TCG TSS 2.0 Enhanced System API (ESAPI) Specification

Version 1.00

Revision 08

May 28, 2020

Contact: admin@trustedcomputinggroup.org

TCG PUBLISHED

TCG

Corrections and Comments

Please send comments and corrections to admin@trustedcomputinggroup.org.

Normative-Informative Language

“SHALL,” “SHALL NOT”, “SHOULD”, ”SHOULD NOT”, “RECOMMENDED”, “MAY” and “OPTIONAL” in this document are normative statements. They are to be interpreted as described in [RFC-2119].

Acknowledgements

TCG and the TSS Work Group would like to thank the following people for their work on this specification.

• Will Arthur	Raytheon
• Brenda Baggaley	OnBoard Security
• Dave Challener	Johns Hopkins University, Applied Physics Lab
• Mike Cox	OnBoard Security
• Andreas Fuchs	Fraunhofer SIT
• Ken Goldman	IBM
• Jürgen Repp	Fraunhofer SIT
• William Roberts	Intel
• Tadeusz Struk	Intel
• Philip Tricca	Intel
• Lee Wilson	OnBoard Security

Table of Contents

1 Scope and Information on the TCG TSS 2.0 Specification Library

1.1 Scope of this Specification

This TCG TSS 2.0 ESAPI Specification defines the middle layer of the user-facing TSS 2.0 APIs and adds the following functionality to that provided by SAPI:

- ESAPI makes the management of TPM 2.0 sessions easier by providing support for establishing encryption of the TPM command/data stream and providing support of TPM enhanced authorization policies.
- ESAPI calculates command and response HMACs automatically, including the so-called TPM resources' public names. This ensures that applications reference the correct object.

The intended audience for this specification includes software developers and designers implementing applications for TPM 2.0 and TCG TSS 2.0.

1.2 Acronyms

For definitions of the acronyms used in the TSS 2.0 specifications please see the TCG TSS 2.0 Overview and Common Structures Specification [22}.

1.3 Key Words

The key words "MUST," "MUST NOT," "REQUIRED," "SHALL," "SHALL NOT," "SHOULD," "SHOULD NOT," "RECOMMENDED," "MAY," and "OPTIONAL" in this document normative statements are to be interpreted as described in RFC-2119, Key words for use in RFCs to Indicate Requirement Levels.

1.4 TCG Software Stack 2.0 (TSS 2.0) Specification Library Structure

At the time of writing, the documents that specify the TSS 2.0 are:

[1] TCG TSS 2.0 Overview and Common Structures Specification

[2] TCG TSS 2.0 TPM Command Transmission Interface (TCTI) API Specification

[3] TCG TSS 2.0 Marshaling/Unmarshaling (MU) API Specification

[4] TCG TSS 2.0 System API (SAPI) Specification

[5] TCG TSS 2.0 Enhanced System API (ESAPI) Specification

[6] TCG TSS 2.0 Feature API (FAPI) Specification

[7] TCG TSS 2.0 TAB and Resource Manager Specification

[8] TCG TSS 2.0 Response Code (RC) API Specification

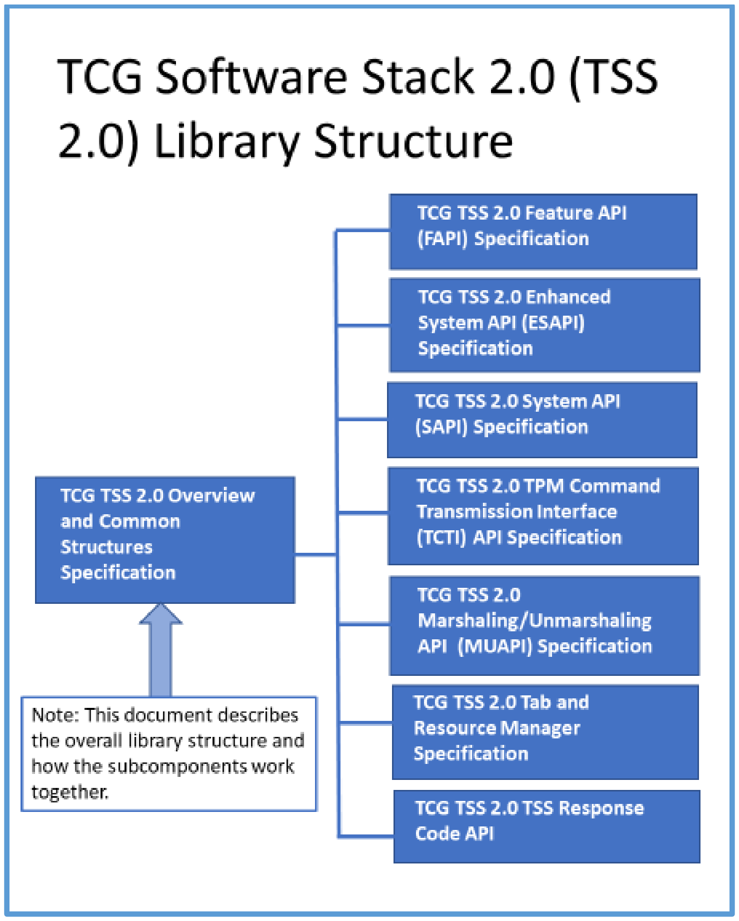

TCG Software Stack 2.0 (TSS 2.0) Library Structure
TCG TSS 2.0 Feature API (FAPI) Specification
TCG TSS 2.0 Enhanced System API (ESAPI) Specification
TCG TSS 2.0 System API (SAPI) Specification
TCG TSS 2.0 TPM Command Transmission Interface (TCTI) API Specification
TCG TSS 2.0 Marshaling/Unmarshaling API (MUAPI) Specification
TCG TSS 2.0 Tab and Resource Manager Specification
TCG TSS 2.0 TSS Response Code API
TCG TSS 2.0 Overview and Common Structures Specification
Note: This document describes the overall library structure and how the subcomponents work together.

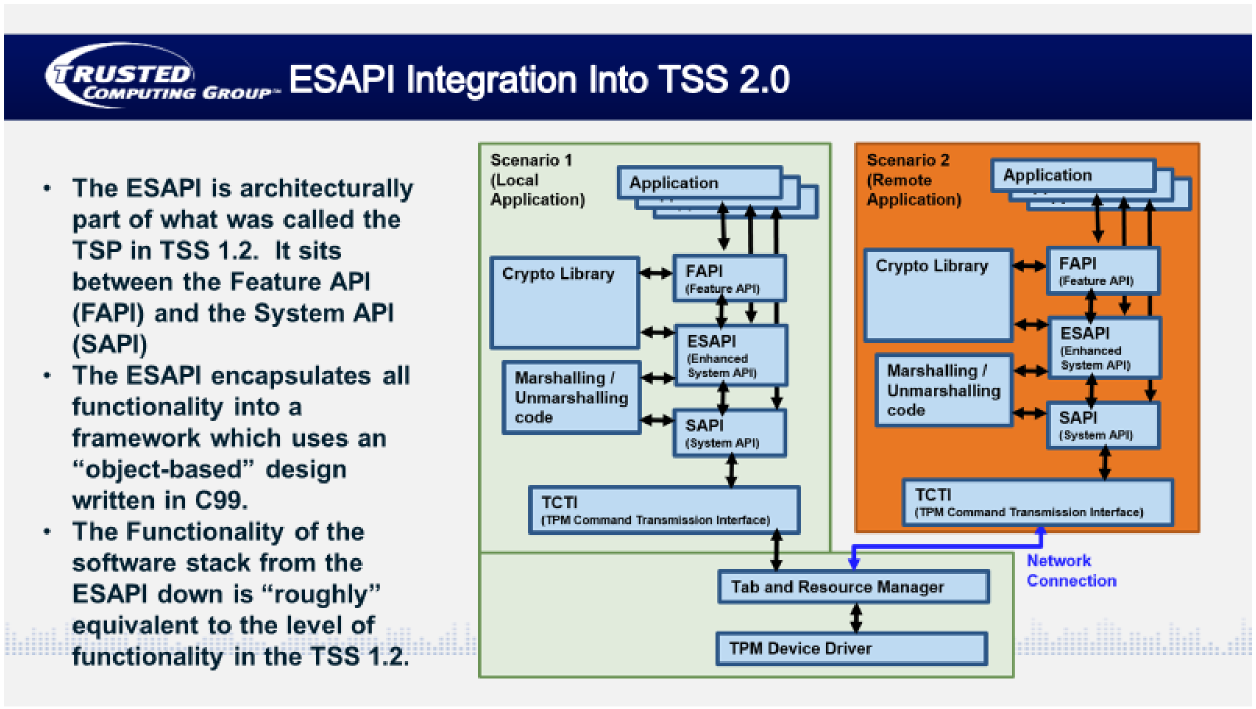
Figure 1: TSS 2.0 Specification Library

2 TSS Enhanced System API Overview

2.1 ESAPI Overview

The Enhanced System API (ESAPI) is an interface that is intended to sit directly above the System API. The primary purpose of the ESAPI is to reduce the programming complexity of applications that desire to send individual "system level" TPM calls to the TPM, but that also require cryptographic operations on the data being passed to and from the TPM. In particular, applications that wish to utilize secure sessions to perform Hash-based Message Authentication Code (HMAC) operations, parameter encryption, parameter decryption, TPM command audit and TPM policy operations could benefit from using the ESAPI. Additionally, context and object management are provided by the ESAPI.

While the ESAPI is significantly less complex to use than the System API for cryptographically protected communications with the TPM, it still requires in-depth understanding about the interface to a TPM 2.0. It is therefore envisioned that only expert applications will utilize the ESAPI and that typical applications would utilize a higher-level interface such as the Feature API. It is, however, expected that Feature API implementations would utilize the ESAPI as appropriate.



ESAPI Integration Into TSS 2.0

- The ESAPI is architecturally part of what was called the TSP in TSS 1.2. It sits between the Feature API (FAPI) and the System API (SAPI)
- The ESAPI encapsulates all functionality into a framework which uses an "object-based" design written in C99.
- The Functionality of the software stack from the ESAPI down is "roughly" equivalent to the level of functionality in the TSS 1.2.

Scenario 1 (Local Application)
Application
Crypto Library
FAPI (Feature API)
ESAPI (Enhanced System API)
Marshalling / Unmarshalling code
SAPI (System API)
TCTI (TPM Command Transmission Interface)
Scenario 2 (Remote Application)
Application
Crypto Library
FAPI (Feature API)
ESAPI (Enhanced System API)
Marshalling / Unmarshalling code
SAPI (System API)
TCTI (TPM Command Transmission Interface)
Network Connection
Tab and Resource Manager
TPM Device Driver

2.2 ESAPI Functional Description

The ESAPI layer of TSS 2.0 has three primary purposes:

- It provides cryptographic functionality for applications wishing to encrypt the data stream from TSS 2.0 to the TPM (thus defeating sideband attacks involving probing of the data bus to the TPM 2.0).
- It provides enhanced session management functionality on top of the base SAPI functionality.
- It provides 100% TPM functionality
  NOTE: The FAPI layer provides only about 80% of the TPM's functionality.

To use the ESAPI, applications (and their programmers) need to understand part 2 and part 3 of the TCG TPM 2.0 specification. The differences between ESAPI and SAPI are as follows.

There are several functions the ESAPI handles more simply (from the application's prespective) than the SAPI does:

- Starting and salting sessions.
- Calculating HMACs.
- Encrypting and decrypting sessions

However implementing an application to the SAPI interface has advantages for the following reasons:

- SAPI implementations have a smaller footprint primarily because they do not include cryptographic functions.
- SAPI can be used in heapless environments.
- SAPI allows saving RAM by underallocating data structures.

2.3 ESAPI Programming Considerations

The ESAPI API has the following high level characteristics:

- The ESAPI is written in C99 to allow it to operate in the broadest range of operating systems and to simplify the writing of language bindings to other languages (e.g. Java, Python, etc.).
- The ESAPI is designed to enable applications to send commands to the TPM using a single or very limited number of function calls when using sessions (when compared with the SAPI).
- The ESAPI allows applications to control/set all input parameters and data elements that are included in TPM commands, except for HMAC session calculation (e.g. nonceCaller).
- The ESAPI manages session handling:
    - Session key calculation
    - Session salt creation and encryption
    - HMAC calculation
    - Keeping track of resources's names
    - Session binding
- The ESAPI abstracts all formatting and crypto handling from the application – it is performed in the ESAPI layer or lower.
- Additionally, the ESAPI does the following:
    - HMAC (calculation/verify) with an authValue input
    - Encryption/decryption of the first parameter with a key input
    - Supports audit sessions
    - Cryptographic flexibility/agility (e.g. The ESAPI caller can perform cryptographic operations with SHA-1, SHA-256, various signing algorithms, various symmetric and asymmetric cryptographic algorithms, etc.)

ESAPI implementations are not required to maintain state in global variables between calls. State information is stored in contexts provided to the application. These ESAPI contexts are accessible simultaneously. No two threads are allowed the same ESAPI context simultaneously. The ESAPI supports both a synchronous and an asynchronous call model.

The ESAPI also meets the following additional low-level design requirements:

- The ESAPI mimics the TPM 2.0 specification, Part 3 command schematics as much as possible. This helps programmers understand the code and easily correlate with the spec. Function input and output parameters are ordered in the way they appear in the Part 3 command schematics and names match as much as possible. Additionally, the ESAPI provides functionality to automatically setup and tear down sessions as part of a larger operation.

- ESAPI gives the application full control over all TPM 2.0 part 3 input parameters and handles.
- The application developer calling ESAPI can get any desired TPM output parameters.
- The application developer calling ESAPI can suppress output parameters from the TPM that are unwanted.

- Memory for ESAPI output parameters is provided by the callee (the ESAPI implementation).
- Memory for ESAPI input parameters is provided by the caller (the application using the ESAPI)
- The Enhanced System API implementation can do all the work for the caller that System API implementations do (with greater ease of use). The following are some examples of simplifications for the application calling the ESAPI:
    - The commandSize field for all commands is calculated dynamically by the ESAPI. The caller doesn't have to do this.
    - The ESAPI operates on complex TPM2B's and special TPML's (like TPML_PCR_SELECT) the same as SAPI with regard to the size field.
    - The ESAPI unmarshals output parameters into C structures before returning them to the caller so that the caller can read fields out of them in a straightforward manner.
- The Enhanced System API implementation handles as much of the cryptographic operation and data management related to TPM 2.0 sessions as possible. Examples of ESAPI implementation operations include:
    - Centralized management of session-related and TPM resource-related data within an "ESAPI Object".
    - Computation of the names of TPM resources.
    - Formatting and calculation of all information required to compute a command; parameter hash (cpHash) and response parameter hash (rpHash).
    - Computation of HMACs for command integrity sent to the TPM.
    - Verification of response HMACs received from the TPM.
    - Encryption of the first parameter of commands intended for the TPM.
    - Decryption of the first parameter of responses encrypted by the TPM.

2.4 Marshaling/Unmarshaling Description

Marshaling and unmarshaling are required by both the SAPI and ESAPI. Rather than writing the marshaling and unmarshalling code separately for the SAPI and ESAPI these pieces of code have been put into a separate namespace that both the SAPI and ESAPI use. The marshaling and unmarshalling API and header files are specified in the TSS 2.0 Marshaling/Unmarshaling API Specification.

3 TSS ESAPI Use Model

3.1 Top-Level ESAPI Usage Model

The application will initialize a context, start sessions and create and load objects via the context, and finally close the context. The following steps are taken by the caller and ESAPI to establish contexts and sessions.

1. Application initializes an an ESAPI context
   a. ESAPI uses the heap, so ESAPI will dynamically allocate memory to track the metadata it needs for each TPM resource.
2. Application calls TPM commands and resource handling is partially automated by the ESAPI.
   a. If a new resource is created by a command then the ESAPI will create a new ESYS_TR object to track it and its metadata.
   b. For static TPM resources (such as NV and persistent keys) ESYS_TR objects can also be serialized and deserialized from disk or created directly from the TPM.
   c. Application calls Esys_StartAuthSession and the ESAPI manages session metadata such as SessionKey and NonceRolling and HMAC calculations.
3. Application flushes or closes ESYS_TR objects via ESAPI calls so ESAPI can free the allocated memory.
   a. If a TPM resource is destroyed by a command then the ESAPI will automatically free the memory used by the object metatdata and mark the ESYS_TR object as invalid.
   b. The application may close the ESYS_TR to release the memory used by the ESAPI without affecting the resource within the TPM.
4. The application closes ESAPI context when done.

3.2 The ESAPI Context

The ESYS_CONTEXT (the ESAPI context) is an opaque structure that contains all information that the ESAPI module needs to store between calls (so that the ESAPI doesn't need to maintain global state variables). The ESYS_CONTEXT contains:

- Information needed to communicate to the TPM (e.g. System API command context and TCTI context).
- Metadata for each ESYS_TR object.
- State information about internally used libraries e.g. cryptographic libraries, and cached TPM capabilities needed.

The memory for the ESYS_CONTEXT is allocated by the ESAPI.

The lifecycle of an ESYS_CONTEXT is briefly described as follows:

- Create an ESYS_CONTEXT using the Esys_Initialize() function.
- Create or deserialize metadata from disk storage for the session and resource information structures.
- Use sessions and resources in TPM commands.
- Serialize resource information structures to disk.
- Close ESYS_CONTEXT with Esys_Finalize().

3.3 ESAPI Resources

TPM resource metadata is stored in an opaque structure associated with an ESYS_CONTEXT, referenced by an ESYS_TR handle. An ESYS_TR object created within one ESYS_CONTEXT can be used only within that ESYS_CONTEXT.

The metadata for an ESYS_TR includes data like:

- The TPM handle for the resource (PCR, NV area, TPM object (key, data), hierarchy).
- The authValue of the resource, if applicable.
- The public area of the resource (TPM2B_PUBLIC or TPM2B_NV_PUBLIC).
- The resource name (can be computed from the above).

The lifecycle for an ESAPI resource is briefly described as follows:

- Create an ESYS_TR object:
    - For transient TPM objects, simply call the appropriate load function (e.g. Esys_Load, Esys_LoadExternal) and the ESAPI module creates a new ESYS_TR object.
    - For persistent resources (e.g. evict keys, NV areas):
        - Create an ESYS_TR using the Esys_TR_FromTPMPublic().
        - Alternatively deserialize metadata from disk using Esys_TR_Deserialize().
    - For TPM metaobjects (e.g. hierarchies and PCRS) each ESYS_CONTEXT contains a preexisting singleton ESYS_TR object that can be used directly.
    - Set the authValue for the resource using the Esys_TR_SetAuth() function.
- Make TPM calls referencing the resource in the handle area using the Esys_<COMMAND_NAME>() template.
- Serialize metadata to disk (e.g. public area of NVSpace, persistent keys, etc.) using the Esys_TR_Serialize() function.
- Destroy an ESYS_TR object:
    - Flush or remove the resource from the TPM when done using e.g. Esys_FlushContext(), Esys_NV_UndefineSpace, Esys_NV_UndefineSpaceSpecial, Esys_EvictControl, functions.
    - Close the object using Esys_TR_Close to release the metadata while leaving the resource in the TPM.

3.4 The ESAPI Session

ESAPI session data is stored in an opaque session information structure in an ESYS_CONTEXT, referenced by an ESYS_TR. The opaque session information structure includes the following types of data:

- TPM handle for the session.
- Information about session attributes to use in the next command.
- Information related to the bind state and encrypted salt of the session.
- Information related to the generation and storage of the session key.
- Session nonces.
- Policy session-specific information.

The lifecycle of an ESAPI session is briefly described as follows:

- Start session using the Esys_StartAuthSession() function.
- Make TPM calls referencing the session using the Esys_<COMMAND_NAME>() function template.
- Close the session:
    - Use the Esys_FlushContext() to immediately evict the session or
    - Set the 'continueSession' bit to false using Esys_TRSess_SetAttributes to evict the session the next time it is used.

3.5 ESAPI Use Model – Command Setup

The following example provides a typical sequence of function calls that an application will do to set up the environment to run a TPM command(s).

1. **Create ESAPI Context**
    a. Open a context by calling Esys_Initialize().
2. **Create session(s) if needed**
    a. Start a TPM session by calling Esys_StartAuthSession().
3. **Load objects into the TPM if needed**
    a. Automatically fill in resource information by calling Esys_Load().
    b. Call Esys_TR_SetAuth().
4. **Set up TPM resource information from disk if needed**
    a. Setup resource information from disk by calling Esys_TR_Deserialize().
    b. Call Esys_TR_SetAuth().
5. **Set up TPM resource information from the TPM if needed**
    a. If there are pre-existing TPM resources required, call Esys_TR_FromTPMPublic().
    b. Call Esys_TR_SetAuth().

3.6 ESAPI Use Model – Policy Setup

The following example provides a typical sequence of function calls for setting up a policy.

1. Perform "Command Setup" (see section **3.5** ESAPI Use Model – Command Setup**)**
2. Determine Desired Policy Digest
    a. Start Trial Session
        i. Call Esys_StartAuthSession() with the session type set to trial session.
    b. Send appropriate TPM 2.0 session commands
        i. Call Esys_PolicyXxx() session commands to update the digest value.
    c. Get final policy digest
        i. Call Esys_PolicyGetDigest()
3. Use obtained policy digest as authPolicy in future structures

3.7 ESAPI Use Model – TPM Command Execution

The following example provides a typical sequence of function calls for typical TPM use.

1. Perform "Command Setup" (see section **3.5** ESAPI Use Model – Command Setup**)**
2. Make TPM calls
    a. Call Esys_<COMMAND_NAME>(). This automatically
        i. Uses System API to help do all formatting, parsing and send/receive to the TPM
        ii. Generates outgoing nonces and handles incoming nonces
        iii. Encrypts command parameters and decrypts response parameters
        iv. Computes resource names, cpHash and rpHash
        v. Computes command HMACs and verifies response HMACs
    b. Call Esys_<COMMAND_NAME>_Async followed by Esys_<COMMAND_NAME>_Finish. These provide the same services as the one-call version of the function (one-call refers to the synchronous version of the function).

3.8 ESAPI Cryptographic Operations

The following example provides a typical sequence of function calls for doing cryptographic operations with ESAPI.

1. **Session setup**
    a. Caller picks algorithms, bind entity, encryptedSalt's target tpmKey, and session attributes
        i. ESAPI module selects nonce
        ii. ESAPI module selects and encrypts salt
        iii. ESAPI module starts session in TPM and retrieves TPM nonce
        iv. ESAPI module computes and stores session key
        v. ESAPI module stores the bind entity's name
2. **Sending a TPM command**
    a. Caller picks parameters, resource handles, session attributes (can use defaults)
        i. ESAPI module computes names of all resources pointed to by the handles area
        ii. ESAPI module encrypts command parameter, if needed
        iii. For each session (as needed)
            1. ESAPI module computes cpHash
            2. ESAPI module generates nonceCaller
            3. ESAPI module generates HMAC (including resource object's authValue as needed)
        iv. ESAPI module sends command to TPM and receives TPM nonces, HMACs, data
        v. For each session (as needed)
            1. ESAPI module computes rpHash
            2. ESAPI module verifies HMAC response

4 TSS Enhanced System API Structures and Functions

4.1 ESAPI Function Summary

The following table provides a quick reference list with minimal description of the ESAPI functions defined in this specification.

Table 1 - ESAPI Command Quick Reference Table

Function Name or Category	Description	Section Reference
ESAPI Context Administration Functions		
Esys_Initialize()	This function opens a new ESYS_CONTEXT before making TPM calls.	6.1
Esys_Finalize()	This function closes an open ESYS_CONTEXT.	6.2
Esys_GetTcti()	This function returns the user-supplied TCTI.	6.3
Esys_GetPollHandles()	This function returns a set of OS-specific handles that can be used for polling the TPM communication channel for asynchonous operation.	6.5
Esys_SetTimeout()	This function sets the amount of time a <COMMAND_NAME>_Finish call will block waiting for a response from the TPM.	6.6
Esys_Free()	This function frees the callee-allocated memory.	6.7
ESAPI General ESYS_TR Functions		
Esys_TR_Serialize()	This function serializes the metadata of a TR-Object into a byte buffer for writing into a file.	7.1
Esys_TR_Deserialize()	This function performs the inverse of Esys_TR_Serialize().	7.2
Esys_TR_FromTPMPublic()	This function instructs the ESAPI implementation to query the TPM for the metadata of a TPM object and to populate a new TR-Object with it **NOTE:** This command SHOULD be used only in combination with a salted authenticating TPM session.	7.3
Esys_TR_SetAuth()	This function sets the authorization value for a resource. This is not a TPM command – that's why it is Esys_TR_. **NOTE:** This function cannot be applied to a TR-SessionObject.	7.4
Esys_TR_GetName()	This function returns the name of ESYS_TR object to the caller for further calculation.	7.5
Esys_TR_Close()	This function releases all resources within the ESAPI related to a TR-Object.	7.6

Function Name or Category	Description	Section Reference
ESAPI Session Administration Functions		
Esys_TRSess_SetAttributes()	This function sets the session attributes to be used the next time the session is used in a TPM command.	8.1
Esys_TRSess_GetAttributes()	This function gets the session attributes to be used the next time the session is used in a TPM command.	8.2
Esys_TRSess_GetNonceTPM	This function gets the current value of the nonceTPM for this session.	8.3
Esys-TPM-Command-Template		
Esys_<COMMAND_NAME>	Function calls based on this template execute a TPM command synchronously.	9.1
Esys_<COMMAND_NAME>_Async	Function calls based on this template initiate an asynchronous TPM command.	9.2
Esys_<COMMAND_NAME>_Finish	Function calls based on this template return the results of an asynchronous TPM command.	9.3

4.2 ESAPI TPM Commands

ESAPI TPM Command Functions Requiring Special Handling		
Esys_StartAuthSession()	See ESAPI Session Administration Functions in section 4.1.	10.1
Esys_Load()	This function sends a TPM2_Load command to the TPM and creates a new ESYS_TR object to hold information about the loaded object.	10.2
Esys_LoadExternal()	This function sends a TPM2_LoadExternal command to the TPM and creates a new ESYS_TR object to hold information about the loaded object.	10.3
Esys_CreateLoaded()	This function sends a TPM2_CreateLoaded command to the TPM and creates a new ESYS_TR object to hold information about the loaded object.	10.4
Esys_HMAC_Start()	This function sends a TPM2_HMAC_Start command to the TPM and creates a new ESYS_TR object to hold information about the new sequence.	10.5
Esys_HashSequenceStart()	This function sends a TPM2_HashSequenceStart command to the TPM and creates a new ESYS_TR to hold information about the new sequence.	10.6

Esys_SequenceComplete()	This function sends a TPM2_SequenceComplete command to the TPM and destroys the ESYS_TR object on successful completion.	10.7
Esys_EventSequenceComplete()	This function sends a TPM2_EventSequenceComplete command to the TPM and destroys the ESYS_TR object on successful completion.	10.8
Esys_PolicyAuthValue()	This function sends a TPM2_PolicyAuthValue to the TPM and mark the session for the ESAPI to automatically include the auth value upon usage.	10.9
Esys_PolicyPassword()	This function sends a TPM2_PolicyPassword to the TPM and mark the session for the ESAPI to automatically use the auth value upon usage.	10.10
Esys_CreatePrimary()	This function sends a TPM2_CreatePrimary command to the TPM and creates a new ESYS_TR object to hold information about the new primary object.	10.11
Esys_HierarchyChangeAuth()	This function sends a TPM2_HierarchyChangeAuth command to the TPM and updates the auth value associated with the supplied ESYS_TR object.	10.12
Esys_ContextSave()	This function sends a TPM2_ContextSave command to the TPM and extends the context blob to include the metadata of the supplied ESYS_TR object. For ESYS_TRSess objects this function also invalidates the ESYS_TR object for all use other than Esys_FlushContext..	10.13
Esys_ContextLoad()	This function extracts the object metadata and context blob from the supplied buffer. Sends a TPM2_ContextLoad command to the TPM. Creates an ESYS_TR object from the metadata and the TPM handle.	10.14
Esys_FlushContext()	This function flushes a session, key, or sequence within the TPM using TPM2_FlushContext and invalidates the corresponding ESYS_TR.	10.15
Esys_EvictControl()	This function sends a TPM2_EvictControl command to the TPM and either creates or destroys an ESYS_TR object for the persistent handle.	10.16
Esys_NV_DefineSpace()	This function sends a TPM2_NV_DefineSpace command to the TPM and creates a new ESYS_TR object to hold information about the new NV space.	10.17

Esys_NV_UndefineSpace()	This function sends a TPM2_NV_UndefineSpace command to the TPM and destroys the ESYS_TR object.	10.18
Esys_NV_UndefineSpaceSpecial()	This function sends a TPM2_NV_UndefineSpaceSpecial command to the TPM and destroys the ESYS_TR object.	10.19
Esys_NV_Write()	This function sends a TPM2_NV_Write command to the TPM and if necessary updates the object name.	10.20
Esys_NV_Increment()	This function sends a TPM2_NV_Increment command to the TPM and if necessary updates the object name.	10.21
Esys_NV_Extend()	This function sends a TPM2_NV_Extend command to the TPM and if necessary updates the object name.	10.22
Esys_NV_SetBits()	This functions sends a TPM2_NV_SetBits command to the TPM and if necessary updates the object name.	10.23
Esys_NV_WriteLock()	This functions sends a TPM2_NV_WriteLock command to the TPM and if necessary updates the object name.	10.24
Esys_NV_ReadLock()	This function sends a TPM2_NV_ReadLock command to the TPM and if necessary updates the object name.	10.25
Esys_NV_ChangeAuth()	This function sends a TPM2_NV_ChangeAuth command to the TPM and updates the auth value associated with the supplied ESYS_TR object.	10.26
ESAPI TPM Command Functions Which Can Be Derived from TPM Comand Template		
Esys_<COMMAND_NAME>()	Unless noted, all other TPM 2.0 commands use this template.	9

5 ESAPI Types and Defines

The ESAPI contains a set of types and defines that are specific to the ESAPI.

5.1 Typedef ESYS_CONTEXT

The execution context for ESAPI is held in an opaque memory blob with the following characteristics:

- This blob is allocated and freed by the ESAPI.
- The application is required to provide a pointer for this blob during initialization and pass the same pointer to all function calls including finalization.

NOTE: an application may have several context blobs, but ESYS_TR objects cannot be passed between two different contexts.

The following typedef defines the C type for ESYS_CONTEXT.

```
typedef struct ESYS_CONTEXT ESYS_CONTEXT;
```

5.2 Typedef ESYS_TR

TPM Resources in ESAPI are represented by an object identifier of type ESYS_TR. This object identifier is used to track the connection between the underlying TPM object and meta-data for this TPM object such as object names or authValues. These objects are created and their identifiers are returned by ESAPI in several function call.

```
typedef uint32_t ESYS_TR;
```

Preallocated identifiers exist for a set of common ESYS_TR objects. Their purpose is to simplify usage in cases where no special construction is required. This set of common ESYS_TR objects includes PCRs and hierarchies and other authorizations.

These constant / singleton handles come from the reserved area of values below 0x1000. Values above 0x1000 are reserved for ESYS_TR object created by the ESYS during operation.

```
#define ESYS_TR_PCR0      0U
#define ESYS_TR_PCR1      1U
#define ESYS_TR_PCR2      2U
#define ESYS_TR_PCR3      3U
#define ESYS_TR_PCR4      4U
#define ESYS_TR_PCR5      5U
#define ESYS_TR_PCR6      6U
#define ESYS_TR_PCR7      7U
#define ESYS_TR_PCR8      8U
#define ESYS_TR_PCR9      9U
#define ESYS_TR_PCR10    10U
#define ESYS_TR_PCR11    11U
#define ESYS_TR_PCR12    12U
#define ESYS_TR_PCR13    13U
#define ESYS_TR_PCR14    14U
#define ESYS_TR_PCR15    15U
#define ESYS_TR_PCR16    16U
#define ESYS_TR_PCR17    17U
#define ESYS_TR_PCR18    18U
#define ESYS_TR_PCR19    19U
#define ESYS_TR_PCR20    20U
#define ESYS_TR_PCR21    21U
```

```
#define ESYS_TR_PCR22   22U
#define ESYS_TR_PCR23   23U
#define ESYS_TR_PCR24   24U
#define ESYS_TR_PCR25   25U
#define ESYS_TR_PCR26   26U
#define ESYS_TR_PCR27   27U
#define ESYS_TR_PCR28   28U
#define ESYS_TR_PCR29   29U
#define ESYS_TR_PCR30   30U
#define ESYS_TR_PCR31   31U

#define ESYS_TR_PASSWORD       0xffU
#define ESYS_TR_NONE           0xfffU

/* From TPM_RH_CONSTANTS */
#define ESYS_TR_RH_OWNER       0x101U
#define ESYS_TR_RH_NULL        0x107U
#define ESYS_TR_RH_LOCKOUT     0x10AU
#define ESYS_TR_RH_ENDORSEMENT 0x10BU
#define ESYS_TR_RH_PLATFORM    0x10CU
#define ESYS_TR_RH_PLATFORM_NV 0x10DU
#define ESYS_TR_RH_AUTH_FIRST  0x110U
#define ESYS_TR_RH_AUTH(x)     (ESYS_TR_RH_AUTH_FIRST + (ESYS_TR)(x))
```

6 ESAPI Context Administration Functions

An ESYS_CONTEXT is a repository for context information for an application using the ESAPI.

There are two primary types of data that are stored within an ESYS_OBJECT – TPM session data and TPM resource data.

TPM session data is used by the ESAPI module to maintain state for a particular TPM session and enable the ESAPI module to use that session to securely communicate with the TPM. This may include information such as the session key, session attributes and session defaults. Some of this information (such as the session key) is likewise held by the TPM, while other information (such as session default attributes) is specifically intended to simplify the requirements on the application and are not shared with the TPM.

TPM resource data is used by the ESAPI module to correlate the handle of a TPM resource with information that the ESAPI module maintains related to that resource. The resource data will typically include the public information about the resource and also the corresponding authorization value (authValue) that can be used to authorize use of that resource.

NOTE: The TPM resource data stored within an ESYS_CONTEXT does not separately include the policy digest (authPolicy) associated with the resource. The ESAPI module does not need to operate on the authPolicy when authorizing commands using a policy session.

The ESAPI includes methods to get and set fields within the ESYS_CONTEXT to enable maximum flexibility. It is not recommended for applications to directly manipulate data within the ESYS_CONTEXT.

The ESAPI implementation SHALL return all "unhandled" error codes from lower layers in the TSS stack to the ESAPI caller without alteration.

6.1 Esys_Initialize()

Description:

The Esys_Initialize() functions initializes an ESYS_CONTEXT context within the ESAPI module supplied memory. The ESAPI context will be used for subsequent ESAPI functions that require an open ESAPI context.

Definition:

```
TSS2_RC Esys_Initialize(
    ESYS_CONTEXT      **esysContext,
    TSS2_TCTI_CONTEXT  *tcti,
    TSS2_ABI_VERSION   *abiVersion);
```

Parameters:

- The 'esysContext' is a reference to a pointer, pointing to the opaque ESYS_CONTEXT blob. It is allocated by the ESAPI. The 'esysContext' MUST NOT be NULL.
- The 'tcti' is a pointer to a TCTI_CONTEXT that will be used by ESAPI to communicate with the TPM. If 'tcti' is NULL the ESAPI will open a connection to the local TPM. The means of this connection – i.e. the TCTI type – is implementation specific, but it MUST refer to the local TPM.
- The 'abiVersion' is a pointer to the ABI_VERSION that the application requests. If this does not match the ESAPI's ABI_VERSION then an error is returned and the ESAPI's default ABI_VERSION is written to those fields. The 'abiVersion' parameter can be NULL. For more information on ABI negotiation see Section 7.10.1.2 of the TSS System Level API and TPM Command Transmission Interface Specification.

Response Codes:

- **TSS2_RC_SUCCESS: if the function call was a success.**
- **TSS2_ESYS_RC_BAD_REFERENCE: if esysContext is NULL.**

- **TSS2_ESYS_RC_MEMORY: if the ESAPI cannot allocate enough memory to create the context.**
- **TSS2_ESYS_RC_ABI_MISMATCH: if the ABI versions of the caller and ESAPI are incompatible.**
- **TSS2_ESYS_RC_INCOMPATIBLE_TCTI: if the TCTI version is unknown or an unusable version of the structure.**
- **TSS2_ESYS_RC_BAD_TCTI_STRUCTURE: if the required TCTI function pointers are NULL.**

6.2 Esys_Finalize()

Description:

The Esys_Finalize() function closes an ESPI_CONTEXT that was initialized by a call to Esys_Initialize(). The ESYS_CONTEXT structure will be cleared. The application SHOULD have flushed all ESYS_TR objects by this time, so that on systems without a auto-flushing ResourceManager the TPM does not become cluttered with objects that are supposed to be flushed.

Definition:

```
void Esys_Finalize(
    ESYS_CONTEXT      **esysContext);
```

Parameters:

- The ‘esysContext’ references the ESYS_CONTEXT to be finalized. The ‘esysContext’ parameter contains the reference to a pointer for this ESYS_CONTEXT. The memory pointed to will be freed and the pointer set to NULL. If the reference or the referenced pointer is NULL, nothing will be done.

Response Codes:

- None

6.3 Esys_GetTcti()

Description:

The Esys_GetTcti() function is provided to allow cleanup of a user-provided TCTI when an ESYS_CONTEXT is closed.

Definition:

```
TSS2_RC Esys_GetTcti(
    ESYS_CONTEXT       *esysContext,
    TSS2_TCTI_CONTEXT **tcti);
```

Parameters:

- The ‘esysContext’ is a pointer to the opaque context blob currently being operated on. This pointer MUST NOT be NULL.
- The ‘tcti’ is a reference to a pointer for a TCTI context. The ‘tcti’ parameter MUST NOT be NULL. The ‘tcti’ will be filled with the pointer that was passed into the ‘tcti’ parameter of the corresponding Esys_Initialize() function for this context. The output value for the Esys_GetTcti() function may be NULL if NULL was passed into Esys_Initialize().

Return Values:

- **TSS2_RC_SUCCESS: if the function call was a success.**
- **TSS2_ESYS_RC_BAD_REFERENCE: if esysContext or tcti is NULL.**
- **TSS2_ESYS_RC_BAD_CONTEXT: if esysContext corruption is detected.**

6.4 Esys_ GetSysContext ()

Description:

The Esys_GetSysContext () function is provided to obtain access to the System API (SAPI) context for lower level access to the TPM. For example, this can be used to get the cp buffer contents.

Definition:

```
TSS2_RC Esys_GetSysContext(
    ESYS_CONTEXT       *esysContext,
    TSS2_SYS_CONTEXT **sysContext);
```

Parameters:

- The ‘esysContext’ is a pointer to the opaque context blob currently being operated on. This pointer MUST NOT be NULL.
- The ‘sysContext’ is a reference to a pointer to a SAPI context. The ‘sysContext’ parameter MUST NOT be NULL. The sysContext will be filled with the pointer generated in Esys_Initialize().

Return Values:

- **TSS2_RC_SUCCESS: if the function call was a success.**
- **TSS2_ESYS_RC_BAD_REFERENCE: if esysContext or tcti is NULL.**
- **TSS2_ESYS_RC_BAD_CONTEXT: if esysContext corruption is detected.**
- **TSS2_ESYS_RC_NOT_IMPLEMENTED: if the ESAPI implementation does not support retrieving a SAPI context.**

6.5 Esys_GetPollHandles()

Description:

The Esys_GetPollHandles() function returns the handles that can be used for poll() or select() or WaitForMultipleObjects(). This function returns a set of handles that can be used to poll for incoming responses from the underlying software stack or TPM. The type of these handles is platform-specific and defined separately in the declaration of TSS2_TCTI_POLL_HANDLE.

Definition:

```
TSS2_RC Esys_GetPollHandles(
    ESYS_CONTEXT           *esysContext,
    TSS2_TCTI_POLL_HANDLE **handles,
    size_t                 *count);
```

Parameters:

- The ‘esysContext’ is a pointer to the opaque context blob currently being operated on. This pointer MUST NOT be NULL.
- The ‘handles’ output parameter is a reference to a pointer to an array of TCTI_POLL_HANDLEs. The array is allocated by the ESAPI and will be freed using the Esys_Free() function. This pointer MUST NOT be NULL.
- The ‘count’ output parameter is a reference to the number of poll handles that will be allocated and returned in the ‘handles’ parameter. This pointer MUST NOT be NULL.

Return Values:

- **TSS2_RC_SUCCESS: if the function call was a success.**
- **TSS2_ESYS_RC_BAD_REFERENCE: if esysContext, count, or handles is NULL.**
- **TSS2_ESYS_RC_BAD_CONTEXT: if esysContext corruption is detected.**
- **TSS2_TCTI_RC_NOT_IMPLEMENTED: if TCTI does not support returning poll handles.**
- **TSS2_ESYS_RC_MEMORY: if the ESAPI cannot allocate enough memory for the response array.**

6.6 Esys_SetTimeout()

Description:

The Esys_SetTimeout() function sets the behavior of the Esys_<COMMAND_NAME>_Finish() functions regarding blocking vs non-blocking and timeouts.

NOTE: The default behavior is non-blocking (i.e. a value of 0).

Definition:

```
TSS2_RC Esys_SetTimeout(
    ESYS_CONTEXT *esysContext,
    int32_t       timeout);
```

Parameters:

- The 'esysContext' is a pointer to the opaque context blob currently being operated on. It MUST NOT be NULL.
- The 'timeout' parameter sets the timeout behavior of the esysContext's Esys_<COMMAND_NAME>_Finish() functions:
    - -1: Blocking: Block until a result or error is available.
    - 0: Return immediately: The Esys_<COMMAND_NAME>_Finish() returns TSS2_ESYS_RC_TRY_AGAIN if result is not ready yet.
    - 1..INT32_MAX: Timeout in milliseconds: The Esys_<COMMAND_NAME>_Finish() returns TSS2_ESYS_RC_TRY_AGAIN if result is not ready within the given number of milliseconds.

Return Values:

- **TSS2_RC_SUCCESS: if the function call was a success.**
- **TSS2_ESYS_RC_BAD_REFERENCE: if the esysContext is NULL.**
- **TSS2_ESYS_RC_BAD_CONTEXT: if esysContext corruption is detected.**
- **TSS2_ESYS_RC_BAD_VALUE: if the supplied timeout is out of range.**

6.7 Esys_Free()

Description:

Many Esys_* functions allocate memory for output parameters (so called callee-allocated memory). This Esys_Free() function frees this allocated memory.

The function's implementation SHOULD map to the free function that corresponds to the allocation functions of the ESAPI library, e.g. free() for malloc/calloc() in C or delete() for new() in C++.

The rationale is that on some operating systems, a library and an application may be linked aganst different C runtime libraries. Even if calloc() based allocation was demanded by this specification, this circumstance can result in different heaps used for the library and the application. In those cases the application's free() function would not be able to free the memory allocated by the library. **Note**: Even if the application and the ESAPI library are known to use the same C runtime library and the standard free() function could be used directly, it SHOULD NOT be used for reasons of portability.

Definition:

```
void Esys_Free(
    void *ptr);
```

Parameters:

- The pointer to memory to be freed.

Return Values:

- **None**

7 ESAPI General ESYS_TR Functions

7.1 Esys_TR_Serialize()

Description:

The Esys_TR_Serialize() function returns a serialized representation of the metadata for an ESYS_TR object. The data returned by this function can be passed to Esys_TR_Deserialize to recreate this object in a different ESYS_CONTEXT such as in a new program invocation.

The primary purpose of this function is to recreate ESYS_TR objects that represent persistent TPM objects or to recreate TPM NV indices in a different ESYS_CONTEXT than they were created in.

NOTE: The canonical representation of this object is implementation specific. Auth values MUST NOT be part of the serialized blob.
NOTE: The auth value has to be set again after deserialization of the object.

Definition:

```
TSS2_RC Esys_TR_Serialize(
    ESYS_CONTEXT       *esysContext,
    ESYS_TR             object,
    uint8_t           **buffer,
    size_t             *buffer_size);
```

Parameters:

- The ‘esysContext’ is a pointer to the opaque context blob currently being operated on. It MUST NOT be NULL.
- The ‘object’ parameter contains the identifier of the ESYS_TR object that will be serialized.
- The ‘buffer’ output parameter contains a reference to a pointer to a byte buffer. This buffer is allocated by the ESAPI and contains a canonical representation of the ESYS_TR object. The allocated memory will be freed using the Esys_Free() function. MAY be NULL, in which case only the required size is returned via the ‘buffer_size’ parameter.
- The ‘buffer_size’ output parameter contains a reference to the size value of the ‘buffer’ parameter. It MUST NOT be NULL.

Return Values:

- **TSS2_RC_SUCCESS: if the function call was a success.**
- **TSS2_ESYS_RC_BAD_REFERENCE: if esysContext or buffer_size is NULL.**
- **TSS2_ESYS_RC_BAD_CONTEXT: if esysContext corruption is detected.**
- **TSS2_ESYS_RC_MEMORY: if the ESAPI cannot allocate enough memory for the response array.**
- **TSS2_ESYS_RC_BAD_TR: if the ESYS_TR object is unknown to the ESYS_CONTEXT.**

7.2 Esys_TR_Deserialize()

Description:

The Esys_TR_Deserialize() function creates an ESYS_TR object from a serialized buffer created by Esys_TR_Serialize. It can be used to recreate this object in a different ESYS_CONTEXT such as in a new program invocation.

The primary purpose of this function is to recreate ESYS_TR objects that represent persistent TPM objects or TPM NV indices in a different ESYS_CONTEXT than in which they were created.

NOTE: The canonical representation of this object is implementation specific.

NOTE: Auth values MUST NOT be part of the serialized blob. The initial authValue is an authValue of length zero. The TPM treats it the same as an authValue of '\0'.

Definition:

```
TSS2_RC Esys_TR_Deserialize(
    ESYS_CONTEXT        *esysContext,
    uint8_t      const *buffer,
    size_t              buffer_size,
    ESYS_TR            *object);
```

Parameters:

- The 'esysContext' is a pointer to the opaque context blob currently being operated on. It MUST NOT be NULL.
- The 'buffer' parameter contains a pointer to a byte buffer. This buffer is allocated by the ESAPI and contains a canonical representation of the ESYS_TR object. It MUST NOT be NULL.
- The 'buffer_size' parameter contains the size value of the 'buffer' parameter.
- The 'object' output parameter contains a reference to the identifier of the ESYS_TR object that will be serialized. It MUST NOT be NULL.

Return Values:

- **TSS2_RC_SUCCESS: if the function call was a success.**
- **TSS2_ESYS_RC_BAD_REFERENCE: if esysContext, buffer or object is NULL.**
- **TSS2_ESYS_RC_BAD_CONTEXT: if esysContext corruption is detected.**
- **TSS2_ESYS_RC_MEMORY: if the ESAPI cannot allocate enough internal memory to represent the object.**

7.3 Esys_TR_FromTPMPublic()

The Esys_TR_FromTPMPublic() function constructs an ESYS_TR object for a given TPM2_HANDLE.

Description:

This can be used to construct an ESYS_TR object for resources inside the TPM for which no serialized blob exists. This is done using the TPM commands TPM2_ReadPublic, TPM2_Getcapability and TPM2_NV_ReadPublic. Typically, these ESYS_TR objects are created either by Esys_TR_Deserialize() or by TPM functions such as Esys_Load(), Esys_StartAuthSession() or Esys_CreatePrimary. Typically, Esys_TR_FromTPMPublic() is used for persistent objects, sessions and NV objects not created by the application itself. Optionally, an HMAC and/or encryption session can be passed into Esys_TR_FromTPMPublic() in order to authenticate that the public data read using this function actually originates from the expected TPM. It is recommended that the caller use a salted HMAC session for NV index handles. For all other handles (besides NV-Index) this function can be used only if the path to the TPM can be trusted. In addition to NV and persistent object handles, Esys_TR_FromTPMPublic() SHALL also support converting sessions represented as a TPM2_HANDLE to an ESYS_TR. The resulting ESYS_TR session object has limited command functionality, notably useful for Esys_FlushContext().

NOTE: If a session is passed into this function, then the ReadPublic command with the TPM will be called twice. The first call is executed without a session in order to retrieve the public name that is then used in the HMAC calculation for the second call. For the synchronous invocation the two TPM calls are made transparently for the application.

NOTE: Since this operation requires interaction with the TPM, an asynchronous version of this function also exists. For the asynchronous invocation with sessions, the function Esys_TR_FromTPMPublic_Finish() will need to be called twice. The first successful invocation will always return TSS2_ESYS_RC_TRY_AGAIN regardless of timeout settings.

NOTE: The initial authValue for the returned object is an authValue of length zero. The TPM treats it the same as an authValue of '\0'.

NOTE: Handles of type TPM2_HANDLE that represent sessions will result in a degraded ESYS_TR object that is only suitable for Esys_FlushContext().

ESAPI implementations MUST evaluate the name against the public area of the object it refers to.

Definition:

```
TSS2_RC Esys_TR_FromTPMPublic(
    ESYS_CONTEXT        *esysContext,
    TPM2_HANDLE          tpm_handle,        /* required */
    ESYS_TR              optionalSession1, /* optional (ESYS_TR_NONE) */
    ESYS_TR              optionalSession2, /* optional (ESYS_TR_NONE) */
    ESYS_TR              optionalSession3, /* optional (ESYS_TR_NONE) */
    ESYS_TR             *object            /* required (non-NULL) */
);

TSS2_RC Esys_TR_FromTPMPublic_Async(
    ESYS_CONTEXT        *esysContext,
    TPM2_HANDLE          tpm_handle,        /* required */
    ESYS_TR              optionalSession1, /* optional (ESYS_TR_NONE) */
    ESYS_TR              optionalSession2, /* optional (ESYS_TR_NONE) */
    ESYS_TR              optionalSession3  /* optional (ESYS_TR_NONE) */
);

TSS2_RC Esys_TR_FromTPMPublic_Finish(
    ESYS_CONTEXT        *esysContext,
    ESYS_TR             *object            /* required (non-NULL) */
);
```

Parameters:

- The 'esysContext' is a pointer to the opaque context blob currently being operated on. It MUST NOT be NULL.
- The 'tpm_handle' parameter references the TPM object for which the ESYS_TR object is being created.
- The 'object' output parameter is a reference to an ESYS_TR object identifier for the newly created ESYS_TR object. It MUST NOT be NULL.
- The 'optionalSession' parameter can be used in order to pass an HMAC session, encryption session or audit session. This parameter can also be ESYS_TR_NONE. It is recommended to pass in one salted HMAC session for untrusted connections to a TPM.

Return Values:

- **TSS2_RC_SUCCESS: if the function call was a success.**
- **TSS2_ESYS_RC_BAD_REFERENCE: if esysContext or object is NULL.**
- **TSS2_ESYS_RC_BAD_CONTEXT: if esysContext corruption is detected.**
- **TSS2_ESYS_RC_MEMORY: if the ESAPI cannot allocate enough internal memory to represent the object.**
- **TSS2_ESYS_RC_BAD_TR: if any the ESYS_TR optional sessions are unknown to the ESYS_CONTEXT.**
- **TSS2_ESYS_RC_BAD_SEQUENCE: if the ESYS Async and Finish functions are not called in matching pairs.**

- **TSS2_ESYS_RC_BAD_VALUE: if the TPM2_HANDLE tpm_handle is not a valid TPM handle.**
- **TSS2_TCTI_RC_IO_ERROR: if the communication with the TPM fails on an I/O level.**
- **TSS2_ESYS_RC_RSP_AUTH_FAILED: if the response HMAC from the TPM could not be verified.**
- **TSS2_TCTI_RC_TRY_AGAIN: if the timeout was reached or non-blocking mode was used, the function needs to be called again.**

7.4 Esys_TR_SetAuth()

The Esys_TR_SetAuth() function sets the authentication value associated with an ESYS_TR object.

Description:

This authentication value is used if the provided ESYS_TR object is used in a command that requires authValue based authentication, such as password-authentication (when no session is provided), HMAC-authentication, or PolicySecret- or PolicyPassword-enabled policy sessions.

NOTE: This authentication value is stored in the esysContext. In order to delete this authentication value from memory, the Esys_TR_SetAuth() function can be called using NULL for 'authValue'.

NOTE: The authentication value is not part of a serialization blob for an ESYS_TR object. After serializing and deserializing an ESYS_TR object, the authentication value needs to be set again by the caller.

Definition:

```
TSS2_RC Esys_TR_SetAuth(
    ESYS_CONTEXT       *esysContext,
    ESYS_TR             handle,        /* required */
    TPM2B_AUTH   const *authValue      /* optional (NULL) */
);
```

Parameters:

- The 'esysContext' is a pointer to the opaque context blob currently being operated on. It MUST NOT be NULL.
- The 'handle' is the identifier of the ESYS_TR object for which the 'authValue' is set.
- The 'authValue' parameter is a pointer to the TPM2B_AUTH that contains the password to be set. The ESAPI implementation will copy this value into its internal state. If 'authValue' is NULL it is interpreted as deleting the authorization value associated with the object (i.e. resetting the object's authValue to its creation value, which is the same as passing in an authValue of length zero or an authValue of '\0').

Return Values:

- **TSS2_RC_SUCCESS: if the function call was a success.**
- **TSS2_ESYS_RC_BAD_REFERENCE: if the esysContext is NULL.**
- **TSS2_ESYS_RC_BAD_CONTEXT: if esysContext corruption is detected.**
- **TSS2_ESYS_RC_MEMORY: if the ESAPI cannot allocate enough internal memory to save the authorization value.**
- **TSS2_ESYS_RC_BAD_TR: if any ESYS_TR handle is unknown to the ESYS_CONTEXT.**

7.5 Esys_TR_GetName()

Description:

The Esys_TR_GetName() function returns the name of an ESYS_TR object to the caller for further usage, such as Esys_PolicyNameHash().

Definition:

```
TSS2_RC Esys_TR_GetName(
    ESYS_CONTEXT       *esysContext,
    ESYS_TR             handle,        /* required */
    TPM2B_NAME        **name           /* required */
);
```

Parameters:

- The 'esysContext' is a pointer to the opaque context blob currently being operated on. It MUST NOT be NULL.
- The 'handle' is the identifier of the ESYS_TR object for which the 'name' is retrieved.
- The 'name' parameter is a reference to a pointer to the TPM2B_NAME for the object's name. The TPM2B_NAME will be allocated by the ESAPI and will be be freed by the caller.

Return Values:

- **TSS2_RC_SUCCESS: if the function call was a success.**
- **TSS2_ESYS_RC_BAD_REFERENCE: if the esysContext or name are NULL.**
- **TSS2_ESYS_RC_BAD_CONTEXT: if esysContext corruption is detected.**
- **TSS2_ESYS_RC_MEMORY: if the ESAPI cannot allocate enough memory to return the name.**
- **TSS2_ESYS_RC_BAD_TR: if any ESYS_TR handle is unknown to the ESYS_CONTEXT.**

7.6 Esys_TR_Close()

Description:

The Esys_TR_Close() function instructs the ESAPI to release the metadata and resources allocated for a specific ESYS_TR object. This is useful for objects representing persistent keys and NV indices, if they are no longer used and/or have been serialized. This function does not free TPM resources associated with the ESYS_TR object.

For transient objects it is recommended to use Esys_FlushContext() instead, which also frees the TPM resources.

Definition:

```
TSS2_RC Esys_TR_Close(
    ESYS_CONTEXT *esysContext,
    ESYS_TR      *object);
```

Parameters:

- The 'esysContext' is a pointer to the opaque context blob currently being operated on. It MUST NOT be NULL.
- The 'object' parameter MUST be a reference to an ESYS_TR identifier for the object to be closed. The reference MUST NOT be NULL and the referenced ESYS_TR value MUST NOT be ESYS_TR_NONE. After successful completion of the function the referenced ESYS_TR value will be set to ESYS_TR_NONE.

Return Values:

- **TSS2_RC_SUCCESS: if the function call was a success.**
- **TSS2_ESYS_RC_BAD_REFERENCE: if the esysContext or object are NULL.**
- **TSS2_ESYS_RC_BAD_CONTEXT: if esysContext corruption is detected.**
- **TSS2_ESYS_RC_BAD_TR: if any ESYS_TR *object is unknown to the ESYS_CONTEXT or is ESYS_TR_NONE.**

7.7 Esys_TR_GetTpmHandle()

Description:

The Esys_TR_GetTpmHandle() function retrieves the TPM2_HANDLE from an ESYS_TR object. This can be used to retrieve the TPM2_HANDLE from an ESYS_TR object for use with the SAPI or comparison against raw handle values. For instance, when retrieving persistent handles from the TPM via Esys_GetCapabilities(), the handle values are of type TPM2_HANDLE or when using TPM2_HR masks.

Definition:

```
TSS2_RC Esys_TR_GetTpmHandle (
    ESYS_CONTEXT        *esysContext,/* required */
    ESYS_TR             object,      /* required */
    TPM2_HANDLE         *tpm_handle  /* required */
);
```

Parameters:

- The 'esysContext' is a pointer to the opaque context blob currently being operated on. It MUST NOT be NULL.
- The 'object' parameter is an ESYS_TR object identifier for the newly created ESYS_TR object.
- The 'tpm_handle' output parameter references a TPM2_HANDLE associated with the ESYS_TR object. It MUST NOT be NULL.

Return Values:

- **TSS2_RC_SUCCESS: if the function call was a success.**
- **TSS2_ESYS_RC_BAD_REFERENCE: if the esysContext or tpm_handle are NULL.**
- **TSS2_ESYS_RC_BAD_CONTEXT: if esysContext corruption is detected.**
- **TSS2_ESYS_RC_BAD_TR: if any ESYS_TR object is unknown to the ESYS_CONTEXT or is ESYS_TR_NONE.**

7.8 TPM Commands Covered by General ESYS_TR Functions

Description:

The follow commands create or flush ESYS_TR objects. See Section 11 for a description.

- Esys_FlushContext
- Esys_ContextSave
- Esys_ContextLoad
- Esys_CreateLoaded
- Esys_CreatePrimary
- Esys_Load
- Esys_EvictControl
- Esys_NV_DefineSpace
- Esys_StartAuthSession
- Esys_HMAC_Start
- Esys_HashSequenceStart
- Esys_SequenceComplete

- Esys_EventSequenceComplete
- Esys_NV_UndefineSpace
- Esys_NV_UndefineSpaceSpecial

8 ESAPI Session Administration Functions

This section describes functions for session administration provided by the ESAPI which do not interact with the TPM. These functions are additional functions provided by the ESAPI for host-side management of sessions. The TPM session administration functions provided by the TPM are also supported using the template described in section 9.

8.1 Esys_TRSess_SetAttributes()

Description:

The Esys_TRSess_SetAttributes() function allows the setting and clearing of session attributes, such as continueSession, encrypt or decrypt. The new session attributes are calculated as follows:

```
attributes = (attributes & ~mask) | (flags & mask)
```

If Esys_TRSess_SetAttributes() is never called the default values for the session flags are: the 'continueSession' bit is set, all other bits are clear.

If the continueSession attribute is CLEAR, the ESYS_TR object for the session is invalidated after the next successful TPM response to a function call using the session. The ESAPI implementation MUST set the internal type/status of the ESYS_TR sequenceHandle to invalid, so it cannot be used anymore. Once ESYS_TR sequenceHandle is invalid, the ESYS_TR object no longer needs to be closed with Esys_TR_Close() or flushed with Esys_FlushContext().

Definition:

```
TSS2_RC Esys_TRSess_SetAttributes(
    ESYS_CONTEXT        *esysContext,
    ESYS_TR              session,
    TPMA_SESSION         flags,
    TPMA_SESSION         mask);
```

Parameters:

- The 'esysContext' is a pointer to the opaque context blob currently being operated on. It MUST NOT be NULL.
- The 'session' parameter refers to the session, for which the flags will be altered.
- The 'flags' parameter denote the flags to be set for this session.
- The 'mask' parameter indicates which bits are modified. Any bits that are clear in the mask will not be modified.
  NOTE: A mask of 0xff can be used to set all bits simultaneously.

Return Values:

- **TSS2_RC_SUCCESS: if the function call was a success.**
- **TSS2_ESYS_RC_BAD_REFERENCE: if the esysContext is NULL.**
- **TSS2_ESYS_RC_BAD_CONTEXT: if esysContext corruption is detected.**
- **TSS2_ESYS_RC_BAD_TR: if any the ESYS_TR session is unknown to the ESYS_CONTEXT or is not a session object.**

8.2 Esys_TRSess_GetAttributes()

Description:

The Esys_TRSess_GetAttributes() function returns the current setting of the sessionAttributes property of the session.

Definition:

```
TSS2_RC Esys_TRSess_GetAttributes(
    ESYS_CONTEXT       *esysContext,
    ESYS_TR             session,
    TPMA_SESSION       *flags);
```

Parameters:

- The ‘esysContext’ is a pointer to the opaque context blob currently being operated on. It MUST NOT be NULL.
- The ‘session’ parameter refers to the session, for which the flags will be returned.
- The ‘flags’ output parameter is a reference to where the flags of this session will be stored. It MUST NOT be NULL.

Return Values:

- **TSS2_RC_SUCCESS: if the function call was a success.**
- **TSS2_ESYS_RC_BAD_REFERENCE: if the esysContext or flags are NULL.**
- **TSS2_ESYS_RC_BAD_CONTEXT: if esysContext corruption is detected.**
- **TSS2_ESYS_RC_BAD_TR: if session is unknown to the ESYS_CONTEXT or is not a session object.**

8.3 Esys_TRSess_GetNonceTPM()

Description:

If Esys_StartAuthSession() is called with a non-NULL TPM key and non-NULL encrypted salt then this function, Esys_TRSess_GetNonceTPM(), is used to get the nonceTPM from the startAuthSession() command in order to calculate the session key.

Definition:

```
TSS2_RC Esys_TRSess_GetNonceTPM(
    ESYS_CONTEXT       *esysContext,
    ESYS_TR             session,
    TPM2B_NONCE       **nonceTPM);
```

Parameters:

- The ‘esysContext’ is a pointer to the opaque context blob currently being operated on. It MUST NOT be NULL.
- The ‘session’ parameter refers to the session, for which the flags are returned.
- The nonceTPM parameter is a refence to the pointer for storing the nonceTPM.

Return Values:

- **TSS2_RC_SUCCESS: if the function call was a success.**
- **TSS2_ESYS_RC_BAD_REFERENCE: if the esysContext or nonceTPM are NULL.**
- **TSS2_ESYS_RC_BAD_CONTEXT: if esysContext corruption is detected.**
- **TSS2_ESYS_RC_BAD_TR: if the ESYS_TR session is unknown to the ESYS_CONTEXT or is not a session object.**
- **TSS2_ESYS_RC_MEMORY: if the ESAPI cannot allocate enough memory for nonceTPM.**

9 ESAPI TPM Command Template

This specification defines a set of functions using a template format. For each command of Part 3 of the TPM 2.0 Library Specification revision 138 the following three functions exist in the ESAPI: Esys_<COMMAND_NAME>, Esys_<COMMAND_NAME>_Async, and Esys_<COMMAND_NAME>_Finish. In each case <COMMAND_NAME> is the name of the TPM command without its "TPM2_" prefix.

The ESAPI allows the invocation of TPM commands in a synchronous as well as an asynchronous manner.

- For synchronous invocation a (blocking) call to Esys_<COMMAND_NAME> is executed.
- For asynchronous invocation the functions Esys_<COMMAND_NAME>_Async and Esys_<COMMAND_NAME>_Finish need to be called in pairs.
  NOTE: If they are not called in pairs then TSS2_ESYS_RC_BAD_SEQUENCE is returned.

9.1 **Esys_<COMMAND_NAME>()**

Description:

The synchronous ESAPI TPM function template is used to create functions that cause the ESAPI to execute a TPM command in a synchronous manner. It will block during the TPM's execution.

The following are general concepts used in these function calls.

- Authorization Values: The handles passed into the function call carry the authorization values that will be used when the TPM requires authorization.
- The caller provides a session for the command and the session's attributes will specify whether the caller wants encryption/decryption. When a session is created, the HMAC parameters (SHA-256 or SHA-1, or other hash algorithms) are provided by the caller.
- Sessions: If a resource passed in via an inHandle requires authorization then:
  - If the corresponding HMAC or policy session is provided the ESAPI calculates the HMAC automatically.
  - If a policy session is provided with PolicyPassword active then the authorization value is passed automatically into the HMAC calculation by ESAPI.
  - If ESYS_TR_PASSWORD is provided in the corresponding session parameter then a password authorization is performed by ESAPI instead of an HMAC calculation.
  - If a session is a bound session then this is handled automatically by ESAPI by matching the object names.
    NOTE: if the name for a persistent TPM object is changed by another ESYS_CONTEXT this cannot be detected by ESAPI. Consequentially the resolution of bound sessions might fail.
- Parameter Encryption:
  - Decrypt Bit: If exactly one of the sessions has the decryption bit set and the command supports encryption of the first parameter of the command then that parameter will be encrypted before being sent to the TPM.
  - Encrypt Bit: If exactly one of the sessions has the encryption bit set and the command supports the encryption of the first parameter of the response then the TPM will send it encrypted and the ESAPI will decrypt it before unmarshaling it and returning it to the application.

- TPM names: When the TPM returns a name for an object, these parameters are recorded by the ESAPI inside the associated ESYS_TR object and therefore not returned to the caller. Hence, the output parameter's 'name' is not present in Esys_Load, Esys_CreatePrimary, Esys_LoadExternal and Esys_CreateLoaded. This rule does not apply to Esys_ReadPublic() and Esys_NV_ReadPublic(), which have an output parameter 'name'.
- TPM error codes: Certain TPM error codes require the resubmission of commands. When the TPM returns TPM2_RC_RETRY, TPM2_RC_YIELDED or TPM2_RC_TESTING, the ESAPI MUST resubmit the command on behalf of the caller at least once. ESAPI implementations SHOULD establish a limit on the number or duration of retries that will be made before the original TPM error code is sent back to the caller.

Definition:

```
 TSS2_RC Esys_<COMMAND_NAME>(
        ESYS_CONTEXT            *esysContext,
        ESYS_TR                  inHandle1,
        ESYS_TR                  inHandle2,
        ESYS_TR                  session1,  /* optional (ESYS_TR_NONE) */
        ESYS_TR                  session2,  /* optional (ESYS_TR_NONE) */
        ESYS_TR                  session3,  /* optional (ESYS_TR_NONE) */
        TPM2_BASEEXAMPLE         inParams,
        TPMA_EXAMPLE             inParams,
        TPM2B_EXAMPLE     const *inParams,  /* optional (NULL) */
        TPML_EXAMPLE      const *inParams,  /* required (non-NULL) */
        TPMS_EXAMPLE      const *inParams,  /* required (non-NULL) */
        TPMT_EXAMPLE      const *inParams,  /* required (non-NULL) */
        ESYS_TR                 *outHandle, /* required (non-NULL) */
        TPM2_BASEEXAMPLE        *outParams, /* optional (NULL-Pointer)*/
        TPMA_EXAMPLE            *outParams, /* optional (NULL-Pointer)*/
        TPM2B_EXAMPLE          **outParams, /* callee-
allocated (use Esys_Free()), optional (NULL-Pointer)*/
        TPML_EXAMPLE           **outParams, /* callee-
allocated (use Esys_Free()), optional (NULL-Pointer)*/
        TPMS_EXAMPLE           **outParams  /* callee-allocated (use
Esys_Free()), optional (NULL-Pointer)*/
);
```

Parameters:

- The 'esysContext' is a pointer to the opaque context blob currently being operated on. It MUST NOT be NULL.
- The 'inHandle' parameters contain the identifiers of ESYS_TR objects to be used as command parameters. The number of these parameters depends on the specific TPM command. Their names are constructed from the TPM command parameters with the word "Handle" appended to them (e.g. "keyHandle").
- The 'session' parameters contain the identifiers for ESYS_TR session objects. Their names are constructed based on the parameters of the TPM command. If the session is used for authenticating one of the handles, the name consists of the handle's name followed by "Session" and the index of the session (e.g. "keyHandleSession2"). If the session is not used for authentication then the name is "optionalSession" followed by the index of the session (e.g. 1, 2 or 3).
  If ESYS_TR_PASSWORD is passed in, a password session is used for authentication instead of an HMAC session. See Trusted Platform Module Library Part 1 Section 19 for an explanation of password sessions.
- The 'inParams' parameters represent the TPM command parameters.

  - If ‘inParams’ refers to a simple type (such as UINT16) or a bitmask (such as TPMA_KEY), they are passed by value.
  - If ‘inParams’ refers to a non-simple type (such as TPM2B_*, TPML_* or TPMS_*) they are passed in as pointers to constant structures.
  - If ‘inParams’ refers to a TPM2B_* then NULL may be passed in, which is interpreted as a zero-length TPM2B (i.e. a TPM2B with the .size field being zero).
    NOTE: This applies to TPM2B_* structs with embedded arrays as well as with embedded TPMT_*, which is required for e.g. Esys_LoadExternal() when loading a public key without an associated private key.
  - If ‘inParams’ refers to a TPM2B_* with an embedded TPMT_* (i.e. not with an embedded array), then the size-field is ignored.
- The ‘outHandle’ output parameter is present if the TPM creates a new TPM object. The ‘outHandle’ is the reference to the identifier of the newly created ESYS_TR object corresponding to the created TPM object. It MUST NOT be NULL.
- The ‘outParams’ parameter represents the TPM response parameters.
  - If ‘outParams’ refers to a simple type (such as UINT16) or a bitmask (such as TPMA_KEY), a reference to the output variable is passed in. ‘outParams’ value may be NULL.
  - If ‘outParams’ refers to a non-simple type (such as TPM2B_*, TPML_* or TPMS_*) they are passed in as references to a pointer for this type. The ESAPI will allocate memory for the type and return its pointer in this parameter. The memory will be freed by the application using Esys_Free(). ‘outParams’ value may be NULL.

Return Values:

- **TSS2_RC_SUCCESS: if the function call was a success.**
- **TSS2_ESYS_RC_BAD_REFERENCE: if the esysContext or required input pointers or required output handle references are NULL.**
- **TSS2_ESYS_RC_BAD_CONTEXT: if esysContext corruption is detected.**
- **TSS2_ESYS_RC_BAD_TR: if any of the ESYS_TR objects are unknown to the ESYS_CONTEXT or are of the wrong type or if required ESYS_TR objects are ESYS_TR_NONE.**
- **TSS2_ESYS_RC_MEMORY: if the ESAPI cannot allocate enough memory for internal operations or return parameters.**
- **TSS2_ESYS_RC_BAD_SEQUENCE: if the context has an asynchronous operation already pending.**
- **TSS2_ESYS_RC_MULTIPLE_DECRYPT_SESSIONS: if more than one session has the 'decrypt' attribute bit set.**
- **TSS2_ESYS_RC_MULTIPLE_ENCRYPT_SESSIONS: if more than one session has the 'encrypt' attribute bit set.**
- **TSS2_ESYS_RC_NO_DECRYPT_PARAM: if one of the sessions has the 'decrypt' attribute set and the command does not support encryption of the first command parameter.**
- **TSS2_ESYS_RC_NO_ENCRYPT_PARAM: if one of the sessions has the 'encrypt' attribute set and the command does not support encryption of the first response parameter.**
- **TSS2_ESYS_RC_INSUFFICIENT_RESPONSE: if the TPM's response does not contain the tag, response length, or response code.**
- **TSS2_ESYS_RC_MALFORMED_RESPONSE: if the TPM's response is corrupted.**
- **TSS2_ESYS_RC_RSP_AUTH_FAILED: if the response HMAC from the TPM could not be verified.**
- **TSS2_RCs produced by lower layers of the software stack may be returned to the caller unaltered unless handled internally.**

9.2 Esys_<COMMAND_NAME>_Async()

Description:

This function template is used to create functions that cause the ESAPI to execute a TPM command in an asynchronous manner. It will not block while the TPM executes the command. The corresponding Esys_<COMMAND_NAME>_Finish needs to be called next to retrieve the TPM's response parameters.

The following are general concepts used in these function calls.

- Authorization Values: The handles passed into the function call carry the authorization values that will be used when the TPM requires authorization.
- The caller provides a session for the command and the session's attributes will specify whether the caller wants encryption/decryption. When a session is created, the HMAC parameters (SHA-256 or SHA-1, or other hash algorithms) are provided by the caller.
- Sessions: If a resource passed in via an inHandle requires authorization then:
    - If the corresponding HMAC or policy session is provided the ESAPI calculates the HMAC automatically.
    - If a policy session is provided with PolicyPassword active then the authorization value is passed automatically into the HMAC calculation by ESAPI.
    - If ESYS_TR_PASSWORD is provided in the corresponding session parameter then a password authorization is performed by ESAPI instead of an HMAC calculation.
    - If a session is a bound session then this is handled automatically by ESAPI by matching the object names.
      NOTE: if the name for a persistent TPM object is changed by another ESYS_CONTEXT this cannot be detected by ESAPI. Consequentially the resolution of bound sessions might fail.
- Parameter Encryption:
    - Decrypt Bit: If exactly one of the sessions has the decryption bit set and the command supports encryption of the first parameter of the command then that parameter will be encrypted before being sent to the TPM.
    - Encrypt Bit: If exactly one of the sessions has the encryption bit set and the command supports the encryption of the first parameter of the response then the TPM will send it encrypted and the ESAPI will decrypt it before unmarshaling it and returning it to the application.

Definition:

```
TSS2_RC Esys_<COMMAND_NAME>_Async(
        ESYS_CONTEXT            *esysContext,

        ESYS_TR                 inHandle1,
        ESYS_TR                 inHandle2,

        ESYS_TR                  session1,  /* optional (ESYS_TR_NONE) */
        ESYS_TR                  session2,  /* optional (ESYS_TR_NONE) */
        ESYS_TR                  session3,  /* optional (ESYS_TR_NONE) */
        TPM2_BASEEXAMPLE         inParams,
        TPMA_EXAMPLE             inParams,
        TPM2B_EXAMPLE      const *inParams,   /* TPM2Bs: optional
(NULL-Pointer) */
        TPML_EXAMPLE       const *inParams,   /* must not be NULL */
        TPMS_EXAMPLE       const *inParams    /* must not be NULL */
        TPMT_EXAMPLE       const *inParams    /* must not be NULL */
);
```

Parameters:

- The ‘esysContext’ is a pointer to the opaque context blob currently being operated on. It MUST NOT be NULL.
- The ‘inHandle’ parameters contain the identifiers of ESYS_TR objects to be used as command parameters. The number of these parameters depends on the specific TPM command. Their names are constructed from the TPM command parameters with the word “Handle” appended to them (e.g. “keyHandle”).
- The ‘session’ parameters contain the identifiers for ESYS_TR session objects. Their names are constructed based on the parameters of the TPM command. If the session is used for authenticating one of the handles, the name consists of the handle’s name followed by “Session” and the index of the session (e.g. “keyHandleSession2”). If the session is not used for authentication then the name is “optionalSession” followed by the index of the session (e.g. 1, 2 or 3).
  If ESYS_TR_PASSWORD is passed in, a password session is used for authentication instead of an HMAC session. See Trusted Platform Module Library Part 1 Section 19 for an explanation of password sessions.
- The ‘inParams’ parameters represent the TPM command parameters.
  - If ‘inParams’ refers to a simple type (such as UINT16) or a bitmask (such as TPMA_KEY), they are passed by value.
  - If ‘inParams’ refers to a non-simple type (such as TPM2B_*, TPML_* or TPMS_*) they are passed in as pointers to constant structures.
  - If ‘inParams’ refers to a TPM2B_* then NULL may be passed in, which is interpreted as a zero-length TPM2B (i.e. a TPM2B with the .size field being zero).
    NOTE: This applies to TPM2B_* structs with embedded arrays as well as with embedded TPMT_*, which is required for e.g. Esys_LoadExternal() when loading a public key without an associated private key.
  - If ‘inParams’ refers to a TPM2B_* with an embedded TPMT_* (i.e. not with an embedded array), then the size-field is ignored.

Return Values:

- **TSS2_RC_SUCCESS: if the function call was a success.**
- **TSS2_ESYS_RC_BAD_REFERENCE: if the esysContext or required input pointers are NULL.**
- **TSS2_ESYS_RC_BAD_CONTEXT: if esysContext corruption is detected.**
- **TSS2_ESYS_RC_BAD_TR: if any of the ESYS_TR objects are unknown to the ESYS_CONTEXT or are of the wrong type or if required ESYS_TR objects are ESYS_TR_NONE.**
- **TSS2_ESYS_RC_MEMORY: if the ESAPI cannot allocate enough memory for internal operations.**
- **TSS2_ESYS_RC_BAD_SEQUENCE: if the context has an asynchronous operation already pending.**

- **TSS2_ESYS_RC_MULTIPLE_DECRYPT_SESSIONS: if more than one session has the 'decrypt' attribute bit set.**
- **TSS2_ESYS_RC_MULTIPLE_ENCRYPT_SESSIONS: if more than one session has the 'encrypt' attribute bit set.**
- **TSS2_ESYS_RC_NO_DECRYPT_PARAM: if one of the sessions has the 'decrypt' attribute set and the command does not support encryption of the first command parameter.**
- **TSS2_ESYS_RC_NO_ENCRYPT_PARAM: if one of the sessions has the 'encrypt' attribute set and the command does not support encryption of the first response parameter.**
- **TSS2_RCs produced by lower layers of the software stack may be returned to the caller unaltered unless handled internally.**

9.3 **Esys_<COMMAND_NAME>_Finish()**

Description:

This function template is used to create functions that cause the ESAPI to get the results of an asynchonous TPM command. It will block according to the timeout set using Esys_SetTimeout (default is non-blocking). The corresponding Esys_<COMMAND_NAME>_Async needs to be called before the Esys_<COMMAND_NAME>_Finish() function to initiate the TPM command asynchronously.

When the TPM returns a name for an object, these parameters are recorded by the ESAPI inside the associated ESYS_TR object and therefore not returned to the caller. Hence, the output parameter's 'name' is not present in Esys_Load, Esys_CreatePrimary, Esys_LoadExternal and Esys_CreateLoaded. This rule does not apply to Esys_ReadPublic() and Esys_NV_ReadPublic(), which have an output parameter 'name'.

TPM error codes: Certain TPM error codes require the resubmission of commands. When the TPM returns TPM2_RC_RETRY, TPM2_RC_YIELDED or TPM2_RC_TESTING, the ESAPI MUST resubmit the command on behalf of the caller at least once.

ESAPI implementations SHOULD establish a limit on the number or duration of retries that will be made before the original TPM error code is sent back to the caller. If the Esys_<COMMAND_NAME>_Finish() call resubmits a command to the TPM it MUST return TSS2_ESYS_RC_TRY_AGAIN immediately after command re-submission to the TPM.

Note: There are a set of recoverable errors or non-error response codes (e.g. TSS2_ESYS_RC_TRY_AGAIN or TSS2_TCTI_RC_TRY_AGAIN). Therefore, the caller might have to call Esys_<COMMAND_NAME>_Finish multiple times in a row until a TPM response is ready or an IO-Error occurs.

Definition:

```
TSS2_RC Esys_<COMMAND_NAME>_Finish(
        ESYS_CONTEXT            *esysContext,
        ESYS_TR                 *outHandle, /* required (non-NULL) */
        TPM2_BASEEXAMPLE        *outParams, /* optional (NULL-Pointer)*/
        TPMA_EXAMPLE            *outParams, /* optional (NULL-Pointer)*/
        TPM2B_EXAMPLE          **outParams, /* callee-
allocated (use Esys_Free()), optional (NULL-Pointer)*/
        TPML_EXAMPLE           **outParams, /* callee-
allocated (use Esys_Free()), optional (NULL-Pointer)*/
        TPMS_EXAMPLE           **outParams  /* callee-allocated (use
Esys_Free()), optional (NULL-Pointer)*/
);
```

Parameters:

- The 'esysContext' is a pointer to the opaque context blob currently being operated on. It MUST NOT be NULL.
- The 'outHandle' output parameter is present if the TPM creates a new TPM object. 'outHandle' is the reference to the identifier of the newly created ESYS_TR object corresponding to the created TPM object. It MUST NOT be NULL.
- The 'outParams' parameter represents the TPM response parameters.
    - If 'outParams' refers to a simple type (such as UINT16) or a bitmask (such as TPMA_KEY), a reference to the output variable is passed in. May be NULL.
    - If 'outParams' refers to a non-simple type (such as TPM2B_*, TPML_* or TPMS_*) they are passed in as references to a pointer for this type. The ESAPI will allocate memory for the type and return its pointer in this parameter. The memory will be freed using Esys_Free() by the application. May be NULL.

Return Values:

- **TSS2_RC_SUCCESS: if the function call was a success.**
- **TSS2_ESYS_RC_BAD_REFERENCE: if the esysContext or required references are NULL.**
- **TSS2_ESYS_RC_BAD_CONTEXT: if esysContext corruption is detected.**
- **TSS2_ESYS_RC_MEMORY: if the ESAPI cannot allocate enough memory for internal operations or return parameters.**
- **TSS2_ESYS_RC_BAD_SEQUENCE: if the context does not have the corresponding asynchronous operation already pending.**
- **TSS2_ESYS_RC_TRY_AGAIN: if the timeout counter expires before the TPM response is received.**
- **TSS2_ESYS_RC_INSUFFICIENT_RESPONSE: if the TPM's response does not at least contain the tag, response length, and response code.**
- **TSS2_ESYS_RC_MALFORMED_RESPONSE: if the TPM's response is corrupted.**
- **TSS2_ESYS_RC_RSP_AUTH_FAILED: if the response HMAC from the TPM could not be verified.**
- **TSS2_RCs produced by lower layers of the software stack may be returned to the caller unaltered unless handled internally.**

10 ESAPI TPM Commands Requiring Special Handling

This section presents cases that have some differences from the standard template presented in section 9. These cases require some degree of special handling beyond what is described in the basic command template. This special handling is mostly requirements for the internal behavior of the ESAPI implementation. Only the differences are described. For everything else, the standard template given in section 9 applies.

10.1 Esys_StartAuthSession Commands

10.1.1 Esys_StartAuthSession()

The special handling for Esys_StartAuthSession() is the same as the special handling for Esys_StartAuthSession_Async() and Esys_StartAuthSession_Finish().

Definition:

```
TSS2_DLL_EXPORT TSS2_RC Esys_StartAuthSession(
    ESYS_CONTEXT                   *esysContext,
    ESYS_TR                         tpmKey,
    ESYS_TR                         bind,
    ESYS_TR                         optionalSession1,
    ESYS_TR                         optionalSession2,
    ESYS_TR                         optionalSession3,
    TPM2B_NONCE              const *nonceCaller,
    TPM2_SE                         sessionType,
    TPMT_SYM_DEF             const *symmetric,
    TPMI_ALG_HASH                   authHash,
    ESYS_TR                        *sessionHandle);
```

10.1.2 Esys_StartAuthSession_Async()

Definition:

```
TSS2_DLL_EXPORT TSS2_RC Esys_StartAuthSession_Async(
    ESYS_CONTEXT                   *esysContext,
    ESYS_TR                         tpmKey,
    ESYS_TR                         bind,
    ESYS_TR                         optionalSession1,
    ESYS_TR                         optionalSession2,
    ESYS_TR                         optionalSession3,
    TPM2B_NONCE              const *nonceCaller,
    TPM2_SE                         sessionType,
    TPMT_SYM_DEF             const *symmetric,
    TPMI_ALG_HASH                   authHash);
```

Parameter descriptions:

- tpmKey: Can be NULL if no salting is required (this represents the case of TPM_RH_NULL from the TPM’s library specification).
- bind: Can be NULL (this represents the case of TPM_RH_NULL from the TPM’s library specification).
- nonceCaller: Can be NULL. If NULL, then the ESAPI generates a nonceCaller value of appropriate length automatically.

Note: The TPM's encryptedSalt parameter is generated by the ESAPI implementation and thus not exposed.

Special Handling Required:

- Handling of tpmKey
    - If tpmKey is NULL
        - No special action needs to be performed, since no salt value is transferred to the TPM.
    - If tpmKey is non-NULL
        - ESAPI needs to check that tpmKey is suitable for encrypting salts. Return TSS2_ESYS_RC_BAD_TR if not suitable.
        - ESAPI generates a salt value using its internal RNG implementation.
        - This salt is encrypted using the tpmKey public key.
        - The encrypted salt is transferred to the TPM.
        - The plain salt value is used for session key calculation during Esys_StartAuthSession_Finish().
- If bind is non-NULL
    - Save the following information in the esysContext in order to fill in the session object on Esys_StartAuthSession_Finish()
        - bindName
        - bindAuth
- Save the following information in the esysContext in order to fill in the session object on Esys_StartAuthSession_Finish()
    - The unencryptedSalt
    - Parameters: sessionType, symmetric, authHash, nonceCaller

Return Values:

- **TSS2_RC_SUCCESS: if the function call was a success.**
- **TSS2_ESYS_RC_BAD_TR: if the tpmKey is provided but cannot be used to encrypt the salt.**
- **TSS2_ESYS_RC_BAD_VALUE: if the tpmKey is ESYS_TR_NONE and the encrypted salt is non-NULL.**
- **Any return value from the generic ESYS_<COMMAND_NAME>_Async() section**

10.1.3 **Esys_StartAuthSession_Finish()**

Special Handling Required:

- Upon successful TPM response:
    - Create an Esys_TR object for ‘sessionHandle’ of type session
    - Set ‘sessionType’ in ESys_TR ‘sessionHandle’ to the esysContext’s sessionType
    - Set ‘symmetric’ in ESys_TR ‘sessionHandle’ to the esysContext’s symmetric algorithm
    - Set ‘authHash’ in ESys_TR ‘sessionHandle’ to the esysContext’s authHash algorithm
    - Calculate the sessionKey from esysContext’s authHash, unencryptedSalt, the bindAuthValue, the nonceTPM and nonceCaller
    - Store the sessionKey in the ESys_TR ‘sessionHandle’
    - Store the response parameter tpmNonce in the ESys_TR ‘sessionHandle’

  - Set ‘bindName’ in ESys_TR ‘sessionHandle’ to the esysContext’s bindName if provided during Esys_StartAuthSession_Async()
  - Set ‘bindAuth’ in ESys_TR sessionHandle to the esysContext’s bindAuth if provided during Esys_StartAuthSession_Async()
- Delete values saved to esysContext: ‘unencryptedSalt’, ‘sessionType’, ‘symmetric’, ‘authHash’, ‘bindName’, ‘bindAuth’, ‘nonceCaller’

10.2 Esys_Load Commands

10.2.1 Esys_Load()

Special Handling Required:

The special handling for Esys_Load() is the same as the special handling for Esys_Load_Async() and Esys_Load_Finish().

10.2.2 Esys_Load_Async()

Special Handling Required:

- Store public from inPublic in the esysContext for use in Esys_Load_Finish()

10.2.3 Esys_Load_Finish()

Special Handling Required:

- If TPM response is successful
  - Create a ESYS_TR objectHandle
  - Store the inPublic from the esysContext within the ESYS_TR objectHandle.
  - Store the response parameter ‘name’ within the ESYS_TR objectHandle.
  - Check ‘name’ against ‘inPublic’
- Delete inPublic from the esysContext.

Return Values:

- **TSS2_ESYS_RC_MALFORMED_RESPONSE: if the 'name' doesn't match the 'inPublic'.**
- **Any return value from the generic Esys_<COMMAND_NAME>_Finish() section.**

10.3 Esys_LoadExternal Commands

10.3.1 Esys_LoadExternal()

Special Handling Required:

The special handling for Esys_LoadExternal() is the same as the special handling for Esys_LoadExternal_Async() and Esys_LoadExternal_Finish().

10.3.2 Esys_LoadExternal_Async()

Special Handling Required:

- Store ‘public’ from ‘inPublic’ in the esysContext for use in Esys_LoadExternal_Finish()

10.3.3 Esys_LoadExternal_Finish()

Special Handling Required:

- If TPM response is successful
    - Create a ESYS_TR objectHandle
    - Store the inPublic from the esysContext within the ESYS_TR objectHandle.
    - Store the response parameter 'name' within the ESYS_TR objectHandle.
    - Check 'name' against 'inPublic'
- Delete inPublic from the esysContext.

Return Values:

- **TSS2_ESYS_RC_MALFORMED_RESPONSE: if the 'name' doesn't match the 'inPublic'.**
- **Any return value from the generic Esys_<COMMAND_NAME>_Finish() section.**

10.4 Esys_CreateLoaded Commands

10.4.1 Esys_CreateLoaded()

Special Handling Required:

The special handling for Esys_CreateLoaded() is the same as the special handling for Esys_CreateLoaded_Async() and Esys_CreateLoaded_Finish().

10.4.2 Esys_CreateLoaded_Async()

Special Handling Required:

- Store the authValue for use during Esys_CreateLoaded_Finish().

10.4.3 Esys_CreateLoaded_Finish()

Special Handling Required:

- If TPM response is successful
    - Create a ESYS_TR objectHandle
    - Store the response parameter outPublic within the ESYS_TR objectHandle.
    - Store the response parameter name within the ESYS_TR objectHandle.
    - Check 'name' against 'outPublic'
    - Store the recorded authValue in the ESYS_TR objectHandle

Return Values:

- **TSS2_ESYS_RC_MALFORMED_RESPONSE: if the 'name' doesn't match the 'inPublic'.**
- **Any return value from the generic Esys_<COMMAND_NAME>_Finish() section.**

10.5 Esys_HMAC_Start Commands

10.5.1 Esys_HMAC_Start()

Special Handling Required:

The special handling for Esys_HMAC_Start() is the same as the special handling for Esys_HMAC_Start_Async() and Esys_HMAC_Start_Finish().

10.5.2 Esys_HMAC_Start_Async()

Special Handling Required:

- Store authValue into the esysContext.
  NOTE: if the user does not want to have it stored, then he/she need to reset it afterwards using Esys_TR_SetAuth().

10.5.3 Esys_HMAC_Start_Finish()

Special Handling Required:

- If TPM response is successful
  - Create a ESYS_TR sequenceHandle
  - Store the authValue from esysContext in the sequenceHandle
    NOTE: The name of a sequence object is an empty buffer.
- Delete authValue from esysContext.

10.6 Esys_HashSequenceStart Commands

Special Handling Required:

The special handling for Esys_HashSequenceStart() is the same as the special handling for Esys_HMAC_Start().

10.7 Esys_SequenceComplete Commands

Special Handling Required:

The ESYS_TR object is invalidated after a successful TPM response.

This means that the ESYS_TR object cannot be used for any future operations and the variable can be discarded.

10.7.1 Esys_SequenceComplete()

Special Handling Required:

The ESYS_TR object is invalidated after a successful TPM response.

ESAPI MUST set the internal type/status of the ESYS_TR sequenceHandle to invalid, so it cannot be used anymore.

10.7.2 Esys_SequenceComplete_Async()

Special Handling Required:

The ESYS_TR object will be stored in the ESYS_CONTEXT for use during the Finish call.

10.7.3 Esys_SequenceComplete_Finish()

Special Handling Required:

The ESYS_TR object is invalidated after a successful TPM response.

ESAPI MUST set the type/status of the ESYS_TR sequenceHandle to invalid, so it cannot be used anymore.

10.8 Esys_EventSequenceComplete Commands

The special handling for Esys_EventSequenceComplete() is the same as the special handling for Esys_SequenceComplete().

10.9 Esys_PolicyAuthValue Commands

10.9.1 Esys_PolicyAuthValue()

Special Handling Required:

The special handling for Esys_PolicyAuthValue() is the same as the special handling for Esys_PolicyAuthValue_Finish().

10.9.2 Esys_PolicyAuthValue_Finish()

Special Handling Required:

On successful TPM response, update the referenced policy session with a marker to include the authvalue in the HMAC key when the authorization HMAC is computed. Also, the marker for PolicyPassword is reset.

10.10 Esys_PolicyPassword Commands

10.10.1 Esys_PolicyPassword()

Special Handling Required:

The special handling for Esys_PolicyPassword() is the same as the special handling for Esys_PolicyPassword_Finish().

10.10.2 Esys_PolicyPassword_Finish()

Special Handling Required:

On successful TPM response, update the session's ESYS_TR object to indicate that the authValue of the authorized object will be checked in cleartext when the session is used for authorization. Also, the marker for PolicyAuthValue is being reset.

10.11 Esys_CreatePrimary Commands

10.11.1 Esys_CreatePrimary()

Special Handling Required:

The special handling for Esys_CreatePrimary() is the same as the special handling for Esys_CreatePrimary_Finish().

10.11.2 Esys_CreatePrimary_Async()

Special Handling Required:

- Store the authValue for use during Finish.

10.11.3 Esys_CreatePrimary_Finish()

Special Handling Required:

- On successful TPM response, a new ESYS_TR object for the primary key is created.

- The name of the ESYS_TR object is filled with 'name'.
- The publicArea of the ESYS_TR object is filled with 'outPublic'
- 'name' and 'outPublic' SHOULD be checked for consistency.
- Store the recorded authValue in the ESYS_TR object

Return Values:

- **TSS2_ESYS_RC_MALFORMED_RESPONSE: if the 'name' doesn't match the 'inPublic'.**
- **Any return value from the generic Esys_<COMMAND_NAME>_Finish() section.**

10.12 Esys_HierarchyChangeAuth Commands

10.12.1 Esys_HierarchyChangeAuth()

Special Handling Required:

On successful TPM response the hierarchy's ESYS_TR object is updated with a new auth value. The newly provided authValue will be used when validating the response HMAC.

10.12.2 Esys_HierarchyChangeAuth_Async()

Special Handling Required:

The newly provided authValue is recorded for further use during the Finish call.

10.12.3 Esys_HierarchyChangeAuth_Finish()

Special Handling Required:

On successful TPM response the hierarchy's ESYS_TR object is updated with a new authValue. The newly provided authValue will be used when validating the response HMAC.

10.13 Esys_ContextSave Commands

If the ESYS_TR object being saved refers to a session, the ESYS_TR object is invalidated.
This means that the ESYS_TR object cannot be used for any future operations and the variable can be discarded.

Futhermore, the ESAPI implementation augments the data inside the saved context blob by the metadata it requires for e.g. object names, if needed. This is done by augmenting the contents of context->contextBlob.buffer (and size). This data is used to restore the ESYS_TR object during ContextLoad.

NOTE: authorization values kept inside the ESYS_TR object metadata are not stored in the context blobs.

The recommended implementation is:

```
typedef TPM2B_EVENT TSS2B_METADATA;
typedef struct {
     UINT32 reserved; /* Must always be zero */
     TPM2B_CONTEXT_DATA tpmContext;
     TPM2B_METADATA esysMetadata;
} ESYS_CONTEXT_DATA;
if (e.g. type == RSA_KEY && type != HashSequence) {
    ESYS_CONTEXT_DATA esyscontextData;
    esyscontextData.reserved = 0;
    esyscontextData.tpmContext.buffer = context->contextBlob.buffer;
    esyscontextData.tpmContext.size = context->contextBlob.size;
    esyscontextData.esysMetadata.buffer = metadata;
    esyscontextData.esysMetadata.size = metadata_size;
    context->contextBlob.buffer = esyscontextData;
    context->contextBlob.size = sizeof(UINT32) +
        sizeof(UINT16) + esyscontextData.tpmContext.size +
        sizeof(UINT16) + esyscontextData.esysMetadata.size;
}
```

10.13.1 Esys_ContextSave()

Special Handling Required:

The special handling for Esys_ContextSave() is the same as the special handling for Esys_ContextSave_Async() and Esys_ContextSave_Finish().

10.13.2 Esys_ContextSave_Async()

Special Handling Required:

Record the ESYS_TR object for handling inside the Esys_ContextSave_Finish() call.

10.13.3 Esys_ContextSave_Finish()

Special Handling Required:

The ESAPI implementation augments the data inside the saved context blob by the metadata it requires for e.g. object names.

On a successful TPM reponse, if the saved ESYS_TR object refers to a session, invalidate this ESYS_TR object.

10.14 Esys_ContextLoad Commands

10.14.1 Esys_ContextLoad()

Special Handling Required:

The special handling for Esys_ContextLoad() is the same as the special handling for Esys_ContextLoad_Async() and Esys_ContextLoad_Finish().

10.14.2 Esys_ContextLoad_Async()

Special Handling Required:

The ESAPI implementation loads the metadata from the context blob and restores the original TPM context blob for loading by the TPM. This metadata is stored inside the EsysContext for use during the complete call.

10.14.3 Esys_ContextLoad_Finish()

Special Handling Required:

If the TPM responds success, then a ESYS_TR object is created and returned. The metadata of this object is set to the metadata provided during the _Async() call.

10.15 Esys_FlushContext Commands

10.15.1 Esys_FlushContext()

Special Handling Required:

The ESYS_TR object is invalidated after a successful TPM response.

Esys MUST set the internal type/status of the ESYS_TR sequenceHandle to invalid, so it cannot be used anymore.

10.15.2 Esys_FlushContext_Async()

Special Handling Required:

The ESYS_TR object is stored in the ESYS_CONTEXT for use during the Finish call.

10.15.3 Esys_FlushContext_Finish()

Special Handling Required:

The ESYS_TR object is invalidated after a successful TPM response.

Esys MUST set the internal type/status of the ESYS_TR sequenceHandle to invalid, so it cannot be used anymore.

10.16 Esys_EvictControl Commands

10.16.1 Esys_EvictControl()

Special Handling Required:

The special handling for Esys_EvictControl() is the same as the special handling for Esys_EvictControl_Async() and Esys_EvictControl_Finish().

Definition:

```
TSS2_RC Esys_EvictControl(
    ESYS_CONTEXT        *esysContext,
    ESYS_TR              auth,
    ESYS_TR              objectHandle,
    ESYS_TR              authSession1,
    ESYS_TR              optionalSession2,
    ESYS_TR              optionalSession3,
    TPMI_DH_PERSISTENT   persistentHandle,
    ESYS_TR             *newObjectHandle);
```

10.16.2 Esys_EvictControl_Async()

Special Handling Required:

If the ESYS_TR ‘objectHandle’ is a persistent handle, the ‘persistentHandle’ parameter is ignored and is set internally (inside the ESAPI) to the same TPM handle as ESYS_TR ‘objectHandle’.

Store all metadata of the object including the authValue inside the esysContext.

Store the persistentHandle inside the esysContext.

10.16.3 Esys_EvictControl_Finish()

Special Handling Required:

The additional ESYS_TR output parameter ‘newObjectHandle’ is added to Esys_EvictControl_Finish().

If the ESYS_TR 'objectHandle' on input was a transient object, an ESYS_TR object will be created and will represent the persistentHandle. The metadata including the authValue from esysContext is stored for the new ESYS_TR object.

The metadata including the authValue and persistentHandle is removed from the esysContext.

If the ESYS_TR object on input was a persistent object, this parameter will not return a ESYS_TR object but instead will return ESYS_TR_NULL. Also the metadata in the ESYS_CONTEXT regarding the ESYS_TR object will be removed and the ESYS_TR object will be marked as invalid.

Definition:

```
TSS2_RC Esys_EvictControl_Finish(
    ESYS_CONTEXT        *esysContext,
    ESYS_TR             *newObjectHandle);
```

10.17 **Esys_NV_DefineSpace Commands**

10.17.1 **Esys_NV_DefineSpace()**

Special Handling Required:

The special handling for Esys_NV_DefineSpace() is the same as the special handling for Esys_NV_DefineSpace_Async and Esys_NV_DefineSpace_Finish().

Definition:

```
TSS2_RC Esys_NV_DefineSpace(
    ESYS_CONTEXT              *esysContext,
    ESYS_TR                    authHandle,
    ESYS_TR                    authHandleSession1,
    ESYS_TR                    optionalSession2,
    ESYS_TR                    optionalSession3,
    TPM2B_AUTH      const *auth,
    TPM2B_NV_PUBLIC const *publicInfo,
    ESYS_TR                   *nvHandle);
```

10.17.2 **Esys_NV_DefineSpace_Async()**

Special Handling Required:

1. Record the public data inside the Esys context for use during Esys_NV_DefineSpace_Finish().
1. Store the authValue for use during Esys_NV_DefineSpace_Finish().

Implementations SHOULD check that TPMA_NV_POLICY_DELETE may be set only if an authPolicy is provided. This is intended to prevent undeletable NV indices. Return TSS2_ ESYS_RC_BAD_VALUE if unset authPolicy is detected.

10.17.3 **Esys_NV_DefineSpace_Finish()**

Special Handling Required:

1. The additional ESYS_TR output parameter 'nvHandle' is added to Esys_NV_DefineSpace_Finish() to represent the newly created NV index.
2. Sets the metadata to the 'name' and 'nvPublic' of the Esys_NV_DefineSpace_Async() call.
3. Store the recorded authValue in the ESYS_TR object.

Definition:

```
TSS2_RC Esys_NV_DefineSpace_Finish(
    ESYS_CONTEXT *esysContext,
    ESYS_TR      *nvHandle);
```

Return Values:

- **TSS2_ESYS_RC_MALFORMED_RESPONSE: if the 'name' doesn't match the 'inPublic'.**
- **Any return value from the generic Esys_<COMMAND_NAME>_Finish() section.**

10.18 Esys_NV_UndefineSpace Commands

10.18.1 Esys_NV_UndefineSpace()

The special handling for Esys_NV_UndefineSpace() is the same as the special handling for Esys_NV_UndefineSpace_Finish().

10.18.2 Esys_NV_UndefineSpace_Async()

Special Handling Required:

The ESYS_TR object is recorded inside the Esys context for use during Esys_NV_UndefineSpace_Finish().

10.18.3 Esys_NV_UndefineSpace_Finish()

Special Handling Required:

The ESYS_TR object is invalidated after a successful TPM response.

ESAPI MUST set the internal type/status of the ESYS_TR sequenceHandle to invalid, so it cannot be used anymore.

10.19 Esys_NV_UndefineSpaceSpecial Commands

10.19.1 Esys_NV_UndefineSpaceSpecial()

Special Handling Required:

The special handling for Esys_NV_UndefineSpaceSpecial() is the same as the special handling for Esys_NV_UndefineSpaceSpecial_Finish().

10.19.2 Esys_NV_UndefineSpaceSpecial_Async()

Special Handling Required:

The ESYS_TR object is recorded inside the Esys context for use during Esys_NV_UndefineSpaceSpecial_Finish().

10.19.3 Esys_NV_UndefineSpaceSpecial_Finish()

Special Handling Required:

If the TPM response is SUCCESS, the HMAC key for response validation uses an empty authValue for the nvIndex's session authentication.

The ESYS_TR object is invalidated after a successful TPM response.

Esys MUST set the internal type/status of the ESYS_TR sequenceHandle to invalid, so it cannot be used anymore.

10.20 Esys_NV_Write Commands

10.20.1 Esys_NV_Write()

Special Handling Required:

The special handling for Esys_NV_Write() is the same as the special handling for Esys_NV_Write_Finish().

10.20.2 Esys_NV_Write_Async()

Special Handling Required:

Record the ESYS_TR object for use during Esys_NV_Write_Finish().

10.20.3 Esys_NV_Write_Finish()

Special Handling Required:

On successful TPM response, set the TPMA_NV_WRITTEN bit in the ESYS_TR object's metadata and recalculate the object's name for response validation and future use.

10.21 Esys_NV_Increment Commands

The special handling for Esys_NV_Increment() is the same as the special handling for Esys_NV_Write().

10.22 Esys_NV_Extend Commands

The special handling for Esys_NV_Extend() is the same as the special handling for Esys_NV_Write().

10.23 Esys_NV_SetBits Commands

The special handling for Esys_NV_SetBits() is the same as the special handling for Esys_NV_Write().

10.24 Esys_NV_WriteLock Commands

10.24.1 Esys_NV_WriteLock()

Special Handling Required:

The special handling for Esys_NV_WriteLock() is the same as the special handling for Esys_NV_WriteLock_Finish().

10.24.2 Esys_NV_WriteLock_Async()

Special Handling Required:

Record the ESYS_TR object for use during Esys_NV_WriteLock_Finish().

10.24.3 Esys_NV_WriteLock_Finish()

Special Handling Required:

On successful TPM response, set the TPMA_NV_WRITELOCKED bit in the ESYS_TR object's metadata and recalculate the name for response validate and future use.

10.25 Esys_NV_ReadLock Commands

10.25.1 Esys_NV_ReadLock()

Special Handling Required:

The special handling for Esys_NV_ReadLock() is the same as the special handling for Esys_NV_ReadLock_Finish.

10.25.2 Esys_NV_ReadLock_Async()

Special Handling Required:

Record the ESYS_TR object for use during Esys_NV_ReadLock_Finish().

10.25.3 Esys_NV_ReadLock_Finish()

Special Handling Required:

On successful TPM response, set the TPMA_NV_READLOCKED bit in the ESYS_TR object's metadata and recalculate the name for response validation and future use.

10.26 Esys_NV_ChangeAuth Commands

10.26.1 Esys_NV_ChangeAuth()

Special Handling Required:

The special handling for Esys_NV_ChangeAuth() is the same as the special handling for Esys_NV_ChangeAuth_Finish().

10.26.2 Esys_NV_ChangeAuth_Async()

Special Handling Required:

The newly provided authValue and the ESYS_TR object is recorded for further use during the complete call.

10.26.3 Esys_NV_ChangeAuth_Finish()

Special Handling Required:

On successful TPM response the ESYS_TR object is updated with a new authValue. A newly provided authValue will be used when validating the response HMAC.

10.27 Esys_PCR_SetAuthValue Commands

10.27.1 Esys_PCR_SetAuthValue()

Special Handling Required:

On successful TPM response the PCR's ESYS_TR object is updated with a new auth value. The newly provided authValue will be used when validating the response HMAC.

10.27.2 Esys_PCR_SetAuthValue_Async()

Special Handling Required:

The newly provided authValue is recorded for further use during the Finish call.

10.27.3 Esys_PCR_SetAuthValue_Finish()

Special Handling Required:

On successful TPM response the PCR's ESYS_TR object is updated with a new authValue. The newly provided authValue will be used when validating the response HMAC.

11 ESAPI Header File (tss2_esys.h)

The ESAPI header file is put in “tss2_esys.h”.

11.1 tss2_esys.h Prelude

```
#ifndef TSS2_ESYS_H
#define TSS2_ESYS_H

#include <stdlib.h>
#include "tss2_common.h"
#include "tss2_tpm2_types.h"
#include "tss2_tcti.h"

#ifndef TSS2_API_VERSION_1_2_1_108
#error Version mismatch among TSS2 header files.
#endif

#ifdef __cplusplus
extern "C" {
#endif
```

11.2 tss2_esys.h Types and Constants

```
/*
 * Enhanced System API Types and Constants
 */

/* Opaque context structure */
typedef struct ESYS_CONTEXT ESYS_CONTEXT;

typedef uint32_t ESYS_TR;
#define ESYS_TR_PCR0     0U
#define ESYS_TR_PCR1     1U
#define ESYS_TR_PCR2     2U
#define ESYS_TR_PCR3     3U
#define ESYS_TR_PCR4     4U
#define ESYS_TR_PCR5     5U
#define ESYS_TR_PCR6     6U
#define ESYS_TR_PCR7     7U
#define ESYS_TR_PCR8     8U
#define ESYS_TR_PCR9     9U
#define ESYS_TR_PCR10   10U
#define ESYS_TR_PCR11   11U
#define ESYS_TR_PCR12   12U
#define ESYS_TR_PCR13   13U
#define ESYS_TR_PCR14   14U
#define ESYS_TR_PCR15   15U
#define ESYS_TR_PCR16   16U
#define ESYS_TR_PCR17   17U
#define ESYS_TR_PCR18   18U
#define ESYS_TR_PCR19   19U
#define ESYS_TR_PCR20   20U
#define ESYS_TR_PCR21   21U
#define ESYS_TR_PCR22   22U
#define ESYS_TR_PCR23   23U
```

```
#define ESYS_TR_PCR24   24U
#define ESYS_TR_PCR25   25U
#define ESYS_TR_PCR26   26U
#define ESYS_TR_PCR27   27U
#define ESYS_TR_PCR28   28U
#define ESYS_TR_PCR29   29U
#define ESYS_TR_PCR30   30U
#define ESYS_TR_PCR31   31U

#define ESYS_TR_PASSWORD        0x0ffU
#define ESYS_TR_NONE            0xfffU

/* From TPM_RH Constants */
#define ESYS_TR_RH_OWNER        0x101U
#define ESYS_TR_RH_NULL         0x107U
#define ESYS_TR_RH_LOCKOUT      0x10AU
#define ESYS_TR_RH_ENDORSEMENT  0x10BU
#define ESYS_TR_RH_PLATFORM     0x10CU
#define ESYS_TR_RH_PLATFORM_NV  0x10DU
#define ESYS_TR_RH_AUTH_FIRST   0x110U
#define ESYS_TR_RH_AUTH(x)      (ESYS_TR_RH_AUTH_FIRST + (ESYS_TR)(x))
```

11.1 tss2_esys.h Context Management Functions

```
/*
 * Enhanced System API Management Functions
 */
TSS2_DLL_EXPORT TSS2_RC Esys_Initialize(
    ESYS_CONTEXT      **esysContext,
    TSS2_TCTI_CONTEXT  *tcti,
    TSS2_ABI_VERSION   *abiVersion);

TSS2_DLL_EXPORT void Esys_Finalize(
    ESYS_CONTEXT      **esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_GetTcti(
    ESYS_CONTEXT       *esysContext,
    TSS2_TCTI_CONTEXT **tcti);

TSS2_DLL_EXPORT TSS2_RC Esys_GetPollHandles(
    ESYS_CONTEXT           *esysContext,
    TSS2_TCTI_POLL_HANDLE **handles,
    size_t                 *count);

TSS2_DLL_EXPORT TSS2_RC Esys_SetTimeout(
    ESYS_CONTEXT   *esysContext,
    int32_t         timeout);

TSS2_DLL_EXPORT void Esys_Free(
    void    *ptr);
```

11.2 tss2_esys.h Object Management Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_TR_Serialize(
    ESYS_CONTEXT       *esysContext,
    ESYS_TR             object,
    uint8_t           **buffer,
    size_t             *buffer_size);

TSS2_DLL_EXPORT TSS2_RC Esys_TR_Deserialize(
    ESYS_CONTEXT       *esysContext,
    uint8_t      const *buffer,
    size_t              buffer_size,
    ESYS_TR            *object);

TSS2_DLL_EXPORT TSS2_RC Esys_TR_FromTPMPublic_Async(
    ESYS_CONTEXT       *esysContext,
    TPM2_HANDLE         tpm_handle,
    ESYS_TR             optionalSession1,
    ESYS_TR             optionalSession2,
    ESYS_TR             optionalSession3);

TSS2_DLL_EXPORT TSS2_RC Esys_TR_FromTPMPublic_Finish(
    ESYS_CONTEXT       *esysContext,
    ESYS_TR            *object);

TSS2_DLL_EXPORT TSS2_RC Esys_TR_FromTPMPublic(
    ESYS_CONTEXT       *esysContext,
    TPM2_HANDLE         tpm_handle,
    ESYS_TR             optionalSession1,
    ESYS_TR             optionalSession2,
    ESYS_TR             optionalSession3,
    ESYS_TR            *object);

TSS2_DLL_EXPORT TSS2_RC Esys_TR_Close(
    ESYS_CONTEXT       *esysContext,
    ESYS_TR            *object);

TSS2_DLL_EXPORT TSS2_RC Esys_TR_SetAuth(
    ESYS_CONTEXT       *esysContext,
    ESYS_TR             handle,
    TPM2B_AUTH   const *authValue);

TSS2_DLL_EXPORT TSS2_RC Esys_TR_GetName(
    ESYS_CONTEXT       *esysContext,
    ESYS_TR             handle,
    TPM2B_NAME        **name);

TSS2_DLL_EXPORT TSS2_RC Esys_TRSess_GetAttributes(
    ESYS_CONTEXT       *esysContext,
    ESYS_TR             session,
    TPMA_SESSION       *flags);

TSS2_DLL_EXPORT TSS2_RC Esys_TRSess_SetAttributes(
    ESYS_CONTEXT       *esysContext,
    ESYS_TR             session,
    TPMA_SESSION        flags,
    TPMA_SESSION        mask);
```

```
TSS2_DLL_EXPORT TSS2_RC Esys_TRSess_GetNonceTPM(
    ESYS_CONTEXT       *esysContext,
    ESYS_TR             session,
    TPM2B_NONCE       **nonceTPM);
```

11.3 tss2_esys.h Functions for Invoking TPM Commands

```
/*
 * The following functions are the Async, Finish, and One-Shot
 * functions corresponding to each command in part 3 of the TPM
 * specification.
 */
```

11.3.1 Esys_Startup Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_Startup_Async(
    ESYS_CONTEXT *esysContext,
    TPM2_SU       startupType);

TSS2_DLL_EXPORT TSS2_RC Esys_Startup_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_Startup(
    ESYS_CONTEXT *esysContext,
    TPM2_SU       startupType);
```

11.3.2 Esys_Shutdown Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_Shutdown_Async(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       optionalSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3,
    TPM2_SU       shutdownType);

TSS2_DLL_EXPORT TSS2_RC Esys_Shutdown_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_Shutdown(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       optionalSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3,
    TPM2_SU       shutdownType);
```

11.3.3 Esys_SelfTest Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_SelfTest_Async(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       optionalSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3,
    TPMI_YES_NO   fullTest);

TSS2_DLL_EXPORT TSS2_RC Esys_SelfTest_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_SelfTest(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       optionalSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3,
    TPMI_YES_NO   fullTest);
```

11.3.4 Esys_IncrementalSelfTest Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_IncrementalSelfTest_Async(
    ESYS_CONTEXT   *esysContext,
    ESYS_TR         optionalSession1,
    ESYS_TR         optionalSession2,
    ESYS_TR         optionalSession3,
    TPML_ALG const *toTest);

TSS2_DLL_EXPORT TSS2_RC Esys_IncrementalSelfTest_Finish(
    ESYS_CONTEXT *esysContext,
    TPML_ALG    **toDoList);

TSS2_DLL_EXPORT TSS2_RC Esys_IncrementalSelfTest(
    ESYS_CONTEXT   *esysContext,
    ESYS_TR         optionalSession1,
    ESYS_TR         optionalSession2,
    ESYS_TR         optionalSession3,
    TPML_ALG const *toTest,
    TPML_ALG      **toDoList);
```

11.3.5 Esys_GetTestResult Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_GetTestResult_Async(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       optionalSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3);

TSS2_DLL_EXPORT TSS2_RC Esys_GetTestResult_Finish(
    ESYS_CONTEXT      *esysContext,
    TPM2B_MAX_BUFFER **outData,
    TPM2_RC           *testResult);

TSS2_DLL_EXPORT TSS2_RC Esys_GetTestResult(
    ESYS_CONTEXT      *esysContext,
    ESYS_TR            optionalSession1,
    ESYS_TR            optionalSession2,
    ESYS_TR            optionalSession3,
    TPM2B_MAX_BUFFER **outData,
    TPM2_RC           *testResult);
```

11.3.6 Esys_StartAuthSession Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_StartAuthSession_Async(
    ESYS_CONTEXT                    *esysContext,
    ESYS_TR                          tpmKey,
    ESYS_TR                          bind,
    ESYS_TR                          optionalSession1,
    ESYS_TR                          optionalSession2,
    ESYS_TR                          optionalSession3,
    TPM2B_NONCE               const *nonceCaller,
    TPM2_SE                          sessionType,
    TPMT_SYM_DEF              const *symmetric,
    TPMI_ALG_HASH                    authHash);

TSS2_DLL_EXPORT TSS2_RC Esys_StartAuthSession_Finish(
    ESYS_CONTEXT *esysContext,
    ESYS_TR      *sessionHandle);

TSS2_DLL_EXPORT TSS2_RC Esys_StartAuthSession(
    ESYS_CONTEXT                    *esysContext,
    ESYS_TR                          tpmKey,
    ESYS_TR                          bind,
    ESYS_TR                          optionalSession1,
    ESYS_TR                          optionalSession2,
    ESYS_TR                          optionalSession3,
    TPM2B_NONCE               const *nonceCaller,
    TPM2_SE                          sessionType,
    TPMT_SYM_DEF              const *symmetric,
    TPMI_ALG_HASH                    authHash,
    ESYS_TR                         *sessionHandle);
```

11.3.7 Esys_PolicyRestart Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_PolicyRestart_Async(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       sessionHandle,
    ESYS_TR       optionalSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicyRestart_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicyRestart(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       sessionHandle,
    ESYS_TR       optionalSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3);
```

11.3.8 Esys_Create Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_Create_Async(
    ESYS_CONTEXT                   *esysContext,
    ESYS_TR                         parentHandle,
    ESYS_TR                         parentHandleSession1,
    ESYS_TR                         optionalSession2,
    ESYS_TR                         optionalSession3,
    TPM2B_SENSITIVE_CREATE const *inSensitive,
    TPM2B_PUBLIC           const *inPublic,
    TPM2B_DATA             const *outsideInfo,
    TPML_PCR_SELECTION     const *creationPCR);

TSS2_DLL_EXPORT TSS2_RC Esys_Create_Finish(
    ESYS_CONTEXT         *esysContext,
    TPM2B_PRIVATE       **outPrivate,
    TPM2B_PUBLIC        **outPublic,
    TPM2B_CREATION_DATA **creationData,
    TPM2B_DIGEST        **creationHash,
    TPMT_TK_CREATION    **creationTicket);

TSS2_DLL_EXPORT TSS2_RC Esys_Create(
    ESYS_CONTEXT                   *esysContext,
    ESYS_TR                         parentHandle,
    ESYS_TR                         parentHandleSession1,
    ESYS_TR                         optionalSession2,
    ESYS_TR                         optionalSession3,
    TPM2B_SENSITIVE_CREATE const *inSensitive,
    TPM2B_PUBLIC           const *inPublic,
    TPM2B_DATA             const *outsideInfo,
    TPML_PCR_SELECTION     const *creationPCR,
    TPM2B_PRIVATE               **outPrivate,
    TPM2B_PUBLIC                **outPublic,
    TPM2B_CREATION_DATA         **creationData,
    TPM2B_DIGEST                **creationHash,
    TPMT_TK_CREATION            **creationTicket);
```

11.3.9 Esys_Load Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_Load_Async(
    ESYS_CONTEXT         *esysContext,
    ESYS_TR               parentHandle,
    ESYS_TR               parentHandleSession1,
    ESYS_TR               optionalSession2,
    ESYS_TR               optionalSession3,
    TPM2B_PRIVATE const *inPrivate,
    TPM2B_PUBLIC  const *inPublic);

TSS2_DLL_EXPORT TSS2_RC Esys_Load_Finish(
    ESYS_CONTEXT *esysContext,
    ESYS_TR      *objectHandle);

TSS2_DLL_EXPORT TSS2_RC Esys_Load(
    ESYS_CONTEXT         *esysContext,
    ESYS_TR               parentHandle,
    ESYS_TR                parentHandleSession1,
    ESYS_TR                optionalSession2,
    ESYS_TR                optionalSession3,
    TPM2B_PRIVATE const *inPrivate,
    TPM2B_PUBLIC  const *inPublic,
    ESYS_TR              *objectHandle);
```

11.3.10 Esys_LoadExternal Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_LoadExternal_Async(
    ESYS_CONTEXT         *esysContext,
    ESYS_TR               optionalSession1,
    ESYS_TR               optionalSession2,
    ESYS_TR               optionalSession3,
    TPM2B_SENSITIVE const *inPrivate,
    TPM2B_PUBLIC    const *inPublic,
    ESYS_TR               hierarchy);

TSS2_DLL_EXPORT TSS2_RC Esys_LoadExternal_Finish(
    ESYS_CONTEXT *esysContext,
    ESYS_TR      *objectHandle);

TSS2_DLL_EXPORT TSS2_RC Esys_LoadExternal(
    ESYS_CONTEXT         *esysContext,
    ESYS_TR               optionalSession1,
    ESYS_TR               optionalSession2,
    ESYS_TR               optionalSession3,
    TPM2B_SENSITIVE const *inPrivate,
    TPM2B_PUBLIC    const *inPublic,
    ESYS_TR               hierarchy,
    ESYS_TR              *objectHandle);
```

11.3.11 Esys_ReadPublic Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_ReadPublic_Async(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       objectHandle,
    ESYS_TR       optionalSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3);

TSS2_DLL_EXPORT TSS2_RC Esys_ReadPublic_Finish(
    ESYS_CONTEXT  *esysContext,
    TPM2B_PUBLIC **outPublic,
    TPM2B_NAME   **name,
    TPM2B_NAME   **qualifiedName);

TSS2_DLL_EXPORT TSS2_RC Esys_ReadPublic(
    ESYS_CONTEXT  *esysContext,
    ESYS_TR        objectHandle,
    ESYS_TR        optionalSession1,
    ESYS_TR        optionalSession2,
    ESYS_TR        optionalSession3,
    TPM2B_PUBLIC **outPublic,
    TPM2B_NAME   **name,
    TPM2B_NAME   **qualifiedName);
```

11.3.12 Esys_ActivateCredential Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_ActivateCredential_Async(
    ESYS_CONTEXT                   *esysContext,
    ESYS_TR                         activateHandle,
    ESYS_TR                         keyHandle,
    ESYS_TR                         activateHandleSession1,
    ESYS_TR                         keyHandleSession2,
    ESYS_TR                         optionalSession3,
    TPM2B_ID_OBJECT        const *credentialBlob,
    TPM2B_ENCRYPTED_SECRET const *secret);

TSS2_DLL_EXPORT TSS2_RC Esys_ActivateCredential_Finish(
    ESYS_CONTEXT   *esysContext,
    TPM2B_DIGEST **certInfo);

TSS2_DLL_EXPORT TSS2_RC Esys_ActivateCredential(
    ESYS_CONTEXT                   *esysContext,
    ESYS_TR                         activateHandle,
    ESYS_TR                         keyHandle,
    ESYS_TR                         activateHandleSession1,
    ESYS_TR                         keyHandleSession2,
    ESYS_TR                         optionalSession3,
    TPM2B_ID_OBJECT        const *credentialBlob,
    TPM2B_ENCRYPTED_SECRET const *secret,
    TPM2B_DIGEST                  **certInfo);
```

11.3.13 Esys_MakeCredential Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_MakeCredential_Async(
    ESYS_CONTEXT         *esysContext,
    ESYS_TR               handle,
    ESYS_TR               optionalSession1,
    ESYS_TR               optionalSession2,
    ESYS_TR               optionalSession3,
    TPM2B_DIGEST const *credential,
    TPM2B_NAME   const *objectName);

TSS2_DLL_EXPORT TSS2_RC Esys_MakeCredential_Finish(
    ESYS_CONTEXT             *esysContext,
    TPM2B_ID_OBJECT        **credentialBlob,
    TPM2B_ENCRYPTED_SECRET **secret);

TSS2_DLL_EXPORT TSS2_RC Esys_MakeCredential(
    ESYS_CONTEXT             *esysContext,
    ESYS_TR                   handle,
    ESYS_TR                   optionalSession1,
    ESYS_TR                   optionalSession2,
    ESYS_TR                   optionalSession3,
    TPM2B_DIGEST       const *credential,
    TPM2B_NAME         const *objectName,
    TPM2B_ID_OBJECT         **credentialBlob,
    TPM2B_ENCRYPTED_SECRET **secret);
```

11.3.14 Esys_Unseal Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_Unseal_Async(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       itemHandle,
    ESYS_TR       itemHandleSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3);

TSS2_DLL_EXPORT TSS2_RC Esys_Unseal_Finish(
    ESYS_CONTEXT          *esysContext,
    TPM2B_SENSITIVE_DATA **outData);

TSS2_DLL_EXPORT TSS2_RC Esys_Unseal(
    ESYS_CONTEXT          *esysContext,
    ESYS_TR                itemHandle,
    ESYS_TR                itemHandleSession1,
    ESYS_TR                optionalSession2,
    ESYS_TR                optionalSession3,
    TPM2B_SENSITIVE_DATA **outData);
```

11.3.15 Esys_ObjectChangeAuth Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_ObjectChangeAuth_Async(
    ESYS_CONTEXT     *esysContext,
    ESYS_TR           objectHandle,
    ESYS_TR           parentHandle,
    ESYS_TR           objectHandleSession1,
    ESYS_TR           optionalSession2,
    ESYS_TR           optionalSession3,
    TPM2B_AUTH const *newAuth);

TSS2_DLL_EXPORT TSS2_RC Esys_ObjectChangeAuth_Finish(
    ESYS_CONTEXT   *esysContext,
    TPM2B_PRIVATE **outPrivate);

TSS2_DLL_EXPORT TSS2_RC Esys_ObjectChangeAuth(
    ESYS_CONTEXT     *esysContext,
    ESYS_TR           objectHandle,
    ESYS_TR           parentHandle,
    ESYS_TR           objectHandleSession1,
    ESYS_TR           optionalSession2,
    ESYS_TR           optionalSession3,
    TPM2B_AUTH const *newAuth,
    TPM2B_PRIVATE   **outPrivate);
```

11.3.16 Esys_CreateLoaded Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_CreateLoaded_Async(
    ESYS_CONTEXT                     *esysContext,
    ESYS_TR                           parentHandle,
    ESYS_TR                           parentHandleSession1,
    ESYS_TR                           optionalSession2,
    ESYS_TR                           optionalSession3,
    TPM2B_SENSITIVE_CREATE const *inSensitive,
    TPM2B_TEMPLATE         const *inPublic);

TSS2_DLL_EXPORT TSS2_RC Esys_CreateLoaded_Finish(
    ESYS_CONTEXT   *esysContext,
    ESYS_TR        *objectHandle,
    TPM2B_PRIVATE **outPrivate,
    TPM2B_PUBLIC  **outPublic);

TSS2_DLL_EXPORT TSS2_RC Esys_CreateLoaded(
    ESYS_CONTEXT                     *esysContext,
    ESYS_TR                           parentHandle,
    ESYS_TR                           parentHandleSession1,
    ESYS_TR                           optionalSession2,
    ESYS_TR                           optionalSession3,
    TPM2B_SENSITIVE_CREATE const *inSensitive,
    TPM2B_TEMPLATE         const *inPublic,
    ESYS_TR                          *objectHandle,
    TPM2B_PRIVATE                   **outPrivate,
    TPM2B_PUBLIC                    **outPublic);
```

11.3.17 Esys_Duplicate Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_Duplicate_Async(
    ESYS_CONTEXT                  *esysContext,
    ESYS_TR                        objectHandle,
    ESYS_TR                        newParentHandle,
    ESYS_TR                        objectHandleSession1,
    ESYS_TR                        optionalSession2,
    ESYS_TR                        optionalSession3,
    TPM2B_DATA          const *encryptionKeyIn,
    TPMT_SYM_DEF_OBJECT const *symmetricAlg);

TSS2_DLL_EXPORT TSS2_RC Esys_Duplicate_Finish(
    ESYS_CONTEXT                *esysContext,
    TPM2B_DATA                 **encryptionKeyOut,
    TPM2B_PRIVATE              **duplicate,
    TPM2B_ENCRYPTED_SECRET **outSymSeed);

TSS2_DLL_EXPORT TSS2_RC Esys_Duplicate(
    ESYS_CONTEXT                  *esysContext,
    ESYS_TR                        objectHandle,
    ESYS_TR                        newParentHandle,
    ESYS_TR                        objectHandleSession1,
    ESYS_TR                        optionalSession2,
    ESYS_TR                        optionalSession3,
    TPM2B_DATA          const *encryptionKeyIn,
    TPMT_SYM_DEF_OBJECT const *symmetricAlg,
    TPM2B_DATA               **encryptionKeyOut,
    TPM2B_PRIVATE            **duplicate,
    TPM2B_ENCRYPTED_SECRET   **outSymSeed);
```

11.3.18 Esys_Rewrap Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_Rewrap_Async(
    ESYS_CONTEXT                 *esysContext,
    ESYS_TR                       oldParent,
    ESYS_TR                       newParent,
    ESYS_TR                       oldParentSession1,
    ESYS_TR                       optionalSession2,
    ESYS_TR                       optionalSession3,
    TPM2B_PRIVATE          const *inDuplicate,
    TPM2B_NAME             const *name,
    TPM2B_ENCRYPTED_SECRET const *inSymSeed);

TSS2_DLL_EXPORT TSS2_RC Esys_Rewrap_Finish(
    ESYS_CONTEXT            *esysContext,
    TPM2B_PRIVATE          **outDuplicate,
    TPM2B_ENCRYPTED_SECRET **outSymSeed);

TSS2_DLL_EXPORT TSS2_RC Esys_Rewrap(
    ESYS_CONTEXT                 *esysContext,
    ESYS_TR                       oldParent,
    ESYS_TR                       newParent,
    ESYS_TR                       oldParentSession1,
    ESYS_TR                       optionalSession2,
    ESYS_TR                       optionalSession3,
    TPM2B_PRIVATE          const *inDuplicate,
    TPM2B_NAME             const *name,
    TPM2B_ENCRYPTED_SECRET const *inSymSeed,
    TPM2B_PRIVATE                **outDuplicate,
    TPM2B_ENCRYPTED_SECRET       **outSymSeed);
```

11.3.19 Esys_Import Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_Import_Async(
    ESYS_CONTEXT                 *esysContext,
    ESYS_TR                       parentHandle,
    ESYS_TR                       parentHandleSession1,
    ESYS_TR                       optionalSession2,
    ESYS_TR                       optionalSession3,
    TPM2B_DATA             const *encryptionKey,
    TPM2B_PUBLIC           const *objectPublic,
    TPM2B_PRIVATE          const *duplicate,
    TPM2B_ENCRYPTED_SECRET const *inSymSeed,
    TPMT_SYM_DEF_OBJECT    const *symmetricAlg);

TSS2_DLL_EXPORT TSS2_RC Esys_Import_Finish(
    ESYS_CONTEXT   *esysContext,
    TPM2B_PRIVATE **outPrivate);
```

```
TSS2_DLL_EXPORT TSS2_RC Esys_Import(
    ESYS_CONTEXT                 *esysContext,
    ESYS_TR                       parentHandle,
    ESYS_TR                       parentHandleSession1,
    ESYS_TR                       optionalSession2,
    ESYS_TR                       optionalSession3,
    TPM2B_DATA             const *encryptionKey,
    TPM2B_PUBLIC           const *objectPublic,
    TPM2B_PRIVATE          const *duplicate,
    TPM2B_ENCRYPTED_SECRET const *inSymSeed,
    TPMT_SYM_DEF_OBJECT    const *symmetricAlg,
    TPM2B_PRIVATE               **outPrivate);
```

11.3.20 Esys_RSA_Encrypt Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_RSA_Encrypt_Async(
    ESYS_CONTEXT               *esysContext,
    ESYS_TR                     keyHandle,
    ESYS_TR                     optionalSession1,
    ESYS_TR                     optionalSession2,
    ESYS_TR                     optionalSession3,
    TPM2B_PUBLIC_KEY_RSA const *message,
    TPMT_RSA_DECRYPT     const *inScheme,
    TPM2B_DATA           const *label);

TSS2_DLL_EXPORT TSS2_RC Esys_RSA_Encrypt_Finish(
    ESYS_CONTEXT          *esysContext,
    TPM2B_PUBLIC_KEY_RSA **outData);

TSS2_DLL_EXPORT TSS2_RC Esys_RSA_Encrypt(
    ESYS_CONTEXT               *esysContext,
    ESYS_TR                     keyHandle,
    ESYS_TR                     optionalSession1,
    ESYS_TR                     optionalSession2,
    ESYS_TR                     optionalSession3,
    TPM2B_PUBLIC_KEY_RSA const *message,
    TPMT_RSA_DECRYPT     const *inScheme,
    TPM2B_DATA           const *label,
    TPM2B_PUBLIC_KEY_RSA      **outData);
```

11.3.21 Esys_RSA_Decrypt Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_RSA_Decrypt_Async(
    ESYS_CONTEXT               *esysContext,
    ESYS_TR                     keyHandle,
    ESYS_TR                     keyHandleSession1,
    ESYS_TR                     optionalSession2,
    ESYS_TR                     optionalSession3,
    TPM2B_PUBLIC_KEY_RSA const *cipherText,
    TPMT_RSA_DECRYPT     const *inScheme,
    TPM2B_DATA           const *label);
```

```
TSS2_DLL_EXPORT TSS2_RC Esys_RSA_Decrypt_Finish(
    ESYS_CONTEXT          *esysContext,
    TPM2B_PUBLIC_KEY_RSA **message);

TSS2_DLL_EXPORT TSS2_RC Esys_RSA_Decrypt(
    ESYS_CONTEXT               *esysContext,
    ESYS_TR                     keyHandle,
    ESYS_TR                     keyHandleSession1,
    ESYS_TR                     optionalSession2,
    ESYS_TR                     optionalSession3,
    TPM2B_PUBLIC_KEY_RSA const *cipherText,
    TPMT_RSA_DECRYPT     const *inScheme,
    TPM2B_DATA           const *label,
    TPM2B_PUBLIC_KEY_RSA      **message);
```

11.3.22 Esys_ECDH_KeyGen Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_ECDH_KeyGen_Async(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       keyHandle,
    ESYS_TR       optionalSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3);

TSS2_DLL_EXPORT TSS2_RC Esys_ECDH_KeyGen_Finish(
    ESYS_CONTEXT     *esysContext,
    TPM2B_ECC_POINT **zPoint,
    TPM2B_ECC_POINT **pubPoint);

TSS2_DLL_EXPORT TSS2_RC Esys_ECDH_KeyGen(
    ESYS_CONTEXT     *esysContext,
    ESYS_TR           keyHandle,
    ESYS_TR           optionalSession1,
    ESYS_TR           optionalSession2,
    ESYS_TR           optionalSession3,
    TPM2B_ECC_POINT **zPoint,
    TPM2B_ECC_POINT **pubPoint);
```

11.3.23 Esys_ECDH_ZGen Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_ECDH_ZGen_Async(
    ESYS_CONTEXT          *esysContext,
    ESYS_TR                keyHandle,
    ESYS_TR                keyHandleSession1,
    ESYS_TR                optionalSession2,
    ESYS_TR                optionalSession3,
    TPM2B_ECC_POINT const *inPoint);

TSS2_DLL_EXPORT TSS2_RC Esys_ECDH_ZGen_Finish(
    ESYS_CONTEXT     *esysContext,
    TPM2B_ECC_POINT **outPoint);

TSS2_DLL_EXPORT TSS2_RC Esys_ECDH_ZGen(
    ESYS_CONTEXT          *esysContext,
    ESYS_TR                keyHandle,
    ESYS_TR                keyHandleSession1,
    ESYS_TR                optionalSession2,
    ESYS_TR                optionalSession3,
    TPM2B_ECC_POINT const *inPoint,
    TPM2B_ECC_POINT      **outPoint);
```

11.3.24 Esys_ECC_Parameters Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_ECC_Parameters_Async(
    ESYS_CONTEXT   *esysContext,
    ESYS_TR         optionalSession1,
    ESYS_TR         optionalSession2,
    ESYS_TR         optionalSession3,
    TPMI_ECC_CURVE curveID);

TSS2_DLL_EXPORT TSS2_RC Esys_ECC_Parameters_Finish(
    ESYS_CONTEXT                *esysContext,
    TPMS_ALGORITHM_DETAIL_ECC **parameters);

TSS2_DLL_EXPORT TSS2_RC Esys_ECC_Parameters(
    ESYS_CONTEXT                *esysContext,
    ESYS_TR                      optionalSession1,
    ESYS_TR                      optionalSession2,
    ESYS_TR                      optionalSession3,
    TPMI_ECC_CURVE               curveID,
    TPMS_ALGORITHM_DETAIL_ECC **parameters);
```

11.3.25 Esys_ZGen_2Phase Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_ZGen_2Phase_Async(
    ESYS_CONTEXT          *esysContext,
    ESYS_TR                keyA,
    ESYS_TR                keyASession1,
    ESYS_TR                optionalSession2,
    ESYS_TR                optionalSession3,
    TPM2B_ECC_POINT const *inQsB,
    TPM2B_ECC_POINT const *inQeB,
    TPMI_ECC_KEY_EXCHANGE  inScheme,
    UINT16                 counter);

TSS2_DLL_EXPORT TSS2_RC Esys_ZGen_2Phase_Finish(
    ESYS_CONTEXT     *esysContext,
    TPM2B_ECC_POINT **outZ1,
    TPM2B_ECC_POINT **outZ2);

TSS2_DLL_EXPORT TSS2_RC Esys_ZGen_2Phase(
    ESYS_CONTEXT          *esysContext,
    ESYS_TR                keyA,
    ESYS_TR                keyASession1,
    ESYS_TR                optionalSession2,
    ESYS_TR                optionalSession3,
    TPM2B_ECC_POINT const *inQsB,
    TPM2B_ECC_POINT const *inQeB,
    TPMI_ECC_KEY_EXCHANGE  inScheme,
    UINT16                 counter,
    TPM2B_ECC_POINT      **outZ1,
    TPM2B_ECC_POINT      **outZ2);
```

11.3.26 Esys_EncryptDecrypt Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_EncryptDecrypt_Async(
    ESYS_CONTEXT           *esysContext,
    ESYS_TR                 keyHandle,
    ESYS_TR                 keyHandleSession1,
    ESYS_TR                 optionalSession2,
    ESYS_TR                 optionalSession3,
    TPMI_YES_NO             decrypt,
    TPMI_ALG_SYM_MODE       mode,
    TPM2B_IV         const *ivIn,
    TPM2B_MAX_BUFFER const *inData);

TSS2_DLL_EXPORT TSS2_RC Esys_EncryptDecrypt_Finish(
    ESYS_CONTEXT      *esysContext,
    TPM2B_MAX_BUFFER **outData,
    TPM2B_IV         **ivOut);

TSS2_DLL_EXPORT TSS2_RC Esys_EncryptDecrypt(
    ESYS_CONTEXT           *esysContext,
    ESYS_TR                 keyHandle,
    ESYS_TR                 keyHandleSession1,
    ESYS_TR                 optionalSession2,
    ESYS_TR                 optionalSession3,
    TPMI_YES_NO             decrypt,
    TPMI_ALG_SYM_MODE       mode,
    TPM2B_IV         const *ivIn,
    TPM2B_MAX_BUFFER const *inData,
    TPM2B_MAX_BUFFER      **outData,
    TPM2B_IV              **ivOut);
```

11.3.27 Esys_EncryptDecrypt2 Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_EncryptDecrypt2_Async(
    ESYS_CONTEXT           *esysContext,
    ESYS_TR                 keyHandle,
    ESYS_TR                 keyHandleSession1,
    ESYS_TR                 optionalSession2,
    ESYS_TR                 optionalSession3,
    TPM2B_MAX_BUFFER const *inData,
    TPMI_YES_NO             decrypt,
    TPMI_ALG_SYM_MODE       mode,
    TPM2B_IV         const *ivIn);

TSS2_DLL_EXPORT TSS2_RC Esys_EncryptDecrypt2_Finish(
    ESYS_CONTEXT      *esysContext,
    TPM2B_MAX_BUFFER **outData,
    TPM2B_IV         **ivOut);

TSS2_DLL_EXPORT TSS2_RC Esys_EncryptDecrypt2(
    ESYS_CONTEXT           *esysContext,
    ESYS_TR                 keyHandle,
    ESYS_TR                 keyHandleSession1,
    ESYS_TR                 optionalSession2,
    ESYS_TR                 optionalSession3,
    TPM2B_MAX_BUFFER const *inData,
    TPMI_YES_NO             decrypt,
    TPMI_ALG_SYM_MODE       mode,
    TPM2B_IV         const *ivIn,
    TPM2B_MAX_BUFFER      **outData,
    TPM2B_IV              **ivOut);
```

11.3.28 Esys_Hash Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_Hash_Async(
    ESYS_CONTEXT           *esysContext,
    ESYS_TR                 optionalSession1,
    ESYS_TR                 optionalSession2,
    ESYS_TR                 optionalSession3,
    TPM2B_MAX_BUFFER const *data,
    TPMI_ALG_HASH           hashAlg,
    ESYS_TR                 hierarchy);

TSS2_DLL_EXPORT TSS2_RC Esys_Hash_Finish(
    ESYS_CONTEXT       *esysContext,
    TPM2B_DIGEST      **outHash,
    TPMT_TK_HASHCHECK **validation);

TSS2_DLL_EXPORT TSS2_RC Esys_Hash(
    ESYS_CONTEXT           *esysContext,
    ESYS_TR                 optionalSession1,
    ESYS_TR                 optionalSession2,
    ESYS_TR                 optionalSession3,
    TPM2B_MAX_BUFFER const *data,
    TPMI_ALG_HASH           hashAlg,
    ESYS_TR                 hierarchy,
    TPM2B_DIGEST          **outHash,
    TPMT_TK_HASHCHECK     **validation);
```

11.3.29 Esys_HMAC Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_HMAC_Async(
    ESYS_CONTEXT           *esysContext,
    ESYS_TR                 handle,
    ESYS_TR                 handleSession1,
    ESYS_TR                 optionalSession2,
    ESYS_TR                 optionalSession3,
    TPM2B_MAX_BUFFER const *buffer,
    TPMI_ALG_HASH           hashAlg);
```

```
TSS2_DLL_EXPORT TSS2_RC Esys_HMAC_Finish(
    ESYS_CONTEXT  *esysContext,
    TPM2B_DIGEST **outHMAC);

TSS2_DLL_EXPORT TSS2_RC Esys_HMAC(
    ESYS_CONTEXT           *esysContext,
    ESYS_TR                 handle,
    ESYS_TR                 handleSession1,
    ESYS_TR                 optionalSession2,
    ESYS_TR                 optionalSession3,
    TPM2B_MAX_BUFFER const *buffer,
    TPMI_ALG_HASH           hashAlg,
    TPM2B_DIGEST          **outHMAC);
```

11.3.30 Esys_GetRandom Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_GetRandom_Async(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       optionalSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3,
    UINT16        bytesRequested);

TSS2_DLL_EXPORT TSS2_RC Esys_GetRandom_Finish(
    ESYS_CONTEXT  *esysContext,
    TPM2B_DIGEST **randomBytes);

TSS2_DLL_EXPORT TSS2_RC Esys_GetRandom(
    ESYS_CONTEXT  *esysContext,
    ESYS_TR        optionalSession1,
    ESYS_TR        optionalSession2,
    ESYS_TR        optionalSession3,
    UINT16         bytesRequested,
    TPM2B_DIGEST **randomBytes);
```

11.3.31 Esys_StirRandom Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_StirRandom_Async(
    ESYS_CONTEXT               *esysContext,
    ESYS_TR                     optionalSession1,
    ESYS_TR                     optionalSession2,
    ESYS_TR                     optionalSession3,
    TPM2B_SENSITIVE_DATA const *inData);

TSS2_DLL_EXPORT TSS2_RC Esys_StirRandom_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_StirRandom(
    ESYS_CONTEXT               *esysContext,
    ESYS_TR                     optionalSession1,
    ESYS_TR                     optionalSession2,
    ESYS_TR                     optionalSession3,
    TPM2B_SENSITIVE_DATA const *inData);
```

11.3.32 Esys_HMAC_Start Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_HMAC_Start_Async(
    ESYS_CONTEXT     *esysContext,
    ESYS_TR           handle,
    ESYS_TR           handleSession1,
    ESYS_TR           optionalSession2,
    ESYS_TR           optionalSession3,
    TPM2B_AUTH const *auth,
    TPMI_ALG_HASH     hashAlg);

TSS2_DLL_EXPORT TSS2_RC Esys_HMAC_Start_Finish(
    ESYS_CONTEXT *esysContext,
    ESYS_TR      *sequenceHandle);

TSS2_DLL_EXPORT TSS2_RC Esys_HMAC_Start(
    ESYS_CONTEXT     *esysContext,
    ESYS_TR           handle,
    ESYS_TR           handleSession1,
    ESYS_TR           optionalSession2,
    ESYS_TR           optionalSession3,
    TPM2B_AUTH const *auth,
    TPMI_ALG_HASH     hashAlg,
    ESYS_TR          *sequenceHandle);
```

11.3.33 Esys_HashSequenceStart Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_HashSequenceStart_Async(
    ESYS_CONTEXT     *esysContext,
    ESYS_TR           optionalSession1,
    ESYS_TR           optionalSession2,
    ESYS_TR           optionalSession3,
    TPM2B_AUTH const *auth,
    TPMI_ALG_HASH     hashAlg);

TSS2_DLL_EXPORT TSS2_RC Esys_HashSequenceStart_Finish(
    ESYS_CONTEXT *esysContext,
    ESYS_TR      *sequenceHandle);

TSS2_DLL_EXPORT TSS2_RC Esys_HashSequenceStart(
    ESYS_CONTEXT     *esysContext,
    ESYS_TR           optionalSession1,
    ESYS_TR           optionalSession2,
    ESYS_TR           optionalSession3,
    TPM2B_AUTH const *auth,
    TPMI_ALG_HASH     hashAlg,
    ESYS_TR          *sequenceHandle);
```

11.3.34 Esys_SequenceUpdate Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_SequenceUpdate_Async(
    ESYS_CONTEXT           *esysContext,
    ESYS_TR                 sequenceHandle,
    ESYS_TR                 sequenceHandleSession1,
    ESYS_TR                 optionalSession2,
    ESYS_TR                 optionalSession3,
    TPM2B_MAX_BUFFER const *buffer);

TSS2_DLL_EXPORT TSS2_RC Esys_SequenceUpdate_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_SequenceUpdate(
    ESYS_CONTEXT           *esysContext,
    ESYS_TR                 sequenceHandle,
    ESYS_TR                 sequenceHandleSession1,
    ESYS_TR                 optionalSession2,
    ESYS_TR                 optionalSession3,
    TPM2B_MAX_BUFFER const *buffer);
```

11.3.35 Esys_SequenceComplete Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_SequenceComplete_Async(
    ESYS_CONTEXT           *esysContext,
    ESYS_TR                 sequenceHandle,
    ESYS_TR                 sequenceHandleSession1,
    ESYS_TR                 optionalSession2,
    ESYS_TR                 optionalSession3,
    TPM2B_MAX_BUFFER const *buffer,
    ESYS_TR                 hierarchy);

TSS2_DLL_EXPORT TSS2_RC Esys_SequenceComplete_Finish(
    ESYS_CONTEXT       *esysContext,
    TPM2B_DIGEST      **result,
    TPMT_TK_HASHCHECK **validation);

TSS2_DLL_EXPORT TSS2_RC Esys_SequenceComplete(
    ESYS_CONTEXT           *esysContext,
    ESYS_TR                 sequenceHandle,
    ESYS_TR                 sequenceHandleSession1,
    ESYS_TR                 optionalSession2,
    ESYS_TR                 optionalSession3,
    TPM2B_MAX_BUFFER const *buffer,
    ESYS_TR                 hierarchy,
    TPM2B_DIGEST          **result,
    TPMT_TK_HASHCHECK     **validation);
```

11.3.36 Esys_EventSequenceComplete Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_EventSequenceComplete_Async(
    ESYS_CONTEXT           *esysContext,
    ESYS_TR                 pcrHandle,
    ESYS_TR                  sequenceHandle,
    ESYS_TR                  pcrHandleSession1,
    ESYS_TR                  sequenceHandleSession2,
    ESYS_TR                  optionalSession3,
    TPM2B_MAX_BUFFER const *buffer);

TSS2_DLL_EXPORT TSS2_RC Esys_EventSequenceComplete_Finish(
    ESYS_CONTEXT        *esysContext,
    TPML_DIGEST_VALUES **results);

TSS2_DLL_EXPORT TSS2_RC Esys_EventSequenceComplete(
    ESYS_CONTEXT           *esysContext,
    ESYS_TR                  pcrHandle,
    ESYS_TR                  sequenceHandle,
    ESYS_TR                  pcrHandleSession1,
    ESYS_TR                  sequenceHandleSession2,
    ESYS_TR                  optionalSession3,
    TPM2B_MAX_BUFFER const *buffer,
    TPML_DIGEST_VALUES    **results);
```

11.3.37 Esys_Certify Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_Certify_Async(
    ESYS_CONTEXT          *esysContext,
    ESYS_TR                objectHandle,
    ESYS_TR                signHandle,
    ESYS_TR                objectHandleSession1,
    ESYS_TR                signHandleSession2,
    ESYS_TR                optionalSession3,
    TPM2B_DATA      const *qualifyingData,
    TPMT_SIG_SCHEME const *inScheme);

TSS2_DLL_EXPORT TSS2_RC Esys_Certify_Finish(
    ESYS_CONTEXT    *esysContext,
    TPM2B_ATTEST   **certifyInfo,
    TPMT_SIGNATURE **signature);

TSS2_DLL_EXPORT TSS2_RC Esys_Certify(
    ESYS_CONTEXT          *esysContext,
    ESYS_TR                objectHandle,
    ESYS_TR                signHandle,
    ESYS_TR                objectHandleSession1,
    ESYS_TR                signHandleSession2,
    ESYS_TR                optionalSession3,
    TPM2B_DATA      const *qualifyingData,
    TPMT_SIG_SCHEME const *inScheme,
    TPM2B_ATTEST         **certifyInfo,
    TPMT_SIGNATURE       **signature);
```

11.3.38 Esys_CertifyCreation Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_CertifyCreation_Async(
    ESYS_CONTEXT           *esysContext,
    ESYS_TR                  signHandle,
    ESYS_TR                  objectHandle,
    ESYS_TR                  signHandleSession1,
    ESYS_TR                  optionalSession2,
    ESYS_TR                  optionalSession3,
    TPM2B_DATA        const *qualifyingData,
    TPM2B_DIGEST      const *creationHash,
    TPMT_SIG_SCHEME   const *inScheme,
    TPMT_TK_CREATION const *creationTicket);

TSS2_DLL_EXPORT TSS2_RC Esys_CertifyCreation_Finish(
    ESYS_CONTEXT     *esysContext,
    TPM2B_ATTEST    **certifyInfo,
    TPMT_SIGNATURE **signature);

TSS2_DLL_EXPORT TSS2_RC Esys_CertifyCreation(
    ESYS_CONTEXT            *esysContext,
    ESYS_TR                  signHandle,
    ESYS_TR                  objectHandle,
    ESYS_TR                  signHandleSession1,
    ESYS_TR                  optionalSession2,
    ESYS_TR                  optionalSession3,
    TPM2B_DATA        const *qualifyingData,
    TPM2B_DIGEST      const *creationHash,
    TPMT_SIG_SCHEME   const *inScheme,
    TPMT_TK_CREATION const *creationTicket,
    TPM2B_ATTEST           **certifyInfo,
    TPMT_SIGNATURE         **signature);
```

11.3.39 Esys_Quote Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_Quote_Async(
    ESYS_CONTEXT              *esysContext,
    ESYS_TR                    signHandle,
    ESYS_TR                    signHandleSession1,
    ESYS_TR                    optionalSession2,
    ESYS_TR                    optionalSession3,
    TPM2B_DATA          const *qualifyingData,
    TPMT_SIG_SCHEME     const *inScheme,
    TPML_PCR_SELECTION const *PCRselect);

TSS2_DLL_EXPORT TSS2_RC Esys_Quote_Finish(
    ESYS_CONTEXT     *esysContext,
    TPM2B_ATTEST    **quoted,
    TPMT_SIGNATURE **signature);

TSS2_DLL_EXPORT TSS2_RC Esys_Quote(
    ESYS_CONTEXT              *esysContext,
    ESYS_TR                    signHandle,
    ESYS_TR                    signHandleSession1,
    ESYS_TR                    optionalSession2,
    ESYS_TR                    optionalSession3,
    TPM2B_DATA          const *qualifyingData,
    TPMT_SIG_SCHEME     const *inScheme,
    TPML_PCR_SELECTION const *PCRselect,
    TPM2B_ATTEST                **quoted,
    TPMT_SIGNATURE              **signature);
```

11.3.40 Esys_GetSessionAuditDigest Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_GetSessionAuditDigest_Async(
    ESYS_CONTEXT              *esysContext,
    ESYS_TR                    privacyAdminHandle,
    ESYS_TR                    signHandle,
    ESYS_TR                    sessionHandle,
    ESYS_TR                    privacyAdminHandleSession1,
    ESYS_TR                    signHandleSession2,
    ESYS_TR                    optionalSession3,
    TPM2B_DATA      const *qualifyingData,
    TPMT_SIG_SCHEME const *inScheme);

TSS2_DLL_EXPORT TSS2_RC Esys_GetSessionAuditDigest_Finish(
    ESYS_CONTEXT    *esysContext,
    TPM2B_ATTEST   **auditInfo,
    TPMT_SIGNATURE **signature);

TSS2_DLL_EXPORT TSS2_RC Esys_GetSessionAuditDigest(
    ESYS_CONTEXT              *esysContext,
    ESYS_TR                    privacyAdminHandle,
    ESYS_TR                    signHandle,
    ESYS_TR                    sessionHandle,
    ESYS_TR                    privacyAdminHandleSession1,
    ESYS_TR                    signHandleSession2,
    ESYS_TR                    optionalSession3,
    TPM2B_DATA      const *qualifyingData,
    TPMT_SIG_SCHEME const *inScheme,
    TPM2B_ATTEST          **auditInfo,
    TPMT_SIGNATURE        **signature);
```

11.3.41 Esys_GetCommandAuditDigest Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_GetCommandAuditDigest_Async(
    ESYS_CONTEXT              *esysContext,
    ESYS_TR                    privacyHandle,
    ESYS_TR                    signHandle,
    ESYS_TR                    privacyHandleSession1,
    ESYS_TR                    signHandleSession2,
    ESYS_TR                    optionalSession3,
    TPM2B_DATA      const *qualifyingData,
    TPMT_SIG_SCHEME const *inScheme);

TSS2_DLL_EXPORT TSS2_RC Esys_GetCommandAuditDigest_Finish(
    ESYS_CONTEXT    *esysContext,
    TPM2B_ATTEST   **auditInfo,
    TPMT_SIGNATURE **signature);

TSS2_DLL_EXPORT TSS2_RC Esys_GetCommandAuditDigest(
    ESYS_CONTEXT              *esysContext,
    ESYS_TR                 privacyHandle,
    ESYS_TR                 signHandle,
    ESYS_TR                 privacyHandleSession1,
    ESYS_TR                 signHandleSession2,
    ESYS_TR                 optionalSession3,
    TPM2B_DATA      const *qualifyingData,
    TPMT_SIG_SCHEME const *inScheme,
    TPM2B_ATTEST          **auditInfo,
    TPMT_SIGNATURE        **signature);
```

11.3.42 Esys_GetTime Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_GetTime_Async(
    ESYS_CONTEXT           *esysContext,
    ESYS_TR                 privacyAdminHandle,
    ESYS_TR                 signHandle,
    ESYS_TR                 privacyAdminHandleSession1,
    ESYS_TR                 signHandleSession2,
    ESYS_TR                 optionalSession3,
    TPM2B_DATA      const *qualifyingData,
    TPMT_SIG_SCHEME const *inScheme);

TSS2_DLL_EXPORT TSS2_RC Esys_GetTime_Finish(
    ESYS_CONTEXT    *esysContext,
    TPM2B_ATTEST   **timeInfo,
    TPMT_SIGNATURE **signature);

TSS2_DLL_EXPORT TSS2_RC Esys_GetTime(
    ESYS_CONTEXT           *esysContext,
    ESYS_TR                 privacyAdminHandle,
    ESYS_TR                 signHandle,
    ESYS_TR                 privacyAdminHandleSession1,
    ESYS_TR                 signHandleSession2,
    ESYS_TR                 optionalSession3,
    TPM2B_DATA      const *qualifyingData,
    TPMT_SIG_SCHEME const *inScheme,
    TPM2B_ATTEST          **timeInfo,
    TPMT_SIGNATURE        **signature);
```

11.3.43 Esys_Commit Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_Commit_Async(
    ESYS_CONTEXT                *esysContext,
    ESYS_TR                      signHandle,
    ESYS_TR                      signHandleSession1,
    ESYS_TR                      optionalSession2,
    ESYS_TR                      optionalSession3,
    TPM2B_ECC_POINT      const *P1,
    TPM2B_SENSITIVE_DATA const *s2,
    TPM2B_ECC_PARAMETER  const *y2);

TSS2_DLL_EXPORT TSS2_RC Esys_Commit_Finish(
    ESYS_CONTEXT     *esysContext,
    TPM2B_ECC_POINT **K,
    TPM2B_ECC_POINT **L,
    TPM2B_ECC_POINT **E,
    UINT16           *counter);

TSS2_DLL_EXPORT TSS2_RC Esys_Commit(
    ESYS_CONTEXT                 *esysContext,
    ESYS_TR                       signHandle,
    ESYS_TR                       signHandleSession1,
    ESYS_TR                       optionalSession2,
    ESYS_TR                       optionalSession3,
    TPM2B_ECC_POINT      const *P1,
    TPM2B_SENSITIVE_DATA const *s2,
    TPM2B_ECC_PARAMETER  const *y2,
    TPM2B_ECC_POINT            **K,
    TPM2B_ECC_POINT            **L,
    TPM2B_ECC_POINT            **E,
    UINT16                       *counter);
```

11.3.44 Esys_EC_Ephemeral Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_EC_Ephemeral_Async(
    ESYS_CONTEXT  *esysContext,
    ESYS_TR        optionalSession1,
    ESYS_TR        optionalSession2,
    ESYS_TR        optionalSession3,
    TPMI_ECC_CURVE curveID);

TSS2_DLL_EXPORT TSS2_RC Esys_EC_Ephemeral_Finish(
    ESYS_CONTEXT     *esysContext,
    TPM2B_ECC_POINT **Q,
    UINT16           *counter);

TSS2_DLL_EXPORT TSS2_RC Esys_EC_Ephemeral(
    ESYS_CONTEXT     *esysContext,
    ESYS_TR           optionalSession1,
    ESYS_TR           optionalSession2,
    ESYS_TR           optionalSession3,
    TPMI_ECC_CURVE    curveID,
    TPM2B_ECC_POINT **Q,
    UINT16           *counter);
```

11.3.45 Esys_VerifySignature Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_VerifySignature_Async(
    ESYS_CONTEXT           *esysContext,
    ESYS_TR                 keyHandle,
    ESYS_TR                 optionalSession1,
    ESYS_TR                 optionalSession2,
    ESYS_TR                 optionalSession3,
    TPM2B_DIGEST   const *digest,
    TPMT_SIGNATURE const *signature);
```

```
TSS2_DLL_EXPORT TSS2_RC Esys_VerifySignature_Finish(
    ESYS_CONTEXT      *esysContext,
    TPMT_TK_VERIFIED **validation);

TSS2_DLL_EXPORT TSS2_RC Esys_VerifySignature(
    ESYS_CONTEXT         *esysContext,
    ESYS_TR               keyHandle,
    ESYS_TR               optionalSession1,
    ESYS_TR               optionalSession2,
    ESYS_TR               optionalSession3,
    TPM2B_DIGEST   const *digest,
    TPMT_SIGNATURE const *signature,
    TPMT_TK_VERIFIED    **validation);
```

11.3.46 Esys_Sign Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_Sign_Async(
    ESYS_CONTEXT            *esysContext,
    ESYS_TR                  keyHandle,
    ESYS_TR                  keyHandleSession1,
    ESYS_TR                  optionalSession2,
    ESYS_TR                  optionalSession3,
    TPM2B_DIGEST      const *digest,
    TPMT_SIG_SCHEME   const *inScheme,
    TPMT_TK_HASHCHECK const *validation);

TSS2_DLL_EXPORT TSS2_RC Esys_Sign_Finish(
    ESYS_CONTEXT    *esysContext,
    TPMT_SIGNATURE **signature);

TSS2_DLL_EXPORT TSS2_RC Esys_Sign(
    ESYS_CONTEXT            *esysContext,
    ESYS_TR                  keyHandle,
    ESYS_TR                  keyHandleSession1,
    ESYS_TR                  optionalSession2,
    ESYS_TR                  optionalSession3,
    TPM2B_DIGEST      const *digest,
    TPMT_SIG_SCHEME   const *inScheme,
    TPMT_TK_HASHCHECK const *validation,
    TPMT_SIGNATURE         **signature);
```

11.3.47 Esys_SetCommandCodeAuditStatus Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_SetCommandCodeAuditStatus_Async(
    ESYS_CONTEXT  *esysContext,
    ESYS_TR        auth,
    ESYS_TR        authSession1,
    ESYS_TR        optionalSession2,
    ESYS_TR        optionalSession3,
    TPMI_ALG_HASH  auditAlg,
    TPML_CC const *setList,
    TPML_CC const *clearList);
```

```
TSS2_DLL_EXPORT TSS2_RC Esys_SetCommandCodeAuditStatus_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_SetCommandCodeAuditStatus(
    ESYS_CONTEXT  *esysContext,
    ESYS_TR        auth,
    ESYS_TR        authSession1,
    ESYS_TR        optionalSession2,
    ESYS_TR        optionalSession3,
    TPMI_ALG_HASH  auditAlg,
    TPML_CC const *setList,
    TPML_CC const *clearList);
```

11.3.48 Esys_PCR_Extend Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_PCR_Extend_Async(
    ESYS_CONTEXT             *esysContext,
    ESYS_TR                   pcrHandle,
    ESYS_TR                   pcrHandleSession1,
    ESYS_TR                   optionalSession2,
    ESYS_TR                   optionalSession3,
    TPML_DIGEST_VALUES const *digests);

TSS2_DLL_EXPORT TSS2_RC Esys_PCR_Extend_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_PCR_Extend(
    ESYS_CONTEXT             *esysContext,
    ESYS_TR                   pcrHandle,
    ESYS_TR                   pcrHandleSession1,
    ESYS_TR                   optionalSession2,
    ESYS_TR                   optionalSession3,
    TPML_DIGEST_VALUES const *digests);
```

11.3.49 Esys_PCR_Event Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_PCR_Event_Async(
    ESYS_CONTEXT      *esysContext,
    ESYS_TR            pcrHandle,
    ESYS_TR            pcrHandleSession1,
    ESYS_TR            optionalSession2,
    ESYS_TR            optionalSession3,
    TPM2B_EVENT const *eventData);

TSS2_DLL_EXPORT TSS2_RC Esys_PCR_Event_Finish(
    ESYS_CONTEXT        *esysContext,
    TPML_DIGEST_VALUES **digests);

TSS2_DLL_EXPORT TSS2_RC Esys_PCR_Event(
    ESYS_CONTEXT        *esysContext,
    ESYS_TR              pcrHandle,
    ESYS_TR              pcrHandleSession1,
    ESYS_TR              optionalSession2,
    ESYS_TR              optionalSession3,
    TPM2B_EVENT   const *eventData,
    TPML_DIGEST_VALUES **digests);
```

11.3.50 Esys_PCR_Read Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_PCR_Read_Async(
    ESYS_CONTEXT              *esysContext,
    ESYS_TR                    optionalSession1,
    ESYS_TR                    optionalSession2,
    ESYS_TR                    optionalSession3,
    TPML_PCR_SELECTION const *pcrSelectionIn);

TSS2_DLL_EXPORT TSS2_RC Esys_PCR_Read_Finish(
    ESYS_CONTEXT         *esysContext,
    UINT32               *pcrUpdateCounter,
    TPML_PCR_SELECTION **pcrSelectionOut,
    TPML_DIGEST        **pcrValues);

TSS2_DLL_EXPORT TSS2_RC Esys_PCR_Read(
    ESYS_CONTEXT              *esysContext,
    ESYS_TR                    optionalSession1,
    ESYS_TR                    optionalSession2,
    ESYS_TR                    optionalSession3,
    TPML_PCR_SELECTION const *pcrSelectionIn,
    UINT32                    *pcrUpdateCounter,
    TPML_PCR_SELECTION       **pcrSelectionOut,
    TPML_DIGEST              **pcrValues);
```

11.3.51 Esys_PCR_Allocate Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_PCR_Allocate_Async(
    ESYS_CONTEXT              *esysContext,
    ESYS_TR                    authHandle,
    ESYS_TR                    authHandleSession1,
    ESYS_TR                    optionalSession2,
    ESYS_TR                    optionalSession3,
    TPML_PCR_SELECTION const *pcrAllocation);

TSS2_DLL_EXPORT TSS2_RC Esys_PCR_Allocate_Finish(
    ESYS_CONTEXT *esysContext,
    TPMI_YES_NO  *allocationSuccess,
    UINT32       *maxPCR,
    UINT32       *sizeNeeded,
    UINT32       *sizeAvailable);

TSS2_DLL_EXPORT TSS2_RC Esys_PCR_Allocate(
    ESYS_CONTEXT              *esysContext,
    ESYS_TR                    authHandle,
    ESYS_TR                    authHandleSession1,
    ESYS_TR                    optionalSession2,
    ESYS_TR                    optionalSession3,
    TPML_PCR_SELECTION const *pcrAllocation,
    TPMI_YES_NO              *allocationSuccess,
    UINT32                   *maxPCR,
    UINT32                   *sizeNeeded,
    UINT32                   *sizeAvailable);
```

11.3.52 Esys_PCR_SetAuthPolicy Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_PCR_SetAuthPolicy_Async(
    ESYS_CONTEXT       *esysContext,
    ESYS_TR             authHandle,
    ESYS_TR             authHandleSession1,
    ESYS_TR             optionalSession2,
    ESYS_TR             optionalSession3,
    TPM2B_DIGEST const *authPolicy,
    TPMI_ALG_HASH       hashAlg,
    ESYS_TR             pcrNum);

TSS2_DLL_EXPORT TSS2_RC Esys_PCR_SetAuthPolicy_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_PCR_SetAuthPolicy(
    ESYS_CONTEXT       *esysContext,
    ESYS_TR             authHandle,
    ESYS_TR             authHandleSession1,
    ESYS_TR             optionalSession2,
    ESYS_TR             optionalSession3,
    TPM2B_DIGEST const *authPolicy,
    TPMI_ALG_HASH       hashAlg,
    ESYS_TR             pcrNum);
```

11.3.53 Esys_PCR_SetAuthValue Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_PCR_SetAuthValue_Async(
    ESYS_CONTEXT       *esysContext,
    ESYS_TR             pcrHandle,
    ESYS_TR             pcrHandleSession1,
    ESYS_TR             optionalSession2,
    ESYS_TR             optionalSession3,
    TPM2B_DIGEST const *auth);

TSS2_DLL_EXPORT TSS2_RC Esys_PCR_SetAuthValue_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_PCR_SetAuthValue(
    ESYS_CONTEXT       *esysContext,
    ESYS_TR             pcrHandle,
    ESYS_TR             pcrHandleSession1,
    ESYS_TR             optionalSession2,
    ESYS_TR             optionalSession3,
    TPM2B_DIGEST const *auth);
```

11.3.54 Esys_PCR_Reset Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_PCR_Reset_Async(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       pcrHandle,
    ESYS_TR       pcrHandleSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3);

TSS2_DLL_EXPORT TSS2_RC Esys_PCR_Reset_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_PCR_Reset(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       pcrHandle,
    ESYS_TR       pcrHandleSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3);
```

11.3.55 Esys_PolicySigned Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_PolicySigned_Async(
    ESYS_CONTEXT        *esysContext,
    ESYS_TR              authObject,
    ESYS_TR              policySession,
    ESYS_TR              optionalSession1,
    ESYS_TR              optionalSession2,
    ESYS_TR              optionalSession3,
    TPM2B_DIGEST   const *cpHashA,
    TPM2B_NONCE    const *policyRef,
    INT32                expiration,
    TPMT_SIGNATURE const *auth);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicySigned_Finish(
    ESYS_CONTEXT   *esysContext,
    TPM2B_TIMEOUT **timeout,
    TPMT_TK_AUTH  **policyTicket);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicySigned(
    ESYS_CONTEXT        *esysContext,
    ESYS_TR              authObject,
    ESYS_TR              policySession,
    ESYS_TR              optionalSession1,
    ESYS_TR              optionalSession2,
    ESYS_TR              optionalSession3,
    TPM2B_DIGEST   const *cpHashA,
    TPM2B_NONCE    const *policyRef,
    INT32                expiration,
    TPMT_SIGNATURE const *auth,
    TPM2B_TIMEOUT       **timeout,
    TPMT_TK_AUTH        **policyTicket);
```

11.3.56 Esys_PolicySecret Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_PolicySecret_Async(
    ESYS_CONTEXT       *esysContext,
    ESYS_TR             authHandle,
    ESYS_TR             policySession,
    ESYS_TR             authHandleSession1,
    ESYS_TR             optionalSession2,
    ESYS_TR             optionalSession3,
    TPM2B_DIGEST const *cpHashA,
    TPM2B_NONCE  const *policyRef,
    INT32               expiration);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicySecret_Finish(
    ESYS_CONTEXT   *esysContext,
    TPM2B_TIMEOUT **timeout,
    TPMT_TK_AUTH  **policyTicket);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicySecret(
    ESYS_CONTEXT       *esysContext,
    ESYS_TR             authHandle,
    ESYS_TR             policySession,
    ESYS_TR             authHandleSession1,
    ESYS_TR             optionalSession2,
    ESYS_TR             optionalSession3,
    TPM2B_DIGEST const *cpHashA,
    TPM2B_NONCE  const *policyRef,
    INT32               expiration,
    TPM2B_TIMEOUT     **timeout,
    TPMT_TK_AUTH      **policyTicket);
```

11.3.57 Esys_PolicyTicket Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_PolicyTicket_Async(
    ESYS_CONTEXT        *esysContext,
    ESYS_TR              policySession,
    ESYS_TR              optionalSession1,
    ESYS_TR              optionalSession2,
    ESYS_TR              optionalSession3,
    TPM2B_TIMEOUT const *timeout,
    TPM2B_DIGEST  const *cpHashA,
    TPM2B_NONCE   const *policyRef,
    TPM2B_NAME    const *authName,
    TPMT_TK_AUTH  const *ticket);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicyTicket_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicyTicket(
    ESYS_CONTEXT        *esysContext,
    ESYS_TR              policySession,
    ESYS_TR              optionalSession1,
    ESYS_TR              optionalSession2,
    ESYS_TR              optionalSession3,
    TPM2B_TIMEOUT const *timeout,
    TPM2B_DIGEST  const *cpHashA,
    TPM2B_NONCE   const *policyRef,
    TPM2B_NAME    const *authName,
    TPMT_TK_AUTH  const *ticket);
```

11.3.58 Esys_PolicyOR Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_PolicyOR_Async(
    ESYS_CONTEXT      *esysContext,
    ESYS_TR            policySession,
    ESYS_TR            optionalSession1,
    ESYS_TR            optionalSession2,
    ESYS_TR            optionalSession3,
    TPML_DIGEST const *pHashList);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicyOR_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicyOR(
    ESYS_CONTEXT      *esysContext,
    ESYS_TR            policySession,
    ESYS_TR            optionalSession1,
    ESYS_TR            optionalSession2,
    ESYS_TR            optionalSession3,
    TPML_DIGEST const *pHashList);
```

11.3.59 Esys_PolicyPCR Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_PolicyPCR_Async(
    ESYS_CONTEXT             *esysContext,
    ESYS_TR                   policySession,
    ESYS_TR                   optionalSession1,
    ESYS_TR                   optionalSession2,
    ESYS_TR                   optionalSession3,
    TPM2B_DIGEST       const *pcrDigest,
    TPML_PCR_SELECTION const *pcrs);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicyPCR_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicyPCR(
    ESYS_CONTEXT             *esysContext,
    ESYS_TR                   policySession,
    ESYS_TR                   optionalSession1,
    ESYS_TR                   optionalSession2,
    ESYS_TR                   optionalSession3,
    TPM2B_DIGEST       const *pcrDigest,
    TPML_PCR_SELECTION const *pcrs);
```

11.3.60 Esys_PolicyLocality Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_PolicyLocality_Async(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       policySession,
    ESYS_TR       optionalSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3,
    TPMA_LOCALITY locality);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicyLocality_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicyLocality(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       policySession,
    ESYS_TR       optionalSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3,
    TPMA_LOCALITY locality);
```

11.3.61 Esys_PolicyNV Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_PolicyNV_Async(
    ESYS_CONTEXT        *esysContext,
    ESYS_TR              authHandle,
    ESYS_TR              nvIndex,
    ESYS_TR              policySession,
    ESYS_TR              authHandleSession1,
    ESYS_TR              optionalSession2,
    ESYS_TR              optionalSession3,
    TPM2B_OPERAND const *operandB,
    UINT16               offset,
    TPM2_EO              operation);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicyNV_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicyNV(
    ESYS_CONTEXT        *esysContext,
    ESYS_TR              authHandle,
    ESYS_TR              nvIndex,
    ESYS_TR              policySession,
    ESYS_TR              authHandleSession1,
    ESYS_TR              optionalSession2,
    ESYS_TR              optionalSession3,
    TPM2B_OPERAND const *operandB,
    UINT16               offset,
    TPM2_EO              operation);
```

11.3.62 Esys_PolicyCounterTimer Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_PolicyCounterTimer_Async(
    ESYS_CONTEXT        *esysContext,
    ESYS_TR              policySession,
    ESYS_TR              optionalSession1,
    ESYS_TR              optionalSession2,
    ESYS_TR              optionalSession3,
    TPM2B_OPERAND const *operandB,
    UINT16               offset,
    TPM2_EO              operation);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicyCounterTimer_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicyCounterTimer(
    ESYS_CONTEXT        *esysContext,
    ESYS_TR              policySession,
    ESYS_TR              optionalSession1,
    ESYS_TR              optionalSession2,
    ESYS_TR              optionalSession3,
    TPM2B_OPERAND const *operandB,
    UINT16               offset,
    TPM2_EO              operation);
```

11.3.63 Esys_PolicyCommandCode Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_PolicyCommandCode_Async(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       policySession,
    ESYS_TR       optionalSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3,
    TPM2_CC       code);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicyCommandCode_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicyCommandCode(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       policySession,
    ESYS_TR       optionalSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3,
    TPM2_CC       code);
```

11.3.64 Esys_PolicyPhysicalPresence Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_PolicyPhysicalPresence_Async(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       policySession,
    ESYS_TR       optionalSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicyPhysicalPresence_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicyPhysicalPresence(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       policySession,
    ESYS_TR       optionalSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3);
```

11.3.65 Esys_PolicyCpHash Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_PolicyCpHash_Async(
    ESYS_CONTEXT       *esysContext,
    ESYS_TR             policySession,
    ESYS_TR             optionalSession1,
    ESYS_TR             optionalSession2,
    ESYS_TR             optionalSession3,
    TPM2B_DIGEST const *cpHashA);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicyCpHash_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicyCpHash(
    ESYS_CONTEXT       *esysContext,
    ESYS_TR             policySession,
    ESYS_TR             optionalSession1,
    ESYS_TR             optionalSession2,
    ESYS_TR             optionalSession3,
    TPM2B_DIGEST const *cpHashA);
```

11.3.66 Esys_PolicyNameHash Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_PolicyNameHash_Async(
    ESYS_CONTEXT       *esysContext,
    ESYS_TR             policySession,
    ESYS_TR             optionalSession1,
    ESYS_TR             optionalSession2,
    ESYS_TR             optionalSession3,
    TPM2B_DIGEST const *nameHash);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicyNameHash_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicyNameHash(
    ESYS_CONTEXT       *esysContext,
    ESYS_TR             policySession,
    ESYS_TR             optionalSession1,
    ESYS_TR             optionalSession2,
    ESYS_TR             optionalSession3,
    TPM2B_DIGEST const *nameHash);
```

11.3.67 Esys_PolicyDuplicationSelect Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_PolicyDuplicationSelect_Async(
    ESYS_CONTEXT     *esysContext,
    ESYS_TR           policySession,
    ESYS_TR           optionalSession1,
    ESYS_TR           optionalSession2,
    ESYS_TR           optionalSession3,
    TPM2B_NAME const *objectName,
    TPM2B_NAME const *newParentName,
    TPMI_YES_NO       includeObject);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicyDuplicationSelect_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicyDuplicationSelect(
    ESYS_CONTEXT     *esysContext,
    ESYS_TR           policySession,
    ESYS_TR           optionalSession1,
    ESYS_TR           optionalSession2,
    ESYS_TR           optionalSession3,
    TPM2B_NAME const *objectName,
    TPM2B_NAME const *newParentName,
    TPMI_YES_NO       includeObject);
```

11.3.68 Esys_PolicyAuthorize Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_PolicyAuthorize_Async(
    ESYS_CONTEXT           *esysContext,
    ESYS_TR                 policySession,
    ESYS_TR                 optionalSession1,
    ESYS_TR                 optionalSession2,
    ESYS_TR                 optionalSession3,
    TPM2B_DIGEST      const *approvedPolicy,
    TPM2B_NONCE       const *policyRef,
    TPM2B_NAME        const *keySign,
    TPMT_TK_VERIFIED const *checkTicket);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicyAuthorize_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicyAuthorize(
    ESYS_CONTEXT           *esysContext,
    ESYS_TR                 policySession,
    ESYS_TR                 optionalSession1,
    ESYS_TR                 optionalSession2,
    ESYS_TR                 optionalSession3,
    TPM2B_DIGEST      const *approvedPolicy,
    TPM2B_NONCE       const *policyRef,
    TPM2B_NAME        const *keySign,
    TPMT_TK_VERIFIED const *checkTicket);
```

11.3.69 Esys_PolicyAuthValue Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_PolicyAuthValue_Async(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       policySession,
    ESYS_TR       optionalSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicyAuthValue_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicyAuthValue(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       policySession,
    ESYS_TR       optionalSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3);
```

11.3.70 Esys_PolicyPassword Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_PolicyPassword_Async(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       policySession,
    ESYS_TR       optionalSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicyPassword_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicyPassword(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       policySession,
    ESYS_TR       optionalSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3);
```

11.3.71 Esys_PolicyGetDigest Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_PolicyGetDigest_Async(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       policySession,
    ESYS_TR       optionalSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicyGetDigest_Finish(
    ESYS_CONTEXT  *esysContext,
    TPM2B_DIGEST **policyDigest);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicyGetDigest(
    ESYS_CONTEXT  *esysContext,
    ESYS_TR        policySession,
    ESYS_TR        optionalSession1,
    ESYS_TR        optionalSession2,
    ESYS_TR        optionalSession3,
    TPM2B_DIGEST **policyDigest);
```

11.3.72 Esys_PolicyNvWritten Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_PolicyNvWritten_Async(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       policySession,
    ESYS_TR       optionalSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3,
    TPMI_YES_NO   writtenSet);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicyNvWritten_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicyNvWritten(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       policySession,
    ESYS_TR       optionalSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3,
    TPMI_YES_NO   writtenSet);
```

11.3.73 Esys_PolicyTemplate Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_PolicyTemplate_Async(
    ESYS_CONTEXT       *esysContext,
    ESYS_TR             policySession,
    ESYS_TR             optionalSession1,
    ESYS_TR             optionalSession2,
    ESYS_TR             optionalSession3,
    TPM2B_DIGEST const *templateHash);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicyTemplate_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicyTemplate(
    ESYS_CONTEXT       *esysContext,
    ESYS_TR             policySession,
    ESYS_TR             optionalSession1,
    ESYS_TR             optionalSession2,
    ESYS_TR             optionalSession3,
    TPM2B_DIGEST const *templateHash);
```

11.3.74 Esys_PolicyAuthorizeNV Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_PolicyAuthorizeNV_Async(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       authHandle,
    ESYS_TR       nvIndex,
    ESYS_TR       policySession,
    ESYS_TR       authHandleSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicyAuthorizeNV_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_PolicyAuthorizeNV(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       authHandle,
    ESYS_TR       nvIndex,
    ESYS_TR       policySession,
    ESYS_TR       authHandleSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3);
```

11.3.75 Esys_CreatePrimary Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_CreatePrimary_Async(
    ESYS_CONTEXT                *esysContext,
    ESYS_TR                      primaryHandle,
    ESYS_TR                      primaryHandleSession1,
    ESYS_TR                      optionalSession2,
    ESYS_TR                      optionalSession3,
    TPM2B_SENSITIVE_CREATE const *inSensitive,
    TPM2B_PUBLIC           const *inPublic,
    TPM2B_DATA             const *outsideInfo,
    TPML_PCR_SELECTION     const *creationPCR);

TSS2_DLL_EXPORT TSS2_RC Esys_CreatePrimary_Finish(
    ESYS_CONTEXT         *esysContext,
    ESYS_TR              *objectHandle,
    TPM2B_PUBLIC        **outPublic,
    TPM2B_CREATION_DATA **creationData,
    TPM2B_DIGEST        **creationHash,
    TPMT_TK_CREATION    **creationTicket);

TSS2_DLL_EXPORT TSS2_RC Esys_CreatePrimary(
    ESYS_CONTEXT                *esysContext,
    ESYS_TR                      primaryHandle,
    ESYS_TR                      primaryHandleSession1,
    ESYS_TR                      optionalSession2,
    ESYS_TR                      optionalSession3,
    TPM2B_SENSITIVE_CREATE const *inSensitive,
    TPM2B_PUBLIC           const *inPublic,
    TPM2B_DATA             const *outsideInfo,
    TPML_PCR_SELECTION     const *creationPCR,
    ESYS_TR                     *objectHandle,
    TPM2B_PUBLIC               **outPublic,
    TPM2B_CREATION_DATA        **creationData,
    TPM2B_DIGEST               **creationHash,
    TPMT_TK_CREATION            **creationTicket);
```

11.3.76 Esys_HierarchyControl Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_HierarchyControl_Async(
    ESYS_CONTEXT   *esysContext,
    ESYS_TR         authHandle,
    ESYS_TR         authHandleSession1,
    ESYS_TR         optionalSession2,
    ESYS_TR         optionalSession3,
    ESYS_TR         enable,
    TPMI_YES_NO     state);

TSS2_DLL_EXPORT TSS2_RC Esys_HierarchyControl_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_HierarchyControl(
    ESYS_CONTEXT   *esysContext,
    ESYS_TR         authHandle,
    ESYS_TR         authHandleSession1,
    ESYS_TR         optionalSession2,
    ESYS_TR         optionalSession3,
    ESYS_TR         enable,
    TPMI_YES_NO     state);
```

11.3.77 Esys_SetPrimaryPolicy Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_SetPrimaryPolicy_Async(
    ESYS_CONTEXT       *esysContext,
    ESYS_TR             authHandle,
    ESYS_TR             authHandleSession1,
    ESYS_TR             optionalSession2,
    ESYS_TR             optionalSession3,
    TPM2B_DIGEST const *authPolicy,
    TPMI_ALG_HASH       hashAlg);

TSS2_DLL_EXPORT TSS2_RC Esys_SetPrimaryPolicy_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_SetPrimaryPolicy(
    ESYS_CONTEXT       *esysContext,
    ESYS_TR             authHandle,
    ESYS_TR             authHandleSession1,
    ESYS_TR             optionalSession2,
    ESYS_TR             optionalSession3,
    TPM2B_DIGEST const *authPolicy,
    TPMI_ALG_HASH       hashAlg);
```

11.3.78 Esys_ChangePPS Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_ChangePPS_Async(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       authHandle,
    ESYS_TR       authHandleSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3);

TSS2_DLL_EXPORT TSS2_RC Esys_ChangePPS_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_ChangePPS(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       authHandle,
    ESYS_TR       authHandleSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3);
```

11.3.79 Esys_ChangeEPS Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_ChangeEPS_Async(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       authHandle,
    ESYS_TR       authHandleSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3);

TSS2_DLL_EXPORT TSS2_RC Esys_ChangeEPS_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_ChangeEPS(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       authHandle,
    ESYS_TR       authHandleSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3);
```

11.3.80 Esys_Clear Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_Clear_Async(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       authHandle,
    ESYS_TR       authHandleSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3);

TSS2_DLL_EXPORT TSS2_RC Esys_Clear_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_Clear(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       authHandle,
    ESYS_TR       authHandleSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3);
```

11.3.81 Esys_ClearControl Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_ClearControl_Async(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       auth,
    ESYS_TR       authSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3,
    TPMI_YES_NO   disable);

TSS2_DLL_EXPORT TSS2_RC Esys_ClearControl_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_ClearControl(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       auth,
    ESYS_TR       authSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3,
    TPMI_YES_NO   disable);
```

11.3.82 Esys_HierarchyChangeAuth Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_HierarchyChangeAuth_Async(
    ESYS_CONTEXT     *esysContext,
    ESYS_TR           authHandle,
    ESYS_TR           authHandleSession1,
    ESYS_TR           optionalSession2,
    ESYS_TR           optionalSession3,
    TPM2B_AUTH const *newAuth);

TSS2_DLL_EXPORT TSS2_RC Esys_HierarchyChangeAuth_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_HierarchyChangeAuth(
    ESYS_CONTEXT     *esysContext,
    ESYS_TR           authHandle,
    ESYS_TR           authHandleSession1,
    ESYS_TR           optionalSession2,
    ESYS_TR           optionalSession3,
    TPM2B_AUTH const *newAuth);
```

11.3.83 Esys_DictionaryAttackLockReset Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_DictionaryAttackLockReset_Async(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       lockHandle,
    ESYS_TR       lockHandleSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3);
```

```
TSS2_DLL_EXPORT TSS2_RC Esys_DictionaryAttackLockReset_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_DictionaryAttackLockReset(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       lockHandle,
    ESYS_TR       lockHandleSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3);
```

11.3.84 Esys_DictionaryAttackParameters Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_DictionaryAttackParameters_Async(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       lockHandle,
    ESYS_TR       lockHandleSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3,
    UINT32        newMaxTries,
    UINT32        newRecoveryTime,
    UINT32        lockoutRecovery);

TSS2_DLL_EXPORT TSS2_RC Esys_DictionaryAttackParameters_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_DictionaryAttackParameters(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       lockHandle,
    ESYS_TR       lockHandleSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3,
    UINT32        newMaxTries,
    UINT32        newRecoveryTime,
    UINT32        lockoutRecovery);
```

11.3.85 Esys_PP_Commands Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_PP_Commands_Async(
    ESYS_CONTEXT  *esysContext,
    ESYS_TR        auth,
    ESYS_TR        authSession1,
    ESYS_TR        optionalSession2,
    ESYS_TR        optionalSession3,
    TPML_CC const *setList,
    TPML_CC const *clearList);

TSS2_DLL_EXPORT TSS2_RC Esys_PP_Commands_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_PP_Commands(
    ESYS_CONTEXT  *esysContext,
    ESYS_TR        auth,
    ESYS_TR        authSession1,
    ESYS_TR        optionalSession2,
    ESYS_TR        optionalSession3,
    TPML_CC const *setList,
    TPML_CC const *clearList);
```

11.3.86 Esys_SetAlgorithmSet Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_SetAlgorithmSet_Async(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       authHandle,
    ESYS_TR       authHandleSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3,
    UINT32        algorithmSet);

TSS2_DLL_EXPORT TSS2_RC Esys_SetAlgorithmSet_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_SetAlgorithmSet(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       authHandle,
    ESYS_TR       authHandleSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3,
    UINT32        algorithmSet);
```

11.3.87 Esys_FieldUpgradeStart Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_FieldUpgradeStart_Async(
    ESYS_CONTEXT         *esysContext,
    ESYS_TR               authorization,
    ESYS_TR               keyHandle,
    ESYS_TR               authorizationSession1,
    ESYS_TR               optionalSession2,
    ESYS_TR               optionalSession3,
    TPM2B_DIGEST   const *fuDigest,
    TPMT_SIGNATURE const *manifestSignature);

TSS2_DLL_EXPORT TSS2_RC Esys_FieldUpgradeStart_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_FieldUpgradeStart(
    ESYS_CONTEXT         *esysContext,
    ESYS_TR               authorization,
    ESYS_TR               keyHandle,
    ESYS_TR               authorizationSession1,
    ESYS_TR               optionalSession2,
    ESYS_TR               optionalSession3,
    TPM2B_DIGEST   const *fuDigest,
    TPMT_SIGNATURE const *manifestSignature);
```

11.3.88 Esys_FieldUpgradeData Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_FieldUpgradeData_Async(
    ESYS_CONTEXT           *esysContext,
    ESYS_TR                 optionalSession1,
    ESYS_TR                 optionalSession2,
    ESYS_TR                 optionalSession3,
    TPM2B_MAX_BUFFER const *fuData);

TSS2_DLL_EXPORT TSS2_RC Esys_FieldUpgradeData_Finish(
    ESYS_CONTEXT *esysContext,
    TPMT_HA     **nextDigest,
    TPMT_HA     **firstDigest);

TSS2_DLL_EXPORT TSS2_RC Esys_FieldUpgradeData(
    ESYS_CONTEXT           *esysContext,
    ESYS_TR                 optionalSession1,
    ESYS_TR                 optionalSession2,
    ESYS_TR                 optionalSession3,
    TPM2B_MAX_BUFFER const *fuData,
    TPMT_HA               **nextDigest,
    TPMT_HA               **firstDigest);
```

11.3.89 Esys_FirmwareRead Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_FirmwareRead_Async(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       optionalSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3,
    UINT32        sequenceNumber);

TSS2_DLL_EXPORT TSS2_RC Esys_FirmwareRead_Finish(
    ESYS_CONTEXT      *esysContext,
    TPM2B_MAX_BUFFER **fuData);

TSS2_DLL_EXPORT TSS2_RC Esys_FirmwareRead(
    ESYS_CONTEXT      *esysContext,
    ESYS_TR            optionalSession1,
    ESYS_TR            optionalSession2,
    ESYS_TR            optionalSession3,
    UINT32             sequenceNumber,
    TPM2B_MAX_BUFFER **fuData);
```

11.3.90 Esys_ContextSave Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_ContextSave_Async(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       saveHandle);

TSS2_DLL_EXPORT TSS2_RC Esys_ContextSave_Finish(
    ESYS_CONTEXT  *esysContext,
    TPMS_CONTEXT **context);
```

```
TSS2_DLL_EXPORT TSS2_RC Esys_ContextSave(
    ESYS_CONTEXT  *esysContext,
    ESYS_TR        saveHandle,
    TPMS_CONTEXT **context);
```

11.3.91 Esys_ContextLoad Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_ContextLoad_Async(
    ESYS_CONTEXT       *esysContext,
    TPMS_CONTEXT const *context);

TSS2_DLL_EXPORT TSS2_RC Esys_ContextLoad_Finish(
    ESYS_CONTEXT *esysContext,
    ESYS_TR      *loadedHandle);

TSS2_DLL_EXPORT TSS2_RC Esys_ContextLoad(
    ESYS_CONTEXT       *esysContext,
    TPMS_CONTEXT const *context,
    ESYS_TR            *loadedHandle);
```

11.3.92 Esys_FlushContext Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_FlushContext_Async(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       flushHandle);

TSS2_DLL_EXPORT TSS2_RC Esys_FlushContext_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_FlushContext(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       flushHandle);
```

11.3.93 Esys_EvictControl Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_EvictControl_Async(
    ESYS_CONTEXT      *esysContext,
    ESYS_TR            auth,
    ESYS_TR            objectHandle,
    ESYS_TR            authSession1,
    ESYS_TR            optionalSession2,
    ESYS_TR            optionalSession3,
    TPMI_DH_PERSISTENT persistentHandle);

TSS2_DLL_EXPORT TSS2_RC Esys_EvictControl_Finish(
    ESYS_CONTEXT *esysContext,
    ESYS_TR      *newObjectHandle);

TSS2_DLL_EXPORT TSS2_RC Esys_EvictControl(
    ESYS_CONTEXT      *esysContext,
    ESYS_TR             auth,
    ESYS_TR             objectHandle,
    ESYS_TR             authSession1,
    ESYS_TR             optionalSession2,
    ESYS_TR             optionalSession3,
    TPMI_DH_PERSISTENT persistentHandle,
    ESYS_TR            *newObjectHandle);
```

11.3.94 Esys_ReadClock Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_ReadClock_Async(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       optionalSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3);

TSS2_DLL_EXPORT TSS2_RC Esys_ReadClock_Finish(
    ESYS_CONTEXT    *esysContext,
    TPMS_TIME_INFO **currentTime);

TSS2_DLL_EXPORT TSS2_RC Esys_ReadClock(
    ESYS_CONTEXT    *esysContext,
    ESYS_TR          optionalSession1,
    ESYS_TR          optionalSession2,
    ESYS_TR          optionalSession3,
    TPMS_TIME_INFO **currentTime);
```

11.3.95 Esys_ClockSet Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_ClockSet_Async(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       auth,
    ESYS_TR       authSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3,
    UINT64        newTime);

TSS2_DLL_EXPORT TSS2_RC Esys_ClockSet_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_ClockSet(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       auth,
    ESYS_TR       authSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3,
    UINT64        newTime);
```

11.3.96 Esys_ClockRateAdjust Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_ClockRateAdjust_Async(
    ESYS_CONTEXT     *esysContext,
    ESYS_TR           auth,
    ESYS_TR           authSession1,
    ESYS_TR           optionalSession2,
    ESYS_TR           optionalSession3,
    TPM2_CLOCK_ADJUST rateAdjust);

TSS2_DLL_EXPORT TSS2_RC Esys_ClockRateAdjust_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_ClockRateAdjust(
    ESYS_CONTEXT     *esysContext,
    ESYS_TR           auth,
    ESYS_TR           authSession1,
    ESYS_TR           optionalSession2,
    ESYS_TR           optionalSession3,
    TPM2_CLOCK_ADJUST rateAdjust);
```

11.3.97 Esys_GetCapability Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_GetCapability_Async(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       optionalSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3,
    TPM2_CAP      capability,
    UINT32        property,
    UINT32        propertyCount);

TSS2_DLL_EXPORT TSS2_RC Esys_GetCapability_Finish(
    ESYS_CONTEXT          *esysContext,
    TPMI_YES_NO           *moreData,
    TPMS_CAPABILITY_DATA **capabilityData);

TSS2_DLL_EXPORT TSS2_RC Esys_GetCapability(
    ESYS_CONTEXT          *esysContext,
    ESYS_TR                optionalSession1,
    ESYS_TR                optionalSession2,
    ESYS_TR                optionalSession3,
    TPM2_CAP               capability,
    UINT32                 property,
    UINT32                 propertyCount,
    TPMI_YES_NO           *moreData,
    TPMS_CAPABILITY_DATA **capabilityData);
```

11.3.98 Esys_TestParms Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_TestParms_Async(
    ESYS_CONTEXT            *esysContext,
    ESYS_TR                  optionalSession1,
    ESYS_TR                  optionalSession2,
    ESYS_TR                  optionalSession3,
    TPMT_PUBLIC_PARMS const *parameters);
```

```
TSS2_DLL_EXPORT TSS2_RC Esys_TestParms_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_TestParms(
    ESYS_CONTEXT            *esysContext,
    ESYS_TR                  optionalSession1,
    ESYS_TR                  optionalSession2,
    ESYS_TR                  optionalSession3,
    TPMT_PUBLIC_PARMS const *parameters);
```

11.3.99 Esys_NV_DefineSpace Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_NV_DefineSpace_Async(
    ESYS_CONTEXT          *esysContext,
    ESYS_TR                authHandle,
    ESYS_TR                authHandleSession1,
    ESYS_TR                optionalSession2,
    ESYS_TR                optionalSession3,
    TPM2B_AUTH      const *auth,
    TPM2B_NV_PUBLIC const *publicInfo);

TSS2_DLL_EXPORT TSS2_RC Esys_NV_DefineSpace_Finish(
    ESYS_CONTEXT *esysContext,
    ESYS_TR      *nvHandle);

TSS2_DLL_EXPORT TSS2_RC Esys_NV_DefineSpace(
    ESYS_CONTEXT          *esysContext,
    ESYS_TR                authHandle,
    ESYS_TR                authHandleSession1,
    ESYS_TR                optionalSession2,
    ESYS_TR                optionalSession3,
    TPM2B_AUTH      const *auth,
    TPM2B_NV_PUBLIC const *publicInfo,
    ESYS_TR               *nvHandle);
```

11.3.100 Esys_NV_UndefineSpace Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_NV_UndefineSpace_Async(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       authHandle,
    ESYS_TR       nvIndex,
    ESYS_TR       authHandleSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3);

TSS2_DLL_EXPORT TSS2_RC Esys_NV_UndefineSpace_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_NV_UndefineSpace(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       authHandle,
    ESYS_TR       nvIndex,
    ESYS_TR       authHandleSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3);
```

11.3.101 Esys_NV_UndefineSpaceSpecial Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_NV_UndefineSpaceSpecial_Async(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       nvIndex,
    ESYS_TR       platform,
    ESYS_TR       nvIndexSession1,
    ESYS_TR       platformSession2,
    ESYS_TR       optionalSession3);

TSS2_DLL_EXPORT TSS2_RC Esys_NV_UndefineSpaceSpecial_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_NV_UndefineSpaceSpecial(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       nvIndex,
    ESYS_TR       platform,
    ESYS_TR       nvIndexSession1,
    ESYS_TR       platformSession2,
    ESYS_TR       optionalSession3);
```

11.3.102 Esys_ReadPublic Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_NV_ReadPublic_Async(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       nvIndex,
    ESYS_TR       optionalSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3);

TSS2_DLL_EXPORT TSS2_RC Esys_NV_ReadPublic_Finish(
    ESYS_CONTEXT     *esysContext,
    TPM2B_NV_PUBLIC **nvPublic,
    TPM2B_NAME      **nvName);

TSS2_DLL_EXPORT TSS2_RC Esys_NV_ReadPublic(
    ESYS_CONTEXT     *esysContext,
    ESYS_TR           nvIndex,
    ESYS_TR           optionalSession1,
    ESYS_TR           optionalSession2,
    ESYS_TR           optionalSession3,
    TPM2B_NV_PUBLIC **nvPublic,
    TPM2B_NAME      **nvName);
```

11.3.103 Esys_NV_Write Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_NV_Write_Async(
    ESYS_CONTEXT                *esysContext,
    ESYS_TR                      authHandle,
    ESYS_TR                      nvIndex,
    ESYS_TR                      authHandleSession1,
    ESYS_TR                      optionalSession2,
    ESYS_TR                      optionalSession3,
    TPM2B_MAX_NV_BUFFER const *data,
    UINT16                       offset);

TSS2_DLL_EXPORT TSS2_RC Esys_NV_Write_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_NV_Write(
    ESYS_CONTEXT                *esysContext,
    ESYS_TR                      authHandle,
    ESYS_TR                      nvIndex,
    ESYS_TR                      authHandleSession1,
    ESYS_TR                      optionalSession2,
    ESYS_TR                      optionalSession3,
    TPM2B_MAX_NV_BUFFER const *data,
    UINT16                       offset);
```

11.3.104 Esys_NV_Increment Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_NV_Increment_Async(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       authHandle,
    ESYS_TR       nvIndex,
    ESYS_TR       authHandleSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3);

TSS2_DLL_EXPORT TSS2_RC Esys_NV_Increment_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_NV_Increment(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       authHandle,
    ESYS_TR       nvIndex,
    ESYS_TR       authHandleSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3);
```

11.3.105 Esys_NV_Extend Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_NV_Extend_Async(
    ESYS_CONTEXT                *esysContext,
    ESYS_TR                      authHandle,
    ESYS_TR                      nvIndex,
    ESYS_TR                      authHandleSession1,
    ESYS_TR                      optionalSession2,
    ESYS_TR                      optionalSession3,
    TPM2B_MAX_NV_BUFFER const *data);
```

```
TSS2_DLL_EXPORT TSS2_RC Esys_NV_Extend_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_NV_Extend(
    ESYS_CONTEXT              *esysContext,
    ESYS_TR                    authHandle,
    ESYS_TR                    nvIndex,
    ESYS_TR                    authHandleSession1,
    ESYS_TR                    optionalSession2,
    ESYS_TR                    optionalSession3,
    TPM2B_MAX_NV_BUFFER const *data);
```

11.3.106 Esys_NV_SetBits Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_NV_SetBits_Async(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       authHandle,
    ESYS_TR       nvIndex,
    ESYS_TR       authHandleSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3,
    UINT64        bits);

TSS2_DLL_EXPORT TSS2_RC Esys_NV_SetBits_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_NV_SetBits(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       authHandle,
    ESYS_TR       nvIndex,
    ESYS_TR       authHandleSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3,
    UINT64        bits);
```

11.3.107 Esys_NV_WriteLock Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_NV_WriteLock_Async(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       authHandle,
    ESYS_TR       nvIndex,
    ESYS_TR       authHandleSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3);

TSS2_DLL_EXPORT TSS2_RC Esys_NV_WriteLock_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_NV_WriteLock(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       authHandle,
    ESYS_TR       nvIndex,
    ESYS_TR       authHandleSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3);
```

11.3.108 Esys_NV_GlobalWriteLock Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_NV_GlobalWriteLock_Async(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       authHandle,
    ESYS_TR       authHandleSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3);

TSS2_DLL_EXPORT TSS2_RC Esys_NV_GlobalWriteLock_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_NV_GlobalWriteLock(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       authHandle,
    ESYS_TR       authHandleSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3);
```

11.3.109 Esys_NV_Read Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_NV_Read_Async(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       authHandle,
    ESYS_TR       nvIndex,
    ESYS_TR       authHandleSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3,
    UINT16        size,
    UINT16        offset);

TSS2_DLL_EXPORT TSS2_RC Esys_NV_Read_Finish(
    ESYS_CONTEXT        *esysContext,
    TPM2B_MAX_NV_BUFFER **data);

TSS2_DLL_EXPORT TSS2_RC Esys_NV_Read(
    ESYS_CONTEXT        *esysContext,
    ESYS_TR              authHandle,
    ESYS_TR              nvIndex,
    ESYS_TR              authHandleSession1,
    ESYS_TR              optionalSession2,
    ESYS_TR              optionalSession3,
    UINT16               size,
    UINT16               offset,
    TPM2B_MAX_NV_BUFFER **data);
```

11.3.110 Esys_NV_ReadLock Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_NV_ReadLock_Async(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       authHandle,
    ESYS_TR       nvIndex,
    ESYS_TR       authHandleSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3);

TSS2_DLL_EXPORT TSS2_RC Esys_NV_ReadLock_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_NV_ReadLock(
    ESYS_CONTEXT *esysContext,
    ESYS_TR       authHandle,
    ESYS_TR       nvIndex,
    ESYS_TR       authHandleSession1,
    ESYS_TR       optionalSession2,
    ESYS_TR       optionalSession3);
```

11.3.111 Esys_NV_ChangeAuth Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_NV_ChangeAuth_Async(
    ESYS_CONTEXT     *esysContext,
    ESYS_TR           nvIndex,
    ESYS_TR           nvIndexSession1,
    ESYS_TR           optionalSession2,
    ESYS_TR           optionalSession3,
    TPM2B_AUTH const *newAuth);

TSS2_DLL_EXPORT TSS2_RC Esys_NV_ChangeAuth_Finish(
    ESYS_CONTEXT *esysContext);

TSS2_DLL_EXPORT TSS2_RC Esys_NV_ChangeAuth(
    ESYS_CONTEXT     *esysContext,
    ESYS_TR           nvIndex,
    ESYS_TR           nvIndexSession1,
    ESYS_TR           optionalSession2,
    ESYS_TR           optionalSession3,
    TPM2B_AUTH const *newAuth);
```

11.3.112 Esys_NV_Certify Functions

```
TSS2_DLL_EXPORT TSS2_RC Esys_NV_Certify_Async(
    ESYS_CONTEXT           *esysContext,
    ESYS_TR                 signHandle,
    ESYS_TR                 authHandle,
    ESYS_TR                 nvIndex,
    ESYS_TR                 signHandleSession1,
    ESYS_TR                 authHandleSession2,
    ESYS_TR                 optionalSession3,
    TPM2B_DATA      const *qualifyingData,
    TPMT_SIG_SCHEME const *inScheme,
    UINT16                 size,
    UINT16                 offset);

TSS2_DLL_EXPORT TSS2_RC Esys_NV_Certify_Finish(
    ESYS_CONTEXT    *esysContext,
    TPM2B_ATTEST   **certifyInfo,
    TPMT_SIGNATURE **signature);

TSS2_DLL_EXPORT TSS2_RC Esys_NV_Certify(
    ESYS_CONTEXT          *esysContext,
    ESYS_TR                signHandle,
    ESYS_TR                authHandle,
    ESYS_TR                nvIndex,
    ESYS_TR                signHandleSession1,
    ESYS_TR                authHandleSession2,
    ESYS_TR                optionalSession3,
    TPM2B_DATA      const *qualifyingData,
    TPMT_SIG_SCHEME const *inScheme,
    UINT16                 size,
    UINT16                 offset,
    TPM2B_ATTEST         **certifyInfo,
    TPMT_SIGNATURE       **signature);
```

11.4 Esys.h Postlude

```
#ifdef __cplusplus
} /* end extern "C" */
#endif

#endif /* TSS2_ESYS_H */
```