) symmetric key on the host using the algorithm parameters in the Profile file using the TPM RNG
        ii. It encrypts the data using this symmetric key on the host, using the crypto mode in the Profile file. This is done in software.
        iii. It wraps the ephemeral key using the key found in keyPath (this key must currently be asymmetric) and sealed to the policy given in policyPath. (This is done with TPM2_Create)
        iv. It adds metadata in front of the encrypted data, which contains the full policy, the key's parent' name and the ephemeral key blob.
    b. If not:
        i. It encrypts the data using the key found in keyPath (this key must currently be asymmetric) and sealed to the policy given in policyPath (This is done with TPM2_Create).
        **ii.** It attaches metadata to the encrypted data, which contains the full policy, and the key's parent's name.

**Metadata format:**

| | | |
|---|---|---|
| uint32_t | sizeOfMetadata | |
| TSS2_SIZED_DATA | policy | |
| TSS2_SIZED_DATA | sizeOfPublicParent | (if absent, persistent SNK is assumed) |
| TSS2_SIZED_DATA | encryptedEphemeralKey | (may be absent, in case b) |

Endianess will be little-endian

Note that the FAPI is responsible for allocating memory for the cipherText buffer, but it is the responsibility of the programmer to free them.

#### 5.7.1.2 Command

```
TSS2_RC Tss2_Data_Encrypt ( TSS2_CONTEXT              *context,      /*input*/
                            const TSS2_SIZED_BUFFER   *plaintext,    /*input*/
                            const char                *keyPath,      /*input*/
                            const char                *policyPath,   /*input*/
                            TSS2_SIZED_BUFFER         *cipherText);  /*output*/
```

#### 5.7.1.3 Returns

| | |
|---|---|
| **TSS2_SUCCESS** | Command Successful |
| **TSS2_PATH_NOT_FOUND** | Path was not found |
| **TSS2_POLICY_PATH_NOT_FOUND** | Path to policy not found |

#### 5.7.1.4 Example

```
TSS2_SIZED_BUFFER myData;
char                myClearText[24]="My data to be encrypted";
myData.size=strlen(myClearText) + 1;
myData.buffer=myClearText;
TSS2_SIZED_BUFFER myCipherText;

Tss2_Data_Encrypt(context, &myData, "SNK", "policy/PCR_0_5", &myCipherText);
```

### 5.7.2 Tss2_Data_Decrypt

#### 5.7.2.1 Description

Input pointer to data to be decrypted, and a place to put the decrypted data

Note that the FAPI is responsible for allocating memory for the plaintext buffer, but it is the responsibility of the programmer to free it. Since the key used to decrypt it may be at a different location than the key used for encryption (and on a different machine), it is located by its name, which is stored with the data.

#### 5.7.2.2 Command

```
TSS2_RC Tss2_Data_Decrypt( TSS2_CONTEXT               *context,      /*input*/
                           const TSS2_SIZED_BUFFER    *cipherText,   /*input*/
                           TSS2_SIZED_BUFFER          *plainText);   /*output*/
```

#### 5.7.2.3 Returns

| | |
|---|---|
| **TSS2_SUCCESS** | Command successful |
| **TSS2_KEY_MISSING** | Can’t find the key necessary to decrypt the file |
| **TSS2_BAD_FILENAME** | Can’t find the encrypted file |
| **TSS2_PCR_MISMATCH** | The PCRs are in the wrong state |

#### 5.7.2.4 Example Usage:

```
TSS2_SIZED_BUFFER myPlainText;
TSS2_SIZED_BUFFER myCipherText;
FILE*  myFile;
myFile=fopen(“myEncryptedFile.bin”,”rb”);

fseek(myFile, 0L, SEEK_END);
myCipherText.size = ftell(myFile);
fseek(fp, 0L, SEEK_SET);

myCipherText.buffer=calloc(myCipherText.size,1);
fread(myCipherText.buffer, myCipherText.size,myFile);

Tss2_Data_Decrypt(context, &myCipherText, &myPlainText);

……
free(myPlainText.buffer);
```

## 5.8 PCRs

### 5.8.1 Tss2_PCR_ReadWithLog

#### 5.8.1.1 Description

Given a set of PCRs, it will fill in an array of the appropriate size with the contents of the PCRs. If you ask for 5 PCRs and the hash algorithm is of size 32 bytes it will return a 5 by 32 array of bytes. It will return a TPM2B with the PCRlog values that correspond to those PCRs. The ~~callee~~ FAPI must allocate continuous memory in which to place the PCR values consecutively.

Note that the FAPI is responsible for allocating memory for the PCRlog buffer, but it is the responsibility of the programmer to free them.

The format of the buffer in the PCRlog output is defined by the PC Client specification.

#### 5.8.1.2 Command format

```
TSS2_RC Tss2_PCR_ReadWithLog(  TSS2_CONTEXT         *context,      /*input*/
                               uint32_t             hashAlg,       /*input*/
                               TSS2_PCRSELECTION    PCRsChosen,    /*input*/
                               uint8_t***           PCRvalues,     /*output*/
                               TSS2_SIZED_BUFFER    *PCRlog)       /*output*/
```

#### 5.8.1.3 Returns

| | |
|---|---|
| **TSS2_SUCCESS** | command successful |

#### 5.8.1.4 Example Usage

Assuming the PCRs are using SHA-1, which is 20 bytes, the following command:

```
char              PCRvalueArray[4][20];
TSS2_SIZED_BUFFER *PCRlog;
Tss2_PCR_ReadWithLog(context, SHA1, {0,2,3,-1}, &PCRvalueArray,PCRlog);
```

This would return:

A 4 by 20 array of bytes such that PCRvalueArray[0] would contain the values of PCR0, PCRvalueArr[1] would contain those of PCR2, PCRvalueArray[2] would contain the values of PCR3.

### 5.8.2 Tss2_PCR_Quote

#### 5.8.2.1 Description

Given a set of PCRs and a restricted signing key, it will sign those PCRs, and return

- The signature
    - The digest signed (The digest is computed as the hash of the concatenation of all of the digest values of the selected PCR.)
- The data that went into creating the hash that was signed in two parts
    - The PCR values that went into the hash
    - The other TPM values (which include the boot counter, etc.)
- Note that the FAPI is responsible for allocating memory for the publicKey and certificate buffers, but it is the responsibility of the programmer to free them.

The FAPI must allocate continuous memory in which to place the PCR values consecutively.

- The TPMdataSigned's buffer structure contains TPMS_ATTEST data.

#### 5.8.2.2 Command format

```
TSS2_RC TSS2_PCR_Quote( TSS2_CONTEXT        *context,                /*input*/
                        TSS2_PCRSELECTION PCRsChosen,                /*input*/
                        const char          *keyPath                 /*input*/
                        TSS2_SIZED_BUFFER *nonce                     /*input*/
                        TSS2_SIZED_BUFFER *signature                 /*output*/
                        TSS2_SIZED_BUFFER *hashSigned                /*output*/
                        uint8_t             *PCRvalues[][],          /*output*/
                        TSS2_SIZED_BUFFER *PCRlog,                   /*output*/
                        TSS2_SIZED_BUFFER *TPMdataSigned,            /*output*/
                        TSS2_SIZED_BUFFER *publicKey,                /*output*/
                        TSS2_SIZED_BUFFER *publicKeycertificate)     /*output*/
```

#### 5.8.2.3 Returns

| | |
|---|---|
| **TSS2_SUCCESS** | command successful |
| **TSS2_KEY_NOT_FOUND** | can't find the key necessary to quote |

#### 5.8.2.4 Example Usage

```
TPM2B         signature;
TPM2B         hashSigned;
char          PCRvalues[3][32];
TPM2_ATTEST   TPMdataSigned
TSS2_SIZED_BUFFER publicKey, certificate;
Tss2_PCR_Quote(   context, {2,3,-1},"SNK/aik",
                  &challengeNonce,&signature,&hashSigned,
                  &PCRvalues,&TPMdataSigned,&publicKey,
                  &certificate);
```

### 5.8.3 Tss2_PCR_Extend

#### 5.8.3.1 Description

The command extends the data in the data parameter into the single PCR listed. The parameter logData, which must be in the format designated by the PC Client event structure, will be extended into the PCR log. If the logData is NULL, only the extend takes place.

#### 5.8.3.2 Command

```
TSS2_RC Tss2_PCR_Extend( TSS2_CONTEXT        *context,      /*input*/
                         uint32_t            hashAlg,       /*input*/
                         int                 PCR,           /*input*/
                         TSS2_SIZED_BUFFER   &data,         /*input*/
                         TSS2_SIZED_BUFFER   &logData);     /*input*/
```

#### 5.8.3.3 Returns

| | |
|---|---|
| **TSS2_SUCCESS** | Command Successful |
| **TSS2_NO_SUCHPCR** | No such PCR exists on this TPM |
| **TSS2_OPERATION_NOT_ALLOWED** | |

#### 5.8.3.4 Example

```
TSS2_SIZED_BUFFER dataToExtend;
int8_t mydata[22]=”Four score and seventy”;
dataToExtend.size=22;
dataToExtend.buffer=mydata;
Tss2_PCR_Extend(context,SHA256, 15, &dataToExtend, NULL);
```

### 5.8.4 Tss2_PCR_Reset

#### 5.8.4.1 Description

Resets a given PCR (currently 0-15 are not resettable under any circumstances).

#### 5.8.4.2 Command

```
TSS2_RC Tss2_PCR_Reset(    TSS2_CONTEXT *context,        /*input*/
                           uint32_t          PCR);       /*input*/
```

#### 5.8.4.3 Returns

| | |
|---|---|
| **TSS2_SUCCESS** | Command Successful |
| **TSS2_NO_SUCHPCR** | No such PCR exists on this TPM |
| **TSS2_PCR_NOT_RESETTABLE** | This is not a resettable PCR |

#### 5.8.4.4 Example

```
Tss2_PCR_Reset(context, 16);
```

## 5.9 NV

- o Administrate
    - Create an NV index, with read/write settings
    - Create PCR
    - Create Counter
- o Use
    - Store the data in the index,
    - Read data from an index.

### 5.9.1 Tss2_NV_CreateWithTemplate

#### 5.9.1.1 Description

This command takes a path(name), a template for an NV index (see 3.3.2), a size, password and policy, and instantiates it in the TPM. The TSS picks the index, which it stores, along with the resultant name in the path.

#### 5.9.1.2 Command

```
TSS2_RC Tss2__NV_CreateWithTemplate( TSS2_CONTEXT      *context,     /*input*/
                                     const char        *NvPath,      /*input*/
                                     uint8_t           NVTemplate,   /*input*/
                                     uint32_t          size,         /*input*/
                                         /*size ignored if template not TSS2_NV_MEMORY
                                         TSS2_NV_TEMP_READ_DISABLE */
                                     TSS2_SignedBuffer *password,    /*input*/
                                     char              *PolicyPath)  /*input*/
```

#### 5.9.1.3 Returns

| | |
|---|---|
| **TSS2_SUCCESS** | Command Successful |
| **TSS2_PATH_ALREADY_EXISTS** | The path name already is populated |
| **TSS2_TEMPLATE_NOT_FOUND** | The NV template name is probably mispelled |
| **TSS2_NOT_ENOUGH_MEMORY** | The TPM doesn't have sufficient NV memory to make this index |
| **TSS2_POLICY_NOT_FOUND** | The policy is not found at this path |

#### 5.9.1.4 Example code

```
uint8  password[7]=”banana”;
TSS2_SIZED_BUFFER myPassword;
myPassword.size=6;
memcpy(myPassword.buffer,password,6);
Tss2_NV_CreateWithTemplate(context, “NV/myHash”,TSS2_NV_MEMORY,32,&myPassword,POLICY_PASSWORD);
Tss2_NV_CreateWithTemplate(context, “NV/departmentMembers”,TSS2_NV_BITMAP,8,NULL_PASSWORD,POLICY_SH);
Tss2_NV_CreateWithTemplate(context, “NV/myCounter”,TSS2_NV_COUNTER,8, NULL_PASSWORD,POLICY_EH);
Tss2_NV_CreateWithTemplate(context, “NV/myAuditPCR”,TSS2_NV_PCR,32, NULL_PASSWORD,POLICY_SH);
Tss2_NV_CreateWithTempalte(context, “NV/myEncryptionKey,TSS2_NV_MEMORY_FRAGILE,32,
                                    NULL_PASSWORD,”Policy/PCR0_thru_5”);
```

### 5.9.2 Tss2_NV_Write

#### 5.9.2.1 Description

If the NV is a standard TSS2_NV_MEMORY, then it can be written with this command.

#### 5.9.2.2 Command

```
TSS2_RC Tss2_NV_Write( TSS2_CONTEXT    *context,    /*input*/
                       const char      *NvPath,     /*input*/   sized_buffer not data nad size…..
                       const uint8_t   *data,       /*input*/
                       size_t          size);       /*input*/  /* Offset always assumed to be zero */
```

#### 5.9.2.3 Returns

| | |
|---|---|
| **TSS2_SUCCESS** | Command Successful |
| **TSS2_PATH_NOT_FOUND** | Can’t find the path |
| **TSS2_NV_NOT_BIG_ENOUGH** | The data to be written is bigger than the NV at that index |
| **TSS2_NV_NOT_WRITEABLE** | The NV referred to in the path is not a writeable index |

#### 5.9.2.4 Example

```
char      data[32];
/* code omitted that populates this with the hash of a public key*/
Tss2_NV_Write(context, “NV/myHash”, data,32);
```

### 5.9.3 Tss2_NV_Extend

#### 5.9.3.1 Description

Extends the data in data into the single NV_PCR listed.

#### 5.9.3.2 Command

```
TSS2_RC Tss2_NV_Extend(   TSS2_CONTEXT        *context,         /*input*/
                          const char          *NvPath,          /*input*/
                          TSS2_SIZED_BUFFER   *DataToExtend);   /*input*/
```

#### 5.9.3.3 Returns

| | |
|---|---|
| **TSS2_SUCCESS** | COMMAND SUCCESSFUL |
| **TSS2_PATH_NOT_FOUND** | Can’t find the path |
| **TSS2_NV_WRONG_TYPE** | The NV referred to in the path is not a NV_PCR index |
| **TSS2_OPERATION_NOT_ALLOWED** | |

#### 5.9.3.4 Example

```
TSS2_SIZED_BUFFER myDataToExtend;
char      data[9]=”hi there”
myDataToExtend.size=9;
myDataToExtend.buffer=data;
Tss2_NV_Extend(context, “NV/myAuditPCR”,&myDataToExtend);
```

### 5.9.4 Tss2_NV_Quote

#### 5.9.4.1 Description:

This command is used to provide a quote over the data stored in an NV location. Normally this NV location contains a NV_PCR, but it need not. Because the command TPM2_NV_Certify may not be an available TPM command, this command uses the session audit command to provide a quote over the data.

Note that the FAPI is responsible for allocating memory for the signature, hashSigned,and PCRlog buffers, but it is the responsibility of the programmer to free them.

#### 5.9.4.2 Command

```
TSS2_RC Tss2_NV_Quote(TSS2_CONTEXT        *context,         /*input*/
                      const char          *NvPath,          /*input*/
                      const char          *keyPath,         /*input*/
                      Tss2_SIZED_BUFFER   *nonce,           /*input*/
                      Tss2_SIZED_BUFFER   *signature,       /*output*/
                      Tss2_SIZED_BUFFER   *hashSigned,      /*ouput*/
                      Tss2_SIZED_BUFFER   *PCRlog,          /*output*/
                      Tss2_SIZED_BUFFER   *TPMdataSigned)   /*output*/  /* Format of data in buffer is that of
                                                                          the TPMS_ATTEST */
```

#### 5.9.4.3 Returns

| | |
|---|---|
| **TSS2_SUCCESS** | **command successful** |
| **TSS2_KEY_MISSING** | **can't find the key necessary to quote** |
| **TSS2_SIGNATURE_ INVALID** | **signature did not match** |
| **TSS2_HASHMISMATCH** | **hash did not match** |

#### 5.9.4.4 Example

```
Tss2_SIZED_BUFFER  nonce,signature,hashSigned,PCRlog;

TPMS_ATTEST      dataSigned;


Tss2_NV_Quote(context, “NV/myNVPCR”,”SNK/myAIK”,&nonce, &signature,&PCRvalue,&LogValue,&dataSigned);
```

### 5.9.5 Tss2_NV_Increment

#### 5.9.5.1 Description

This increments a NV index that is a counter by 1.

#### 5.9.5.2 Command

```
TSS2_RC Tss2_NV_Increment (TSS2_CONTEXT    *context,       /*input*/
                           const char      *NvPath)        /*input*/
```

#### 5.9.5.3 Returns

| | |
|---|---|
| **TSS2_SUCCESS** | **Command Successful** |
| **TSS2_PATH_NOT_FOUND** | **Can't find the path** |
| **TSS2_NV_WRONG_TYPE** | **The NV referred to in the path is not a NV_COUNTER index** |

#### 5.9.5.4 Example

```
Tss2_NV_Increment(context, “NV/myAuditCounter”);
```

### 5.9.6 Tss2_NV_SetBits

#### 5.9.6.1 Description

This command is used to SET bits in an NV Index that was created as a bit field. Any number of bits from 0 to 64 may be SET. The contents of *data* are ORed with the current contents of the NV Index starting at *offset*.

#### 5.9.6.2 Command

```
TSS2_RC Tss2_NV_SetBits(    TSS2_CONTEXT    *context,       /*input*/
                            char            *path,          /*input*/
                            uint64_t        bitsToSet)      /*input*/
```

#### 5.9.6.3 Returns

| | |
|---|---|
| **TSS2_SUCCESS** | Command Successful |
| **TSS2_PATH_NOT_FOUND** | Can’t find the path |
| **TSS2_NV_WRONG_TYPE** | The NV referred to in the path is not a NV_BITMAP index |

#### 5.9.6.4 Example

```
uint64_t    myBit7=0b1000000;
/*This sets bit number 7*
Tss2_NV_SetBits(context, “/NV/myDepartmentMembers”,myBit7)
```

### 5.9.7 Tss2_NV_Read

#### 5.9.7.1 Description

Any NV may be read with this command.

Note that the FAPI is responsible for allocating memory for the data buffer, but it is the responsibility of the programmer to free them.

#### 5.9.7.2 Command

```
TSS2_RC Tss2_NV_Read( TSS2_CONTEXT       *context,    /*input*/
                      const char         *NvPath,     /*input*/
                      TSS2_SIZED_BUFFER  *data);      /*input*/    /* Offset always assumed to be zero */
```

#### 5.9.7.3 RETURNS

| | |
|---|---|
| **TSS2_SUCCESS** | Command Successful |
| **TSS2_PATH_NOT_FOUND** | Can’t find the path |
| **TSS2_NV_NOT_READABLE** | The NV referred to in the path is not a readable index |

#### 5.9.7.4 Example

```
TSS2_SIZED_BUFFER data;
/* code omitted that populates this with the hash of a public key*/

Tss2_NV_Read(context, “NV/myHash”, &data);
printf(“Size of data read is %d”,data.size);
int i;
printf(“Data in Hex is:\n”);
for(i=0;i<data.size;++i) printf(“%2x “,data.buffer[i]);
```

### 5.9.8 Tss2_NV_MakeNVUnreadableThisBootSequence

#### 5.9.8.1 Description

If an index is a **TSS2_NV_TEMP_READ_DISABLE**, then it can be made unreadable for the rest of a boot sequence. This is usefull for passing data into an OS kernel during a boot sequence, and then making it unreadable afterwards.

#### 5.9.8.2 Command

```
TSS2_RC Tss2_NV_MakeNVUnreadableThisBootSequence(   TSS2_CONTEXT   *context,   /*input*/
                                                    char           *path);     /*input*/
```

#### 5.9.8.3 Returns

| | |
|---|---|
| **TSS2_SUCCESS** | Command Successful |
| **TSS2_PATH_NOT_FOUND** | Can’t find the path |
| **TSS2_NV_WRONG_TYPE** | The NV referred to in the path is not a TSS2_NV_TEMP_READ_DISABLE |

#### 5.9.8.4 Example

```
Tss2_NV_MakeNVUnreadableThisBootSequence(context, “NV/myEncryptionKey”);
```

## 5.10 Path Commands

### 5.10.1 Tss2_Path_SetShortNameAndDescription

#### 5.10.1.1 Description

Every path addressable entity has a description field that can be set in order to store additional information in its path entry. Short name shall be no more than 30 characters. Description can be as no longer than 1024 characters.

#### 5.10.1.2 Command

```
TSS2_RC Tss2_Path_SetShortNameAndDescription(   TSS2_CONTEXT    *context,       /*input*/
                                                const char      *path,          /*input*/
                                                const char      *shortName      /*input*/
                                                const char      *description    /*input*/
                                              );
```

#### 5.10.1.3 Returns

| | |
|---|---|
| **TSS2_SUCCESS** | Command Successful |
| **TSS2_PATH_NOT_FOUND** | Can’t find the path |

#### 5.10.1.4 Example:

```
Tss2_Key_Create(context, "/HS/UNK/fraunhoferSIT/TNCClientKey", .....);
Tss2_Path_SetShortNameAndDescription(context, "/HS/UNK/fraunhoferSIT/TNCClientKey", “VPN key”,"Key that is
used to authenticate on VPN-Establishment using the TNC-Client");
```

### 5.10.2 Tss2_Path_GetShortNameAndDescription

#### 5.10.2.1 Description

Every object has a description field that can be retrieved in order to obtain additional information in its “path” entry. Most have a shortName as well. The command will be allocating memory for the buffer of the shortName and description, but it is the responsibility of the programmer to free that memory.

#### 5.10.2.2 Command

```
TSS2_RC Tss2_Path_GetShortNameAndDescription(   TSS2_CONTEXT   *context,       /*input*/
                                                const char     *path,          /*input*/
                                                char           **shortName,    /*output*/
                                                char           **description   /*output*/
                                                );
```

#### 5.10.2.3 Returns

| | |
|---|---|
| **TSS2_SUCCESS** | Command Successful |
| **TSS2_PATH_NOT_FOUND** | Can’t find the path |

#### 5.10.2.4 Example:

```
Tss2_Path_GetShortNameAndDescription(context, "/HS/UNK/fraunhoferSIT/TNCClientKey",&shortName”,
&description);
  printf("Key %s: %s", pathNameList[i], description);

/* what is printed is: Key that is used to authenticate on VPN-Establishment using the TNC-Client */

 free(shortName);
 free(description);
```

# 6 Satisfying an EA Policy

When a TSS has to satisfy a policy, there are only three cases of Policy ACEs that may require interaction with either the user or and external device. The policy may have branches (PolicyOR) or PolicyAuth with multiple approved policies, it may require a password to be entered, or it may require that some data be signed. We handle these three cases with callbacks. The application is responsible for registering functions to handle these callbacks.

In the case that a signature is to be handled by external hardware, the hardware may have a device driver that handles the callback mechanism. This can be registered in a policy form as described elsewhere.

## 6.1 PolicyCallBack Commands

### 6.1.1 Tss2_PolicyBranchSelectionCallback

#### 6.1.1.1 Description:

This can take place when the Policy contains a PolicyOR (with more than one branch), or a PolicyAuthorize(which has more than one approved policy)

In this case, the TSS will execute a callback which will contain three things:

- An arbitrary pointer supplied by the application when the callback was registered
- The number of policies/branches to choose from
- The names associated with those policies/branches
- The description of the entity being authorized

Note the selectBranch returned will be the index of the selected branch so the range is 0 based.

#### 6.1.1.2 Command

```
typedef TSS2_RC (*Tss2_PolicyBranchSelectionCallback)(
                    TSS2_CONTEXT        context,            /*input*/
                    char                *description,       /*input*/
                    void                *userData,          /*input*/
                    size_t              numBranches         /*input*/
                    char const          **branchNames,      /*input*/
                    size_t              *selectedBranch);   /*output*/
          );
```

#### 6.1.1.3 Returns

| | |
|---|---|
| **TSS2_SUCCESS** | Command Successful |
| **TSS2_BRANCH_NOT_FOUND** | The selected Branch is not one of the choices |

#### 6.1.1.4 Example

```
TSS2_RC myBranchSelectionCallback(
   TSS2_CONTEXT      context,          /*input*/
   void              *userData,        /*input*/
   size_t            numBranches       /*input*/
   char const        **branchNames,    /*input*/
   size_t            *selectedBranch)  /*output*/
{
   if (numBranches == 0)
       return TSS2_BAD_PARAMETER;
   // This callback always chooses the first branch
    *selectedBranch=0;
   return TSS2_SUCCESS;
}


Tss2_SetPolicyBranchSelectionCallback(context, myBranchSelectionCallback, NULL);
```

### 6.1.2 Tss2_PolicyAuthCallback

#### 6.1.2.1 Description

In this case the TSS will execute a callback which will contain the description of the entity for which authorization is required. Note that the FAPI is responsible for allocating memory for the auth buffer, but it is the responsibility of the programmer to free it. This command Returns the authorization value back to the FAPI. The FAPI is responsible for creating the HMAC value necessary to provide authentication to the TPM.

#### 6.1.2.2 Command

```
typedef TSS2_RC (*Tss2_PolicyAuthCallback)( TSS2_CONTEXT       context,        /*input*/
                                            void               *userData,      /*input*/
                                            char const         *description,   /*input*/
                                            TSS2_SIZED_BUFFER *auth);          /*output*/
```

#### 6.1.2.3 Returns

| | |
|---|---|
| **TSS2_SUCCESS** | Command successful |

#### 6.1.2.4 Example

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <string>
#include <unistd.h>
#include <termios.h>

/* A callback that will prompt the user for a password, read it in, and return it to the FAPI.  Note this
uses malloc to allocate auth->buffer, the FAPI must call free to release the memory.
      Also note that this implementation leaves the password in memory.
   A proper implementation should zero out the memory in 'password' before it goes out of scope.
*/
Tss2_RC passwordCallback(    TSS2_CONTEXT      context,
                             char              *description,
                             void              *userData,
                             char        const *description,
                             TSS2_SIZED_BUFFER *auth)
{
    /* Sanity check inputs */
    if (!description || !auth)
      return TSS2_E_BAD_PARAMETER;

    /* Disable echo of keyboard input. */
    int fd = STDIN_FILENO;  /* We're controlling stdin. */
    struct termios termios;
    if (tcgetattr(fd, &termios) == -1)
      return TSS2_E_IO_ERROR;
    /* Save the old setting */
    tcflag_t savedFlags = termios.P_lflag;
    /* Turn off echo */
    termios.P_lflag &= ~(ECHO | ECHOE | ECHONL);
    if (tcsetattr(fd, TCSADRAIN, &termios) == -1)
```

```
      return TSS2_E_IO_ERROR;

    /* Prompt the user for the password. */
    printf("Enter your password for %s: ", description);
    fflush(stdout);

    /* Read the password */
    std::string password;
    int c = getchar();
    while ((c != '\n') && (c != EOF))
    {
        if (c == 0x7f) /* DEL */
          password.pop_back();
        else
          password.pusHback(c);
        c = getchar();
    }
    if (c == '\n')
      printf("\n");

    /* Restore the terminal state */
    termios.P_lflag = savedFlags;
    if (tcsetattr(fd, TCSADRAIN, &termios) == -1)
      return TSS2_E_IO_ERROR;

    /* Copy the password to the output parameter and return success */
    /* Don't include the trailing '\0'.   */
    uint8_t *returnPtr = (uint8_t*) malloc(password.size());
    if (!returnPtr)
      return TSS2_E_OUT_OF_MEMORY;
    auth->size = password.size();
    auth->buffer = returnPtr;
    memcpy(auth->buffer, password.P_str(), auth->size);

    return TSS2_SUCCESS;
}

int main( int argc, char *argv[])
{

result = Tss2_Initialize(&context, NULL);
assert(result);

result = Tss2_SetPolicyAuthCallback(context, myAuthCallback, NULL);
assert(result);

result = Tss2_Do_Something_With_Authorization(context, .....);
/* Might call passwordCallback() */

Tss2_Finalize(context);

return 0;
}
```

### 6.1.3 Tss2_PolicyHmacCallback

#### 6.1.3.1 Description

In this case the TSS will execute a callback which will contain the description of the entity for which authorization is required. Note that the FAPI is responsible for allocating memory for the auth buffer, but it is the responsibility of the programmer to free it. The program is responsible for returning an HMAC of the data for a PolicyAuthValue command, rather than returning merely the authorization itself. This is likely to be used as a hardware callback, but if the program decides to to the HMAC itself rather than let the FAPI do, it can use this command.

In order to do the HMAC, a number of parameters need to be passed back to the program (or hardware)

$$\begin{aligned}\mathrm{HMAC}_{\mathit{sessionAlg}}((&\mathit{sessionKey} \,||\, \mathit{authValue}),\\ &(\mathit{pHash} \,//\, \mathit{nonceNewer} \,//\, \mathrm{nonceOlder}\\ &\{ //\, \mathrm{nonceTPM}_{\mathrm{decrypt}} \} \{ //\, \mathrm{nonceTPM}_{\mathrm{encrypt}} \}\\ &//\, \mathit{sessionAttributes}))\end{aligned}$$

#### 6.1.3.2 Command

```
typedef TSS2_RC (*Tss2_PolicyHmacCallback)(TSS2_CONTEXT        context,        /*input*/
                                           void                *userData,      /*input*/
                                           char const          *description,   /*input*/
                                           TSS2_SIZED_BUFFER *priorToAuth,     /*input*/
                                           TSS2_SIZED_BUFFER *afterAuth,       /*input*/
                                           TSS2_SIZED_BUFFER *hmac);           /*output*/
```

#### 6.1.3.3 Returns

| | |
|---|---|
| **TSS2_SUCCESS** | Command successful |

#### 6.1.3.4 Example

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <string>
#include <unistd.h>
#include <termios.h>

Tss2_RC hmacCallback(    TSS2_CONTEXT       context,
                         char                  *description,
                         void              *userData,
                         char        const *description,
                         int32             HASH_ALG_SIZE;
                         TSS2_SIZED_BUFFER *priorToAuth,
                         TSS2_SIZED_BUFFER *afterAuth,
                         TSS2_SIZED_BUFFER *auth)
{
    /* Sanity check inputs */
    if (!description || !auth)
      return TSS2_E_BAD_PARAMETER;

    /* Disable echo of keyboard input. */
    int fd = STDIN_FILENO;  /* We're controlling stdin. */
    struct termios termios;
    if (tcgetattr(fd, &termios) == -1)
      return TSS2_E_IO_ERROR;
    /* Save the old setting */
    tcflag_t savedFlags = termios.P_lflag;
    /* Turn off echo */
    termios.P_lflag &= ~(ECHO | ECHOE | ECHONL);
    if (tcsetattr(fd, TCSADRAIN, &termios) == -1)
      return TSS2_E_IO_ERROR;

    /* Prompt the user for the password. */
    printf("Enter your password for %s: ", description);
    fflush(stdout);

    /* Read the password */
    std::string password;
    int c = getchar();
    while ((c != '\n') && (c != EOF))
    {
        if (c == 0x7f) /* DEL */
          password.pop_back();
        else
          password.pusHback(c);
        c = getchar();
    }
    if (c == '\n')
      printf("\n");

    /* Restore the terminal state */
    termios.P_lflag = savedFlags;
    if (tcsetattr(fd, TCSADRAIN, &termios) == -1)
      return TSS2_E_IO_ERROR;

    /* Copy the password to the output parameter and return success */
    /* Don't include the trailing '\0'.   */
    uint8_t *returnPtr = (uint8_t*) malloc(password.size());
    if (!returnPtr)
      return TSS2_E_OUT_OF_MEMORY;
    auth->size = password.size();
    auth->buffer = returnPtr;
    memcpy(auth->buffer, password.P_str(), auth->size);
   TSS2_SIZED_BUFFER hmac;
   hmac->size= HASH_ALG_SIZE;
   hmac->buffer=malloc(HASH_ALG_SIZE);
   hmac(priorToAuth, &auth, afterAuth, &hmac);

    return TSS2_SUCCESS;
}


int main( int argc, char *argv[])
{

result = Tss2_Initialize(&context, NULL);
assert(result);

result = Tss2_SetPolicyHmacCallback(context, myHmacCallback, NULL);
```

```
assert(result);

result = Tss2_Do_Something_With_Authorization(context, .....);
/* Might call passwordCallback() */

Tss2_Finalize(context);

return 0;
}
```

### 6.1.4 Tss2_PolicySignatureCallback

#### 6.1.4.1 Description

In this case the TSS will execute a callback which contains five things:

- An arbitrary pointer supplied by the application when the callback was registered
- A description of the public key whose private portion must create the signature
- The policyRef
- The information necessary to create the hash which is to be signed.
- An IP PORT (if any) registered by a device that is waiting to handle this signature. This value will be 0 if no device is registered to handle the callback. If a port is available, it promises that the device has a daemon listening on the port, running a service that when provided the parameters will return the signature.

(This is in case a device such as a biometric reader or smartcard read has registered to handle this signature. If this is the case, the second, third, and fourth parts may be simply shipped off to that IP port.)

#### 6.1.4.2 Command

```
typedef TSS2_RC (*Tss2_PolicySignatureCallback)(
                    TSS2_CONTEXT                context,        /*input*/
                    void                        *userData,      /*input*/
                    char const                  *description,   /*input*/
                    uint32_t                    hashAlg,        /*input*/
                    TSS2_SIZED_BUFFER const     *dataToSign,    /*input/output*/
                    uint16_t                    ipPort,         /*input*/
                    TSS2_SIZED_BUFFER           *signature      /*output*/
            );
```

Note: the “dataToSign” buffer should containe nonceTPM || expiration || cpHashA || policyRef. In most cases, expiration will be a uint32 set equal to 0, so no ticket is created. However, the actual signing authority will decide what the expiration and policyRef will be that are signed. Note that no policyRef may actually be needed by the Policy.

#### 6.1.4.3 Returns

| | |
|---|---|
| **TSS2_SUCCESS** | Command successful |
| **TSS2_CANT_PROVIDE_SIGNATURE** | **Cannote provide signature** |
| **TSS2_SIGNATURE_INVALID** | The signature is not valid |

#### 6.1.4.4 Example

### 6.1.5 Tss2_SetPolicyBranchSelectionCallback

#### 6.1.5.1 Description

This function registers a callback that will be invoked whenever the TSS has to decide which branch of a Policy-OR policy to use to authorize a particular TSS operation. Since the TSS does not know which branch is appropriate, the application-defined callback is used to make the choice for the TSS. The callback and user data pointers are associated with the context. The userData parameter is a pointer to application-defined data that will be passed to the callback each time it is invoked. The userData is intended to hold application-specific state as needed, and may be NULL if no such state is required. If the callback function pointer is NULL this clears the callback, and any attempt to use a policy that include an OR branch will fail.

#### 6.1.5.2 Command

```
Tss2_SetPolicyBranchSelectionCallback(   TSS2_CONTEXT    context,                      /*input*/
                                         Tss2_PolicyBranchSelectionCallback callback,  /*input*/
                                         void * userData                               /*input*/
                                         );
```

#### 6.1.5.3 Returns

| | |
|---|---|
| **TSS2_SUCCESS** | Command Successful |
| **TSS2_BAD_PARAMETER** | The context is not valid |

#### 6.1.5.4 Example:

```
TSS2_RC myBranchSelectionCallback(
   TSS2_CONTEXT      context,          /*input*/
   void              *userData,        /*input*/
   size_t            numBranches       /*input*/
   char const        **branchNames,    /*input*/
   size_t            *selectedBranch)  /*output*/
{
   if (numBranches == 0)
      return TSS2_BAD_PARAMETER;
   // This callback always chooses the first branch
    *selectedBranch=0;
   return TSS2_SUCCESS;
}


Tss2_SetPolicyBranchSelectionCallback(context, myBranchSelectionCallback, NULL);
```

### 6.1.6 Tss2_Path_SetPolicyAuthCallback

#### 6.1.6.1 Description

This function registers an application-defined function as a callback to allow the TSS to get authorization values from the application. The callback and user data pointers are saved within the context and the callback is invoked whenever an authorization value is needed. The userData parameter is a pointer to application-defined data that will be passed to the callback each time it is invoked. The userData is intended to hold application-specific state as needed, and may be NULL if no such state is required. If the callback function pointer is NULL this clears the callback, and any attempt to use a policy that requires user-supplied authorization will fail.

#### 6.1.6.2 Command

```
Tss2_Path_SetPolicyAuthCallback( TSS2_CONTEXT context,                /*input*/
                                 Tss2_PolicyAuthCallback callback,    /*input*/
                                 void * userData                      /*input*/
                                 );
```

#### 6.1.6.3 Returns

| | |
|---|---|
| **TSS2_SUCCESS** | Command Successful |
| **TSS2_PATH_NOT_FOUND** | Can't find the path |
| **TSS2_BAD_PARAMETER** | The context is not valid |

#### 6.1.6.4 Example:

/* This code sets the password to “horse” */

```
TSS2_RC myAuthCallback(  TSS2_CONTEXT        context,        /*input*/
                         void                *userData,      /*input*/
                         char const          *description,   /*input*/
                         TSS2_SIZED_BUFFER   *auth)          /*output*/
{
   if (auth == NULL)
      return TSS2_BAD_PARAMETER;
   auth->size = 5;
   auth->buffer = (uint8_t*) malloc(auth->size);
   memset(auth->buffer, “horse”, auth->size);
   return TSS2_SUCCESS;
}


Tss2_SetPolicyAuthCallback(context, myAuthCallback, NULL);
```

### 6.1.7 Tss2_Path_SetPolicyHmacCallback

#### 6.1.7.1 Description

This function registers an application-defined function as a callback to allow the TSS to get authorization values from the application. The callback and user data pointers are saved within the context and the callback is invoked whenever an authorization value is needed. The userData parameter is a pointer to application-defined data that will be passed to the callback each time it is invoked. The userData is intended to hold application-specific state as needed, and may be NULL if no such state is required. If the callback function pointer is NULL this clears the callback, and any attempt to use a policy that requires user-supplied authorization will fail.

#### 6.1.7.2 Command

```
Tss2_Path_SetPolicyHmacCallback(TSS2_CONTEXT            context,    /*input*/
                                Tss2_PolicyHmacCallback callback,   /*input*/
                                void * userData                     /*input*/
                                );
```

#### 6.1.7.3 Returns

| | |
|---|---|
| **TSS2_SUCCESS** | Command Successful |
| **TSS2_PATH_NOT_FOUND** | Can’t find the path |
| **TSS2_BAD_PARAMETER** | The context is not valid |

#### 6.1.7.4 Example:

### 6.1.8 Tss2_Path_SetPolicySignatureCallback

#### 6.1.8.1 Description

This function registers an application-defined function as a callback to allow the TSS to get signatures authorizing use of TPM objects. The callback and user data pointers are saved within the context and the callback is invoked whenever a signature-based policy is used to authorize a command. The userData parameter is a pointer to application-defined data that will be passed to the callback each time it is invoked. The userData is intended to hold application-specific state as needed, and may be NULL if no such state is required. If the callback function pointer is NULL this clears the callback, and any attempt to use a policy that requires a signature-based authorization will fail.

#### 6.1.8.2 Command

```
Tss2_Path_SetPolicySignatureCallback(TSS2_CONTEXT context,                /*input*/
                                     Tss2_PolicySignatureCallback callback, /*input*/
                                     void * userData                        /*input*/
                                     );
```

#### 6.1.8.3 Returns

| | |
|---|---|
| **TSS2_SUCCESS** | Command successful |
| **TSS2_BAD_PARAMETER** | The context is not valid |

#### 6.1.8.4 Example:

```
TSS2_RC mySignatureCallback(
   TSS2_CONTEXT             context,      /*input*/
   void                     *userData,    /*input*/
   char const               *description, /*input*/
   TSS2_SIZED_BUFFER const  *policyRef,   /*input*/
   TSS2_SIZED_BUFFER const  *dataToSign,  /*input*/
   uint16_t                 ipPort,       /*input*/
   TSS2_SIZED_BUFFER        *signature    /*output*/
{
   …
}


Tss2_SetPolicySignatureCallback(context, mySignatureCallback, NULL);
```

# 7 Appendix: List of Error Codes

| | |
|---|---|
| **General** | |
| **TSS2_SUCCESS** | Command Successful |
| **TSS2_OPERATION_NOT_ALLOWED** | |
| **TSS2_BAD_PARAMETER** | |
| **Authorization** | |
| **TSS2_AUTHORIZATION_FAILURE** | The authorization failed to work. |
| **TSS2_PCR_MISMATCH** | Sorry, the values currently in the PCRs don't match what is required |
| **Attestation** | |
| **TSS2_HASH_MISMATCH** | The hash doesn't match that used by the signature |
| **TSS2_SIGNATURE_INVALID** | The signature is not valid |
| **Mispelllings** | |
| **TSS2_PATH_NOT_DUPLICATABLE** | This path doesn't point to a duplicable file |
| **TSS2_PATH_ALREADY_EXISTS** | Can't create the path, because it already exists |
| **TSS2_PATH_NOT_FOUND** | No such path exists |
| **TSS2_KEY_NOT_FOUND** | Can't find a key at this path |
| **TSS2_POLICY_NOT_FOUND** | Can't find a policy at this path |
| **TSS2_POLICY_FORM_NOT_FOUND** | One of the Policy Forms was not found |
| **TSS2_BRANCH_NOT_FOUND** | The selected Branch is not one of the choices |
| **TSS2_BAD_FILENAME** | Sorry, can't find that filename |
| **TSS2_NO_SUCH_PCR** | Sorry, can't find that PCR |
| **TSS2_TEMPLATE_NOT_FOUND** | The template name is probably mispelled |
| **Wrong types** | |
| **TSS2_NOT_A_STORAGE_KEY** | The key pointed to isn't a storage key |
| **TSS2_ENTITY_NOT_DELETABLE** | The entity is not deletable (e.g. a PCR or hierarchy) |
| **TSS2_PCR_NOT_RESETTABLE** | This PCR is not resettable |
| **TSS2_NOT_ENOUGH_MEMORY** | The TPM doesn't have sufficient NV memory to make this index |
| **TSS2_NV_NOT_WRITEABLE** | The NV referred to in the path is not a writeable index |
| **TSS2_NV_NOT_READABLE** | The NV referred to in the path is not a readable index |
| **TSS2_NV_WRONG_TYPE** | This is the wrong kind of NV for this command |
| **TSS2_TOO_BIG** | Can't produce a random number this big. |
| **TSS2_WRONG_FORMAT** | Profile file is not in correct format |