GLOBALPLATFORM®

THE STANDARD FOR MANAGING APPLICATIONS ON SECURE CHIP TECHNOLOGY

# GlobalPlatform Device Technology

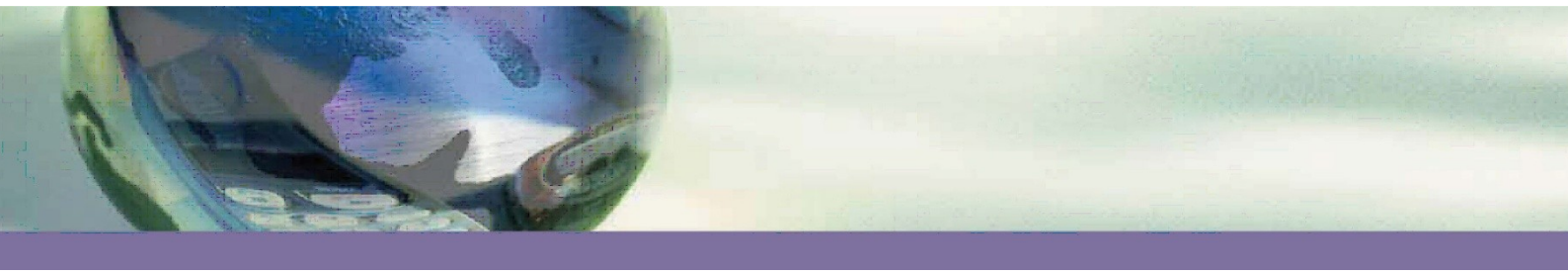
# TEE Secure Element API

Version 1.1.1

Public Release

November 2016

Document Reference: GPD_SPE_024



THIS SPECIFICATION OR OTHER WORK PRODUCT IS BEING OFFERED WITHOUT ANY WARRANTY WHATSOEVER, AND IN PARTICULAR, ANY WARRANTY OF NON-INFRINGEMENT IS EXPRESSLY DISCLAIMED. ANY IMPLEMENTATION OF THIS SPECIFICATION OR OTHER WORK PRODUCT SHALL BE MADE ENTIRELY AT THE IMPLEMENTER'S OWN RISK, AND NEITHER THE COMPANY, NOR ANY OF ITS MEMBERS OR SUBMITTERS, SHALL HAVE ANY LIABILITY WHATSOEVER TO ANY IMPLEMENTER OR THIRD PARTY FOR ANY DAMAGES OF ANY NATURE WHATSOEVER DIRECTLY OR INDIRECTLY ARISING FROM THE IMPLEMENTATION OF THIS SPECIFICATION OR OTHER WORK PRODUCT.

# Contents

# Figures



# Tables

# 1 Introduction

This document specifies the syntax and semantics of the TEE Secure Element API.

## 1.1 Audience

This document is suitable for software developers implementing Trusted Applications running inside the Trusted Execution Environment (TEE) which need to expose an externally visible interface to Client Applications.

This document is also intended for implementers of the TEE itself, its ***Trusted OS***, ***Trusted Core Framework***, the TEE APIs, and the communications infrastructure required to access Trusted Applications.

## 1.2 IPR Disclaimer

Attention is drawn to the possibility that some of the elements of this GlobalPlatform specification or other work product may be the subject of intellectual property rights (IPR) held by GlobalPlatform members or others. For additional information regarding any such IPR that have been brought to the attention of GlobalPlatform, please visit https://www.globalplatform.org/specificationsipdisclaimers.asp. GlobalPlatform shall not be held responsible for identifying any or all such IPR, and takes no position concerning the possible existence or the evidence, validity, or scope of any such IPR.

## 1.3 References

**Table 1-1: Normative References**

| Standard / Specification | Description | Ref |
|---|---|---|
| GPD_SPE_007 | GlobalPlatform Device Technology<br>TEE Client API Specification | [TEE Client API] |
| GPD_SPE_010 | GlobalPlatform Device Technology<br>TEE Internal Core API Specification | [TEE Core API] |
| GPD_SPE_020 | GlobalPlatform Device<br>Trusted User Interface API | [TEE TUI API] |
| GPD_SPE_025 | GlobalPlatform Device Technology<br>TEE TA Debug Specification | [TEE Debug] |
| GPD_SPE_027 | GlobalPlatform Device Technology<br>TEE Administration Framework | [TEE Admin] |
| GPC_SPE_034 | GlobalPlatform Card Specification | [GPCS] |
| GPC_SPE_014 | GlobalPlatform Card Specification – Amendment D<br>Secure Channel Protocol '03' | [Amd D] |
| GPC_SPE_093 | GlobalPlatform Card Specification – Amendment F<br>Secure Channel Protocol '11' | [Amd F] |
| Open Mobile API | SIMalliance<br>Open Mobile API Specification | [Open Mobile] |

| Standard / Specification | Description | Ref |
|---|---|---|
| ISO/IEC 7816-3 | Identification cards – Integrated circuit cards – Part 4: Organization, security and commands for interchange | [ISO 7816-3] |
| ISO/IEC 7816-4 | Identification cards – Integrated circuit cards – Part 4: Organization, security and commands for interchange | [ISO 7816-4] |

**Table 1-2: Informative References**

| Standard / Specification | Description | Ref |
|---|---|---|
| GPD_SPE_009 | GlobalPlatform Device Technology<br>TEE System Architecture | [TEE Sys Arch] |
| GP_GUI_001 | GlobalPlatform Technology<br>Document Management Guide | [Doc Mgmt] |
| PC/SC | PC/SC Specification<br>http://www.pcscworkgroup.com/specifications/overview.php | [PC/SC] |
| RFC 2119 | Key words for use in RFCs to Indicate Requirement Levels | [RFC 2119] |

## 1.4 Terminology and Definitions

The following meanings apply to SHALL, SHALL NOT, MUST, MUST NOT, SHOULD, SHOULD NOT, and MAY in this document (refer to [RFC 2119]):

- **SHALL** indicates an absolute requirement, as does **MUST**.
- **SHALL NOT** indicates an absolute prohibition, as does **MUST NOT**.
- **SHOULD** and **SHOULD NOT** indicate recommendations.
- **MAY** indicates an option.

**Table 1-3: Terminology and Definitions**

| Term | Definition |
|---|---|
| Applet | General term for Secure Element application: An application as described in GlobalPlatform Card Specification [GPCS] which is installed in the SE and runs within the SE. |
| Client Application | An application running outside of the Trusted Execution Environment (TEE) making use of the TEE Client API [TEE Client API] to access facilities provided by Trusted Applications inside the TEE.<br>Contrast *Trusted Application*. |
| Device/Terminal/Mobile application | An application which is installed in the mobile device and runs within the mobile device. |
| Data Object | An object containing a data stream but no key material. |
| Data Stream | Data associated with a Persistent Object (excluding Object Attributes and metadata). |

| Term | Definition |
|---|---|
| Historical Bytes | The historical bytes describe operating characteristics (e.g. capabilities or issuance data) of the Secure Element. The historical bytes may be optionally included in the ATR as defined in [ISO 7816-3] in Chapter 8. Their structure and content shall be as specified in [ISO 7816-4] in Chapter 8. |
| Initialized Object | A Transient Object whose attributes have been populated. |
| Instance | A particular execution of a Trusted Application, having physical memory space that is separated from the physical memory space of all other TA instances. |
| Object Attribute | Small amounts of data used to store key material in a structured way. |
| Object Handle | An opaque reference that identifies a particular object. |
| Object Identifier | A variable-length binary buffer identifying a persistent object. |
| Operation Parameter | A data item passed in a command, which can contain integer values or references to client-owned shared memory blocks. Each command contains (in addition to a Command Identifier), four Operation Parameters. |
| Panic | An exception that kills a whole TA instance as a result of calling one of the API functions. |
| Parameter Annotation | Denotes the pattern of usage of a function parameter or pair of function parameters. |
| Persistent Object | An object identified by an Object Identifier and including a Data Stream.<br>Contrast *Transient Object*. |
| Property | An immutable value identified by a name. |
| Property Set | Any of the following:<br>• The configuration properties of a Trusted Application<br>• Properties associated with a Client Application by the Rich Execution Environment<br>• Properties describing characteristics of a TEE implementation. |
| REE Time | A time value that is as trusted as the REE. |
| Rich Execution Environment (REE) | An environment that is provided and governed by a Rich OS, potentially in conjunction with other supporting operating systems and hypervisors; it is outside of the TEE. This environment and applications running on it are considered un-trusted.<br>Contrast *Trusted Execution Environment (TEE)*. |
| Rich OS | Typically an OS providing a much wider variety of features than that of the OS running inside the TEE. It is very open in its ability to accept applications. It will have been developed with functionality and performance as key goals, rather than security. Due to the size and needs of the Rich OS it will run in an execution environment outside of the TEE hardware (often called an REE – Rich Execution Environment) with much lower physical security boundaries. From the TEE viewpoint, everything in the REE has to be considered un-trusted, though from the Rich OS point of view there may be internal trust structures.<br>Contrast *Trusted OS*. |

| Term | Definition |
|---|---|
| Secure Element (SE) | A tamper resistant component which is used in a device to provide the security, confidentiality, and multiple application environment required to support various business models. May exist in any form factor, such as embedded SE, SIM, UICC, smartSD, smart microSD, etc. |
| Transient Object | An object containing attributes but no data stream, which is reclaimed when closed or when the TA instance is destroyed.<br>Contrast *Persistent Object*. |
| Trusted Application | An application running inside the Trusted Execution Environment (TEE) that provides security related functionality to Client Applications outside of the TEE or to other Trusted Applications inside the TEE.<br>Contrast *Client Application*. |
| Trusted Core Framework or “Framework” | The part of the Trusted OS responsible for implementing the Trusted Core Framework API[1] that provides OS-like facilities to Trusted Applications and a way for the Trusted OS to interact with the Trusted Applications. |
| Trusted Execution Environment (TEE) | An execution environment that runs alongside but isolated from an REE. A TEE has security capabilities and meets certain security-related requirements: It protects TEE assets from general software attacks, defines rigid safeguards as to data and functions that a program can access, and resists a set of defined threats. There are multiple technologies that can be used to implement a TEE, and the level of security achieved varies accordingly.<br>Contrast *Rich Execution Environment (REE)*. |
| Trusted OS | An operating system running in the TEE providing the TEE Internal API [TEE Core API] to Trusted Applications. |

## 1.5 Abbreviations and Notations

**Table 1-4: Abbreviations and Notations**

| Abbreviation / Notation | Meaning |
|---|---|
| APDU | Application Protocol Data Unit – Format for messages exchanged with SEs |
| AES | Advanced Encryption Standard |
| API | Application Programming Interface |
| ATR | Answer To Reset (see [ISO 7816-3] in Chapter 8) |
| CA | Client Application |
| CMAC | Cipher-based MAC |
| DES | Data Encryption Standard |
| DSA | Digital Signature Algorithm |
| ETSI | European Telecommunications Standards Institute |

[1] The Trusted Core Framework API is described in Chapter 4 of [TEE Core API].

| Abbreviation / Notation | Meaning |
|---|---|
| HMAC | Hash-based Message Authentication Code |
| IEEE | Institute of Electrical and Electronics Engineers |
| IETF | Internet Engineering Task Force |
| IPR | Intellectual Property Rights |
| ISO | International Organization for Standardization |
| IV | Initialization Vector |
| MAC | Message Authentication Code |
| MD5 | Message Digest 5 |
| MGF | Mask Generating Function |
| NIST | National Institute of Standards and Technology |
| OAEP | Optimal Asymmetric Encryption Padding |
| OS | Operating System |
| PC/SC | Personal Computer/Smart Card |
| PKCS | Public Key Cryptography Standards |
| PSS | Probabilistic Signature Scheme |
| REE | Rich Execution Environment |
| RFC | Request For Comments; may denote a memorandum published by the IETF |
| SC | Secure Channel |
| SD | Secure Digital |
| SE | Secure Element |
| SHA | Secure Hash Algorithm |
| SIM | Subscriber Identity Module |
| SW1 | Status Word One |
| SW2 | Status Word Two |
| TA | Trusted Application |
| TEE | Trusted Execution Environment |
| TPDU | Transport Protocol Data Unit |
| USB | Universal Serial Bus |
| UTC | Coordinated Universal Time |
| UTF | Unicode Transformation Format |
| UUID | Universally Unique Identifier |
| XTS | XEX-based Tweaked Codebook mode with cipher text stealing (CTS) |

## 1.6 Revision History

**Table 1-5: Revision History**

| Date | Version | Description |
|---|---|---|
| July 2013 | 1.0 | Public Release |
| July 2015 | 1.1 | Public Release<br>• Added Specification Version Number Property section<br>• Added Service Layer APIs chapter with Discovery API and Secure Channel API.<br>• Added `TEE_ERROR_CANCEL` as a return value<br>• Added `TEE_SESecureChannelOpen` function |
| November 2016 | 1.1.1 | Public Release showing all non-trivial changes since v1.1.<br>Significant changes include:<br>• Specification of version definitions in header file<br>• Correction of OIDs<br>• Correction of typos<br>• Clarification on service life cycle<br>• Clarification on basic channel management<br>• Precision on parameter usage on different functions<br>• Clarification on APDU buffer usage<br>• Clarification on response handling during a transmit operation<br>• SCP03 mode C_ENC_CR_MAC added<br>• Clarification on support of extended length<br>Minor differences since v1.1 that are not revision marked include changes to punctuation and capitalization.<br>Note: Two new sections are boxed rather than revision marked. |

# 2 Background

For a general understanding of the TEE Internal API interfaces, the reader is encouraged to review Chapter 2 of the TEE Internal Core API Specification [TEE Core API].

This document is based on the SIMalliance Open Mobile API Specification [Open Mobile].

Perceived relationships between the TEE and an SE are enhanced by the presence of a defined interface.

- Tamper resistance. While the TEE potentially provides high speed, high bandwidth cryptographic processing with remote attack protections, it does not claim to have the tamper resistance security capabilities of an SE. As such, the use of an SE in conjunction with a TEE can provide the best of both worlds.
- The TEE may be implemented with a Trusted UI capability [TEE TUI API]. When this is in place then the SE can have a relationship with a trustworthy interface to the user.

While it would be desirable, to enable the above use cases, for all SE access to be routed through the TEE, it is clear that this is not the case in current devices. As such, the channel of communication between SE and TEE may be direct (based on hardware connection that the REE cannot intercept by software but that is exposed to hardware attacks) or indirect (where communications may be intercepted by the REE). To communicate securely over either type of channel requires the use of a secure protocol.

## 2.1 C Language vs. ISO 7816-4 Language Conventions [new section in v1.1.1]

This document describes a TEE API that can be used to implement use cases involving both the TEE and an SE.

In this document, C language API definitions use C language conventions.

Interactions with the SE use the conventions found in [ISO 7816-4]. The following conventions, in particular, should be noted:

The use of 'xx' denotes a byte, represented as a pair of hexadecimal digits, where each x can take one of the following values: ‘0’..‘9’ for the decimal values 0 to 9; ‘A’..‘F’ for the decimal values 10 to 15; ‘X’ for any decimal value between 0 and 15.

To guarantee an unambiguous mapping to bytes, this syntax can only be used with an even number of hexadecimal digits (e.g. 'A34' is illegal).

An unlimited number of bytes can be concatenated within a pair of straight single quotes.

The following table gives examples of the correspondence between C language and [ISO 7816-4].

**Table 2-1: Language Conventions: C Language and ISO 7816-4**

| C Language | ISO 7816-4 | Meaning |
|---|---|---|
| 0x2A | '2A' | A single byte with hexadecimal value 2A |
| 0x4E2A | '4E2A' | Two bytes, the first with hexadecimal value 4E and the second with hexadecimal value 2A |
| – | '61XX' | Two bytes, the first with hexadecimal value 61 and the second with any legal value for a byte |

**Note:** It is legal C99 to place a sequence of one or more characters between single quotes; such a literal is of type int, and takes an implementation defined value. However, since it is almost certain that 0x42EA != '42EA', implementers are advised to translate [ISO 7816-4] literals into idiomatic C language forms in their source code.

# 3 Requirements for TEE Secure Element API

## 3.1 Assumptions and Scope

The TEE Secure Element API is an enabling thin layer to support communication to Secure Elements (SE) connected to the device within which the TEE is implemented. This API defines only a transport interface based on [Open Mobile]. SE support services may be the subject of a future specification.

Any communication going through the REE shall be considered unsecure. The communication could be intercepted or manipulated by an attacker. If a communication from a TEE TA to an SE passing through the REE needs to be protected against those threats, it is up to the TEE TA and SE Applet implementers to put in place security mechanisms such as a secure channel, for example.

This API does not specify the physical layers which are used to communicate with the SE. The use by the TEE of an REE driver to access an SE may be excluded by the certification program.

### 3.1.1 Error Handling

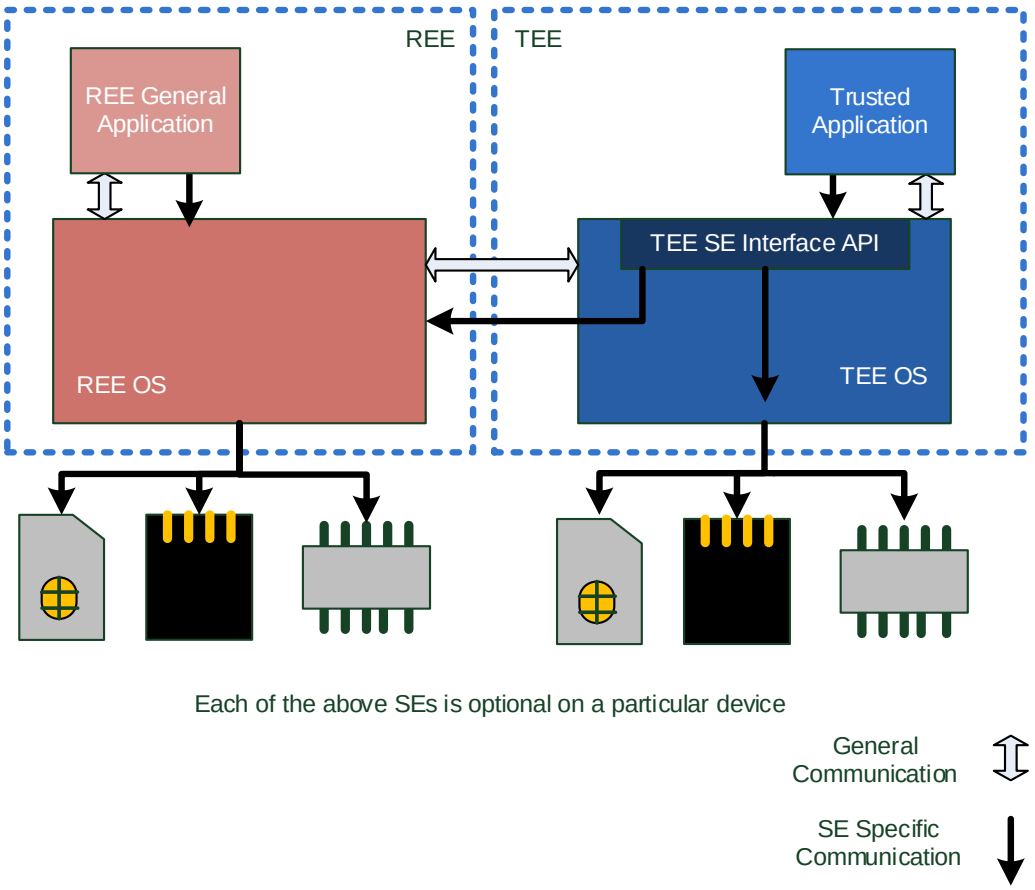
This API follows the TEE Internal API philosophy that any programmer avoidable errors will result in a TA level PANIC. See section 2.2 of [TEE Core API].

### 3.1.2 Implementations

The TEE Secure Element API defines the interface exposed to Trusted Applications using [TEE Core API]. It makes no restrictions on the methods of implementation of the actual connections. For example, such connections may be direct to drivers talking to hardware inside the TEE or indirect via facilities resident in the REE.

# 4 API Overview

**Figure 4-1: Typical Device with Multiple SE Readers**




Secure Elements (SEs) may be connected to the REE or exclusively to the TEE.

- An SE connected exclusively to the TEE is accessible by a TA without using any resources from the REE. Thus the communication is considered trusted.
- An SE connected to the REE is accessible by a TA using resources lying in the REE. It is recommended to use a secure channel (i.e. by using the Secure Channel API) to protect the communication between the TA and the SE against attacks in the REE.

A TA can check whether an SE is exclusively connected to the TEE or not by using `TEE_SEReaderProperties` of this specification which is described in section 5.2.2.

An SE is always located within a reader no matter whether it is permanently fixed to the device or is plugged into a physical card reader (e.g. an SD card reader) or slot (e.g. a SIM socket). A device may support any number of SEs and readers but only as many SEs at any one time as it has readers.

The set of readers associated with the device on which the TEE is executing is available to Trusted Applications running within the TEE as a set of `TEE_SEReaderHandle`. Each `TEE_SEReaderHandle` is associated with properties specified as a string (maximum length 64 characters without including the null-terminator '\0') which identifies the reader. Readers are present in this list even if there is no SE in the reader. It is possible that in some systems the list of available readers may change when for example a USB SD card or smart card reader is attached to the device.

The strings returned to identify the readers shall be unique within a device[2] and shall be human readable, i.e. they shall be suitable for use in prompts to select an SE to interact with.

Once a connection has been made to the reader it is possible to query the ATR of the contained SE and then to make APDU based connections to the applications within the SE.

[2] This string should not be the ATR or anything else specific to the SE within the reader since by the time the connection is made to the reader the SE could have changed.

# 5 Transport Layer API

This chapter specifies the TEE Secure Element API header file, constants, and API.

Data types `bool`, `uint8_t`, and `TEE_Result` are as defined in [TEE Core API].

Parameter annotations *[in]*, *[inbuf]*, *[inout]*, *[out]*, *[outbuf]*, and *[outstring]* are as defined in [TEE Core API].

## 5.1 Header File

The header file for the TEE SE API must have the name "`tee_internal_se_api.h`".

```
#include "tee_internal_se_api.h"
```

### 5.1.1 API Version [new section in v1.1.1]

The header file SHALL contain version specific definitions to allow compilation options specific to this version of the specification.

```
#define TEE_SE_API_MAJOR_VERSION ([Major version number])
#define TEE_SE_API_MINOR_VERSION ([Minor version number])
#define TEE_SE_API_MAINTENANCE_VERSION ([Maintenance version number])
#define TEE_SE_API_VERSION (TEE_SE_API_MAJOR_VERSION << 24) +
                           (TEE_SE_API_MINOR_VERSION << 16) +
                           (TEE_SE_API_MAINTENANCE_VERSION << 8)
```

`TEE_SE_API_MAJOR_VERSION` indicates the major version number of the TEE SE API. It SHALL be set to the major version number of this specification.

`TEE_SE_API_MINOR_VERSION` indicates the minor version number of the TEE SE API. It SHALL be set to the minor version number of this specification. If the minor version is zero, then one zero shall be present.

`TEE_SE_API_MAINTENANCE_VERSION` indicates the maintenance version number of the TEE SE API. It SHALL be set to the maintenance version number of this specification. If the maintenance version is zero, then one zero shall be present.

The definitions of “Major Version”, “Minor Version”, and “Maintenance Version” in the revision number of this specification are determined as defined in [Doc Mgmt] section 4.3. In particular, the value of `TEE_SE_API_MAINTENANCE_VERSION` SHALL be zero if it is not already defined as part of the revision number of this document. The “Draft Revision” number SHALL NOT be provided as an API version indication.

A compound value SHALL also be defined. If the Maintenance version number is 0, the compound value SHALL be defined as:

```
#define TEE_SE_API_[Major version number]_[Minor version number]
```

If the Maintenance version number is not zero, the compound value SHALL be defined as:

```
#define TEE_SE_API_[Major version number]_[Minor version
number]_[Maintenance version number]
```

Some examples of version definitions:

For GlobalPlatform TEE SE API Specification v1.3, these would be:

```
#define TEE_SE_API_MAJOR_VERSION        (1)
#define TEE_SE_API_MINOR_VERSION        (3)
#define TEE_SE_API_MAINTENANCE_VERSION  (0)
#define TEE_SE_API_1_3
```

And the value of `TEE_SE_API_VERSION` would be 0x01030000.

For a maintenance release of the specification as v2.14.7, these would be:

```
#define TEE_SE_API_MAJOR_VERSION        (2)
#define TEE_SE_API_MINOR_VERSION        (14)
#define TEE_SE_API_MAINTENANCE_VERSION  (7)
#define TEE_SE_API_2_14_7
```

And the value of `TEE_SE_API_VERSION` would be 0x020E0700.

## 5.2 Constants

### 5.2.1 Return Codes

Return codes from [TEE Core API] are used. The following additional return codes are defined.

**Table 5-1: API Return Codes**

| Constant Name | Value |
|---|---|
| TEE_ERROR_61_SHORT_BUFFER | 0xF0240001 |
| TEE_ERROR_6C_SHORT_BUFFER | 0xF0240002 |
| TEE_SE_SESSION_OPEN | 0x00240000 |

### 5.2.2 Secure Element Reader Properties

This type is used to return information about a Secure Element reader.

```
typedef struct __TEE_SEReaderProperties
{
   bool sePresent;           // true if an SE is present in the reader
   bool teeOnly;             // true if this reader is accessible
                             //    only via the TEE
   bool selectResponseEnable; // true if the response to a SELECT is
                             //    available in the TEE
} TEE_SEReaderProperties;
```

The `teeOnly` property is set internally in the TEE. Thus it is trustable.

If `teeOnly` is true the `properties sePresent and selectResponseEnable` are trustable, otherwise these properties cannot be trusted.

### 5.2.3 Secure Element Applet's AID

This type is used to pass the AID of the Applet that a TA wants to communicate with.

```
typedef struct __TEE_SEAID
{
    uint8_t *buffer;      // the value of the applet's AID
    uint32_t bufferLen;   // length of the applet's AID
} TEE_SEAID;
```

### 5.2.4 Handles

These handles are opaque handles on a Service, Reader, Session, or Channel. These handles are returned by the open-like functions (see below).

```
typedef struct __TEE_SEServiceHandle* TEE_SEServiceHandle;
typedef struct __TEE_SEReaderHandle* TEE_SEReaderHandle;
typedef struct __TEE_SESessionHandle* TEE_SESessionHandle;
typedef struct __TEE_SEChannelHandle* TEE_SEChannelHandle;
```

## 5.3 API Levels and Classes

The functions defined by the API are grouped as described in Table 5-2.

**Table 5-2: API Levels and Classes**

| Description of Level | Modeled by: | See section: |
|---|---|---|
| The service level is the entry point to the API, allowing one to gain access to the TEE SE API. | SEService class | 5.5 |
| The reader level allows an application to choose a reader once enumerated by the service level. Sessions are opened on a Secure Element inserted in a reader. | SEReader class | 5.6 |
| The session level is used to retrieve the ATR and to open channels to SE applications. | SESession class | 5.7 |
| The channel level is used by applications to exchange APDUs with the SE application. | SEChannel class | 5.8 |

**Note:** The service level and reader level have direct corresponding features in PC/SC [PC/SC] with respectively `SCardEstablishContext/SCardReleaseContext`, combined with the reader enumeration API, and with `SCardConnect, SCardDisconnect, SCardReconnect.` The session and channel levels do not have any direct correspondence in PC/SC, although one can consider the channel level equivalent to the PC/SC `SCardTransmit` method, mapped on a channel (basic or logical).

## 5.4 Specification Version Number Property

This specification defines a TEE property containing the version number of the specification the implementation conforms to. The property can be retrieved using the normal Property Access Functions defined in [TEE Core API]. The property SHALL be named `gpd.tee.~~tui.~~seapi.version` and SHALL be of integer type with the interpretation given below.

The specification version number property consists of four positions; major, minor, maintenance, and RFU. These four bytes are combined into a 32-bit unsigned integer as follows:

- The major version number of the specification is placed in the most significant byte.
- The minor version number of the specification is placed in the second most significant byte.
- The maintenance version number of the specification is placed in the second least significant byte. If the version is not a maintenance version, this SHALL be zero.
- The least significant byte is reserved for future use. Currently this byte SHALL be zero.

**Table 5-3: Specification Version Number Property – 32-bit Integer Structure**

| Bits 24-31 (MSB) | Bits 16-23 | Bits 8-15 | Bits 0-7 (LSB) |
|---|---|---|---|
| Major version number of the specification | Minor version number of the specification | Maintenance version number of the specification | Reserved for future use.<br>Currently SHALL be zero. |

So for example:

- Specification version 1.1 will be held as 0x01010000 (16842752 in base 10)
- Specification version 1.2 will be held as 0x01020000 (16908288 in base 10)
- Specification version 1.2.3 will be held as 0x01020300 (16909056 in base 10)
- Specification version 12.13.14 will be held as 0x0C0D0E00 (202182144 in base 10)
- Specification version 212.213.214 will be held as 0xD4D5D600 (3570783744 in base 10)

This places the following requirement on ~~the usage of~~ the version numbering.

- No document can have a Major or Minor or Maintenance version number greater than 255.

~~This value shall be defined in the header file to allow compilation options specific to this version of specification:~~

```
~~#define TEE_SE_API_1_1~~
```

## 5.5 SEService Class

### 5.5.1 TEE_SEServiceOpen

```
TEE_Result TEE_SEServiceOpen(
   [out] TEE_SEServiceHandle *seServiceHandle,
);
```

**Description**

The `TEE_SEServiceOpen` function allocates a handle for a new connection that can be used to connect to all the Secure Elements available to the TEE.

The function will block until a valid handle can be returned or an error state occurs.

An `seServiceHandle` gives access from a TA to all readers and Secure Elements available.

If this routine is called twice by a TA, then the `TEE_SEServiceOpen` function should return `TEE_ERROR_ACCESS_CONFLICT`.

**Parameters**

- `seServiceHandle`: Reference to SEService handle

**Specification Number:** 24 **Function Number:** 0x0101

**Return Value**

- `TEE_SUCCESS:` If a valid handle has been returned
- `TEE_ERROR_ACCESS_CONFLICT:` If the TA already has an open handle
- `TEE_ERROR_OUT_OF_MEMORY:` If not enough resources are available to perform the operation
- `TEE_ERROR_CANCEL:` If the command has been cancelled. See [TEE Core API] section 2.1.4.

**Panic Reasons**

- The Implementation detects any error which is not explicitly associated with a defined return code for this function.
- `seServiceHandle` is not a valid reference (`NULL` included) on a `TEE_SEServiceHandle`.

### 5.5.2 TEE_SEServiceClose

```
void TEE_SEServiceClose(
   TEE_SEServiceHandle seServiceHandle,
);
```

**Description**

The `TEE_SEServiceClose` function releases all Secure Elements resources allocated by this seService.

It is recommended that this function be called in the termination method of the calling application (or part of this application) which is bound to this seService. No operation will be performed if the `seServiceHandle` is already closed or invalid.

**Parameters**

- `seServiceHandle:` Reference to seService handle

**Specification Number:** 24 **Function Number:** 0x0102

**Return Value**

- None

**Panic Reasons**

- The Implementation detects any error which is not explicitly associated with a defined return code for this function.
- ~~`seServiceHandle` is not a valid reference (`NULL` included) on a `TEE_SEServiceHandle`.~~

### 5.5.3 TEE_SEServiceGetReaders

```
TEE_Result TEE_SEServiceGetReaders(
           TEE_SEServiceHandle seServiceHandle,
   [out]   TEE_SEReaderHandle* seReaderHandleList,
   [inout] uint32_t*           seReaderHandleListLen
);
```

**Description**

The `TEE_SEServiceGetReaders` function returns the list of available Secure Element reader handles. There must be no duplicated objects in the returned list.

This function always returns the same handle for the same logical reader during the service life cycle.

**Parameters**

- `seServiceHandle:` Reference to seService handle
- `seReaderHandleList:` Reference to seReader handle list
- `seReaderHandleListLen:` Length of the seReader handle list

**Specification Number:** 24 **Function Number:** 0x0103

**Return Value**

- `TEE_SUCCESS:` In case of success
- `TEE_ERROR_ITEM_NOT_FOUND:` If no reader is found
- `TEE_ERROR_OUT_OF_MEMORY:` If not enough resources are available to perform the operation
- `TEE_ERROR_SHORT_BUFFER:` If `*seReaderHandleListLen` is too small to count all readers

**Panic Reasons**

- The Implementation detects any error which is not explicitly associated with a defined return code for this function.
- `seServiceHandle` is not a valid handle on a `TEE_seServiceHandle`.
- `seReaderHandleListLen` is not a valid reference (`NULL` included) on a `uint32_t`.
- `*seReaderHandleListLen` is `0`.

## 5.6 SEReader Class

### 5.6.1 TEE_SEReaderGetProperties

```
void TEE_SEReaderGetProperties(
   TEE_SEReaderHandle seReaderHandle,
   [out] TEE_SEReaderProperties* readerProperties
);
```

**Description**

The `TEE_SEReaderGetProperties` function returns the reader properties as defined in section 5.2.2. The properties include the following information: whether an SE is present in the reader, whether the SE is connected directly to the TEE, and whether the response to a SELECT is available in the TEE.

If the SE is not present then the ATR shall be the empty string.

**Parameters**

- `seReaderHandle:` Reference to seReader handle
- `readerProperties:` Reference to the reader properties as defined in section 5.2.2

**Specification Number:** 24 **Function Number:** 0x0201

**Return Value**

- None

**Panic Reasons**

- The Implementation detects any error which is not explicitly associated with a defined return code for this function.
- `seReaderHandle` is not a valid handle on a `TEE_seReaderHandle`.
- `readerProperties` is not a valid reference (`NULL` included) on a `TEE_SEReaderProperties`.

### 5.6.2 TEE_SEReaderGetName

```
TEE_Result TEE_SEReaderGetName(
   TEE_SEReaderHandle seReaderHandle,
   [outstring] char* readerName, uint32_t* readerNameLen
);
```

**Description**

The `TEE_SEReaderGetName` function returns the user-friendly name of this reader.

- If this reader is a SIM reader, then its name must start with the “SIM” prefix.
- If the reader is a SD or micro SD reader, then its name must start with the “SD” prefix.
- If the reader is an embedded SE reader, then its name must start with the “eSE” prefix.

**Parameters**

- `seReaderHandle:` Reference to seReader handle
- `readerName, readerNameLen:` Buffer with the reader name

**Specification Number:** 24 **Function Number:** 0x0202

**Return Value**

- `TEE_SUCCESS:` In case of success
- `TEE_ERROR_SHORT_BUFFER:` If `readerName` buffer is too small to hold the whole reader name

**Panic Reasons**

- The Implementation detects any error which is not explicitly associated with a defined return code for this function.
- `seReaderHandle` is not a valid handle on a `TEE_seReaderHandle`.
- `readerName` is not a valid reference (`NULL` included) on a `char`.
- `readerNameLen` is not a valid reference (`NULL` included) on a `uint32_t`.
- `*readerNameLen` is `0`.

### 5.6.3 TEE_SEReaderOpenSession

```
TEE_Result TEE_SEReaderOpenSession (
   TEE_SEReaderHandle seReaderHandle,
   [out] TEE_SESessionHandle* seSessionHandle
);
```

**Description**

The `TEE_SEReaderOpenSession` function connects to a Secure Element in this reader.

This method prepares (initializes) the Secure Element for communication before the Session object is returned (i.e. powers the Secure Element if it's not already on).

There might be multiple sessions opened at the same time on the same reader. The system ensures the interleaving of APDUs between the respective sessions.

**Parameters**

- `seReaderHandle:` Reference to seReader handle
- `seSessionHandle:` Reference to seSession handle

**Specification Number:** 24 **Function Number:** 0x0203

**Return Value**

- `TEE_SUCCESS:` In case of success
- `TEE_ERROR_COMMUNICATION:` If seReader is not able to open a session with the SE because the SE could not be prepared
- `TEE_ERROR_OUT_OF_MEMORY:` If not enough resources are available to perform the operation
- `TEE_ERROR_CANCEL:` If the command has been cancelled. See [TEE Core API] section 2.1.4.

**Panic Reasons**

- The Implementation detects any error which is not explicitly associated with a defined return code for this function.
- `seReaderHandle` is not a valid handle on a `TEE_seReaderHandle`.
- `seSessionHandle` is not a valid reference (`NULL` included) on a `TEE_seSessionHandle`.

### 5.6.4 TEE_SEReaderCloseSessions

```
void TEE_SEReaderCloseSessions (
   TEE_SEReaderHandle seReaderHandle
);
```

**Description**

The `TEE_SEReaderCloseSessions` function closes all the sessions opened on this reader. All the channels opened by all these sessions will be closed.

**Parameters**

- `seReaderHandle:` Reference to seReader handle

**Specification Number:** 24 **Function Number:** 0x0204

**Return Value**

- None

**Panic Reasons**

- The Implementation detects any error which is not explicitly associated with a defined return code for this function.
- `seReaderHandle` is not a valid handle on a `TEE_seReaderHandle`.

## 5.7 SESession Class

### 5.7.1 TEE_SESessionGetATR

```
TEE_Result TEE_SESessionGetATR(
   TEE_SESessionHandle seSessionHandle,
   [outbuf]  void* atr, uint32_t* atrLen
);
```

**Description**

The `TEE_SESessionGetATR` function returns the ATR of this Secure Element.

The returned byte array can be `NULL` if the ATR for this Secure Element is not available.

**Parameters**

- `seSessionHandle:` Reference to seSession handle
- `atr, atrLen:` Output buffer containing the ATR as a byte array

**Specification Number:** 24 **Function Number:** 0x0301

**Return Value**

- `TEE_SUCCESS:` In case of success
- `TEE_ERROR_COMMUNICATION:` If the ATR is not available or if there is an I/O error
- `TEE_ERROR_SHORT_BUFFER:` If `atr` buffer is too small to hold the whole session ATR

**Panic Reasons**

- The Implementation detects any error which is not explicitly associated with a defined return code for this function.
- `seSessionHandle` is not a valid handle on a `TEE_seSessionHandle`.
- `atr` is not a valid reference (`NULL` included) on a `void`.
- `atrLen` is not a valid reference (`NULL` included) on a `uint32_t`.
- `*atrLen` is `0` and no error is returned.

### 5.7.2 TEE_SESessionIsClosed

```
TEE_Result TEE_SESessionIsClosed(
   TEE_SESessionHandle seSessionHandle
);
```

**Description**

The TEE_SESessionIsClosed function returns with success if this session is closed, returns TEE_SE_SESSION_OPEN if the session is open, and TEE_ERROR_COMMUNICATION if the state of the session is closed or cannot be detected.

**Parameters**

- seSessionHandle: Reference to seSession handle

**Specification Number:** 24 **Function Number:** 0x0302

**Return Value**

- TEE_SUCCESS: If session is closed, or handle is invalid
- TEE_SE_SESSION_OPEN: If session is open
- TEE_ERROR_COMMUNICATION: If the SE is not present or the state cannot be detected (e.g. if there is an I/O error)

**Panic Reasons**

- The Implementation detects any error which is not explicitly associated with a defined return code for this function.
- ~~seSessionHandle is not a valid handle on a TEE_seSessionHandle.~~

### 5.7.3 TEE_SESessionClose

```
void TEE_SESessionClose(
   TEE_SESessionHandle seSessionHandle
);
```

**Description**

The `TEE_SESessionClose` function closes any channel opened on this session and the session itself. No operation will be performed if the session is already closed or invalid.

**Parameters**

- `seSessionHandle:` Reference to seSession handle

**Specification Number:** 24 **Function Number:** 0x0303

**Return Value**

- None

**Panic Reasons**

- The Implementation detects any error which is not explicitly associated with a defined return code for this function.
- ~~`seSessionHandle` is not a valid handle on a `TEE_seSessionHandle`.~~

### 5.7.4 TEE_SESessionCloseChannels

```
void TEE_SESessionCloseChannels(
   TEE_SESessionHandle seSessionHandle
);
```

**Description**

The `TEE_SESessionCloseChannels` function closes any channels opened on this session.

The session itself is not closed.

**Parameters**

- `seSessionHandle:` Reference to seSession handle

**Specification Number:** 24 **Function Number:** 0x0304

**Return Value**

- None

**Panic Reasons**

- The Implementation detects any error which is not explicitly associated with a defined return code for this function.
- `seSessionHandle` is not a valid handle on a `TEE_seSessionHandle`.

### 5.7.5 TEE_SESessionOpenBasicChannel

```
TEE_Result TEE_SESessionOpenBasicChannel(
        TEE_SESessionHandle   seSessionHandle,
  [in]  TEE_SEAID            *seAID,
  [out] TEE_SEChannelHandle  *seChannelHandle
);
```

**Description**

The `TEE_SESessionOpenBasicChannel` function obtains access to the basic channel, as defined in ISO/IEC 7816-4 ([ISO 7816-4]) (the channel that has number `0`).

If the `seAID->bufferLen` is `0` or `seAID is NULL`, which means no SE application is to be selected on this channel, the default SE application is used, else the corresponding SE application is selected.

Once this channel has been opened by a device application, it is considered "locked" ~~by this device application~~, and for other calls ~~to~~ this method will ~~return~~ set `*seChannelHandle to NULL`, until the channel is closed. Some Secure Elements (such as the UICC) might always keep the basic channel locked ~~(i.e. return NULL to applications)~~, to prevent access to the basic channel, while ~~some~~ others might return a channel object implementing some kind of filtering on the commands, restricting the set of accepted command to a smaller set.

The SELECT response data can be retrieved with `TEE_SEChannelGetSelectResponse`.

This method shall be based on a SELECT command as defined in GlobalPlatform Card Specification [GPCS].

**Parameters**

- `seSessionHandle:` Reference to seSession handle
- `seAID:` Reference to the AID with which the channel is to be opened
- `seChannelHandle:` Reference to seChannel handle

**Specification Number:** 24 **Function Number:** 0x0305

**Return Value**

- `TEE_SUCCESS:` In case of success
- `TEE_ERROR_COMMUNICATION:` If the SE is not present or if there is an I/O error
- `TEE_ERROR_BAD_STATE:` If the seSession is closed
- `TEE_ERROR_BAD_PARAMETERS:` If the `seAID->bufferLen` is not within 5 to 16 (inclusive) and is not `0`
- `TEE_ERROR_NOT_SUPPORTED:` If the Secure Element, ~~or~~ the AID on the Secure Element, ~~is not available or a logical~~ or the basic channel is already ~~open to a non-multiselectable Applet~~ in use
- `TEE_ERROR_SECURITY:` If the calling application cannot be granted access to this AID or the default SE application on this session
- `TEE_ERROR_CANCEL:` If the command has been cancelled

**Panic Reasons**

- The Implementation detects any error which is not explicitly associated with a defined return code for this function.
- `seSessionHandle` is not a valid handle on a `TEE_seSessionHandle`.
- `seAID` is not a valid reference (`NULL` not included) on a `TEE_SEAID`.
- `seAID->buffer` is not a valid reference (`NULL` included) on a `uint8_t`.
- `seChannelHandle` is not a valid reference (`NULL` included) on a `TEE_seChannelHandle`.

### 5.7.6 TEE_SESessionOpenLogicalChannel

```
TEE_Result TEE_SESessionOpenLogicalChannel(
        TEE_SESessionHandle  seSessionHandle,
  [in]  TEE_SEAID           *seAID,
  [out] TEE_SEChannelHandle *seChannelHandle
);
```

**Description**

The `TEE_SESessionOpenLogicalChannel` function opens a logical channel with the Secure Element, selecting the application represented by the given AID.

If the `seAID->bufferLen` is `0` or `seAID is NULL`, which means no application is to be selected on this channel, the default application is used. The Secure Element chooses which logical channel will be used.

The SELECT response data can be retrieved with `TEE_SEChannelGetSelectResponse`.

This method shall be based on a SELECT command as defined in [GPCS].

**Parameters**

- `seSessionHandle:` Reference to seSession handle
- `seAID:` Reference to the AID with which the channel is to be opened
- `seChannelHandle:` Reference to seChannel handle

**Specification Number:** 24 **Function Number:** 0x0306

**Return Value**

- `TEE_SUCCESS:` In case of success
- `TEE_ERROR_COMMUNICATION:` If the SE is not present or if there is an I/O error
- `TEE_ERROR_BAD_STATE:` If the seSession is closed
- `TEE_ERROR_BAD_PARAMETERS:` If the AID length is not within 5 to 16 (inclusive) and is not `0`
- `TEE_ERROR_NOT_SUPPORTED:` If the Secure Element or the AID on the Secure Element is not available or a logical channel is already open to a non-multiselectable Applet or all logical channels are already allocated
- `TEE_ERROR_SECURITY:` If the calling application cannot be granted access to this AID or the default SE application on this session
- `TEE_ERROR_CANCEL:` If the command has been cancelled. See [TEE Core API] section 2.1.4.

**Panic Reasons**

- The Implementation detects any error which is not explicitly associated with a defined return code for this function.
- `seSessionHandle` is not a valid handle on a `TEE_seSessionHandle`.
- `seAID` is not a valid reference (`NULL` not included) on a `TEE_SEAID`.
- `seAID->buffer` is not a valid reference (`NULL` included) on a `char`.

- `seChannelHandle` is not a valid reference (`NULL` included) on a `TEE_seChannelHandle`.

## 5.8 SEChannel Class

### 5.8.1 TEE_SEChannelClose

```
void TEE_SEChannelClose(
   TEE_SEChannelHandle   seChannelHandle
);
```

**Description**

The `TEE_SEChannelClose` function closes this channel to the Secure Element. The channel will be in closed state after the successful execution of this function. If the function is called when the channel is already closed or not valid, the function call will be ignored.

The `TEE_SEChannelClose` function shall wait for completion of any pending `TEE_SEChannelTransmit` before closing the channel.

**Parameters**

- `seChannelHandle:` Reference to seChannel handle

**Specification Number:** 24 **Function Number:** 0x0401

**Return Value**

- None

**Panic Reasons**

- The Implementation detects any error which is not explicitly associated with a defined return code for this function.
- ~~`seChannelHandle` is not a valid handle on a `TEE_seChannelHandle`.~~

### 5.8.2 TEE_SEChannelSelectNext

```
TEE_Result TEE_SEChannelSelectNext (
   TEE_SEChannelHandle seChannelHandle
);
```

**Description**

The TEE_SEChannelSelectNext function performs a selection of the next Applet on this channel. This method can be used by a device application in order to iterate through all Applets matching to the same partial AID.

A partial AID could match to several Applets residing in an SE. If a device application is using a partial AID on TEE_SESessionOpenBasicChannel or TEE_SESessionOpenLogicalChannel in order to establish a communication channel, then the first Applet in the SE matching to this partial AID will be selected by the card manager on the SE (the order of selections might depend on the card manager implementation). In order to select the next Applet matching to the partial AID, the function TEE_SEChannelSelectNext() can be used. If TEE_SEChannelSelectNext() returns success a new Applet was successfully selected on this channel. If no further Applet exists with matches to the partial AID this method returns TEE_ERROR_ITEM_NOT_FOUND and the already selected Applet remains selected.

If a device application is using a full AID on TEE_SESessionOpenBasicChannel or TEE_SESessionOpenLogicalChannel, this method should always return TEE_ERROR_ITEM_NOT_FOUND. Note: Since the API cannot distinguish between a partial and full AID, the API shall rely on the response of the Secure Element for the return value of this function.

The SELECT response data can be retrieved with TEE_SEChannelGetSelectResponse.

This function shall be based on a SELECT command as defined in [GPCS].

**Parameters**

- seChannelHandle: Reference to seChannel handle

**Specification Number:** 24 **Function Number:** 0x0402

**Return Value**

- TEE_SUCCESS: In case of success
- TEE_ERROR_ITEM_NOT_FOUND: If a next Applet is not available. The last selected Applet remains selected.
- TEE_ERROR_NOT_SUPPORTED: If this function is not supported by the API implementation
- TEE_ERROR_COMMUNICATION: If the SE is not present or if there is an I/O error
- TEE_ERROR_BAD_STATE: If the seSession is closed
- TEE_ERROR_CANCEL: If the command has been cancelled. See [TEE Core API] section 2.1.4.

**Panic Reasons**

- The Implementation detects any error which is not explicitly associated with a defined return code for this function.
- seChannelHandle is not a valid handle on a TEE_seChannelHandle.

### 5.8.3 TEE_SEChannelGetSelectResponse

```
TEE_Result TEE_SEChannelGetSelectResponse (
   TEE_SEChannelHandle seChannelHandle,
  [outbuf]  void* response, uint32_t *responseLen
);
```

**Description**

The `TEE_SEChannelGetSelectResponse` function returns the response data and the hexadecimal status word resulting from the SELECT command used to open the channel.

Three functional return cases are possible:

- The data as returned by the application SELECT command, including the status word
- Only the status word if the application SELECT command has no returned data
- `TEE_ERROR_NO_DATA` if an application SELECT command has not been performed or the selection response cannot be retrieved by the reader implementation (i.e. the `selectResponseEnable` in the reader properties is set to `False`)

The returned byte array contains the data bytes in the following order:

[<first data byte>,…, <last data byte>, <SW1>, <SW2>]

**Parameters**

- `seChannelHandle:` Reference to seChannel handle
- `response, responseLen:` Output buffer containing the SELECT response as a byte array

**Specification Number:** 24 **Function Number:** 0x0403

**Return Value**

- `TEE_SUCCESS:` In case of success
- `TEE_ERROR_COMMUNICATION:` If the SE is not present or if there is an I/O error
- `TEE_ERROR_BAD_STATE:` If the seSession is closed
- `TEE_ERROR_NO_DATA:` If no data is available (see above)
- `TEE_ERROR_SHORT_BUFFER:` If `response` buffer is too small to hold the whole response of the SELECT command.

**Panic Reasons**

- The Implementation detects any error which is not explicitly associated with a defined return code for this function.
- `seChannelHandle` is not a valid handle on a `TEE_seChannelHandle`.
- `response` is not a valid reference (`NULL` included) on a `void`.
- `responseLen` is not a valid reference (`NULL` included) on a `uint32_t`.
- `*responseLen` is `0`.

Please see [TEE Core API] section 3.4.4 with regard to handling `TEE_ERROR_SHORT_BUFFER` errors.

### 5.8.4 TEE_SEChannelTransmit

```
TEE_Result TEE_SEChannelTransmit(
                        TEE_SEChannelHandle     seChannelHandle,
   [inbuf][inout(commandLen)]   void*                        command,
   [in]                 uint32_t                commandLen,
   [outbuf]             void*                   response,
   [out]                uint32_t*               responseLen
);
```

**Description**

The `TEE_SEChannelTransmit` function transmits an APDU command (as per [ISO 7816-4]) to the Secure Element. The underlying layers generate as many TPDUs as necessary to transport this APDU. The transport part is invisible from the application. The generated response is the response of the APDU.

The system ensures synchronization between concurrent calls to this method, and that only one APDU will be sent at a time, irrespective of the number of TPDUs that might be required to transport it to the SE.

The channel information in the class byte in the APDU will be ignored. The system will add any required information to ensure that the APDU is transported on this channel.

There are restrictions on the set of commands that can be sent:

- MANAGE_CHANNEL commands are not allowed.
- SELECT by DF Name (P1='04') is not allowed. (See [ISO 7816-4] section 8.2)
- CLA bytes with channel numbers are de-masked.

In case of T=0 the GET RESPONSE APDU to be sent to the card when the card is indicating that it has response data is handled within this API. The caller does not need to handle it.

This method shall be based on APDU commands as defined in [GPCS].

**Note:** If a secure messaging session is established for the `seChannelHandle`, then the `command` and `response` buffers require additional space. For more details about the additional buffer size which is required for secure messaging see section 6.2.5, `TEE_SESecureChannelOpen`.

Command APDU: If the provided command APDU defines a response length in the LE byte which does not correlate to the actual response APDU length, the SE indicates the right length with the hexadecimal status word '61XX' or '6CXX' (see [ISO 7816-4]). Depending on this status word, the response APDU has be fetched either by re-issuing the command APDU with the correct LE byte (i.e. if the `responseLen` >= XX from '6CXX' then the API needs to re-issue the C-APDU with the correct LE byte) or by using the GET RESPONSE command (i.e. if the SE returns '61XX' then the API sends a GET RESPONSE which requests XX bytes response data). The API performs this response handling automatically. However, if the provided `response` buffer is too small this response handling has to be performed by the calling application itself. In such a case this routine returns `TEE_ERROR_61_SHORT_BUFFER` or `TEE_ERROR_6C_SHORT_BUFFER`. The correct response length from the status word can be obtained from `TEE_SEChannelGetResponseLength`. However, in some cases this status word information is not available (depending on the underlying drivers and controllers) and `TEE_ERROR_SHORT_BUFFER` is returned without further information.

Response APDU: The caller of this routine has to take care that the provided `response` buffer has a sufficient buffer size. If the response APDU is larger than the provided `response` buffer, this routine returns `TEE_ERROR_SHORT_BUFFER`. It has to be considered that secure messaging requires additional space.

**Note:** If `TEE_ERROR_SHORT_BUFFER` is returned, the Command APDU has to be re-sent by the TA with an appropriate buffer size. The parameters `response, responseLen` will not be modified in this case.

**Note:** If a security error occurred with the Secure Channel (e.g. MAC verification failed), then the Secure Channel Session will be aborted and the function returns with `TEE_ERROR_SECURITY`. The security level will be reset to `NO_SECURITY_LEVEL`.

**Note:** If the reader in use supports extended length APDUs, then the API implementation converts the Command APDUs to extended length APDUs automatically.

**Parameters**

- `seChannelHandle:` Reference to seChannel handle
- `command, commandLen:` ~~Input~~ Buffer containing the command to send as a byte array; needs to be big enough to allow secure messaging wrapping inside the provided buffer
- `response, responseLen:` Output buffer containing the response as a byte array; needs to be big enough to allow secure messaging unwrapping inside the provided buffer

**Specification Number:** 24 **Function Number:** 0x0404

**Return Value**

- `TEE_SUCCESS:` In case of success
- `TEE_ERROR_COMMUNICATION:` If the SE is not present or if there is an I/O error
- `TEE_ERROR_BAD_STATE:` If the seSession is closed
- `TEE_ERROR_BAD_PARAMETERS:` If `commandLen` is less than 4 bytes
- `TEE_ERROR_BAD_PARAMETERS:` If `commandLen` is not consistent with APDU length
- `TEE_ERROR_SHORT_BUFFER:` If `response` buffer is too small to hold the whole response APDU
- `TEE_ERROR_61_SHORT_BUFFER:` If the SE returned the hexadecimal status word '61XX' ~~is returned by the SE~~
- `TEE_ERROR_6C_SHORT_BUFFER:` If the SE returned the hexadecimal status word '6CXX' ~~is returned by the SE~~
- `TEE_ERROR_SECURITY:` If a security error is found (e.g. the command is filtered out by the restrictions described above or the secure channel session was aborted)
- `TEE_ERROR_CANCEL:` If the command has been cancelled. See [TEE Core API] section 2.1.4.

**Panic Reasons**

- The Implementation detects any error which is not explicitly associated with a defined return code for this function.
- `seChannelHandle` is not a valid handle on a `TEE_seChannelHandle`.

- `command` is not a valid reference (`NULL` included) on a `void`.
- `commandLen` is `0`.
- `response` is not a valid reference (`NULL` included) on a `void`.
- `responseLen` is not a valid reference (`NULL` included) on a `uint32_t`.
- `*responseLen` is `0`.

Please see [TEE Core API] section 3.4.4 with regard to handling `TEE_ERROR_SHORT_BUFFER` errors.

### 5.8.5 TEE_SEChannelGetResponseLength

```
TEE_Result TEE_SEChannelGetResponseLength (
   TEE_SEChannelHandle seChannelHandle,
   [outbuf]  uint32_t *responseLen
);
```

**Description**

The `TEE_SEChannelGetResponseLength` function returns the actual length of response data as indicated by the SE with the hexadecimal status word '61XX' or '6CXX' resulting from a transmit operation performed with `TEE_SEChannelTransmit`.

This function does provide the length value gained from the status word '61XX' or '6CXX' (see [ISO 7816-4]) which is received during a transmit operation with `TEE_SEChannelTransmit`. This length value is available only if the `response` buffer, defined as a parameter on `TEE_SEChannelTransmit`, was too small to perform a response handling. In such a case the application has to implement the response handling by itself. Depending on the status word '61XX' or '6CXX' a GET RESPONSE command has to be used to fetch the response or the command has to be re-issued with the correct LE byte.

**Note:** In some cases the status word information ('61XX' or '6CXX') is not available (depending on the underlying drivers and controllers) and this routine returns `TEE_ERROR_NO_DATA`. In such cases it is recommended to try the transaction again, increasing the `response` buffer to an estimated size.

**Parameters**

- `seChannelHandle:` Reference to seChannel handle
- `responseLen:` The actual response length as indicated by the SE

**Specification Number:** 24 **Function Number:** 0x0405

**Return Value**

- `TEE_SUCCESS:` In case of success
- `TEE_ERROR_BAD_STATE:` If the seSession is closed
- `TEE_ERROR_NO_DATA:` If no response length information is available

**Panic Reasons**

- The Implementation detects any error which is not explicitly associated with a defined return code for this function.
- `seChannelHandle` is not a valid handle on a `TEE_seChannelHandle`.
- `responseLen` is not a valid reference (`NULL` included) on a `uint32_t`.

# 6 Service Layer APIs

The Service Layer APIs consist of a Discovery API and Secure Channel API and are both optional supplementary APIs to the TEE Secure Element API. The availability can be checked by the presence and values of associated properties. They provide high level functions for Secure Element based operations to Trusted Applications in a TEE. The intention of these APIs is to simplify the development of Trusted Applications using the Secure Element.

A Service API relies on functions as defined in Chapter 5 for performing the communication and transport layer management to the Secure Element.

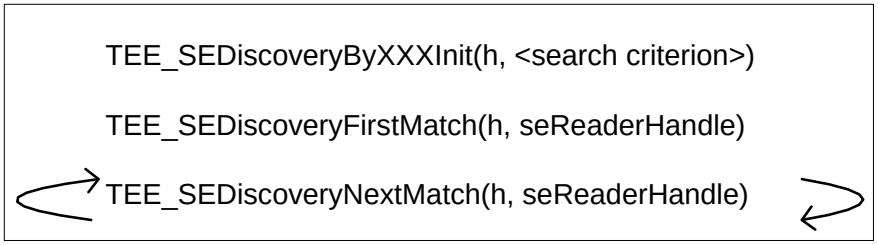
This Service API concept is inspired by [Open Mobile].

## 6.1 Discovery API

In order to simplify the selection of Secure Elements, the Discovery API provides a discovery mechanism based on a search criterion (e.g. the AID of the SE Applet). A discovery engine performs a search through all Secure Elements available to the TEE. All Secure Elements which match the defined search criterion are offered to the TA. Then the TA can choose which of these SEs to use for selecting the SE Applet and performing the operations.

The following figure outlines the concept of the Discovery API. During the initialization the TA can define a search criterion. After the initialization the search can be performed with dedicated functions in a loop.

**Figure 6-1: Discovery Mechanism**




### 6.1.1 Property

The Discovery API is implemented only if the `gpd.tee.seapi.service.discovery` property (a `Boolean`) is present and is set to `True`. If the property is missing or is set to `False`, then the Discovery API routines are not implemented.

### 6.1.2 Discovery Handle

The Discovery API requires a handle which is needed to perform an iteration based search through different Secure Elements.

```
typedef struct __TEE_SEDiscoveryHandle* TEE_SEDiscoveryHandle;
```

This handle is an opaque handle which is provided by the API after the discovery initialization (see below).

### 6.1.3 TEE_SEDiscoveryByAIDInit

```
TEE_Result TEE_SEDiscoveryByAIDInit (
   TEE_SEServiceHandle seServiceHandle,
   [out]  TEE_SEDiscoveryHandle* seDiscoveryHandle,
   [in]   TEE_SEAID *seAID
);
```

**Description**

The `TEE_SEDiscoveryByAIDInit` function initializes the discovery engine based on the AID as search criterion. It returns an `seDiscoveryHandle` which can be used for the search.

**Parameters**

- `seServiceHandle:` Reference to seService handle
- `seDiscoveryHandle:` Reference to seDiscovery handle
- `*seAID:` The AID of the Applet to be searched for

**Specification Number:** 24 **Function Number:** 0x0501

**Return Value**

- `TEE_SUCCESS:` In case of success
- `TEE_ERROR_COMMUNICATION:` If the SE is not present or if there is an I/O error
- `TEE_ERROR_BAD_STATE:` If the `seServiceHandle` is closed
- `TEE_ERROR_CANCEL:` If the command has been cancelled. See [TEE Core API] section 2.1.4.

**Panic Reasons**

- The Implementation detects any error which is not explicitly associated with a defined return code for this function.
- `seServiceHandle` is not a valid handle on a `TEE_seServiceHandle`.
- `seDiscoveryHandle` is not a valid handle on a `TEE_seDiscoveryHandle`.
- `seAID` is not a valid reference (`NULL` not included) on a `TEE_SEAID`.
- `seAID->buffer` is not a valid reference (`NULL` included) on a `char`.
- `seAID->length` is `0`.

### 6.1.4 TEE_SEDiscoveryByHistoricalBytesInit

```
TEE_Result TEE_SEDiscoveryByHistoricalBytesInit (
   TEE_SEServiceHandle seServiceHandle,
   [out]  TEE_SEDiscoveryHandle* seDiscoveryHandle,
   [inbuf]   void*  histBytes, uint32_t histBytesLen
);
```

**Description**

The `TEE_SEDiscoveryByHistoricalBytesInit` function initializes the discovery engine based on the historical bytes (see Chapter 8 in [ISO 7816-4]) as search criterion. It returns an `seDiscoveryHandle` which can be used for the search.

**Parameters**

- `seServiceHandle:` Reference to seService handle
- `seDiscoveryHandle:` Reference to seDiscovery handle
- `*histBytes:` The historical bytes to be searched for
- `histBytesLen:` The length of the historical bytes

**Specification Number:** 24 **Function Number:** 0x0502

**Return Value**

- `TEE_SUCCESS:` In case of success
- `TEE_ERROR_COMMUNICATION:` If the SE is not present or if there is an I/O error
- `TEE_ERROR_BAD_STATE:` If the `seServiceHandle` is closed
- `TEE_ERROR_CANCEL:` If the command has been cancelled. See [TEE Core API] section 2.1.4.

**Panic Reasons**

- The Implementation detects any error which is not explicitly associated with a defined return code for this function.
- `seServiceHandle` is not a valid handle on a `TEE_seServiceHandle`.
- `seDiscoveryHandle` is not a valid handle on a `TEE_seDiscoveryHandle`.
- `histBytes` is not a valid reference (`NULL` included) on a `char`.
- `histBytesLen` is `0`.

### 6.1.5 TEE_SEDiscoveryByATRInit

```
TEE_Result TEE_SEDiscoveryByATRInit (
   TEE_SEServiceHandle seServiceHandle,
   [out] TEE_SEDiscoveryHandle* seDiscoveryHandle,
   [inbuf] void* atr, uint32_t atrLen,
   [inbuf] void* mask, uint32_t maskLen
);
```

**Description**

The `TEE_SEDiscoveryByATRInit` function initializes the discovery engine based on the ATR as search criterion. The relevant ATR bytes for the match have to be defined with a mask. Only those bytes which are set in the mask are used as search criterion. This means the mask is a byte array which has to be applied with an AND-operation to the Secure Element ATR byte array values before comparison with the searched value. The byte array of the mask and the ATR must have the same length. This function returns an `seDiscoveryHandle` which can be used for the search.

**Parameters**

- `seServiceHandle:` Reference to seService handle
- `seDiscoveryHandle:` Reference to seDiscovery handle
- `*atr:` The ATR to be searched for
- `atrLen:` The length of the ATR
- `*mask:` The mask to be applied to the ATR
- `maskLen:` The length of the mask

**Specification Number:** 24 **Function Number:** 0x0503

**Return Value**

- `TEE_SUCCESS:` In case of success
- `TEE_ERROR_COMMUNICATION:` If the SE is not present or if there is an I/O error
- `TEE_ERROR_BAD_STATE:` If the `seServiceHandle` is closed
- `TEE_ERROR_CANCEL:` If the command has been cancelled. See [TEE Core API] section 2.1.4.

**Panic Reasons**

- `seServiceHandle` is not a valid handle on a `TEE_seServiceHandle`.
- `seDiscoveryHandle` is not a valid handle on a `TEE_seDiscoveryHandle`.
- `atr` is not a valid reference (`NULL` included) on a `char`.
- `atrLen` is `0`.
- `mask` is not a valid reference (`NULL` included) on a `char`.
- `maskLen` is `0`.

### 6.1.6 TEE_SEDiscoveryFirstMatch

```
TEE_Result TEE_SEDiscoveryFirstMatch (
   TEE_SEDiscoveryHandle      seDiscoveryHandle,
   [out]  TEE_SEReaderHandle* seReaderHandle
);
```

**Description**

The `TEE_SEDiscoveryFirstMatch` function locates the first Reader that contains a Secure Element that matches the defined search criterion. It returns an `seReaderHandle` which is connected to the identified Reader.

**Parameters**

- `seDiscoveryHandle:` Reference to seDiscovery handle
- `seReaderHandle:` Reference to the first Reader which matches the search criterion

**Specification Number:** 24 **Function Number:** 0x0504

**Return Value**

- `TEE_SUCCESS:` In case of success
- `TEE_ERROR_COMMUNICATION:` If the SE is not present or if there is an I/O error
- `TEE_ERROR_BAD_STATE:` If the `seServiceHandle` used during initialization is closed
- `TEE_ERROR_ITEM_NOT_FOUND:` If a Reader couldn't be found for this search criterion
- `TEE_ERROR_CANCEL:` If the command has been cancelled. See [TEE Core API] section 2.1.4.

**Panic Reasons**

- The Implementation detects any error which is not explicitly associated with a defined return code for this function.
- `seDiscoveryHandle` is not a valid handle on a `TEE_seDiscoveryHandle`.

### 6.1.7 TEE_SEDiscoveryNextMatch

```
TEE_Result TEE_SEDiscoveryNextMatch (
   TEE_SEDiscoveryHandle seDiscoveryHandle,
   [out]  TEE_SEReaderHandle* seReaderHandle
);
```

**Description**

The `TEE_SEDiscoveryNextMatch` function locates the next Reader that contains a Secure Element that matches the defined search criterion. It returns an `seReaderHandle` which is connected to the identified Reader.

**Parameters**

- `seDiscoveryHandle:` Reference to seDiscovery handle
- `seReaderHandle:` Reference to the next Reader which matches the search criterion

**Specification Number:** 24 **Function Number:** 0x0505

**Return Value**

- `TEE_SUCCESS:` In case of success
- `TEE_ERROR_COMMUNICATION:` If the SE is not present or if there is an I/O error
- `TEE_ERROR_BAD_STATE:` If the `seServiceHandle` used during initialization is closed
- `TEE_ERROR_ITEM_NOT_FOUND:` If a Reader couldn't be found for this search criterion
- `TEE_ERROR_CANCEL:` If the command has been cancelled. See [TEE Core API] section 2.1.4.

**Panic Reasons**

- The Implementation detects any error which is not explicitly associated with a defined return code for this function.
- `seDiscoveryHandle` is not a valid handle on a `TEE_seDiscoveryHandle`.

### 6.1.8 TEE_SEDiscoveryIsDone

```
bool TEE_SEDiscoveryIsDone (
   TEE_SEDiscoveryHandle seDiscoveryHandle
);
```

**Description**

The `TEE_SEDiscoveryIsDone` function indicates whether the discovery engine is able to provide further Secure Elements which match to the defined search criterion. It returns `TRUE` if a further Secure Element exists matching the defined search criterion. Otherwise it returns `FALSE.`

**Parameters**

- `seDiscoveryHandle:` Reference to seDiscovery handle

**Specification Number:** 24 **Function Number:** 0x0506

**Return Value**

- `TRUE:` The search is done. No further SEs match the criterion.
- `FALSE:` The search is not done. Further SEs match the criterion.

**Panic Reasons**

- `seDiscoveryHandle` is not a valid handle on a `TEE_seDiscoveryHandle`.

### 6.1.9 TEE_SEDiscoveryClose

```
void TEE_SEDiscoveryClose (
   TEE_SEDiscoveryHandle seDiscoveryHandle
);
```

**Description**

The `TEE_SEDiscoveryClose` function terminates the discovery engine. It closes all resources which are allocated by the discovery engine and sets `seDiscoveryHandle to TEE_HANDLE_NULL.`

**Parameters**

- `seDiscoveryHandle:` Reference to seDiscovery handle

**Specification Number:** 24 **Function Number:** 0x0507

**Return Value**

- None

**Panic Reasons**

- The Implementation detects any error which is not explicitly associated with a defined return code for this function.
- `seDiscoveryHandle` is not a valid handle on a `TEE_seDiscoveryHandle`.

## 6.2 Secure Channel API

The Secure Channel API provides functions for TAs which can be used to enable secure channel communication between TEE and Secure Element. The secure channel session is performed by the TEE and works completely transparently from a TA point of view. This means the usage of a secure channel does not require changes in the design of a TA.

This specification considers the GlobalPlatform Secure Channel Protocols SCP02, SCP03, and SCP11 as defined in [GPCS]; GPCS Amendment D, Secure Channel Protocol '03' [Amd D]; and GPCS Amendment F, Secure Channel Protocol '11' [Amd F] for the secure channel communication. Further protocols may be considered in future versions of this specification. The design of the Secure Channel API also allows the usage of other secure channel protocols, even those which are not specified in GlobalPlatform.

A Secure Channel Protocol can be specified by an OID (by selecting an OID from a list of accepted OIDs). The functions of the Secure Channel API are based on a generic concept which also allows the usage of customized parameter structures needed for a specific secure channel protocol which are chosen to be implemented by individual TEE vendors. However, the usage of these customized structures and protocols beyond SCP02, SCP03, and SCP11 is out of scope of this document and is implementer dependent.

### 6.2.1 Property

The Secure Channel API is implemented only if the `gpd.tee.seapi.service.securechannel` property (a `Boolean`) is present and is set to `True`. If the property is missing or is set to `False`, then the Secure Channel API routines are not implemented.

### 6.2.2 Secure Channel Parameters

#### 6.2.2.1 TEE_SC_Params

This type defines the parameters which are needed to set up a secure channel.

```
typedef struct __TEE_SC_Params
{
   uint8_t scType;                       // the SC type
   TEE_SC_OID scOID;                     // the SC type defined by OID
   TEE_SC_SecurityLevel scSecurityLevel; // the SC security level
   TEE_SC_CardKeyRef scCardKeyRef;       // reference to SC card keys
   TEE_SC_DeviceKeyRef scDeviceKeyRef;   // reference to SC device keys
} TEE_SC_Params;
```

The Secure Channel type can be specified either by defining an OID or by a constant. The constant `scType` will apply only if `scOID.buffer` is set to `NULL` and `scOID.bufferLen` is set to `0`.

#### 6.2.2.2 TEE_SC_OID

This type defines the type of protocol which shall be used for the secure channel.

```
typedef struct __TEE_SC_OID
{
    uint8_t *buffer;      // the value of the OID
    uint32_t bufferLen;   // length of the SC OID
} TEE_SC_OID;
```

#### 6.2.2.3 TEE_SC_SecurityLevel

This enumeration lists the Security Levels which can be applied for a secure channel.

```
typedef enum
{
   TEE_SC_NO_SECURE_MESSAGING = 0x00,   // Nothing will be applied
   TEE_SC_AUTHENTICATE = 0x80,  // Command, Response APDU not be secured
   TEE_SC_C_MAC = 0x01,       // Command APDU shall be MAC protected
   TEE_SC_R_MAC = 0x10,       // Response APDU shall be MAC protected
   TEE_SC_CR_MAC = 0x11,      // Command, Response APDU shall be MAC
                              //    protected
   TEE_SC_C_ENC_MAC = 0x03,   // Command APDU shall be encrypted and
                              //    MAC protected
   TEE_SC_R_ENC_MAC = 0x30,   // Response APDU encrypted, MAC protected
   TEE_SC_CR_ENC_MAC = 0x33, // Command, Response APDU encrypted and
                              //    MAC protected
   TEE_SC_C_ENC_CR_MAC = 0x13 // Command APDU encrypted;
                              //   Command, Response APDU MAC protected
} TEE_SC_SecurityLevel;

#define TEE_AUTHENTICATE (TEE_SC_AUTHENTICATE)
                     // deprecated: Command, Response APDU not secured
```

**Notes:**

- `TEE_SC_AUTHENTICATE` shall be applied if only authentication between TEE and SE is expected. The Command and Response APDUs transferred after this authentication are not secured.
- A Security Level with response encryption cannot be applied on SCP02. If this is attempted, `TEE_SESecureChannelOpen` will respond with the error `TEE_ERROR_BAD_PARAMETERS`.
- SCP11 only allows the security levels `TEE_SC_CR_MAC` and `TEE_SC_CR_ENC_MAC`. All other options will result in an error.

For more details about the Secure Channel security levels refer to [GPCS] (see the coding in section 10.6, Table 10-1) or [Amd D] (see the coding in section 7.1.2, Table 7-3).

#### 6.2.2.4 TEE_SC_CardKeyRef

This type defines the reference to the card keys which shall be used for the secure channel.

```
typedef struct __TEE_SC_CardKeyRef
{
   uint8_t scKeyID;           // key identifier of the SC card key
   uint8_t scKeyVersion;      // key version of the SC card key
} TEE_SC_CardKeyRef;
```

#### 6.2.2.5 TEE_SC_DeviceKeyRef

This type defines the reference to the device keys which shall be used for the secure channel.

```
typedef struct __TEE_SC_DeviceKeyRef
{
   TEE_SC_KeyType scKeyType;           // type of SC keys
   union
   {
   TEE_ObjectHandle scBaseKeyHandle;   // SC base key (acc. to SCP02)
   TEE_SC_KeySetRef scKeySetRef;       // Key-ENC, Key-MAC (acc. to
                                       //    SCP02, SCP03)
   } __TEE_key
} TEE_SC_DeviceKeyRef;
```

**Note:** SCP02 and SCP03 require a set of static keys (Key-MAC and Key-ENC) from which session keys (S-MAC and S-ENC) are derived during protocol initialization. The session keys are used for the secure messaging session. As an alternative, SCP02 allows the usage of a single base key from which the session keys S-MAC and S-ENC are derived during protocol initialization. ~~SCP11b~~ SCP11a requires an ECC key pair SK.OCE.ECKA/PK.OCE.ECKA [Amd F] which shall be referenced by the base key object handle. ~~SCP11a~~ SCP11b does not require any keys on the device side and `scBaseKeyHandle` shall be set to `NULL`.

**Note:** SCP11 requires passing CERT.OCE.ECKA and PK.CA-KLCC.ECDSA for the protocol execution. This version of the API specification requires passing these parameters either by proprietary extensions or by API internal implementations with an internal certificate store.

#### 6.2.2.6 TEE_SC_KeyType

This enumeration lists the options for key types.

```
typedef enum
{
   TEE_SC_BASE_KEY = 0,   // A base key acc. to SCP02
   TEE_SC_KEY_SET = 1     // A key set (key-ENC, key-MAC) acc. to SCP02,
                          //    SCP03
} TEE_SC_KeyType;
```

#### 6.2.2.7 TEE_SC_KeySetRef

This type can be used to define a key set with Key-MAC and Key-ENC according to SCP02 and SCP03.

```
typedef struct __TEE_SC_KeySetRef
{
    TEE_ObjectHandle scKeyEncHandle;  // Key-ENC (static encryption key)
    TEE_ObjectHandle scKeyMacHandle;  // Key-MAC (static MAC key)
} TEE_SC_KeySetRef;
```

The TA has to define object handles which refer to the secure channel keys stored in the TEE. The TA can obtain these object handles from its private Trusted Storage or from a certain Security Domain. How a TA obtains a key object handle is defined in [TEE Core API] and in TEE Administration Framework [TEE Admin].

**Note:**

- Secure channel keys can be transient objects. The usage of transient objects is described in [TEE Core API].
- The specified Key-ENC and Key-MAC must have the same type and length. Otherwise the function `TEE_SESecureChannelOpen` will respond with the error `TEE_ERROR_BAD_PARAMETERS.`
- SCP02 requires a double length DES key with 16 bytes as key value. SCP03 requires an AES key which may have a length of 16, 24, or 32 bytes as key value. If the length of the specified key value is inconsistent with the chosen protocol or the key cannot be applied on the cryptographic algorithm, the function `TEE_SESecureChannelOpen` will respond with the error `TEE_ERROR_BAD_PARAMETERS.`

### 6.2.3 Secure Channel Protocol Support

The following protocol types are mandatory to support for the Secure Channel API.

**Table 6-1: Secure Channel Protocol Type OIDs**

| Protocols | OID | Meaning |
|---|---|---|
| SCP02 | { 0x06, 0x08, 0x2A, 0x86, 0x48, 0x86, 0xFC, 0x6B, 0x04, 0x02 } | SCP02 according to [GPCS]:<br>iso(1) member-body(2) us(840) gp(114283) scp(04) v(2) |
| SCP03 | { 0x06, 0x08, 0x2A, 0x86, 0x48, 0x86, 0xFC, 0x6B, 0x04, 0x03 } | SCP03 according to [Amd D]:<br>iso(1) member-body(2) us(840) gp(114283) scp(04) v(3) |
| SCP11a | { 0x06, ~~0x08,~~ 0x09, 0x2A, 0x86, 0x48, 0x86, 0xFC, 0x6B, 0x04, 0x0B, 0x01 } | SCP11 according to [Amd F]:<br>iso(1) member-body(2) us(840) gp(114283) scp(04) v(11) i(01) |
| SCP11b | { 0x06, ~~0x08,~~ 0x09, 0x2A, 0x86, 0x48, 0x86, 0xFC, 0x6B, 0x04, 0x0B, 0x02 } | SCP11 according to [Amd F]:<br>iso(1) member-body(2) us(840) gp(114283) scp(04) v(11) i(02) |

**Table 6-2: Secure Channel Type Constants**

| Constant Name | Value | Meaning |
|---|---|---|
| `TEE_SC_TYPE_SCP02` | 0x00 | SCP02 according to [GPCS] |
| `TEE_SC_TYPE_SCP03` | 0x01 | SCP03 according to [Amd D] |
| `TEE_SC_TYPE_SCP11a` | 0x02 | SCP11a according to [Amd F] |
| `TEE_SC_TYPE_SCP11b` | 0x03 | SCP11b according to [Amd F] |
| | 0x02–0x7F | Reserved for future use |
| | 0x80–0xFF | Reserved for proprietary usage |

When the Secure Element API~~, when supporting~~ supports section 6.2, ~~then~~ the interface shall implement at least the following protocol features.

**Table 6-3: Secure Channel Protocol Features**

| Protocols | Features |
|---|---|
| SCP02 | C_ENC, C_MAC, R_MAC for secure messaging<br>SC base key, SC key set (MAC-key, ENC-key)<br>ICV set to zero, ICV encryption for C-MAC session<br>**Features not to be supported:**<br>R_MAC switch on and off during secure channel session<br>Implicit Secure Channel |
| SCP03 | R_ENC, C_ENC, C_MAC, R_MAC for secure messaging<br>AES-128, AES-192, AES-256 for Key-ENC, Key-MAC without secure messaging<br>**Features not to be supported:**<br>R-MAC switch on and off during secure channel session |
| SCP11 | SCP11a (providing mutual authentication)<br>SCP11b (providing authentication of the Secure Element only)<br>(C_ENC, R_ENC, C_MAC, R_MAC) and (C_MAC, R_MAC) for secure messaging<br>AES-128, AES-192, AES-256 for Key-ENC, Key-MAC |

### 6.2.4 Security Levels

The following constants can be used to check the current Security Level.

**Table 6-4: Security Level Constants**

| Constant Name | Value |
|---|---|
| TEE_SC_LEVEL_NO_SECURITY | 0x00 |
| TEE_SC_LEVEL_AUTHENTICATED | 0x80 |
| TEE_SC_LEVEL_C_MAC | 0x01 |
| TEE_SC_LEVEL_R_MAC | 0x10 |
| TEE_SC_LEVEL_CR_MAC | 0x11 |
| TEE_SC_LEVEL_C_ENC_MAC | 0x03 |
| TEE_SC_LEVEL_R_ENC_MAC | 0x30 |
| TEE_SC_LEVEL_CR_ENC_MAC | 0x33 |
| TEE_SC_LEVEL_C_ENC_CR_MAC | 0x13 |

For more details about the Secure Channel security levels refer to [GPCS] (see the coding in section 10.6, Table 10-1) or [Amd D] (see the coding in section 7.1.2, Table 7-3).

### 6.2.5 TEE_SESecureChannelOpen

```
TEE_Result TEE_SESecureChannelOpen(
        TEE_SEChannelHandle  seChannelHandle,
  [in]  TEE_SC_Params *scParams
);
```

**Description**

The `TEE_SESecureChannelOpen` function establishes a secure channel to the Secure Element. This secure channel is established on the communication channel which is specified by the parameter `TEE_SEChannelHandle`.

Once this function is called the API starts to set up the secure channel based on the specified protocol parameters. If this function returns with success the secure channel is established for the defined `TEE_SEChannelHandle` and subsequently all APDUs transmitted based on this handle with `TEE_SEChannelTransmit` are automatically secured (encrypted and/or MAC protected), depending on the secure channel parameter options defined on `TEE_SESecureChannelOpen`. The secure channel can be terminated with `TEE_SESecureChannelClose`. Before a new secure channel session may be established on the same Channel Handle, the TA needs to close the existing secure channel that is already established on the Channel Handle. Otherwise the error `TEE_ERROR_ACCESS_CONFLICT` will be returned by this function.

**Note:** Once a secure channel is established with `TEE_SESecureChannelOpen`, the `command` and `response` buffers defined by `TEE_SEChannelTransmit` require additional space for performing secure messaging. According to SCP02 and SCP03 the encryption of an APDU requires a padding of up to 16 bytes. The MAC code which is added to each message has a length of 8 bytes. Thus, an additional buffer size of up 24 bytes is required.

**Parameters**

- `seChannelHandle:` Reference to seChannel handle
- `scParams:` Reference to parameters for the Secure Channel Protocol

**Specification Number:** 24 **Function Number:** 0x0601

**Return Value**

- `TEE_SUCCESS:` In case of success
- `TEE_ERROR_COMMUNICATION:` If the SE is not present or if there is an I/O error
- `TEE_ERROR_BAD_STATE:` If the seChannel is closed
- `TEE_ERROR_BAD_PARAMETERS:` If `scParams` includes wrongly encoded data
- `TEE_ERROR_NOT_SUPPORTED:` If the `scParams` is not supported
- `TEE_ERROR_SECURITY:` If the authentication failed or a higher Security Level is expected
- `TEE_ERROR_CANCEL:` If the command has been cancelled
- `TEE_ERROR_ACCESS_CONFLICT:` If a secure channel already exists on seChannel

**Panic Reasons**

- The Implementation detects any error which is not explicitly associated with a defined return code for this function.

- `seChannelHandle` is not a valid reference (`NULL` included) on a `TEE_seChannelHandle`.
- `scParams` is not a valid reference (`NULL` not included) on a `TEE_SC_Params.`
- `scParams->scSecurityLevel` is not a valid reference (`NULL` included) on a `TEE_SC_SecurityLevel`.
- `scParams->scCardKeyRef` is not a valid reference (`NULL` included) on a `TEE_SC_CardKeyRef`.
- `scParams->scDeviceKeyRef` is not a valid reference (`NULL` included) on a `TEE_SC_DeviceKeyRef`.

### 6.2.6 TEE_SESecureChannelGetSecurityLevel

```
int TEE_SESecureChannelGetSecurityLevel(
   TEE_SEChannelHandle seChannelHandle
);
```

**Description**

The `TEE_SESecureChannelGetSecurityLevel` function returns the Security Level which is currently established with the Secure Element on the defined `seChannelHandle.` If the function is called when no Secure Channel is currently established, the function returns NO_SECURITY_LEVEL.

The following table lists possible values for the Security Level.

**Table 6-5: Security Level Coding**

| Option | Value | Meaning |
|---|---|---|
| NO_SECURITY_LEVEL | 0x00 | Secure Channel Session is terminated or not yet fully initiated. |
| AUTHENTICATED | 0x80 | Successful authentication was processed during this session. |
| C_MAC | 0x01 | Command APDUs are MAC protected in the current session. |
| C_ENC | 0x02 | Command APDUs are encrypted in the current session. |
| R_MAC | 0x10 | Response APDUs are MAC protected in the current session. |
| R_ENC | 0x20 | Response APDUs are encrypted in the current session. |
| CR_MAC | 0x11 | Command and Response APDUs are MAC protected in the current session. |
| C_ENC_MAC | 0x03 | Command APDUs are encrypted and MAC protected in the current session. |
| R_ENC_MAC | 0x30 | Response APDUs are encrypted and MAC protected in the current session. |
| CR_ENC_MAC | 0x33 | Command and Response APDUs are encrypted and MAC protected in the current session. |
| C_ENC_CR_MAC | 0x13 | Command APDUs are encrypted and Command and Response APDUs are MAC protected in the current session. |

For more details about the Secure Channel security levels refer to [GPCS] (see the coding in section 10.6, Table 10-1) or [Amd D] (see the coding in section 7.1.2, Table 7-3). The constants as defined in Table 6-4 can be used to check the current Security Level.

**Note:** The Security Level might change during the course of the session.

The Security Level returned by this function is a bit mask based on this coding. For more details about these options and possible combinations refer to [GPCS] or [Amd D]. To check the Security Level value returned by this function, the Security Level constants as defined in section 6.2.4 can be used.

**Parameters**

- `seChannelHandle:` Reference to seChannel handle

**Specification Number:** 24 **Function Number:** 0x0602

**Return Value**

- The Security Level encoded in an integer. See Table 6-5.

**Panic Reasons**

- The Implementation detects any error which is not explicitly associated with a defined return code for this function.
- `seChannelHandle` is not a valid handle on a `TEE_seChannelHandle`.

### 6.2.7 TEE_SESecureChannelClose

```
void TEE_SESecureChannelClose(
   TEE_SEChannelHandle seChannelHandle
);
```

**Description**

The `TEE_SESecureChannelClose` function terminates the secure channel which is currently established to the Secure Element. The channel itself is not closed and can be used for further unsecured communication. If the function is called with an established unsecured channel, the function call will be ignored. After executing this function, all APDUs transmitted with `TEE_SEChannelTransmit` on `seChannelHandle` are not secured.

**Note:** [GPCS] does not define an explicit command for terminating a secure channel. Therefore this function could use the command INITIALIZE UPDATE which effects a secure channel termination. After executing the command INITIALIZE UPDATE the Secure Channel is terminated and the Security Level is set to TEE_SC_LEVEL_NO_SECURITY.

The `TEE_SESecureChannelClose` function shall wait for completion of any pending `TEE_SEChannelTransmit` before terminating the secure channel.

**Parameters**

- `seChannelHandle:` Reference to seChannel handle

**Specification Number:** 24 **Function Number:** 0x0603

**Return Value**

- None

**Panic Reasons**

- The Implementation detects any error which is not explicitly associated with a defined return code for this function.
- `seChannelHandle` is not a valid handle on a `TEE_seChannelHandle`.

# Annex A Panicked Function Identification

If this specification is used in conjunction with the TEE TA Debug Specification [TEE Debug] then the specification number is 24 and the following values must be associated with the function declared.

**Table A-1: Function Identification Values**

| Category | Function | Function Number in hexadecimal | Function Number in decimal |
|---|---|---|---|
| SE Service | | | |
| | TEE_SEServiceOpen | 0x0101 | 257 |
| | TEE_SEServiceClose | 0x0102 | 258 |
| | TEE_SEServiceGetReaders | 0x0103 | 259 |
| SE Reader | | | |
| | TEE_SEReaderGetProperties | 0x0201 | 513 |
| | TEE_SEReaderGetName | 0x0202 | 514 |
| | TEE_SEReaderOpenSession | 0x0203 | 515 |
| | TEE_SEReaderCloseSessions | 0x0204 | 516 |
| SE Session | | | |
| | TEE_SESessionGetATR | 0x0301 | 769 |
| | TEE_SESessionIsClosed | 0x0302 | 770 |
| | TEE_SESessionClose | 0x0303 | 771 |
| | TEE_SESessionCloseChannels | 0x0304 | 772 |
| | TEE_SESessionOpenBasicChannel | 0x0305 | 773 |
| | TEE_SESessionOpenLogicalChannel | 0x0306 | 774 |
| SE Channel | | | |
| | TEE_SEChannelClose | 0x0401 | 1025 |
| | TEE_SEChannelSelectNext | 0x0402 | 1026 |
| | TEE_SEChannelGetSelectResponse | 0x0403 | 1027 |
| | TEE_SEChannelTransmit | 0x0404 | 1028 |
| | TEE_SEChannelGetResponseLength | 0x0405 | 1029 |

| Category | Function | Function Number in hexadecimal | Function Number in decimal |
|---|---|---|---|
| SE Discovery | | | |
| | TEE_SEDiscoveryByAIDInit | 0x0501 | 1281 |
| | TEE_SEDiscoveryByHistoricalBytesInit | 0x0502 | 1282 |
| | TEE_SEDiscoveryByATRInit | 0x0503 | 1283 |
| | TEE_SEDiscoveryFirstMatch | 0x0504 | 1284 |
| | TEE_SEDiscoveryNextMatch | 0x0505 | 1285 |
| | TEE_SEDiscoveryIsDone | 0x0506 | 1286 |
| | TEE_SEDiscoveryClose | 0x0507 | 1287 |
| SE Secure Channel | | | |
| | TEE_SESecureChannelOpen | 0x0601 | 1537 |
| | TEE_SESecureChannelGetSecurityLevel | 0x0602 | 1538 |
| | TEE_SESecureChannelClose | 0x0603 | 1539 |