# Portable Programing Environment and API on Trusted Execution Environment (TEE)

# Specification

The National Institute of Advanced Industrial Science and Technology

2022-02-10

# 1 Class Index

## 1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

# 2 File Index

## 2.1 File List

Here is a list of all files with brief descriptions:

# 3 Class Documentation

## 3.1 __TEE_ObjectHandle Struct Reference

`#include <tee_api_tee_types.h>`

### Public Attributes

- unsigned int type
- int flags
- int desc
- mbedtls_aes_context persist_ctx
- unsigned char persist_iv [TEE_OBJECT_NONCE_SIZE]
- unsigned char public_key [TEE_OBJECT_KEY_SIZE]
- unsigned char private_key [TEE_OBJECT_SKEY_SIZE]

### 3.1.1 Member Data Documentation

#### 3.1.1.1 desc `int __TEE_ObjectHandle::desc`

#### 3.1.1.2 flags `int __TEE_ObjectHandle::flags`

#### 3.1.1.3 persist_ctx `mbedtls_aes_context __TEE_ObjectHandle::persist_ctx`

#### 3.1.1.4 persist_iv `unsigned char __TEE_ObjectHandle::persist_iv`

#### 3.1.1.5 private_key `unsigned char __TEE_ObjectHandle::private_key`

#### 3.1.1.6 public_key `unsigned char __TEE_ObjectHandle::public_key`

#### 3.1.1.7 type `unsigned int __TEE_ObjectHandle::type`

The documentation for this struct was generated from the following files:

- ta-ref/api/keystone/tee_api_tee_types.h
- ta-ref/api/sgx/tee_api_tee_types.h

## 3.2 __TEE_OperationHandle Struct Reference

`#include <tee_api_tee_types.h>`

### Public Attributes

- int mode
- int flags
- int alg
- sha3_ctx_t ctx
- mbedtls_aes_context aectx
- mbedtls_gcm_context aegcmctx
- int aegcm_state
- unsigned char aeiv [TEE_OBJECT_NONCE_SIZE]
- unsigned char aekey [32]
- unsigned char pubkey [TEE_OBJECT_KEY_SIZE]
- unsigned char prikey [TEE_OBJECT_SKEY_SIZE]

### 3.2.1 Member Data Documentation

#### 3.2.1.1 aectx mbedtls_aes_context __TEE_OperationHandle::aectx

#### 3.2.1.2 aegcm_state int __TEE_OperationHandle::aegcm_state

#### 3.2.1.3 aegcmctx mbedtls_gcm_context __TEE_OperationHandle::aegcmctx

#### 3.2.1.4 aeiv unsigned char __TEE_OperationHandle::aeiv

#### 3.2.1.5 aekey unsigned char __TEE_OperationHandle::aekey

#### 3.2.1.6 alg int __TEE_OperationHandle::alg

#### 3.2.1.7 ctx sha3_ctx_t __TEE_OperationHandle::ctx

#### 3.2.1.8 flags int __TEE_OperationHandle::flags

#### 3.2.1.9 mode int __TEE_OperationHandle::mode

#### 3.2.1.10 prikey unsigned char __TEE_OperationHandle::prikey

#### 3.2.1.11 pubkey unsigned char __TEE_OperationHandle::pubkey

The documentation for this struct was generated from the following files:

- ta-ref/api/keystone/tee_api_tee_types.h
- ta-ref/api/sgx/tee_api_tee_types.h

## 3.3 _sgx_errlist_t Struct Reference

`#include <types.h>`

### Public Attributes

- sgx_status_t err
- const char * msg
- const char * sug

#### 3.3.1 Member Data Documentation

##### 3.3.1.1 err `sgx_status_t _sgx_errlist_t::err`

##### 3.3.1.2 msg `const char* _sgx_errlist_t::msg`

##### 3.3.1.3 sug `const char* _sgx_errlist_t::sug`

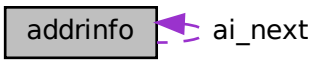
The documentation for this struct was generated from the following file:

- ta-ref/api/include/types.h

## 3.4 addrinfo Struct Reference

`#include <tee_api_types.h>`

Collaboration diagram for addrinfo:



**Public Attributes**

- int ai_flags
- int ai_family
- int ai_socktype
- int ai_protocol
- socklen_t ai_addrlen
- struct sockaddr * ai_addr
- char * ai_canonname
- struct addrinfo * ai_next

### 3.4.1 Member Data Documentation

#### 3.4.1.1 ai_addr `struct sockaddr* addrinfo::ai_addr`

#### 3.4.1.2 ai_addrlen `socklen_t addrinfo::ai_addrlen`

#### 3.4.1.3 ai_canonname `char* addrinfo::ai_canonname`

#### 3.4.1.4 ai_family `int addrinfo::ai_family`

#### 3.4.1.5 ai_flags `int addrinfo::ai_flags`

#### 3.4.1.6 ai_next `struct addrinfo* addrinfo::ai_next`

#### 3.4.1.7 ai_protocol `int addrinfo::ai_protocol`

#### 3.4.1.8 ai_socktype `int addrinfo::ai_socktype`

The documentation for this struct was generated from the following file:

- ta-ref/api/include/tee_api_types.h

## 3.5 enclave_report Struct Reference

`#include <report.h>`

### Public Attributes

- uint8_t hash [MDSIZE]
- uint64_t data_len
- uint8_t data [ATTEST_DATA_MAXLEN]
- uint8_t signature [SIGNATURE_SIZE]

### 3.5.1 Member Data Documentation

#### 3.5.1.1 data `uint8_t enclave_report::data[ATTEST_DATA_MAXLEN]`

#### 3.5.1.2 data_len `uint64_t enclave_report::data_len`

#### 3.5.1.3 hash `uint8_t enclave_report::hash[MDSIZE]`

#### 3.5.1.4 signature `uint8_t enclave_report::signature[SIGNATURE_SIZE]`

The documentation for this struct was generated from the following file:

- ta-ref/api/include/report.h

## 3.6 out_fct_wrap_type Struct Reference

### Public Attributes

- void(∗ fct )(char character, void ∗arg)
- void ∗ arg

### 3.6.1 Member Data Documentation

#### 3.6.1.1 arg `void * out_fct_wrap_type::arg`

#### 3.6.1.2 fct `void(* out_fct_wrap_type::fct)(char character, void *arg)`

The documentation for this struct was generated from the following files:

- ta-ref/api/keystone/vsnprintf.c
- ta-ref/api/sgx/vsnprintf.c

## 3.7 pollfd Struct Reference

`#include <tee_api_types.h>`

### Public Attributes

- int fd
- short int events
- short int revents

### 3.7.1 Member Data Documentation

#### 3.7.1.1 events `short int pollfd::events`

#### 3.7.1.2 fd `int pollfd::fd`

#### 3.7.1.3 revents `short int pollfd::revents`

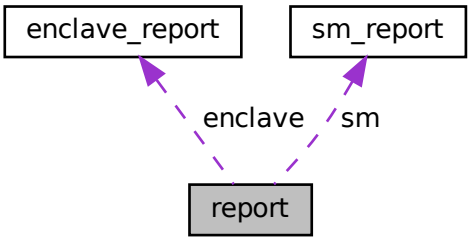
The documentation for this struct was generated from the following file:

- ta-ref/api/include/tee_api_types.h

## 3.8 report Struct Reference

`#include <report.h>`

Collaboration diagram for report:




### Public Attributes

- struct enclave_report enclave
- struct sm_report sm
- uint8_t dev_public_key [PUBLIC_KEY_SIZE]

### 3.8.1 Member Data Documentation

#### 3.8.1.1 dev_public_key `uint8_t report::dev_public_key[PUBLIC_KEY_SIZE]`

#### 3.8.1.2 enclave `struct enclave_report report::enclave`

#### 3.8.1.3 sm `struct sm_report report::sm`

The documentation for this struct was generated from the following file:

- ta-ref/api/include/report.h

## 3.9 sm_report Struct Reference

`#include <report.h>`

#### Public Attributes

- uint8_t hash [MDSIZE]
- uint8_t public_key [PUBLIC_KEY_SIZE]
- uint8_t signature [SIGNATURE_SIZE]

### 3.9.1 Member Data Documentation

#### 3.9.1.1 hash `uint8_t sm_report::hash[MDSIZE]`

#### 3.9.1.2 public_key `uint8_t sm_report::public_key[PUBLIC_KEY_SIZE]`

#### 3.9.1.3 signature `uint8_t sm_report::signature[SIGNATURE_SIZE]`

The documentation for this struct was generated from the following file:

- ta-ref/api/include/report.h

## 3.10 TEE_Attribute Struct Reference

`#include <tee_api_types.h>`

#### Public Attributes

- uint32_t attributeID
- union {
  struct {
  void * buffer
  uint32_t length
  } ref
  struct {
  uint32_t a
  uint32_t b
  } value
  } content

### 3.10.1 Member Data Documentation

#### 3.10.1.1 a `uint32_t TEE_Attribute::a`

#### 3.10.1.2 attributeID `uint32_t TEE_Attribute::attributeID`

#### 3.10.1.3 b `uint32_t TEE_Attribute::b`

#### 3.10.1.4 buffer `void* TEE_Attribute::buffer`

#### 3.10.1.5 `union { ... } TEE_Attribute::content`

#### 3.10.1.6 length `uint32_t TEE_Attribute::length`

#### 3.10.1.7 `struct { ... } TEE_Attribute::ref`

#### 3.10.1.8 `struct { ... } TEE_Attribute::value`

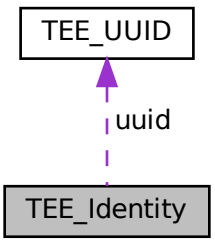
The documentation for this struct was generated from the following file:

- ta-ref/api/include/tee_api_types.h

## 3.11 TEE_Identity Struct Reference

`#include <tee_api_types.h>`

Collaboration diagram for TEE_Identity:



#### Public Attributes

- uint32_t login
- TEE_UUID uuid

### 3.11.1 Member Data Documentation

#### 3.11.1.1 login `uint32_t TEE_Identity::login`

#### 3.11.1.2 uuid `TEE_UUID TEE_Identity::uuid`

The documentation for this struct was generated from the following file:

- ta-ref/api/include/tee_api_types.h

## 3.12 TEE_ObjectInfo Struct Reference

```
#include <tee_api_types.h>
```

#### Public Attributes

- uint32_t objectType
- union {
  uint32_t keySize
  uint32_t objectSize
  };

- union {
  uint32_t maxKeySize
  uint32_t maxObjectSize
  };

- uint32_t objectUsage
- uint32_t dataSize
- uint32_t dataPosition
- uint32_t handleFlags

### 3.12.1 Member Data Documentation

#### 3.12.1.1 `__extension__ union { ... } TEE_ObjectInfo::@3`

#### 3.12.1.2 __extension__ union { ... } TEE_ObjectInfo::@5

#### 3.12.1.3 dataPosition uint32_t TEE_ObjectInfo::dataPosition

#### 3.12.1.4 dataSize uint32_t TEE_ObjectInfo::dataSize

#### 3.12.1.5 handleFlags uint32_t TEE_ObjectInfo::handleFlags

#### 3.12.1.6 keySize uint32_t TEE_ObjectInfo::keySize

#### 3.12.1.7 maxKeySize uint32_t TEE_ObjectInfo::maxKeySize

#### 3.12.1.8 maxObjectSize uint32_t TEE_ObjectInfo::maxObjectSize

#### 3.12.1.9 objectSize uint32_t TEE_ObjectInfo::objectSize

#### 3.12.1.10 objectType uint32_t TEE_ObjectInfo::objectType

#### 3.12.1.11 objectUsage uint32_t TEE_ObjectInfo::objectUsage

The documentation for this struct was generated from the following file:

- ta-ref/api/include/tee_api_types.h

## 3.13 TEE_OperationInfo Struct Reference

`#include <tee_api_types.h>`

### Public Attributes

- uint32_t algorithm
- uint32_t operationClass
- uint32_t mode
- uint32_t digestLength
- uint32_t maxKeySize
- uint32_t keySize
- uint32_t requiredKeyUsage
- uint32_t handleState

#### 3.13.1 Member Data Documentation

##### 3.13.1.1 algorithm `uint32_t TEE_OperationInfo::algorithm`

##### 3.13.1.2 digestLength `uint32_t TEE_OperationInfo::digestLength`

##### 3.13.1.3 handleState `uint32_t TEE_OperationInfo::handleState`

##### 3.13.1.4 keySize `uint32_t TEE_OperationInfo::keySize`

##### 3.13.1.5 maxKeySize `uint32_t TEE_OperationInfo::maxKeySize`

##### 3.13.1.6 mode `uint32_t TEE_OperationInfo::mode`

##### 3.13.1.7 operationClass `uint32_t TEE_OperationInfo::operationClass`

#### 3.13.1.8 requiredKeyUsage `uint32_t TEE_OperationInfo::requiredKeyUsage`

The documentation for this struct was generated from the following file:

- ta-ref/api/include/tee_api_types.h

## 3.14 TEE_OperationInfoKey Struct Reference

`#include <tee_api_types.h>`

### Public Attributes

- uint32_t keySize
- uint32_t requiredKeyUsage

### 3.14.1 Member Data Documentation

#### 3.14.1.1 keySize `uint32_t TEE_OperationInfoKey::keySize`

#### 3.14.1.2 requiredKeyUsage `uint32_t TEE_OperationInfoKey::requiredKeyUsage`

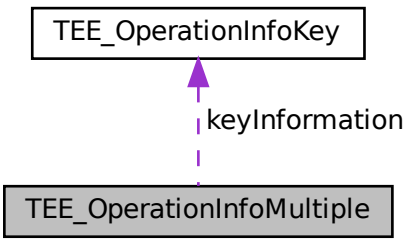
The documentation for this struct was generated from the following file:

- ta-ref/api/include/tee_api_types.h

## 3.15 TEE_OperationInfoMultiple Struct Reference

`#include <tee_api_types.h>`

Collaboration diagram for TEE_OperationInfoMultiple:



### Public Attributes

- uint32_t algorithm
- uint32_t operationClass
- uint32_t mode
- uint32_t digestLength
- uint32_t maxKeySize
- uint32_t handleState
- uint32_t operationState
- uint32_t numberOfKeys
- TEE_OperationInfoKey keyInformation [ ]

## 3.15.1 Member Data Documentation

### 3.15.1.1 algorithm `uint32_t TEE_OperationInfoMultiple::algorithm`

### 3.15.1.2 digestLength `uint32_t TEE_OperationInfoMultiple::digestLength`

### 3.15.1.3 handleState `uint32_t TEE_OperationInfoMultiple::handleState`

### 3.15.1.4 keyInformation `TEE_OperationInfoKey TEE_OperationInfoMultiple::keyInformation[ ]`

### 3.15.1.5 maxKeySize `uint32_t TEE_OperationInfoMultiple::maxKeySize`

### 3.15.1.6 mode `uint32_t TEE_OperationInfoMultiple::mode`

### 3.15.1.7 numberOfKeys `uint32_t TEE_OperationInfoMultiple::numberOfKeys`

### 3.15.1.8 operationClass `uint32_t TEE_OperationInfoMultiple::operationClass`

#### 3.15.1.9 operationState `uint32_t TEE_OperationInfoMultiple::operationState`

The documentation for this struct was generated from the following file:

- ta-ref/api/include/tee_api_types.h

## 3.16 TEE_Param Union Reference

`#include <tee_api_types.h>`

### Public Attributes

- struct {
  void * buffer
  uint32_t size
  } memref

- struct {
  uint32_t a
  uint32_t b
  } value

### 3.16.1 Member Data Documentation

#### 3.16.1.1 a `uint32_t TEE_Param::a`

#### 3.16.1.2 b `uint32_t TEE_Param::b`

#### 3.16.1.3 buffer `void* TEE_Param::buffer`

#### 3.16.1.4 `struct { ... } TEE_Param::memref`

#### 3.16.1.5 size `uint32_t TEE_Param::size`

#### 3.16.1.6 struct { ... } TEE_Param::value

The documentation for this union was generated from the following file:

- ta-ref/api/include/tee_api_types.h

## 3.17 TEE_SEAID Struct Reference

`#include <tee_api_types.h>`

### Public Attributes

- uint8_t * buffer
- size_t bufferLen

### 3.17.1 Member Data Documentation

#### 3.17.1.1 buffer uint8_t* TEE_SEAID::buffer

#### 3.17.1.2 bufferLen size_t TEE_SEAID::bufferLen

The documentation for this struct was generated from the following file:

- ta-ref/api/include/tee_api_types.h

## 3.18 TEE_SEReaderProperties Struct Reference

`#include <tee_api_types.h>`

### Public Attributes

- bool sePresent
- bool teeOnly
- bool selectResponseEnable

### 3.18.1 Member Data Documentation

#### 3.18.1.1 selectResponseEnable `bool TEE_SEReaderProperties::selectResponseEnable`

#### 3.18.1.2 sePresent `bool TEE_SEReaderProperties::sePresent`

#### 3.18.1.3 teeOnly `bool TEE_SEReaderProperties::teeOnly`

The documentation for this struct was generated from the following file:

- ta-ref/api/include/tee_api_types.h

## 3.19 TEE_Time Struct Reference

`#include <tee_api_types.h>`

### Public Attributes

- uint32_t seconds
- uint32_t millis

### 3.19.1 Member Data Documentation

#### 3.19.1.1 millis `uint32_t TEE_Time::millis`

#### 3.19.1.2 seconds `uint32_t TEE_Time::seconds`

The documentation for this struct was generated from the following file:

- ta-ref/api/include/tee_api_types.h

## 3.20 TEE_UUID Struct Reference

`#include <tee_api_types.h>`

**Public Attributes**

- uint32_t timeLow
- uint16_t timeMid
- uint16_t timeHiAndVersion
- uint8_t clockSeqAndNode [8]

### 3.20.1 Member Data Documentation

#### 3.20.1.1 clockSeqAndNode `uint8_t TEE_UUID::clockSeqAndNode[8]`

#### 3.20.1.2 timeHiAndVersion `uint16_t TEE_UUID::timeHiAndVersion`

#### 3.20.1.3 timeLow `uint32_t TEE_UUID::timeLow`

#### 3.20.1.4 timeMid `uint16_t TEE_UUID::timeMid`

The documentation for this struct was generated from the following file:

- ta-ref/api/include/tee_api_types.h

## 3.21 TEEC_Context Struct Reference

`#include <tee_client_api.h>`

**Public Attributes**

- int fd
- bool reg_mem

### 3.21.1 Detailed Description

struct TEEC_Context - Represents a connection between a client application and a TEE.

### 3.21.2 Member Data Documentation

#### 3.21.2.1 fd `int TEEC_Context::fd`

#### 3.21.2.2 reg_mem `bool TEEC_Context::reg_mem`

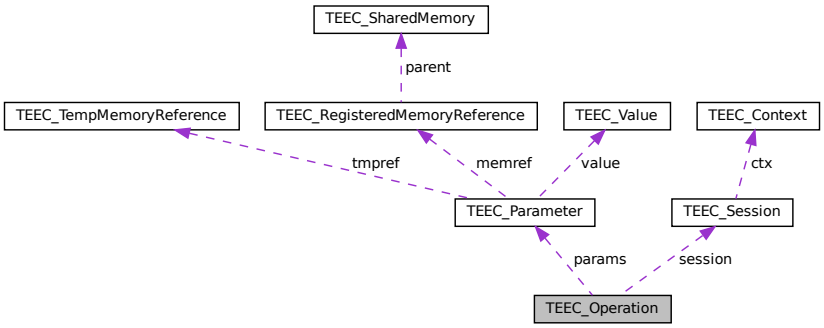
The documentation for this struct was generated from the following file:

- ta-ref/api/include/tee_client_api.h

## 3.22 TEEC_Operation Struct Reference

`#include <tee_client_api.h>`

Collaboration diagram for TEEC_Operation:




### Public Attributes

- uint32_t started
- uint32_t paramTypes
- TEEC_Parameter params [TEEC_CONFIG_PAYLOAD_REF_COUNT]
- TEEC_Session * session

### 3.22.1 Detailed Description

struct TEEC_Operation - Holds information and memory references used in TEEC_InvokeCommand().

**Parameters**

| | |
|---|---|
| *started* | Client must initialize to zero if it needs to cancel an operation about to be performed. |
| *paramTypes* | Type of data passed. Use TEEC_PARAMS_TYPE macro to create the correct flags. 0 means TEEC_NONE is passed for all params. |
| *params* | Array of parameters of type TEEC_Parameter. |
| *session* | Internal pointer to the last session used by TEEC_InvokeCommand with this operation. |

### 3.22.2 Member Data Documentation

#### 3.22.2.1 params `TEEC_Parameter TEEC_Operation::params[TEEC_CONFIG_PAYLOAD_REF_COUNT]`

#### 3.22.2.2 paramTypes `uint32_t TEEC_Operation::paramTypes`

#### 3.22.2.3 session `TEEC_Session* TEEC_Operation::session`

#### 3.22.2.4 started `uint32_t TEEC_Operation::started`

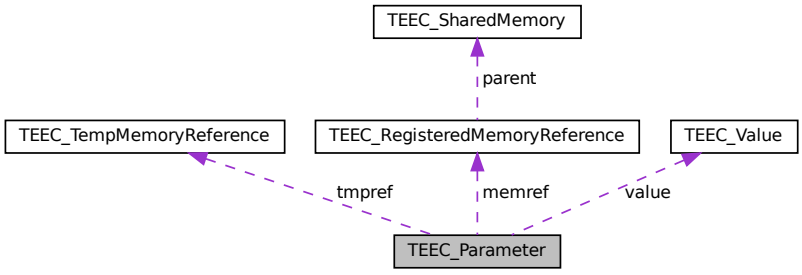
The documentation for this struct was generated from the following file:

- ta-ref/api/include/tee_client_api.h

## 3.23 TEEC_Parameter Union Reference

`#include <tee_client_api.h>`

Collaboration diagram for TEEC_Parameter:




### Public Attributes

- TEEC_TempMemoryReference tmpref
- TEEC_RegisteredMemoryReference memref
- TEEC_Value value

### 3.23.1 Detailed Description

union TEEC_Parameter - Memory container to be used when passing data between client application and trusted code.

Either the client uses a shared memory reference, parts of it or a small raw data container.

**Parameters**

| | |
|---|---|
| *tmpref* | A temporary memory reference only valid for the duration of the operation. |
| *memref* | The entire shared memory or parts of it. |
| *value* | The small raw data container to use |

### 3.23.2 Member Data Documentation

#### 3.23.2.1 memref `TEEC_RegisteredMemoryReference TEEC_Parameter::memref`

#### 3.23.2.2 tmpref `TEEC_TempMemoryReference TEEC_Parameter::tmpref`

#### 3.23.2.3 value `TEEC_Value TEEC_Parameter::value`

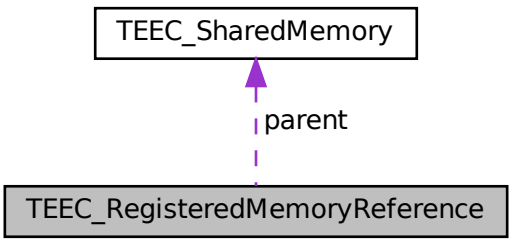
The documentation for this union was generated from the following file:

- ta-ref/api/include/tee_client_api.h

## 3.24 TEEC_RegisteredMemoryReference Struct Reference

```
#include <tee_client_api.h>
```

Collaboration diagram for TEEC_RegisteredMemoryReference:



### Public Attributes

- TEEC_SharedMemory * parent
- size_t size
- size_t offset

### 3.24.1 Detailed Description

struct TEEC_RegisteredMemoryReference - use a pre-registered or pre-allocated shared memory block of memory to transfer data between a client application and trusted code.

**Parameters**

| | |
|---|---|
| *parent* | Points to a shared memory structure. The memory reference may utilize the whole shared memory or only a part of it. Must not be NULL |
| *size* | The size, in bytes, of the memory buffer. |
| *offset* | The offset, in bytes, of the referenced memory region from the start of the shared memory block. |

### 3.24.2 Member Data Documentation

#### 3.24.2.1 offset `size_t TEEC_RegisteredMemoryReference::offset`

#### 3.24.2.2 parent `TEEC_SharedMemory* TEEC_RegisteredMemoryReference::parent`

#### 3.24.2.3 size `size_t TEEC_RegisteredMemoryReference::size`

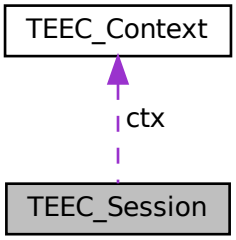
The documentation for this struct was generated from the following file:

- ta-ref/api/include/tee_client_api.h

## 3.25 TEEC_Session Struct Reference

`#include <tee_client_api.h>`

Collaboration diagram for TEEC_Session:




### Public Attributes

- TEEC_Context * ctx
- uint32_t session_id

### 3.25.1 Detailed Description

struct TEEC_Session - Represents a connection between a client application and a trusted application.

### 3.25.2 Member Data Documentation

#### 3.25.2.1 ctx `TEEC_Context* TEEC_Session::ctx`

#### 3.25.2.2 session_id `uint32_t TEEC_Session::session_id`

The documentation for this struct was generated from the following file:

- ta-ref/api/include/tee_client_api.h

## 3.26 TEEC_SharedMemory Struct Reference

`#include <tee_client_api.h>`

### Public Attributes

- void * buffer
- size_t size
- uint32_t flags
- int id
- size_t alloced_size
- void * shadow_buffer
- int registered_fd
- bool buffer_allocated

#### 3.26.1 Detailed Description

struct TEEC_SharedMemory - Memory to transfer data between a client application and trusted code.

**Parameters**

| | |
|---|---|
| *buffer* | The memory buffer which is to be, or has been, shared with the TEE. |
| *size* | The size, in bytes, of the memory buffer. |
| *flags* | Bit-vector which holds properties of buffer. The bit-vector can contain either or both of the TEEC_MEM_INPUT and TEEC_MEM_OUTPUT flags. |

A shared memory block is a region of memory allocated in the context of the client application memory space that can be used to transfer data between that client application and a trusted application. The user of this struct is responsible to populate the buffer pointer.

#### 3.26.2 Member Data Documentation

##### 3.26.2.1 alloced_size `size_t TEEC_SharedMemory::alloced_size`

##### 3.26.2.2 buffer `void* TEEC_SharedMemory::buffer`

##### 3.26.2.3 buffer_allocated `bool TEEC_SharedMemory::buffer_allocated`

##### 3.26.2.4 flags `uint32_t TEEC_SharedMemory::flags`

#### 3.26.2.5 id `int TEEC_SharedMemory::id`

#### 3.26.2.6 registered_fd `int TEEC_SharedMemory::registered_fd`

#### 3.26.2.7 shadow_buffer `void* TEEC_SharedMemory::shadow_buffer`

#### 3.26.2.8 size `size_t TEEC_SharedMemory::size`

The documentation for this struct was generated from the following file:

- ta-ref/api/include/tee_client_api.h

## 3.27 TEEC_TempMemoryReference Struct Reference

`#include <tee_client_api.h>`

### Public Attributes

- void * buffer
- size_t size

### 3.27.1 Detailed Description

struct TEEC_TempMemoryReference - Temporary memory to transfer data between a client application and trusted code, only used for the duration of the operation.

**Parameters**

| | |
|---|---|
| *buffer* | The memory buffer which is to be, or has been shared with the TEE. |
| *size* | The size, in bytes, of the memory buffer. |

A memory buffer that is registered temporarily for the duration of the operation to be called.

### 3.27.2 Member Data Documentation

#### 3.27.2.1 buffer `void* TEEC_TempMemoryReference::buffer`

#### 3.27.2.2 size `size_t TEEC_TempMemoryReference::size`

The documentation for this struct was generated from the following file:

- ta-ref/api/include/tee_client_api.h

## 3.28 TEEC_UUID Struct Reference

`#include <tee_client_api.h>`

### Public Attributes

- uint32_t timeLow
- uint16_t timeMid
- uint16_t timeHiAndVersion
- uint8_t clockSeqAndNode [8]

### 3.28.1 Detailed Description

This type contains a Universally Unique Resource Identifier (UUID) type as defined in RFC4122. These UUID values are used to identify Trusted Applications.

### 3.28.2 Member Data Documentation

#### 3.28.2.1 clockSeqAndNode `uint8_t TEEC_UUID::clockSeqAndNode[8]`

#### 3.28.2.2 timeHiAndVersion `uint16_t TEEC_UUID::timeHiAndVersion`

#### 3.28.2.3 timeLow `uint32_t TEEC_UUID::timeLow`

#### 3.28.2.4 timeMid `uint16_t TEEC_UUID::timeMid`

The documentation for this struct was generated from the following file:

- ta-ref/api/include/tee_client_api.h

## 3.29 TEEC_Value Struct Reference

`#include <tee_client_api.h>`

### Public Attributes

- uint32_t a
- uint32_t b

### 3.29.1 Detailed Description

struct TEEC_Value - Small raw data container

Instead of allocating a shared memory buffer this structure can be used to pass small raw data between a client application and trusted code.

**Parameters**

| | |
|---|---|
| *a* | The first integer value. |
| *b* | The second second value. |

### 3.29.2 Member Data Documentation

#### 3.29.2.1 a `uint32_t TEEC_Value::a`

#### 3.29.2.2 b `uint32_t TEEC_Value::b`

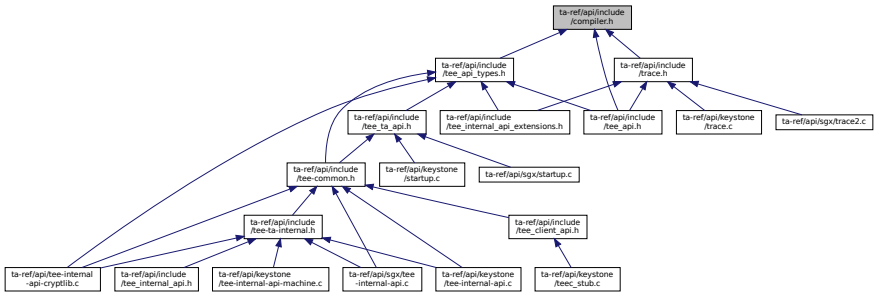
The documentation for this struct was generated from the following file:

- ta-ref/api/include/tee_client_api.h

# 4 File Documentation

## 4.1 ta-ref/api/include/compiler.h File Reference

This graph shows which files directly or indirectly include this file:




## 4.2 compiler.h

Go to the documentation of this file.

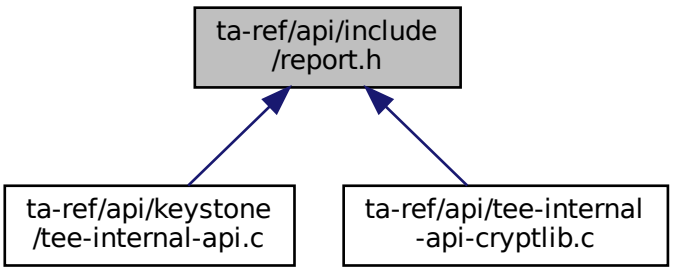
```
1 /*
2  * Copyright (c) 2014, STMicroelectronics International N.V.
3  * All rights reserved.
4  *
5  * Redistribution and use in source and binary forms, with or without
6  * modification, are permitted provided that the following conditions are met:
7  *
8  * 1. Redistributions of source code must retain the above copyright notice,
9  * this list of conditions and the following disclaimer.
10 *
11 * 2. Redistributions in binary form must reproduce the above copyright notice,
12 * this list of conditions and the following disclaimer in the documentation
13 * and/or other materials provided with the distribution.
14 *
15 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
16 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
17 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
18 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
19 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
20 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
21 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
22 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
23 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
24 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
25 * POSSIBILITY OF SUCH DAMAGE.
26 */
27
28 #ifndef COMPILER_H
29 #define COMPILER_H
30
31 #ifndef DOXYGEN_SHOULD_SKIP_THIS
32 /*
33  * Macros that should be used instead of using __attribute__ directly to
34  * ease portability and make the code easier to read.
35  */
36
37 #define __deprecated    __attribute__((deprecated))
38 #define __packed    __attribute__((packed))
39 #define __weak      __attribute__((weak))
40 #define __noreturn  __attribute__((noreturn))
41 #define __pure      __attribute__((pure))
42 #define __aligned(x)    __attribute__((aligned(x)))
43 #define __printf(a, b)  __attribute__((format(printf, a, b)))
44 #define __noinline  __attribute__((noinline))
45 #define __attr_const    __attribute__((__const__))
46 #define __unused    __attribute__((unused))
47 #define __maybe_unused  __attribute__((unused))
```

```
48 #define __used      __attribute__((__used__))
49 #define __must_check    __attribute__((warn_unused_result))
50 #define __cold      __attribute__((__cold__))
51 #define __section(x)    __attribute__((section(x)))
52 #define __data      __section(".data")
53 #define __bss       __section(".bss")
54 #define __rodata    __section(".rodata")
55 #define __rodata_unpaged __section(".rodata.__unpaged")
56 #define __early_ta  __section(".rodata.early_ta")
57 #define __noprof    __attribute__((no_instrument_function))
58
59 #define __compiler_bswap64(x)   __builtin_bswap64((x))
60 #define __compiler_bswap32(x)   __builtin_bswap32((x))
61 #define __compiler_bswap16(x)   __builtin_bswap16((x))
62
63 #define __GCC_VERSION (__GNUC__ * 10000 + __GNUC_MINOR__ * 100 + \
64                __GNUC_PATCHLEVEL__)
65
66 #if __GCC_VERSION >= 50100 && !defined(__CHECKER__)
67 #define __HAVE_BUILTIN_OVERFLOW 1
68 #endif
69
70 #ifdef __HAVE_BUILTIN_OVERFLOW
71 #define __compiler_add_overflow(a, b, res) \
72     __builtin_add_overflow((a), (b), (res))
73
74 #define __compiler_sub_overflow(a, b, res) \
75     __builtin_sub_overflow((a), (b), (res))
76
77 #define __compiler_mul_overflow(a, b, res) \
78     __builtin_mul_overflow((a), (b), (res))
79 #else
80 /*
81  * Copied/inspired from https://www.fefe.de/intof.html
82  */
83 #define __INTOF_HALF_MAX_SIGNED(type) ((type)1 « (sizeof(type)*8-2))
84 #define __INTOF_MAX_SIGNED(type) (__INTOF_HALF_MAX_SIGNED(type) - 1 + \
85                 __INTOF_HALF_MAX_SIGNED(type))
86 #define __INTOF_MIN_SIGNED(type) (-1 - __INTOF_MAX_SIGNED(type))
87
88 #define __INTOF_MIN(type) ((type)-1 < 1?__INTOF_MIN_SIGNED(type):(type)0)
89 #define __INTOF_MAX(type) ((type)~__INTOF_MIN(type))
90
91 #define __INTOF_ASSIGN(dest, src) (__extension__({ \
92     typeof(src) __intof_x = (src); \
93     typeof(dest) __intof_y = __intof_x; \
94     (((uintmax_t)__intof_x == (uintmax_t)__intof_y) && \
95      ((__intof_x < 1) == (__intof_y < 1)) ? \
96         (void)((dest) = __intof_y) , 0 : 1); \
97 }))
98
99 #define __INTOF_ADD(c, a, b) (__extension__({ \
100     typeof(a) __intofa_a = (a); \
101     typeof(b) __intofa_b = (b); \
102     \
103     __intofa_b < 1 ? \
104         ((__INTOF_MIN(typeof(c)) - __intofa_b <= __intofa_a) ? \
105             __INTOF_ASSIGN((c), __intofa_a + __intofa_b) : 1) : \
106         ((__INTOF_MAX(typeof(c)) - __intofa_b >= __intofa_a) ? \
107             __INTOF_ASSIGN((c), __intofa_a + __intofa_b) : 1); \
108 }))
109
110 #define __INTOF_SUB(c, a, b) (__extension__({ \
111     typeof(a) __intofs_a = a; \
112     typeof(b) __intofs_b = b; \
113     \
114     __intofs_b < 1 ? \
115         ((__INTOF_MAX(typeof(c)) + __intofs_b >= __intofs_a) ? \
116             __INTOF_ASSIGN((c), __intofs_a - __intofs_b) : 1) : \
117         ((__INTOF_MIN(typeof(c)) + __intofs_b <= __intofs_a) ? \
118             __INTOF_ASSIGN((c), __intofs_a - __intofs_b) : 1); \
119 }))
120
121 /*
122  * Dealing with detecting overflow in multiplication of integers.
123  *
124  * First step is to remove two corner cases with the minum signed integer
125  * which can't be represented as a positive integer + sign.
126  * Multiply with 0 or 1 can't overflow, no checking needed of the operation,
127  * only if it can be assigned to the result.
128  *
129  * After the corner cases are eliminated we convert the two factors to
130  * positive unsigned values, keeping track of the original in another
131  * variable which is used at the end to determine the sign of the product.
132  *
133  *
```

```
 * The two terms (a and b) are divided into upper and lower half (x1 upper
 * and x0 lower), so the product is:
 * ((a1 « hshift) + a0) * ((b1 « hshift) + b0)
 * which also is:
 * ((a1 * b1) « (hshift * 2)) +                (T1)
 * ((a1 * b0 + a0 * b1) « hshift) +                (T2)
 * (a0 * b0)                                (T3)
 *
 * From this we can tell and (a1 * b1) has to be 0 or we'll overflow, that
 * is, at least one of a1 or b1 has to be 0. Once this has been checked the
 * addition: ((a1 * b0) « hshift) + ((a0 * b1) « hshift)
 * isn't an addition as one of the terms will be 0.
 *
 * Since each factor in: (a0 * b0)
 * only uses half the capicity of the underlaying type it can't overflow
 *
 * The addition of T2 and T3 can overflow so we use __INTOF_ADD() to
 * perform that addition. If the addition succeeds without overflow the
 * result is assigned the required sign and checked for overflow again.
 */

#define __intof_mul_negate  ((__intof_oa < 1) != (__intof_ob < 1))
#define __intof_mul_hshift  (sizeof(uintmax_t) * 8 / 2)
#define __intof_mul_hmask   (UINTMAX_MAX » __intof_mul_hshift)
#define __intof_mul_a0      ((uintmax_t)(__intof_a) » __intof_mul_hshift)
#define __intof_mul_b0      ((uintmax_t)(__intof_b) » __intof_mul_hshift)
#define __intof_mul_a1      ((uintmax_t)(__intof_a) & __intof_mul_hmask)
#define __intof_mul_b1      ((uintmax_t)(__intof_b) & __intof_mul_hmask)
#define __intof_mul_t       (__intof_mul_a1 * __intof_mul_b0 + \
                __intof_mul_a0 * __intof_mul_b1)

#define __INTOF_MUL(c, a, b) (__extension__({ \
    typeof(a) __intof_oa = (a); \
    typeof(a) __intof_a = __intof_oa < 1 ? -__intof_oa : __intof_oa; \
    typeof(b) __intof_ob = (b); \
    typeof(b) __intof_b = __intof_ob < 1 ? -__intof_ob : __intof_ob; \
    typeof(c) __intof_c; \
    \
    __intof_oa == 0 || __intof_ob == 0 || \
    __intof_oa == 1 || __intof_ob == 1 ? \
        __INTOF_ASSIGN((c), __intof_oa * __intof_ob) : \
    (__intof_mul_a0 && __intof_mul_b0) || \
     __intof_mul_t > __intof_mul_hmask ?  1 : \
    __INTOF_ADD((__intof_c), __intof_mul_t « __intof_mul_hshift, \
                __intof_mul_a1 * __intof_mul_b1) ? 1 : \
    __intof_mul_negate ? __INTOF_ASSIGN((c), -__intof_c) : \
                __INTOF_ASSIGN((c), __intof_c); \
}))

#define __compiler_add_overflow(a, b, res) __INTOF_ADD(*(res), (a), (b))
#define __compiler_sub_overflow(a, b, res) __INTOF_SUB(*(res), (a), (b))
#define __compiler_mul_overflow(a, b, res) __INTOF_MUL(*(res), (a), (b))

#endif
#define __compiler_compare_and_swap(p, oval, nval) \
    __atomic_compare_exchange_n((p), (oval), (nval), true, \
                    __ATOMIC_ACQUIRE, __ATOMIC_RELAXED) \

#define __compiler_atomic_load(p) __atomic_load_n((p), __ATOMIC_RELAXED)
#define __compiler_atomic_store(p, val) \
    __atomic_store_n((p), (val), __ATOMIC_RELAXED)

#endif /*DOXYGEN_SHOULD_SKIP_THIS*/
#endif /*COMPILER_H*/
```

## 4.3 ta-ref/api/include/report.h File Reference

This graph shows which files directly or indirectly include this file:




### Classes

- struct enclave_report
- struct sm_report
- struct report

## 4.4 report.h

Go to the documentation of this file.

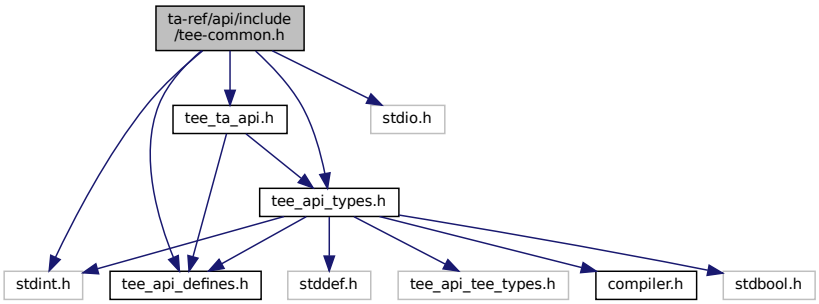
```
1
2 #ifndef _REPORT_H
3 #define _REPORT_H
4
5 #ifndef DOXYGEN_SHOULD_SKIP_THIS
6 #define MDSIZE  64
7 #define SIGNATURE_SIZE  64
8 #define PUBLIC_KEY_SIZE 32
9 #define ATTEST_DATA_MAXLEN  1024
10 #endif /*DOXYGEN_SHOULD_SKIP_THIS*/
11
12 /* attestation reports */
13 struct enclave_report
14 {
15   uint8_t hash[MDSIZE];
16   uint64_t data_len;
17   uint8_t data[ATTEST_DATA_MAXLEN];
18   uint8_t signature[SIGNATURE_SIZE];
19 };
20
21 struct sm_report
22 {
23   uint8_t hash[MDSIZE];
24   uint8_t public_key[PUBLIC_KEY_SIZE];
25   uint8_t signature[SIGNATURE_SIZE];
26 };
27
28 struct report
29 {
30   struct enclave_report enclave;
31   struct sm_report sm;
32   uint8_t dev_public_key[PUBLIC_KEY_SIZE];
33 };
34
35 #endif // _REPORT_H
```

## 4.5 ta-ref/api/include/tee-common.h File Reference

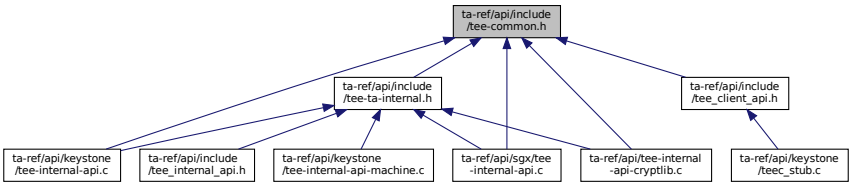
Common type and definitions of RISC-V TEE.

```
#include <stdint.h>
#include <stdio.h>
#include <tee_api_defines.h>
#include <tee_api_types.h>
#include <tee_ta_api.h>
```

Include dependency graph for tee-common.h:




This graph shows which files directly or indirectly include this file:




### 4.5.1 Detailed Description

Common type and definitions of RISC-V TEE.

draft RISC-V Internal TEE API

**Author**

Akira Tsukamoto, AIST

**Date**

2019/09/25

## 4.6 tee-common.h

Go to the documentation of this file.

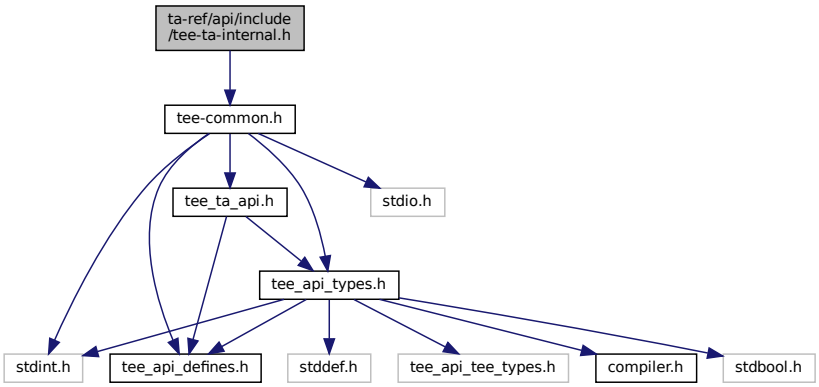
```c
/*
 * SPDX-License-Identifier: BSD-2-Clause
 *
 * Copyright (C) 2019 National Institute of Advanced Industrial Science
 *                          and Technology (AIST)
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 * this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 * this list of conditions and the following disclaimer in the documentation
 * and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */
#ifndef TEE_COMMON_H
#define TEE_COMMON_H

#include <stdint.h>
#include <stdio.h>

#ifdef __cplusplus
extern "C" {
#endif

#ifndef DOXYGEN_SHOULD_SKIP_THIS
#ifdef DEBUG
#define pr_deb(...)       do { printf(__VA_ARGS__); } while (0)
#else
#define pr_deb(...)       do { } while (0)
#endif /* DEBUG */
#endif /*DOXYGEN_SHOULD_SKIP_THIS*/

//#include <tee_api.h>
#include <tee_api_defines.h>
#include <tee_api_types.h>
#include <tee_ta_api.h>

//typedef uint32_t TEE_Result;

#ifdef __cplusplus
}
#endif

#endif /* TEE_COMMON_H */
```

## 4.7 ta-ref/api/include/tee-ta-internal.h File Reference

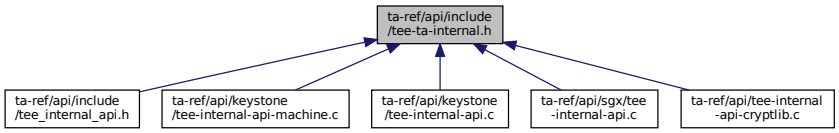
Candidate API list for Global Platform like RISC-V TEE.

```
#include "tee-common.h"
```

Include dependency graph for tee-ta-internal.h:




This graph shows which files directly or indirectly include this file:




### Functions

- void __attribute__ ((noreturn)) TEE_Panic(unsigned long code)
- void TEE_GetREETime (TEE_Time *time)

  *Core Functions, Time Functions.*
- void TEE_GetSystemTime (TEE_Time *time)

  *Core Functions, Time Functions.*
- TEE_Result GetRelTimeStart (uint64_t start)

  *Core Functions, Time Functions.*
- TEE_Result GetRelTimeEnd (uint64_t end)

  *Core Functions, Time Functions.*
- TEE_Result TEE_CreatePersistentObject (uint32_t storageID, const void *objectID, uint32_t objectIDLen, uint32_t flags, TEE_ObjectHandle attributes, const void *initialData, uint32_t initialDataLen, TEE_ObjectHandle *object)

  *Core Functions, Secure Storage Functions (data is isolated for each TA)*
- TEE_Result TEE_OpenPersistentObject (uint32_t storageID, const void *objectID, uint32_t objectIDLen, uint32_t flags, TEE_ObjectHandle *object)

  *Core Functions, Secure Storage Functions (data is isolated for each TA)*
- TEE_Result TEE_GetObjectInfo1 (TEE_ObjectHandle object, TEE_ObjectInfo *objectInfo)

  *Core Functions, Secure Storage Functions (data is isolated for each TA)*
- TEE_Result TEE_WriteObjectData (TEE_ObjectHandle object, const void *buffer, uint32_t size)

  *Core Functions, Secure Storage Functions (data is isolated for each TA)*

- TEE_Result TEE_ReadObjectData (TEE_ObjectHandle object, void *buffer, uint32_t size, uint32_t *count)

  *Core Functions, Secure Storage Functions (data is isolated for each TA)*
- void TEE_CloseObject (TEE_ObjectHandle object)

  *Core Functions, Secure Storage Functions (data is isolated for each TA)*
- void TEE_GenerateRandom (void *randomBuffer, uint32_t randomBufferLen)

  *Crypto, common.*
- TEE_Result TEE_AllocateOperation (TEE_OperationHandle *operation, uint32_t algorithm, uint32_t mode, uint32_t maxKeySize)

  *Crypto, for all Crypto Functions.*
- void TEE_FreeOperation (TEE_OperationHandle operation)

  *Crypto, for all Crypto Functions.*
- void TEE_DigestUpdate (TEE_OperationHandle operation, const void *chunk, uint32_t chunkSize)

  *Crypto, Message Digest Functions.*
- TEE_Result TEE_DigestDoFinal (TEE_OperationHandle operation, const void *chunk, uint32_t chunkLen, void *hash, uint32_t *hashLen)
- TEE_Result TEE_SetOperationKey (TEE_OperationHandle operation, TEE_ObjectHandle key)

  *Crypto, Authenticated Encryption with Symmetric key Verification Functions.*
- TEE_Result TEE_AEInit (TEE_OperationHandle operation, const void *nonce, uint32_t nonceLen, uint32_t tagLen, uint32_t AADLen, uint32_t payloadLen)

  *Crypto, Authenticated Encryption with Symmetric key Verification Functions.*
- TEE_Result TEE_AEUpdate (TEE_OperationHandle operation, const void *srcData, uint32_t srcLen, void *destData, uint32_t *destLen)

  *Crypto, Authenticated Encryption with Symmetric key Verification Functions.*
- void TEE_AEUpdateAAD (TEE_OperationHandle operation, const void *AADdata, uint32_t AADdataLen)

  *Crypto, Authenticated Encryption with Symmetric key Verification Functions.*
- TEE_Result TEE_AEEncryptFinal (TEE_OperationHandle operation, const void *srcData, uint32_t srcLen, void *destData, uint32_t *destLen, void *tag, uint32_t *tagLen)

  *Crypto, Authenticated Encryption with Symmetric key Verification Functions.*
- TEE_Result TEE_AEDecryptFinal (TEE_OperationHandle operation, const void *srcData, uint32_t srcLen, void *destData, uint32_t *destLen, void *tag, uint32_t tagLen)

  *Crypto, Authenticated Encryption with Symmetric key Verification Functions.*
- void TEE_CipherInit (TEE_OperationHandle operation, const void *nonce, uint32_t nonceLen)

  *Crypto, Authenticated Encryption with Symmetric key Verification Functions.*
- TEE_Result TEE_CipherUpdate (TEE_OperationHandle operation, const void *srcData, uint32_t srcLen, void *destData, uint32_t *destLen)

  *Crypto, Authenticated Encryption with Symmetric key Verification Functions.*
- TEE_Result TEE_GenerateKey (TEE_ObjectHandle object, uint32_t keySize, const TEE_Attribute *params, uint32_t paramCount)

  *Crypto, Asymmetric key Verification Functions.*
- TEE_Result TEE_AllocateTransientObject (TEE_ObjectType objectType, uint32_t maxKeySize, TEE_ObjectHandle *object)

  *Crypto, Asymmetric key Verification Functions.*
- void TEE_InitRefAttribute (TEE_Attribute *attr, uint32_t attributeID, const void *buffer, uint32_t length)

  *Crypto, Asymmetric key Verification Functions.*
- void TEE_InitValueAttribute (TEE_Attribute *attr, uint32_t attributeID, uint32_t a, uint32_t b)

  *Crypto, Asymmetric key Verification Functions.*
- void TEE_FreeTransientObject (TEE_ObjectHandle object)

  *Crypto, Asymmetric key Verification Functions.*
- TEE_Result TEE_AsymmetricSignDigest (TEE_OperationHandle operation, const TEE_Attribute *params, uint32_t paramCount, const void *digest, uint32_t digestLen, void *signature, uint32_t *signatureLen)

  *Crypto, Asymmetric key Verification Functions.*
- TEE_Result TEE_AsymmetricVerifyDigest (TEE_OperationHandle operation, const TEE_Attribute *params, uint32_t paramCount, const void *digest, uint32_t digestLen, const void *signature, uint32_t signatureLen)

  *Crypto, Asymmetric key Verification Functions.*

### 4.7.1 Detailed Description

Candidate API list for Global Platform like RISC-V TEE.

draft RISC-V Internal TEE API

**Author**

Akira Tsukamoto, AIST

**Date**

2019/09/25

### 4.7.2 Function Documentation

#### 4.7.2.1 __attribute__() `void __attribute__ ( (noreturn) )`

TEE_Panic() - Raises a panic in the Trusted Application instance.

When a Trusted Application calls the TEE_Panic function, the current instance shall be destroyed and all the resources opened by the instance shall be reclaimed. All sessions opened from the panicking instance on another TA shall be gracefully closed and all cryptographic objects and operations shall be closed properly.

**Parameters**

| | |
|---|---|
| *code* | An informative panic code defined by the TA. |

**Returns**

panic code will be returned.

TEE_Panic() - Raises a Panic in the Trusted Application instance

When a Trusted Application calls the TEE_Panic function, the current instance shall be destroyed and all the resources opened by the instance shall be reclaimed.

**Parameters**

| | |
|---|---|
| *ec* | An informative panic code defined by the TA. May be displayed in traces if traces are available. |

#### 4.7.2.2 GetRelTimeEnd() `TEE_Result GetRelTimeEnd (`

```
            uint64_t end )
```

Core Functions, Time Functions.

Return the elapsed.

GetRelTimeEnd() - finds the real time of the end timing.

This function prints the ending time.

**Parameters**

| | |
|---|---|
| *end* | End timing |

**Returns**

0 If success

GetRelTimeStart() - find the real time of the end timing.

This function prints the End timing.

**Parameters**

| | |
|---|---|
| *end* | End timing |

**Returns**

0 if success else error occured

#### 4.7.2.3 GetRelTimeStart()

```
TEE_Result GetRelTimeStart (
            uint64_t start )
```

Core Functions, Time Functions.

Fast relative Time function which guarantees no hart switch or context switch between Trusted and Untrusted sides.

Most of the time ending up writing similar functions when only measuring the relative time in usec resolution which do not require the quality of the time itself but the distance of the two points.
For the usage above, the function does not have to return wall clock time.
Not prepared in both Keystone and GP.

GetRelTimeStart() - Gets the real time of the start timing.

This function prints the starting time.

**Parameters**

| | |
|---|---|
| *start* | Start timing |

**Returns**

0 on success

GetRelTimeStart() - Gets the real time of the start timing.

Ths function prints the start timing.

**Parameters**

| | |
|---|---|
| *start* | start timing |

**Returns**

0 if success else error occured.

#### 4.7.2.4 TEE_AEDecryptFinal()

```
TEE_Result TEE_AEDecryptFinal (
        TEE_OperationHandle operation,
        const void * srcData,
        uint32_t srcLen,
        void * destData,
        uint32_t * destLen,
        void * tag,
        uint32_t tagLen )
```

Crypto, Authenticated Encryption with Symmetric key Verification Functions.

Supports TEE_ALG_AES_CCM, TEE_ALG_AES_GCM.

TEE_AEDecryptFinal() - Processes data that has not been processed by previous calls to TEE_AEUpdate as well as data supplied in srcData.

This function completes the AE operation and compares the computed tag with the tag supplied in the parameter tag .The operation handle can be reused or newly initialized.The buffers srcData and destData shall be either completely disjoint or equal in their starting positions.The operation may be in either initial or active state and enters initial state afterwards.

**Parameters**

| | |
|---|---|
| *operation* | Handle of a running AE operation |
| *srcData* | Reference to final chunk of input data to be encrypted |
| *srcLen* | length of the input data |
| *destData* | Output buffer. Can be omitted if the output is to be discarded. |
| *destLen* | length of the buffer. |
| *tag* | Output buffer filled with the computed tag |
| *tagLen* | length of the tag. |

**Returns**

0 on success.

TEE_ERROR_SHORT_BUFFER If the output buffer is not large enough to contain the output

TEE_ERROR_MAC_INVALID If the computed tag does not match the supplied tag

#### 4.7.2.5 TEE_AEEncryptFinal()

```
TEE_Result TEE_AEEncryptFinal (
        TEE_OperationHandle operation,
        const void * srcData,
        uint32_t srcLen,
        void * destData,
        uint32_t * destLen,
        void * tag,
        uint32_t * tagLen )
```

Crypto, Authenticated Encryption with Symmetric key Verification Functions.

Supports TEE_ALG_AES_CCM, TEE_ALG_AES_GCM.

TEE_AEEncryptFinal() - processes data that has not been processed by previous calls to TEE_AEUpdate as well as data supplied in srcData .

TEE_AEEncryptFinal completes the AE operation and computes the tag. The operation handle can be reused or newly initialized. The buffers srcData and destData SHALL be either completely disjoint or equal in their starting positions.The operation may be in either initial or active state and enters initial state afterwards.

**Parameters**

| | |
|---|---|
| *operation* | Handle of a running AE operation |
| *srcData* | Reference to final chunk of input data to be encrypted |
| *srcLen* | length of the input data |
| *destData* | Output buffer. Can be omitted if the output is to be discarded. |
| *destLen* | length of the buffer. |
| *tag* | Output buffer filled with the computed tag |
| *tagLen* | length of the tag. |

**Returns**

0 on success.

TEE_ERROR_SHORT_BUFFER If the output or tag buffer is not large enoughto contain the output.

#### 4.7.2.6 TEE_AEInit()

```
TEE_Result TEE_AEInit (
        TEE_OperationHandle operation,
        const void * nonce,
        uint32_t nonceLen,
        uint32_t tagLen,
```

```
uint32_t AADLen,
uint32_t payloadLen )
```

Crypto, Authenticated Encryption with Symmetric key Verification Functions.

Supports TEE_ALG_AES_CCM, TEE_ALG_AES_GCM.

TEE_AEInit() - Initializes an Authentication Encryption operation.

The operation must be in initial state and remains in the initial state afterwards.

**Parameters**

| | |
|---|---|
| *operation* | A handle on the operation. |
| *nonce* | The operation nonce or IV |
| *nonceLen* | length of nonce |
| *tagLen* | Size in bits of the tag |
| *AADLen* | Length in bytes of the AAD |
| *payloadLen* | Length in bytes of the payload. |

**Returns**

0 on success.

TEE_ERROR_NOT_SUPPORTED If the tag length is not supported by the algorithm.

#### 4.7.2.7 TEE_AEUpdate()

```
TEE_Result TEE_AEUpdate (
TEE_OperationHandle operation,
const void * srcData,
uint32_t srcLen,
void * destData,
uint32_t * destLen )
```

Crypto, Authenticated Encryption with Symmetric key Verification Functions.

Supports TEE_ALG_AES_CCM, TEE_ALG_AES_GCM.

TEE_AEUpdate() - Accumulates data for an Authentication Encryption operation

This function describes Input data does not have to be a multiple of block size. Subsequent calls to this function are possible. Unless one or more calls of this function have supplied sufficient input data,no output is generated. when using this routine to decrypt the returned data may be corrupt since the integrity check is not performed until all the data has been processed. If this is a concern then only use the TEE_AEDecryptFinal routine.

**Parameters**

| | |
|---|---|
| *operation* | Handle of a running AE operation. |
| *srcData* | Input data buffer to be encrypted or decrypted |
| *srcLen* | length of the input buffer. |
| *destData* | Output buffer |
| *destLen* | length of the out put buffer. |

**Returns**

0 on success.

TEE_ERROR_SHORT_BUFFER if the output buffer is not large enough to contain the output.

#### 4.7.2.8 TEE_AEUpdateAAD()

```
void TEE_AEUpdateAAD (
        TEE_OperationHandle operation,
        const void * AADdata,
        uint32_t AADdataLen )
```

Crypto, Authenticated Encryption with Symmetric key Verification Functions.

Supports TEE_ALG_AES_CCM, TEE_ALG_AES_GCM.

TEE_AEUpdateAAD() - Feeds a new chunk of Additional Authentication Data (AAD) to the AE operation. Subsequent calls to this function are possible.

The TEE_AEUpdateAAD function feeds a new chunk of Additional Authentication Data (AAD) to the AE operation. Subsequent calls to this function are possible.The buffers srcData and destData shall be either completely disjoint or equal in their starting positions.The operation SHALL be in initial state and remains in initial state afterwards.

**Parameters**

| | |
|---|---|
| *operation* | Handle on the AE operation |
| *AADdata* | Input buffer containing the chunk of AAD |
| *AADdataLen* | length of the chunk of AAD. |

#### 4.7.2.9 TEE_AllocateOperation()

```
TEE_Result TEE_AllocateOperation (
        TEE_OperationHandle * operation,
        uint32_t algorithm,
        uint32_t mode,
        uint32_t maxKeySize )
```

Crypto, for all Crypto Functions.

All Crypto Functions use TEE_OperationHandle∗ operation instances.
Create Crypto instance.

TEE_AllocateOperation() - Allocates a handle for a new cryptographic operation and sets the mode and algorithm type.

If this function does not return with TEE_SUCCESS then there is no valid handle value.Once a cryptographic operation has been created, the implementation shall guarantee that all resources necessary for the operation are allocated and that any operation with a key of at most maxKeySize bits can be performed. For algorithms that take multiple keys, for example the AES XTS algorithm, the maxKeySize parameter specifies the size of the largest key. It is up to the implementation to properly allocate space for multiple keys if the algorithm so requires.

**Parameters**

| | |
|---|---|
| *operation* | reference to generated operation handle. |
| *algorithm* | One of the cipher algorithms. |
| *mode* | The operation mode. |
| *maxKeySize* | Maximum key size in bits for the operation. |

**Returns**

0 in case of success

TEE_ERROR_OUT_OF_MEMORY If there are not enough resources to allocate the operation.

TEE_ERROR_NOT_SUPPORTED If the mode is not compatible with the algorithm or key size or if the algorithm is not one of the listed algorithms or if maxKeySize is not appropriate for the algorithm.

#### 4.7.2.10 TEE_AllocateTransientObject()

```
TEE_Result TEE_AllocateTransientObject (
    TEE_ObjectType objectType,
    uint32_t maxKeySize,
    TEE_ObjectHandle * object )
```

Crypto, Asymmetric key Verification Functions.

Create object storing asymmetric key.

TEE_AllocateTransientObject() - Allocates an uninitialized transient object. Transient objects are used to hold a cryptographic object (key or key-pair).

The value TEE_KEYSIZE_NO_KEY should be used for maxObjectSize for object types that do not require a key so that all the container resources can be pre-allocated. As allocated, the container is uninitialized. It can be initialized by subsequently importing the object material,generating an object, deriving an object, or loading an object from the Trusted Storage.

**Parameters**

| | |
|---|---|
| *objectType* | Type of uninitialized object container to be created |
| *maxKeySize* | Key Size of the object. |
| *object* | Filled with a handle on the newly created key container. |

**Returns**

0 on success

TEE_ERROR_OUT_OF_MEMORY If not enough resources are available to allocate the object handle.

TEE_ERROR_NOT_SUPPORTED If the key size is not supported or the object type is not supported.

#### 4.7.2.11 TEE_AsymmetricSignDigest()

```
TEE_Result TEE_AsymmetricSignDigest (
        TEE_OperationHandle operation,
        const TEE_Attribute * params,
        uint32_t paramCount,
        const void * digest,
        uint32_t digestLen,
        void * signature,
        uint32_t * signatureLen )
```

Crypto, Asymmetric key Verification Functions.

Sign a message digest within an asymmetric key operation.
Keystone has ed25519_sign().
Equivalent in openssl is EVP_DigestSign().

TEE_AsymmetricSignDigest() - Signs a message digest within an asymmetric operation.

**Parameters**

| | |
|---|---|
| *operation* | Handle on the operation, which SHALL have been suitably set up with an operation key. |
| *params* | Optional operation parameters |
| *paramCount* | size of param |
| *digest* | Input buffer containing the input message digest |
| *digestLen* | length of input buffer. |
| *signature* | Output buffer written with the signature of the digest |
| *signatureLen* | length of output buffer. |

**Returns**

0 on sccess

TEE_ERROR_SHORT_BUFFER If the signature buffer is not large enough to hold the result

#### 4.7.2.12 TEE_AsymmetricVerifyDigest()

```
TEE_Result TEE_AsymmetricVerifyDigest (
        TEE_OperationHandle operation,
        const TEE_Attribute * params,
        uint32_t paramCount,
        const void * digest,
        uint32_t digestLen,
        const void * signature,
        uint32_t signatureLen )
```

Crypto, Asymmetric key Verification Functions.

Verifies a message digest signature within an asymmetric key operation.
Keystone has ed25519_verify().
Equivalent in openssl is EVP_DigestVerify().

TEE_AsymmetricVerifyDigest() - verifies a message digest signature within an asymmetric operation.

This function describes the message digest signature verify by calling ed25519_verify().

**Parameters**

| | |
|---|---|
| *operation* | Handle on the operation, which SHALL have been suitably set up with an operation key. |
| *params* | Optional operation parameters |
| *paramCount* | size of param. |
| *digest* | Input buffer containing the input message digest |
| *digestLen* | length of input buffer. |
| *signature* | Output buffer written with the signature of the digest |
| *signatureLen* | length of output buffer. |

**Returns**

TEE_SUCCESS on success

TEE_ERROR_SIGNATURE_INVALID if the signature is invalid.

#### 4.7.2.13 TEE_CipherInit()

```
void TEE_CipherInit (
        TEE_OperationHandle operation,
        const void * nonce,
        uint32_t nonceLen )
```

Crypto, Authenticated Encryption with Symmetric key Verification Functions.

Supports TEE_ALG_AES_CBC.

TEE_CipherInit() - starts the symmetric cipher operation.

The operation shall have been associated with a key. If the operation is in active state, it is reset and then initialized. If the operation is in initial state, it is moved to active state.

**Parameters**

| | |
|---|---|
| *operation* | A handle on an opened cipher operation setup with a key |
| *nonce* | Buffer containing the operation Initialization Vector as appropriate. |
| *nonceLen* | length of the buffer |

#### 4.7.2.14 TEE_CipherUpdate()

```
TEE_Result TEE_CipherUpdate (
        TEE_OperationHandle operation,
        const void * srcData,
        uint32_t srcLen,
        void * destData,
        uint32_t * destLen )
```

Crypto, Authenticated Encryption with Symmetric key Verification Functions.

Supports TEE_ALG_AES_CBC.

TEE_CipherUpdate() - encrypts or decrypts input data.

Input data does not have to be a multiple of block size. Subsequent calls to this function are possible. Unless one or more calls of this function have supplied sufficient input data, no output is generated. The cipher operation is finalized with a call to TEE_CipherDoFinal .The buffers srcData and destData SHALL be either completely disjoint or equal in their starting positions.The operation SHALL be in active state.

**Parameters**

| | |
|---|---|
| *operation* | Handle of a running Cipher operation |
| *srcData* | Input data buffer to be encrypted or decrypted |
| *srcLen* | length of input buffer |
| *destData* | output buffer |
| *destLen* | ouput buffer length. |

**Returns**

0 on success else

TEE_ERROR_SHORT_BUFFER If the output buffer is not large enough to contain the output. In this case, the input is not fed into the algorithm.

#### 4.7.2.15 TEE_CloseObject() `void TEE_CloseObject ( TEE_ObjectHandle object )`

Core Functions, Secure Storage Functions (data is isolated for each TA)

Destroy object (key, key-pair or Data).

TEE_CloseObject() - Closes an opened object handle.

The object can be persistent or transient.For transient objects, TEE_CloseObject is equivalent to TEE_Free↩ TransientObject.

**Parameters**

| | |
|---|---|
| *object* | Handle of the object. |

**Returns**

TEE_SUCCESS if success else error occured.

TEE_CloseObject() - Function closes an opened object handle.

The object can be persistent or transient.For transient objects, TEE_CloseObject is equivalent to TEE_Free↩ TransientObject.

**Parameters**

| | |
|---|---|
| *object* | Handle of the object |

**Returns**

TEE_SUCCESS if success else error occured.

#### 4.7.2.16 TEE_CreatePersistentObject()

```
TEE_Result TEE_CreatePersistentObject (
        uint32_t storageID,
        const void * objectID,
        uint32_t objectIDLen,
        uint32_t flags,
        TEE_ObjectHandle attributes,
        const void * initialData,
        uint32_t initialDataLen,
        TEE_ObjectHandle * object )
```

Core Functions, Secure Storage Functions (data is isolated for each TA)

Create persistent object (key, key-pair or Data).
For the people who have not written code on GP then probably do not need to care the meaning of what is Persistent Object is, since the following are enough to use secure storage feature.

TEE_CreatePersistentObject() - Creates a persistent object with initial attributes.

In this function an initial data stream content returns either a handle on the created object or TEE_HANDLE_NULL upon failure.

**Parameters**

| | |
|---|---|
| *storageID* | The storage to use. |
| *objectID* | The object identifier |
| *objectIDLen* | The object identifier |
| *flags* | The flags which determine the settings under which the object is opened. |
| *attributes* | A handle on a persistent object or an initialized transient object from which to take the persistent object attributes |
| *initialData* | The initial data content of the persistent object |
| *initialDataLen* | The initial data content of the persistent object |
| *object* | A pointer to the handle which contains the opened handle upon successful completion |

**Returns**

0 if success else error occured.

TEE_CreatePersistentObject() - Creates a persistent object with initial attributes.

An initial data stream content, and optionally returns either a handle on the created object, or TEE_HANDLE_NULL upon failure.

**Parameters**

| | |
|---|---|
| *storageID* | The storage to use. |
| *objectID* | The object identifier |
| *objectIDLen* | The object identifier |
| *flags* | The flags which determine the settings under which the object is opened. |
| *attributes* | A handle on a persistent object or an initialized transient object from which to take the persistent object attributes |
| *initialData* | The initial data content of the persistent object |
| *initialDataLen* | The initial data content of the persistent object |
| *object* | A pointer to the handle, which contains the opened handle upon successful completion |

**Returns**

0 if success, else error occured.

#### 4.7.2.17 TEE_DigestDoFinal()

```
TEE_Result TEE_DigestDoFinal (
        TEE_OperationHandle operation,
        const void * chunk,
        uint32_t chunkLen,
        void * hash,
        uint32_t * hashLen )
```

Function accumulates message data for hashing.

TEE_DigestDoFinal() - Finalizes the message digest operation and produces the message hash.

This function finalizes the message digest operation and produces the message hash. Afterwards the Message Digest operation is reset to initial state and can be reused.

**Parameters**

| | |
|---|---|
| *operation* | Handle of a running Message Digest operation. |
| *chunk* | Chunk of data to be hashed. |
| *chunkLen* | size of the chunk. |
| *hash* | Output buffer filled with the message hash. |
| *hashLen* | lenth of the mesaage hash. |

**Returns**

0 on success

TEE_ERROR_SHORT_BUFFER If the output buffer is too small. In this case, the operation is not finalized.

#### 4.7.2.18 TEE_DigestUpdate()

```
void TEE_DigestUpdate (
        TEE_OperationHandle operation,
        const void * chunk,
        uint32_t chunkSize )
```

Crypto, Message Digest Functions.

Function accumulates message data for hashing.

TEE_DigestUpdate()- Accumulates message data for hashing.

This function describes the message does not have to be block aligned. Subsequent calls to this function are possible.The operation may be in either initial or active state and becomes active.

**Parameters**

| | |
|---|---|
| *operation* | Handle of a running Message Digest operation. |
| *chunk* | Chunk of data to be hashed |
| *chunkSize* | size of the chunk. |

#### 4.7.2.19 TEE_FreeOperation()

```
void TEE_FreeOperation (
        TEE_OperationHandle operation )
```

Crypto, for all Crypto Functions.

All Crypto Functions use TEE_OperationHandle∗ operation instances.
Destroy Crypto instance.

TEE_FreeOperation() - Deallocates all resources associated with an operation handle.

This function deallocates all resources associated with an operation handle. After this function is called, the operation handle is no longer valid. All cryptographic material in the operation is destroyed. The function does nothing if operation is TEE_HANDLE_NULL.

**Parameters**

| | |
|---|---|
| *operation* | Reference to operation handle. |

**Returns**

nothing after the operation free.

#### 4.7.2.20 TEE_FreeTransientObject()

```
void TEE_FreeTransientObject (
        TEE_ObjectHandle object )
```

Crypto, Asymmetric key Verification Functions.

Destroy object storing asymmetric key.

TEE_FreeTransientObject() - Deallocates a transient object previously allocated with TEE_AllocateTransientObject .

this function describes the object handle is no longer valid and all resources associated with the transient object shall have been reclaimed after the TEE_AllocateTransientObject() call.

**Parameters**

| | |
|---|---|
| *object* | Handle on the object to free. |

#### 4.7.2.21 TEE_GenerateKey()

```
TEE_Result TEE_GenerateKey (
        TEE_ObjectHandle object,
        uint32_t keySize,
        const TEE_Attribute * params,
        uint32_t paramCount )
```

Crypto, Asymmetric key Verification Functions.

Generate asymmetric keypair.

TEE_GenerateKey () - Generates a random key or a key-pair and populates a transient key object with the generated key material.

The size of the desired key is passed in the keySize parameter and shall be less than or equal to the maximum key size specified when the transient object was created.

**Parameters**

| | |
|---|---|
| *object* | Handle on an uninitialized transient key to populate with the generated key. |
| *keySize* | Requested key size shall be less than or equal to the maximum key size specified when the object container was created |
| *params* | Parameters for the key generation. |
| *paramCount* | The values of all parameters are copied nto the object so that the params array and all the memory buffers it points to may be freed after this routine returns without affecting the object. |

**Returns**

0 on succes

TEE_ERROR_BAD_PARAMETERS If an incorrect or inconsistent attribute is detected. The checks that are performed depend on the implementation.

#### 4.7.2.22 TEE_GenerateRandom() `void TEE_GenerateRandom (`

```
void * randomBuffer,
uint32_t randomBufferLen )
```

Crypto, common.

Random Data Generation Function. The quality of the random is implementation dependent.
I am not sure this should be in Keystone or not, but it is very handy.
Good to have adding a way to check the quality of the random implementation.

TEE_GenerateRandom() - Generates random data.

This function generates random data of random buffer length and is stored in to random Buffer by calling wc_↩RNG_GenerateBlock().If ret is not equal
to 0 then TEE_Panic is called.

**Parameters**

| | |
|---|---|
| *randomBuffer* | Reference to generated random data |
| *randomBufferLen* | Byte length of requested random data |

**Returns**

random data random data will be returned.

TEE_GenerateRandom() - Generates random data.

This function generates random data of random bufferlength and is stored in to randomBuffer by calling sgx_read↩_rand().

**Parameters**

| | |
|---|---|
| *randomBuffer* | Reference to generated random data |
| *randomBufferLen* | Byte length of requested random data |

#### 4.7.2.23 TEE_GetObjectInfo1() `TEE_Result TEE_GetObjectInfo1 (`

```
TEE_ObjectHandle object,
TEE_ObjectInfo * objectInfo )
```

Core Functions, Secure Storage Functions (data is isolated for each TA)

Get length of object required before reading the object.

TEE_GetObjectInfo1() - Returns the characteristics of an object.

This function returns a handle which can be used to access the object's attributes and data stream.

**Parameters**

| | |
|---|---|
| *objectInfo* | Pointer to a structure filled with the object information |
| *object* | Handle of the object |

**Returns**

0 if success else error occured.

TEE_GetObjectInfo1() - Function returns the characteristics of an object.

It returns a handle that can be used to access the object's attributes and data stream.

**Parameters**

| | |
|---|---|
| *objectInfo* | Pointer to a structure filled with the object information |
| *object* | Handle of the object |

**Returns**

0 if success else error occured.

#### 4.7.2.24 TEE_GetREETime() `void TEE_GetREETime ( TEE_Time * time )`

Core Functions, Time Functions.

Wall clock time of host OS, expressed in the number of seconds since 1970-01-01 UTC.
This could be implemented on Keystone using ocall.

TEE_GetREETime() - Retrieves the current REE system time.

This function retrieves the current time as seen from the point of view of
the REE.

**Parameters**

| | |
|---|---|
| *time* | Filled with the number of seconds and milliseconds |

TEE_GetREETime() - Function retrieves the current REE system time.

This function retrieves the current time as seen from the point of view of the REE.

**Parameters**

| | |
|---|---|
| *time* | Filled with the number of seconds and milliseconds. |

#### 4.7.2.25 TEE_GetSystemTime() `void TEE_GetSystemTime (`

```
TEE_Time * time )
```

Core Functions, Time Functions.

Time of TEE-controlled secure timer or Host OS time, implementation dependent.

TEE_GetSystemTime() - Retrieves the current system time.

This function describes the system time has an arbitrary implementation
defined origin that can vary across TA instances. The minimum guarantee
is that the system time shall be monotonic for a given TA instance.

**Parameters**

| | |
|---|---|
| *time* | Filled with the number of seconds and milliseconds |

TEE_GetSystemTime() - Retrieves the current system time.

The system time has an arbitrary implementation-defined origin that can vary across TA instances

**Parameters**

| | |
|---|---|
| *time* | Filled with the number of seconds and milliseconds. |

#### 4.7.2.26 TEE_InitRefAttribute() `void TEE_InitRefAttribute (`

```
TEE_Attribute * attr,
uint32_t attributeID,
const void * buffer,
uint32_t length )
```

Crypto, Asymmetric key Verification Functions.

Storing asymmetric key.

TEE_InitRefAttribute() - The helper function can be used to populate a single attribute either with a reference to a buffer or with integer values.

In TEE_InitRefAttribute () only the buffer pointer is copied, not the content of the buffer. This means that the attribute structure maintains a pointer back to the supplied buffer. It is the responsibility of the TA author to ensure that the contents of the buffer maintain their value until the attributes array is no longer in use.

**Parameters**

| | |
|---|---|
| *attr* | attribute structure to initialize. |
| *attributeID* | Identifier of the attribute to populate. |
| *buffer* | input buffer that holds the content of the attribute. |
| *length* | buffer length. |

#### 4.7.2.27 TEE_InitValueAttribute()

```
void TEE_InitValueAttribute (
        TEE_Attribute * attr,
        uint32_t attributeID,
        uint32_t a,
        uint32_t b )
```

Crypto, Asymmetric key Verification Functions.

Storing asymmetric key.

TEE_InitValueAttribute() - The helper function can be used to populate a single attribute either with a reference to a buffer or with integer values.

**Parameters**

| | |
|---|---|
| *attr* | attribute structure to initialize. |
| *attributeID* | Identifier of the attribute to populate. |
| *a* | unsigned integer value to assign to the a member of the attribute structure. |
| *b* | unsigned integer value to assign to the b member of the attribute structure |

#### 4.7.2.28 TEE_OpenPersistentObject()

```
TEE_Result TEE_OpenPersistentObject (
        uint32_t storageID,
        const void * objectID,
        uint32_t objectIDLen,
        uint32_t flags,
        TEE_ObjectHandle * object )
```

Core Functions, Secure Storage Functions (data is isolated for each TA)

Open persistent object.

TEE_OpenPersistentObject() - Opens a handle on an existing persistent object.

This function returns a handle which can be used to access the object's attributes and data stream.

**Parameters**

| | |
|---|---|
| *storageID* | The storage to use |
| *objectID* | The object identifier |
| *objectIDLen* | The object identifier |
| *flags* | The flags which determine the settings under which the object is opened. |
| *object* | A pointer to the handle, which contains the opened handle upon successful completion |

**Returns**

0 if success else error occured.

TEE_OpenPersistentObject() - Opens a handle on an existing persistent object.

This function returns a handle that can be used to access the object's attributes and data stream.

**Parameters**

| | |
|---|---|
| *storageID* | The storage to use. |
| *objectID* | The object identifier |
| *objectIDLen* | The object identifier |
| *flags* | The flags which determine the settings under which the object is opened. |
| *object* | A pointer to the handle, which contains the opened handle upon successful completion |

**Returns**

0 if success, else error occured.

#### 4.7.2.29 TEE_ReadObjectData()

```
TEE_Result TEE_ReadObjectData (
        TEE_ObjectHandle object,
        void * buffer,
        uint32_t size,
        uint32_t * count )
```

Core Functions, Secure Storage Functions (data is isolated for each TA)

Read object.

TEE_ReadObjectData() - Attempts to read size bytes from the data stream associated with the object into the buffer pointed to by buffer.

The bytes are read starting at the position in the data stream currently stored in the object handle. The handle's position is incremented by the number of bytes actually read. On completion of TEE_ReadObjectData sets the number of bytes actually read in the "uint32_t" pointed to by count. The value written to ∗count may be less than size if the number of bytes until the end-of3067 stream is less than size. It is set to 0 if the position at the start of the read operation is at or beyond the end-of-stream. These are the only cases where ∗count may be less than size.

**Parameters**

| | |
|---|---|
| *object* | Handle of the object |
| *buffer* | The buffer containing the data to be written |
| *size* | The number of bytes to write |
| *count* | size of the buffer. |

**Returns**

TEE_SUCCESS if success else error occured.

TEE_ReadObjectData() - Attempts to read size bytes from the data stream associated with the object object into the buffer pointed to by buffer.

The bytes are read starting at the position in the data stream currently stored in the object handle. The handle's position is incremented by the number of bytes actually read. On completion TEE_ReadObjectData sets the number of bytes actually read in the uint32_t pointed to by count. The value written to ∗count may be less than size if the number of bytes until the end-of3067 stream is less than size. It is set to 0 if the position at the start of the read operation is at or beyond the end-of-stream. These are the only cases where ∗count may be less than size.

**Parameters**

| | |
|---|---|
| *object* | Handle of the object |
| *buffer* | The buffer containing the data to be written |
| *size* | The number of bytes to write |
| *count* | size of the buffer. |

**Returns**

TEE_SUCCESS if success, else error occured.

#### 4.7.2.30 TEE_SetOperationKey() `TEE_Result TEE_SetOperationKey ( TEE_OperationHandle operation, TEE_ObjectHandle key )`

Crypto, Authenticated Encryption with Symmetric key Verification Functions.

Set symmetric key used in operation.

TEE_SetOperationKey() - Programs the key of an operation; that is, it associates an operation with a key.

The key material is copied from the key object handle into the operation. After the key has been set, there is no longer any link between the operation and the key object. The object handle can be closed or reset and this will not affect the operation. This copied material exists until the operation is freed using TEE_FreeOperation or another key is set into the operation.

**Parameters**

| | |
|---|---|
| *operation* | Operation handle. |
| *key* | A handle on a key object. |

**Returns**

0 on success return

TEE_ERROR_CORRUPT_OBJECT If the object is corrupt. The object handle is closed.

TEE_ERROR_STORAGE_NOT_AVAILABLE If the persistent object is stored in a storage area which is currently inaccessible.

#### 4.7.2.31 TEE_WriteObjectData()

```
TEE_Result TEE_WriteObjectData (
    TEE_ObjectHandle object,
    const void * buffer,
    uint32_t size )
```

Core Functions, Secure Storage Functions (data is isolated for each TA)

Write object.

TEE_WriteObjectData() - Writes the buffer data in to persistent objects.

In this function it checks if object is present or not, the encryption/ decryption buffer is taken by calling mbedtls_aes↩ _crypt_cbc() then that buffer data is encrypted and mapped to object.On the base of object creation TEE_SUCCESS appears else TEE_ERROR_GENERIC appears.

**Parameters**

| | |
|---|---|
| *object* | Handle of the object |
| *buffer* | The buffer containing the data to be written |
| *size* | The number of bytes to write |

**Returns**

TEE_SUCCESS if success else error occured.

TEE_WriteObjectData() - writes size bytes from the buffer pointed to by buffer to the data stream associated with the open object handle object.

If the current data position points before the end-of-stream, then size bytes are written to the data stream, overwriting bytes starting at the current data position. If the current data position points beyond the stream's end, then the data stream is first extended with zero bytes until the length indicated by the data position indicator is reached, and then size bytes are written to the stream.

**Parameters**

| | |
|---|---|
| *object* | Handle of the object |
| *buffer* | The buffer containing the data to be written |
| *size* | The number of bytes to write |

**Returns**

TEE_SUCCESS if success else error occured.

## 4.8 tee-ta-internal.h

Go to the documentation of this file.

```
1 /*
2  * SPDX-License-Identifier: BSD-2-Clause
3  *
4  * Copyright (C) 2019 National Institute of Advanced Industrial Science
5  *                          and Technology (AIST)
6  * All rights reserved.
7  *
8  * Redistribution and use in source and binary forms, with or without
9  * modification, are permitted provided that the following conditions are met:
10  *
11  * 1. Redistributions of source code must retain the above copyright notice,
12  * this list of conditions and the following disclaimer.
13  *
14  * 2. Redistributions in binary form must reproduce the above copyright notice,
15  * this list of conditions and the following disclaimer in the documentation
16  * and/or other materials provided with the distribution.
17  *
18  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
19  * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
20  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
21  * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
22  * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
23  * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
24  * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
25  * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
26  * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
27  * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
28  * POSSIBILITY OF SUCH DAMAGE.
29  */
37 #ifndef TA_INTERNAL_TEE_H
38 #define TA_INTERNAL_TEE_H
39
40 #include "tee-common.h"
41
42 #ifdef __cplusplus
43 extern "C" {
44 #endif
45
46 void __attribute__((noreturn)) TEE_Panic(unsigned long code);
47
49
53 void TEE_GetREETime(TEE_Time *time);
54
56
58 /* Wall clock time is important for verifying certificates. */
59 void TEE_GetSystemTime(TEE_Time *time);
60
62
69 /* Start timer */
70 TEE_Result GetRelTimeStart(uint64_t start);
71
73
76 TEE_Result GetRelTimeEnd(uint64_t end);
77
79
85 TEE_Result TEE_CreatePersistentObject(uint32_t storageID, const void *objectID,
86                                       uint32_t objectIDLen, uint32_t flags,
87                                       TEE_ObjectHandle attributes,
88                                       const void *initialData,
```

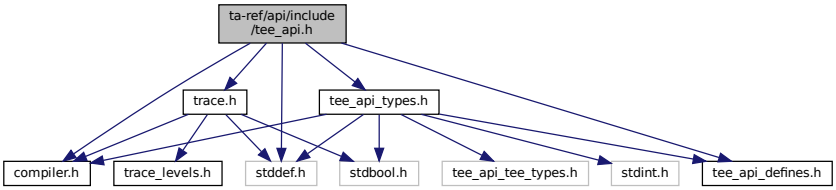
```
89                                      uint32_t initialDataLen,
90                                      TEE_ObjectHandle *object);
92
93 TEE_Result TEE_OpenPersistentObject(uint32_t storageID, const void *objectID,
94                                      uint32_t objectIDLen, uint32_t flags,
95                                      TEE_ObjectHandle *object);
97
98 TEE_Result TEE_GetObjectInfo1(TEE_ObjectHandle object, TEE_ObjectInfo *objectInfo);
100
101 TEE_Result TEE_WriteObjectData(TEE_ObjectHandle object, const void *buffer,
102                                uint32_t size);
105 TEE_Result TEE_ReadObjectData(TEE_ObjectHandle object, void *buffer,
106                               uint32_t size, uint32_t *count);
108
109 void TEE_CloseObject(TEE_ObjectHandle object);
110
111
113
119 void TEE_GenerateRandom(void *randomBuffer, uint32_t randomBufferLen);
120
122
124 TEE_Result TEE_AllocateOperation(TEE_OperationHandle *operation,
125                                  uint32_t algorithm, uint32_t mode,
126                                  uint32_t maxKeySize);
128
130 void TEE_FreeOperation(TEE_OperationHandle operation);
131
132
134
135 void TEE_DigestUpdate(TEE_OperationHandle operation,
136                       const void *chunk, uint32_t chunkSize);
138 TEE_Result TEE_DigestDoFinal(TEE_OperationHandle operation, const void *chunk,
139                              uint32_t chunkLen, void *hash, uint32_t *hashLen);
140
142
143 TEE_Result TEE_SetOperationKey(TEE_OperationHandle operation,
144                     TEE_ObjectHandle key);
146
147 TEE_Result TEE_AEInit(TEE_OperationHandle operation, const void *nonce,
148                       uint32_t nonceLen, uint32_t tagLen, uint32_t AADLen,
149                       uint32_t payloadLen);
151
152 TEE_Result TEE_AEUpdate(TEE_OperationHandle operation, const void *srcData,
153                         uint32_t srcLen, void *destData, uint32_t *destLen);
155
156 void TEE_AEUpdateAAD(TEE_OperationHandle operation, const void *AADdata,
157              uint32_t AADdataLen);
159
160 TEE_Result TEE_AEEncryptFinal(TEE_OperationHandle operation,
161                               const void *srcData, uint32_t srcLen,
162                               void *destData, uint32_t *destLen, void *tag,
163                               uint32_t *tagLen);
165
166 TEE_Result TEE_AEDecryptFinal(TEE_OperationHandle operation,
167                               const void *srcData, uint32_t srcLen,
168                               void *destData, uint32_t *destLen, void *tag,
169                               uint32_t tagLen);
170
172
173 void TEE_CipherInit(TEE_OperationHandle operation, const void *nonce,
174            uint32_t nonceLen);
176
177 TEE_Result TEE_CipherUpdate(TEE_OperationHandle operation, const void *srcData,
178                 uint32_t srcLen, void *destData, uint32_t *destLen);
179
181
182 TEE_Result TEE_GenerateKey(TEE_ObjectHandle object, uint32_t keySize,
183                const TEE_Attribute *params, uint32_t paramCount);
185
186 TEE_Result TEE_AllocateTransientObject(TEE_ObjectType objectType,
187                                        uint32_t maxKeySize,
188                                        TEE_ObjectHandle *object);
190
191 void TEE_InitRefAttribute(TEE_Attribute *attr, uint32_t attributeID,
192                           const void *buffer, uint32_t length);
194
195 void TEE_InitValueAttribute(TEE_Attribute *attr, uint32_t attributeID,
196                 uint32_t a, uint32_t b);
198
199 void TEE_FreeTransientObject(TEE_ObjectHandle object);
200
202
206 TEE_Result TEE_AsymmetricSignDigest(TEE_OperationHandle operation,
207                                     const TEE_Attribute *params,
208                                     uint32_t paramCount, const void *digest,
```

```
209                                     uint32_t digestLen, void *signature,
210                                     uint32_t *signatureLen);
212
216 TEE_Result TEE_AsymmetricVerifyDigest(TEE_OperationHandle operation,
217                                       const TEE_Attribute *params,
218                                       uint32_t paramCount, const void *digest,
219                                       uint32_t digestLen, const void *signature,
220                                       uint32_t signatureLen);
221
222 #ifdef __cplusplus
223 }
224 #endif
225
226 #endif /* TA_INTERNAL_TEE_H */
```

## 4.9 ta-ref/api/include/tee_api.h File Reference

```
#include <stddef.h>
#include <compiler.h>
#include <tee_api_defines.h>
#include <tee_api_types.h>
#include <trace.h>
```

Include dependency graph for tee_api.h:




### Functions

- TEE_Result TEE_GetPropertyAsString (TEE_PropSetHandle propsetOrEnumerator, const char *name, char *valueBuffer, uint32_t *valueBufferLen)
- TEE_Result TEE_GetPropertyAsBool (TEE_PropSetHandle propsetOrEnumerator, const char *name, bool *value)
- TEE_Result TEE_GetPropertyAsU32 (TEE_PropSetHandle propsetOrEnumerator, const char *name, uint32_t *value)
- TEE_Result TEE_GetPropertyAsBinaryBlock (TEE_PropSetHandle propsetOrEnumerator, const char *name, void *valueBuffer, uint32_t *valueBufferLen)
- TEE_Result TEE_GetPropertyAsUUID (TEE_PropSetHandle propsetOrEnumerator, const char *name, TEE_UUID *value)
- TEE_Result TEE_GetPropertyAsIdentity (TEE_PropSetHandle propsetOrEnumerator, const char *name, TEE_Identity *value)
- TEE_Result TEE_AllocatePropertyEnumerator (TEE_PropSetHandle *enumerator)
- void TEE_FreePropertyEnumerator (TEE_PropSetHandle enumerator)
- void TEE_StartPropertyEnumerator (TEE_PropSetHandle enumerator, TEE_PropSetHandle propSet)
- void TEE_ResetPropertyEnumerator (TEE_PropSetHandle enumerator)
- TEE_Result TEE_GetPropertyName (TEE_PropSetHandle enumerator, void *nameBuffer, uint32_t *name↩ BufferLen)
- TEE_Result TEE_GetNextProperty (TEE_PropSetHandle enumerator)
- void TEE_Panic (TEE_Result panicCode)

- TEE_Result TEE_OpenTASession (const TEE_UUID *destination, uint32_t cancellationRequestTimeout, uint32_t paramTypes, TEE_Param params[TEE_NUM_PARAMS], TEE_TASessionHandle *session, uint32_t *returnOrigin)
- void TEE_CloseTASession (TEE_TASessionHandle session)
- TEE_Result TEE_InvokeTACommand (TEE_TASessionHandle session, uint32_t cancellationRequestTimeout, uint32_t commandID, uint32_t paramTypes, TEE_Param params[TEE_NUM_PARAMS], uint32_t *returnOrigin)
- bool TEE_GetCancellationFlag (void)
- bool TEE_UnmaskCancellation (void)
- bool TEE_MaskCancellation (void)
- TEE_Result TEE_CheckMemoryAccessRights (uint32_t accessFlags, void *buffer, uint32_t size)
- void TEE_SetInstanceData (const void *instanceData)
- const void * TEE_GetInstanceData (void)
- void * TEE_Malloc (uint32_t size, uint32_t hint)
- void * TEE_Realloc (void *buffer, uint32_t newSize)
- void TEE_Free (void *buffer)
- void * TEE_MemMove (void *dest, const void *src, uint32_t size)
- int32_t TEE_MemCompare (const void *buffer1, const void *buffer2, uint32_t size)
- void * TEE_MemFill (void *buff, uint32_t x, uint32_t size)
- void TEE_GetObjectInfo (TEE_ObjectHandle object, TEE_ObjectInfo *objectInfo)
- TEE_Result TEE_GetObjectInfo1 (TEE_ObjectHandle object, TEE_ObjectInfo *objectInfo)

  *Core Functions, Secure Storage Functions (data is isolated for each TA)*
- void TEE_RestrictObjectUsage (TEE_ObjectHandle object, uint32_t objectUsage)
- TEE_Result TEE_RestrictObjectUsage1 (TEE_ObjectHandle object, uint32_t objectUsage)
- TEE_Result TEE_GetObjectBufferAttribute (TEE_ObjectHandle object, uint32_t attributeID, void *buffer, uint32_t *size)
- TEE_Result TEE_GetObjectValueAttribute (TEE_ObjectHandle object, uint32_t attributeID, uint32_t *a, uint32_t *b)
- void TEE_CloseObject (TEE_ObjectHandle object)

  *Core Functions, Secure Storage Functions (data is isolated for each TA)*
- TEE_Result TEE_AllocateTransientObject (TEE_ObjectType objectType, uint32_t maxKeySize, TEE_ObjectHandle *object)

  *Crypto, Asymmetric key Verification Functions.*
- void TEE_FreeTransientObject (TEE_ObjectHandle object)

  *Crypto, Asymmetric key Verification Functions.*
- void TEE_ResetTransientObject (TEE_ObjectHandle object)
- TEE_Result TEE_PopulateTransientObject (TEE_ObjectHandle object, const TEE_Attribute *attrs, uint32_t attrCount)
- void TEE_InitRefAttribute (TEE_Attribute *attr, uint32_t attributeID, const void *buffer, uint32_t length)

  *Crypto, Asymmetric key Verification Functions.*
- void TEE_InitValueAttribute (TEE_Attribute *attr, uint32_t attributeID, uint32_t a, uint32_t b)

  *Crypto, Asymmetric key Verification Functions.*
- void TEE_CopyObjectAttributes (TEE_ObjectHandle destObject, TEE_ObjectHandle srcObject)
- TEE_Result TEE_CopyObjectAttributes1 (TEE_ObjectHandle destObject, TEE_ObjectHandle srcObject)
- TEE_Result TEE_GenerateKey (TEE_ObjectHandle object, uint32_t keySize, const TEE_Attribute *params, uint32_t paramCount)

  *Crypto, Asymmetric key Verification Functions.*
- TEE_Result TEE_OpenPersistentObject (uint32_t storageID, const void *objectID, uint32_t objectIDLen, uint32_t flags, TEE_ObjectHandle *object)

  *Core Functions, Secure Storage Functions (data is isolated for each TA)*
- TEE_Result TEE_CreatePersistentObject (uint32_t storageID, const void *objectID, uint32_t objectIDLen, uint32_t flags, TEE_ObjectHandle attributes, const void *initialData, uint32_t initialDataLen, TEE_ObjectHandle *object)

  *Core Functions, Secure Storage Functions (data is isolated for each TA)*

- void TEE_CloseAndDeletePersistentObject (TEE_ObjectHandle object)
- TEE_Result TEE_CloseAndDeletePersistentObject1 (TEE_ObjectHandle object)
- TEE_Result TEE_RenamePersistentObject (TEE_ObjectHandle object, const void *newObjectID, uint32_t newObjectIDLen)
- TEE_Result TEE_AllocatePersistentObjectEnumerator (TEE_ObjectEnumHandle *objectEnumerator)
- void TEE_FreePersistentObjectEnumerator (TEE_ObjectEnumHandle objectEnumerator)
- void TEE_ResetPersistentObjectEnumerator (TEE_ObjectEnumHandle objectEnumerator)
- TEE_Result TEE_StartPersistentObjectEnumerator (TEE_ObjectEnumHandle objectEnumerator, uint32_t storageID)
- TEE_Result TEE_GetNextPersistentObject (TEE_ObjectEnumHandle objectEnumerator, TEE_ObjectInfo *objectInfo, void *objectID, uint32_t *objectIDLen)
- TEE_Result TEE_ReadObjectData (TEE_ObjectHandle object, void *buffer, uint32_t size, uint32_t *count)

  *Core Functions, Secure Storage Functions (data is isolated for each TA)*
- TEE_Result TEE_WriteObjectData (TEE_ObjectHandle object, const void *buffer, uint32_t size)

  *Core Functions, Secure Storage Functions (data is isolated for each TA)*
- TEE_Result TEE_TruncateObjectData (TEE_ObjectHandle object, uint32_t size)
- TEE_Result TEE_SeekObjectData (TEE_ObjectHandle object, int32_t offset, TEE_Whence whence)
- TEE_Result TEE_AllocateOperation (TEE_OperationHandle *operation, uint32_t algorithm, uint32_t mode, uint32_t maxKeySize)

  *Crypto, for all Crypto Functions.*
- void TEE_FreeOperation (TEE_OperationHandle operation)

  *Crypto, for all Crypto Functions.*
- void TEE_GetOperationInfo (TEE_OperationHandle operation, TEE_OperationInfo *operationInfo)
- TEE_Result TEE_GetOperationInfoMultiple (TEE_OperationHandle operation, TEE_OperationInfoMultiple *operationInfoMultiple, uint32_t *operationSize)
- void TEE_ResetOperation (TEE_OperationHandle operation)
- TEE_Result TEE_SetOperationKey (TEE_OperationHandle operation, TEE_ObjectHandle key)

  *Crypto, Authenticated Encryption with Symmetric key Verification Functions.*
- TEE_Result TEE_SetOperationKey2 (TEE_OperationHandle operation, TEE_ObjectHandle key1, TEE_ObjectHandle key2)
- void TEE_CopyOperation (TEE_OperationHandle dstOperation, TEE_OperationHandle srcOperation)
- TEE_Result TEE_IsAlgorithmSupported (uint32_t algId, uint32_t element)
- void TEE_DigestUpdate (TEE_OperationHandle operation, const void *chunk, uint32_t chunkSize)

  *Crypto, Message Digest Functions.*
- TEE_Result TEE_DigestDoFinal (TEE_OperationHandle operation, const void *chunk, uint32_t chunkLen, void *hash, uint32_t *hashLen)
- void TEE_CipherInit (TEE_OperationHandle operation, const void *IV, uint32_t IVLen)

  *Crypto, Authenticated Encryption with Symmetric key Verification Functions.*
- TEE_Result TEE_CipherUpdate (TEE_OperationHandle operation, const void *srcData, uint32_t srcLen, void *destData, uint32_t *destLen)

  *Crypto, Authenticated Encryption with Symmetric key Verification Functions.*
- TEE_Result TEE_CipherDoFinal (TEE_OperationHandle operation, const void *srcData, uint32_t srcLen, void *destData, uint32_t *destLen)
- void TEE_MACInit (TEE_OperationHandle operation, const void *IV, uint32_t IVLen)
- void TEE_MACUpdate (TEE_OperationHandle operation, const void *chunk, uint32_t chunkSize)
- TEE_Result TEE_MACComputeFinal (TEE_OperationHandle operation, const void *message, uint32_t messageLen, void *mac, uint32_t *macLen)
- TEE_Result TEE_MACCompareFinal (TEE_OperationHandle operation, const void *message, uint32_t messageLen, const void *mac, uint32_t macLen)
- TEE_Result TEE_AEInit (TEE_OperationHandle operation, const void *nonce, uint32_t nonceLen, uint32_t tagLen, uint32_t AADLen, uint32_t payloadLen)

  *Crypto, Authenticated Encryption with Symmetric key Verification Functions.*
- void TEE_AEUpdateAAD (TEE_OperationHandle operation, const void *AADdata, uint32_t AADdataLen)

  *Crypto, Authenticated Encryption with Symmetric key Verification Functions.*

- TEE_Result TEE_AEUpdate (TEE_OperationHandle operation, const void *srcData, uint32_t srcLen, void *destData, uint32_t *destLen)

  *Crypto, Authenticated Encryption with Symmetric key Verification Functions.*
- TEE_Result TEE_AEEncryptFinal (TEE_OperationHandle operation, const void *srcData, uint32_t srcLen, void *destData, uint32_t *destLen, void *tag, uint32_t *tagLen)

  *Crypto, Authenticated Encryption with Symmetric key Verification Functions.*
- TEE_Result TEE_AEDecryptFinal (TEE_OperationHandle operation, const void *srcData, uint32_t srcLen, void *destData, uint32_t *destLen, void *tag, uint32_t tagLen)

  *Crypto, Authenticated Encryption with Symmetric key Verification Functions.*
- TEE_Result TEE_AsymmetricEncrypt (TEE_OperationHandle operation, const TEE_Attribute *params, uint32_t paramCount, const void *srcData, uint32_t srcLen, void *destData, uint32_t *destLen)
- TEE_Result TEE_AsymmetricDecrypt (TEE_OperationHandle operation, const TEE_Attribute *params, uint32_t paramCount, const void *srcData, uint32_t srcLen, void *destData, uint32_t *destLen)
- TEE_Result TEE_AsymmetricSignDigest (TEE_OperationHandle operation, const TEE_Attribute *params, uint32_t paramCount, const void *digest, uint32_t digestLen, void *signature, uint32_t *signatureLen)

  *Crypto, Asymmetric key Verification Functions.*
- TEE_Result TEE_AsymmetricVerifyDigest (TEE_OperationHandle operation, const TEE_Attribute *params, uint32_t paramCount, const void *digest, uint32_t digestLen, const void *signature, uint32_t signatureLen)

  *Crypto, Asymmetric key Verification Functions.*
- void TEE_DeriveKey (TEE_OperationHandle operation, const TEE_Attribute *params, uint32_t paramCount, TEE_ObjectHandle derivedKey)
- void TEE_GenerateRandom (void *randomBuffer, uint32_t randomBufferLen)

  *Crypto, common.*
- void TEE_GetSystemTime (TEE_Time *time)

  *Core Functions, Time Functions.*
- TEE_Result TEE_Wait (uint32_t timeout)
- TEE_Result TEE_GetTAPersistentTime (TEE_Time *time)
- TEE_Result TEE_SetTAPersistentTime (const TEE_Time *time)
- void TEE_GetREETime (TEE_Time *time)

  *Core Functions, Time Functions.*
- uint32_t TEE_BigIntFMMSizeInU32 (uint32_t modulusSizeInBits)
- uint32_t TEE_BigIntFMMContextSizeInU32 (uint32_t modulusSizeInBits)
- void TEE_BigIntInit (TEE_BigInt *bigInt, uint32_t len)
- void TEE_BigIntInitFMMContext (TEE_BigIntFMMContext *context, uint32_t len, const TEE_BigInt *modulus)
- void TEE_BigIntInitFMM (TEE_BigIntFMM *bigIntFMM, uint32_t len)
- TEE_Result TEE_BigIntConvertFromOctetString (TEE_BigInt *dest, const uint8_t *buffer, uint32_t bufferLen, int32_t sign)
- TEE_Result TEE_BigIntConvertToOctetString (uint8_t *buffer, uint32_t *bufferLen, const TEE_BigInt *bigInt)
- void TEE_BigIntConvertFromS32 (TEE_BigInt *dest, int32_t shortVal)
- TEE_Result TEE_BigIntConvertToS32 (int32_t *dest, const TEE_BigInt *src)
- int32_t TEE_BigIntCmp (const TEE_BigInt *op1, const TEE_BigInt *op2)
- int32_t TEE_BigIntCmpS32 (const TEE_BigInt *op, int32_t shortVal)
- void TEE_BigIntShiftRight (TEE_BigInt *dest, const TEE_BigInt *op, size_t bits)
- bool TEE_BigIntGetBit (const TEE_BigInt *src, uint32_t bitIndex)
- uint32_t TEE_BigIntGetBitCount (const TEE_BigInt *src)
- void TEE_BigIntAdd (TEE_BigInt *dest, const TEE_BigInt *op1, const TEE_BigInt *op2)
- void TEE_BigIntSub (TEE_BigInt *dest, const TEE_BigInt *op1, const TEE_BigInt *op2)
- void TEE_BigIntNeg (TEE_BigInt *dest, const TEE_BigInt *op)
- void TEE_BigIntMul (TEE_BigInt *dest, const TEE_BigInt *op1, const TEE_BigInt *op2)
- void TEE_BigIntSquare (TEE_BigInt *dest, const TEE_BigInt *op)
- void TEE_BigIntDiv (TEE_BigInt *dest_q, TEE_BigInt *dest_r, const TEE_BigInt *op1, const TEE_BigInt *op2)
- void TEE_BigIntMod (TEE_BigInt *dest, const TEE_BigInt *op, const TEE_BigInt *n)

- void TEE_BigIntAddMod (TEE_BigInt *dest, const TEE_BigInt *op1, const TEE_BigInt *op2, const TEE_BigInt *n)
- void TEE_BigIntSubMod (TEE_BigInt *dest, const TEE_BigInt *op1, const TEE_BigInt *op2, const TEE_BigInt *n)
- void TEE_BigIntMulMod (TEE_BigInt *dest, const TEE_BigInt *op1, const TEE_BigInt *op2, const TEE_BigInt *n)
- void TEE_BigIntSquareMod (TEE_BigInt *dest, const TEE_BigInt *op, const TEE_BigInt *n)
- void TEE_BigIntInvMod (TEE_BigInt *dest, const TEE_BigInt *op, const TEE_BigInt *n)
- bool TEE_BigIntRelativePrime (const TEE_BigInt *op1, const TEE_BigInt *op2)
- void TEE_BigIntComputeExtendedGcd (TEE_BigInt *gcd, TEE_BigInt *u, TEE_BigInt *v, const TEE_BigInt *op1, const TEE_BigInt *op2)
- int32_t TEE_BigIntIsProbablePrime (const TEE_BigInt *op, uint32_t confidenceLevel)
- void TEE_BigIntConvertToFMM (TEE_BigIntFMM *dest, const TEE_BigInt *src, const TEE_BigInt *n, const TEE_BigIntFMMContext *context)
- void TEE_BigIntConvertFromFMM (TEE_BigInt *dest, const TEE_BigIntFMM *src, const TEE_BigInt *n, const TEE_BigIntFMMContext *context)
- void TEE_BigIntFMMConvertToBigInt (TEE_BigInt *dest, const TEE_BigIntFMM *src, const TEE_BigInt *n, const TEE_BigIntFMMContext *context)
- void TEE_BigIntComputeFMM (TEE_BigIntFMM *dest, const TEE_BigIntFMM *op1, const TEE_BigIntFMM *op2, const TEE_BigInt *n, const TEE_BigIntFMMContext *context)

### 4.9.1 Function Documentation

#### 4.9.1.1 TEE_AEDecryptFinal()

```
TEE_Result TEE_AEDecryptFinal (
        TEE_OperationHandle operation,
        const void * srcData,
        uint32_t srcLen,
        void * destData,
        uint32_t * destLen,
        void * tag,
        uint32_t tagLen )
```

Crypto, Authenticated Encryption with Symmetric key Verification Functions.

TEE_AEDecryptFinal() - Processes data that has not been processed by previous calls to TEE_AEUpdate as well as data supplied in srcData.

This function completes the AE operation and compares the computed tag with the tag supplied in the parameter tag .The operation handle can be reused or newly initialized.The buffers srcData and destData shall be either completely disjoint or equal in their starting positions.The operation may be in either initial or active state and enters initial state afterwards.

**Parameters**

| | |
|---|---|
| *operation* | Handle of a running AE operation |
| *srcData* | Reference to final chunk of input data to be encrypted |
| *srcLen* | length of the input data |
| *destData* | Output buffer. Can be omitted if the output is to be discarded. |
| *destLen* | length of the buffer. |
| *tag* | Output buffer filled with the computed tag |
| *tagLen* | length of the tag. |

**Returns**

0 on success.

TEE_ERROR_SHORT_BUFFER If the output buffer is not large enough to contain the output

TEE_ERROR_MAC_INVALID If the computed tag does not match the supplied tag

#### 4.9.1.2 TEE_AEEncryptFinal()

```
TEE_Result TEE_AEEncryptFinal (
        TEE_OperationHandle operation,
        const void * srcData,
        uint32_t srcLen,
        void * destData,
        uint32_t * destLen,
        void * tag,
        uint32_t * tagLen )
```

Crypto, Authenticated Encryption with Symmetric key Verification Functions.

TEE_AEEncryptFinal() - processes data that has not been processed by previous calls to TEE_AEUpdate as well as data supplied in srcData .

TEE_AEEncryptFinal completes the AE operation and computes the tag. The operation handle can be reused or newly initialized. The buffers srcData and destData SHALL be either completely disjoint or equal in their starting positions.The operation may be in either initial or active state and enters initial state afterwards.

**Parameters**

| | |
|---|---|
| *operation* | Handle of a running AE operation |
| *srcData* | Reference to final chunk of input data to be encrypted |
| *srcLen* | length of the input data |
| *destData* | Output buffer. Can be omitted if the output is to be discarded. |
| *destLen* | length of the buffer. |
| *tag* | Output buffer filled with the computed tag |
| *tagLen* | length of the tag. |

**Returns**

0 on success.

TEE_ERROR_SHORT_BUFFER If the output or tag buffer is not large enoughto contain the output.

#### 4.9.1.3 TEE_AEInit()

```
TEE_Result TEE_AEInit (
        TEE_OperationHandle operation,
        const void * nonce,
        uint32_t nonceLen,
        uint32_t tagLen,
        uint32_t AADLen,
        uint32_t payloadLen )
```

Crypto, Authenticated Encryption with Symmetric key Verification Functions.

TEE_AEInit() - Initializes an Authentication Encryption operation.

The operation must be in initial state and remains in the initial state afterwards.

**Parameters**

| | |
|---|---|
| *operation* | A handle on the operation. |
| *nonce* | The operation nonce or IV |
| *nonceLen* | length of nonce |
| *tagLen* | Size in bits of the tag |
| *AADLen* | Length in bytes of the AAD |
| *payloadLen* | Length in bytes of the payload. |

**Returns**

0 on success.

TEE_ERROR_NOT_SUPPORTED If the tag length is not supported by the algorithm.

#### 4.9.1.4 TEE_AEUpdate()

```
TEE_Result TEE_AEUpdate (
        TEE_OperationHandle operation,
        const void * srcData,
        uint32_t srcLen,
        void * destData,
        uint32_t * destLen )
```

Crypto, Authenticated Encryption with Symmetric key Verification Functions.

TEE_AEUpdate() - Accumulates data for an Authentication Encryption operation

This function describes Input data does not have to be a multiple of block size. Subsequent calls to this function are possible. Unless one or more calls of this function have supplied sufficient input data,no output is generated. when using this routine to decrypt the returned data may be corrupt since the integrity check is not performed until all the data has been processed. If this is a concern then only use the TEE_AEDecryptFinal routine.

**Parameters**

| | |
|---|---|
| *operation* | Handle of a running AE operation. |
| *srcData* | Input data buffer to be encrypted or decrypted |
| *srcLen* | length of the input buffer. |
| *destData* | Output buffer |
| *destLen* | length of the out put buffer. |

**Returns**

0 on success.

TEE_ERROR_SHORT_BUFFER if the output buffer is not large enough to contain the output.

#### 4.9.1.5 TEE_AEUpdateAAD()

```
void TEE_AEUpdateAAD (
        TEE_OperationHandle operation,
        const void * AADdata,
        uint32_t AADdataLen )
```

Crypto, Authenticated Encryption with Symmetric key Verification Functions.

TEE_AEUpdateAAD() - Feeds a new chunk of Additional Authentication Data (AAD) to the AE operation. Subsequent calls to this function are possible.

The TEE_AEUpdateAAD function feeds a new chunk of Additional Authentication Data (AAD) to the AE operation. Subsequent calls to this function are possible.The buffers srcData and destData shall be either completely disjoint or equal in their starting positions.The operation SHALL be in initial state and remains in initial state afterwards.

**Parameters**

| | |
|---|---|
| *operation* | Handle on the AE operation |
| *AADdata* | Input buffer containing the chunk of AAD |
| *AADdataLen* | length of the chunk of AAD. |

#### 4.9.1.6 TEE_AllocateOperation()

```
TEE_Result TEE_AllocateOperation (
        TEE_OperationHandle * operation,
        uint32_t algorithm,
        uint32_t mode,
        uint32_t maxKeySize )
```

Crypto, for all Crypto Functions.

TEE_AllocateOperation() - Allocates a handle for a new cryptographic operation and sets the mode and algorithm type.

If this function does not return with TEE_SUCCESS then there is no valid handle value.Once a cryptographic operation has been created, the implementation shall guarantee that all resources necessary for the operation are allocated and that any operation with a key of at most maxKeySize bits can be performed. For algorithms that take multiple keys, for example the AES XTS algorithm, the maxKeySize parameter specifies the size of the largest key. It is up to the implementation to properly allocate space for multiple keys if the algorithm so requires.

**Parameters**

| | |
|---|---|
| *operation* | reference to generated operation handle. |
| *algorithm* | One of the cipher algorithms. |
| *mode* | The operation mode. |
| *maxKeySize* | Maximum key size in bits for the operation. |

**Returns**

0 in case of success

TEE_ERROR_OUT_OF_MEMORY If there are not enough resources to allocate the operation.

TEE_ERROR_NOT_SUPPORTED If the mode is not compatible with the algorithm or key size or if the algorithm is not one of the listed algorithms or if maxKeySize is not appropriate for the algorithm.

#### 4.9.1.7 TEE_AllocatePersistentObjectEnumerator() TEE_Result

```
TEE_AllocatePersistentObjectEnumerator (
            TEE_ObjectEnumHandle * objectEnumerator )
```

#### 4.9.1.8 TEE_AllocatePropertyEnumerator() TEE_Result TEE_AllocatePropertyEnumerator (

```
            TEE_PropSetHandle * enumerator )
```

#### 4.9.1.9 TEE_AllocateTransientObject() TEE_Result TEE_AllocateTransientObject (

```
            TEE_ObjectType objectType,
            uint32_t maxKeySize,
            TEE_ObjectHandle * object )
```

Crypto, Asymmetric key Verification Functions.

TEE_AllocateTransientObject() - Allocates an uninitialized transient object. Transient objects are used to hold a cryptographic object (key or key-pair).

The value TEE_KEYSIZE_NO_KEY should be used for maxObjectSize for object types that do not require a key so that all the container resources can be pre-allocated. As allocated, the container is uninitialized. It can be initialized by subsequently importing the object material,generating an object, deriving an object, or loading an object from the Trusted Storage.

**Parameters**

| | |
|---|---|
| *objectType* | Type of uninitialized object container to be created |
| *maxKeySize* | Key Size of the object. |
| *object* | Filled with a handle on the newly created key container. |

**Returns**

0 on success

TEE_ERROR_OUT_OF_MEMORY If not enough resources are available to allocate the object handle.

TEE_ERROR_NOT_SUPPORTED If the key size is not supported or the object type is not supported.

#### 4.9.1.10 TEE_AsymmetricDecrypt()

```
TEE_Result TEE_AsymmetricDecrypt (
    TEE_OperationHandle operation,
    const TEE_Attribute * params,
    uint32_t paramCount,
    const void * srcData,
    uint32_t srcLen,
    void * destData,
    uint32_t * destLen )
```

#### 4.9.1.11 TEE_AsymmetricEncrypt()

```
TEE_Result TEE_AsymmetricEncrypt (
    TEE_OperationHandle operation,
    const TEE_Attribute * params,
    uint32_t paramCount,
    const void * srcData,
    uint32_t srcLen,
    void * destData,
    uint32_t * destLen )
```

#### 4.9.1.12 TEE_AsymmetricSignDigest()

```
TEE_Result TEE_AsymmetricSignDigest (
    TEE_OperationHandle operation,
    const TEE_Attribute * params,
    uint32_t paramCount,
    const void * digest,
    uint32_t digestLen,
    void * signature,
    uint32_t * signatureLen )
```

Crypto, Asymmetric key Verification Functions.

TEE_AsymmetricSignDigest() - Signs a message digest within an asymmetric operation.

**Parameters**

| | |
|---|---|
| *operation* | Handle on the operation, which SHALL have been suitably set up with an operation key. |
| *params* | Optional operation parameters |
| *paramCount* | size of param |
| *digest* | Input buffer containing the input message digest |
| *digestLen* | length of input buffer. |
| *signature* | Output buffer written with the signature of the digest |
| *signatureLen* | length of output buffer. |

**Returns**

0 on sccess

TEE_ERROR_SHORT_BUFFER If the signature buffer is not large enough to hold the result

#### 4.9.1.13 TEE_AsymmetricVerifyDigest()

```
TEE_Result TEE_AsymmetricVerifyDigest (
    TEE_OperationHandle operation,
    const TEE_Attribute * params,
    uint32_t paramCount,
    const void * digest,
    uint32_t digestLen,
    const void * signature,
    uint32_t signatureLen )
```

Crypto, Asymmetric key Verification Functions.

TEE_AsymmetricVerifyDigest() - verifies a message digest signature within an asymmetric operation.

This function describes the message digest signature verify by calling ed25519_verify().

**Parameters**

| | |
|---|---|
| *operation* | Handle on the operation, which SHALL have been suitably set up with an operation key. |
| *params* | Optional operation parameters |
| *paramCount* | size of param. |
| *digest* | Input buffer containing the input message digest |
| *digestLen* | length of input buffer. |
| *signature* | Output buffer written with the signature of the digest |
| *signatureLen* | length of output buffer. |

**Returns**

TEE_SUCCESS on success

TEE_ERROR_SIGNATURE_INVALID if the signature is invalid.

#### 4.9.1.14 TEE_BigIntAdd()

```
void TEE_BigIntAdd (
    TEE_BigInt * dest,
    const TEE_BigInt * op1,
    const TEE_BigInt * op2 )
```

#### 4.9.1.15 TEE_BigIntAddMod()

```
void TEE_BigIntAddMod (
    TEE_BigInt * dest,
    const TEE_BigInt * op1,
    const TEE_BigInt * op2,
    const TEE_BigInt * n )
```

#### 4.9.1.16 TEE_BigIntCmp()

```
int32_t TEE_BigIntCmp (
    const TEE_BigInt * op1,
    const TEE_BigInt * op2 )
```

#### 4.9.1.17 TEE_BigIntCmpS32()

```
int32_t TEE_BigIntCmpS32 (
    const TEE_BigInt * op,
    int32_t shortVal )
```

#### 4.9.1.18 TEE_BigIntComputeExtendedGcd()

```
void TEE_BigIntComputeExtendedGcd (
    TEE_BigInt * gcd,
    TEE_BigInt * u,
    TEE_BigInt * v,
    const TEE_BigInt * op1,
    const TEE_BigInt * op2 )
```

#### 4.9.1.19 TEE_BigIntComputeFMM()

```
void TEE_BigIntComputeFMM (
    TEE_BigIntFMM * dest,
    const TEE_BigIntFMM * op1,
    const TEE_BigIntFMM * op2,
    const TEE_BigInt * n,
    const TEE_BigIntFMMContext * context )
```

#### 4.9.1.20 TEE_BigIntConvertFromFMM()

```
void TEE_BigIntConvertFromFMM (
    TEE_BigInt * dest,
    const TEE_BigIntFMM * src,
    const TEE_BigInt * n,
    const TEE_BigIntFMMContext * context )
```

#### 4.9.1.21 TEE_BigIntConvertFromOctetString()

```
TEE_Result TEE_BigIntConvertFromOctetString (
    TEE_BigInt * dest,
    const uint8_t * buffer,
    uint32_t bufferLen,
    int32_t sign )
```

#### 4.9.1.22 TEE_BigIntConvertFromS32()

```
void TEE_BigIntConvertFromS32 (
    TEE_BigInt * dest,
    int32_t shortVal )
```

#### 4.9.1.23 TEE_BigIntConvertToFMM()

```
void TEE_BigIntConvertToFMM (
    TEE_BigIntFMM * dest,
    const TEE_BigInt * src,
    const TEE_BigInt * n,
    const TEE_BigIntFMMContext * context )
```

#### 4.9.1.24 TEE_BigIntConvertToOctetString()

```
TEE_Result TEE_BigIntConvertToOctetString (
    uint8_t * buffer,
    uint32_t * bufferLen,
    const TEE_BigInt * bigInt )
```

#### 4.9.1.25 TEE_BigIntConvertToS32()

```
TEE_Result TEE_BigIntConvertToS32 (
    int32_t * dest,
    const TEE_BigInt * src )
```

#### 4.9.1.26 TEE_BigIntDiv()

```
void TEE_BigIntDiv (
    TEE_BigInt * dest_q,
    TEE_BigInt * dest_r,
    const TEE_BigInt * op1,
    const TEE_BigInt * op2 )
```

#### 4.9.1.27 TEE_BigIntFMMContextSizeInU32()

```
uint32_t TEE_BigIntFMMContextSizeInU32 (
    uint32_t modulusSizeInBits )
```

#### 4.9.1.28 TEE_BigIntFMMConvertToBigInt()

```
void TEE_BigIntFMMConvertToBigInt (
    TEE_BigInt * dest,
    const TEE_BigIntFMM * src,
    const TEE_BigInt * n,
    const TEE_BigIntFMMContext * context )
```

#### 4.9.1.29 TEE_BigIntFMMSizeInU32()

```
uint32_t TEE_BigIntFMMSizeInU32 (
    uint32_t modulusSizeInBits )
```

#### 4.9.1.30 TEE_BigIntGetBit()

```
bool TEE_BigIntGetBit (
    const TEE_BigInt * src,
    uint32_t bitIndex )
```

#### 4.9.1.31 TEE_BigIntGetBitCount()

```
uint32_t TEE_BigIntGetBitCount (
    const TEE_BigInt * src )
```

#### 4.9.1.32 TEE_BigIntInit()

```
void TEE_BigIntInit (
    TEE_BigInt * bigInt,
    uint32_t len )
```

#### 4.9.1.33 TEE_BigIntInitFMM()

```
void TEE_BigIntInitFMM (
    TEE_BigIntFMM * bigIntFMM,
    uint32_t len )
```

#### 4.9.1.34 TEE_BigIntInitFMMContext()

```
void TEE_BigIntInitFMMContext (
    TEE_BigIntFMMContext * context,
    uint32_t len,
    const TEE_BigInt * modulus )
```

#### 4.9.1.35 TEE_BigIntInvMod()

```
void TEE_BigIntInvMod (
    TEE_BigInt * dest,
    const TEE_BigInt * op,
    const TEE_BigInt * n )
```

#### 4.9.1.36 TEE_BigIntIsProbablePrime()

```
int32_t TEE_BigIntIsProbablePrime (
    const TEE_BigInt * op,
    uint32_t confidenceLevel )
```

#### 4.9.1.37 TEE_BigIntMod()

```
void TEE_BigIntMod (
    TEE_BigInt * dest,
    const TEE_BigInt * op,
    const TEE_BigInt * n )
```

#### 4.9.1.38 TEE_BigIntMul()

```
void TEE_BigIntMul (
    TEE_BigInt * dest,
    const TEE_BigInt * op1,
    const TEE_BigInt * op2 )
```

#### 4.9.1.39 TEE_BigIntMulMod()

```
void TEE_BigIntMulMod (
    TEE_BigInt * dest,
    const TEE_BigInt * op1,
    const TEE_BigInt * op2,
    const TEE_BigInt * n )
```

#### 4.9.1.40 TEE_BigIntNeg()

```
void TEE_BigIntNeg (
    TEE_BigInt * dest,
    const TEE_BigInt * op )
```

#### 4.9.1.41 TEE_BigIntRelativePrime()

```
bool TEE_BigIntRelativePrime (
    const TEE_BigInt * op1,
    const TEE_BigInt * op2 )
```

#### 4.9.1.42 TEE_BigIntShiftRight()

```
void TEE_BigIntShiftRight (
    TEE_BigInt * dest,
    const TEE_BigInt * op,
    size_t bits )
```

#### 4.9.1.43 TEE_BigIntSquare()

```
void TEE_BigIntSquare (
    TEE_BigInt * dest,
    const TEE_BigInt * op )
```

#### 4.9.1.44 TEE_BigIntSquareMod()

```
void TEE_BigIntSquareMod (
    TEE_BigInt * dest,
    const TEE_BigInt * op,
    const TEE_BigInt * n )
```

#### 4.9.1.45 TEE_BigIntSub()

```
void TEE_BigIntSub (
    TEE_BigInt * dest,
    const TEE_BigInt * op1,
    const TEE_BigInt * op2 )
```

#### 4.9.1.46 TEE_BigIntSubMod()

```
void TEE_BigIntSubMod (
    TEE_BigInt * dest,
    const TEE_BigInt * op1,
    const TEE_BigInt * op2,
    const TEE_BigInt * n )
```

#### 4.9.1.47 TEE_CheckMemoryAccessRights()

```
TEE_Result TEE_CheckMemoryAccessRights (
    uint32_t accessFlags,
    void * buffer,
    uint32_t size )
```

#### 4.9.1.48 TEE_CipherDoFinal()

```
TEE_Result TEE_CipherDoFinal (
    TEE_OperationHandle operation,
    const void * srcData,
    uint32_t srcLen,
    void * destData,
    uint32_t * destLen )
```

TEE_CipherDoFinal() - Finalizes the cipher operation, processing data that has not been processed by previous calls to TEE_CipherUpdate as well as data supplied in srcData .

This function describes The operation handle can be reused or re-initialized. The buffers srcData and destData shall be either completely disjoint or equal in their starting positions.The operation SHALL be in active state and is set to initial state afterwards.

**Parameters**

| | |
|---|---|
| *operation* | Handle of a running Cipher operation |
| *srcData* | Input data buffer to be encrypted or decrypted |
| *srcLen* | length of input buffer |
| *destData* | output buffer |
| *destLen* | ouput buffer length. |

**Returns**

0 on success

TEE_ERROR_SHORT_BUFFER If the output buffer is not large enough to contain the output

#### 4.9.1.49 TEE_CipherInit()

```
void TEE_CipherInit (
    TEE_OperationHandle operation,
    const void * nonce,
    uint32_t nonceLen )
```

Crypto, Authenticated Encryption with Symmetric key Verification Functions.

TEE_CipherInit() - starts the symmetric cipher operation.

The operation shall have been associated with a key. If the operation is in active state, it is reset and then initialized. If the operation is in initial state, it is moved to active state.

**Parameters**

| | |
|---|---|
| *operation* | A handle on an opened cipher operation setup with a key |
| *nonce* | Buffer containing the operation Initialization Vector as appropriate. |
| *nonceLen* | length of the buffer |

#### 4.9.1.50 TEE_CipherUpdate()

```
TEE_Result TEE_CipherUpdate (
    TEE_OperationHandle operation,
    const void * srcData,
    uint32_t srcLen,
    void * destData,
    uint32_t * destLen )
```

Crypto, Authenticated Encryption with Symmetric key Verification Functions.

TEE_CipherUpdate() - encrypts or decrypts input data.

Input data does not have to be a multiple of block size. Subsequent calls to this function are possible. Unless one or more calls of this function have supplied sufficient input data, no output is generated. The cipher operation is finalized with a call to TEE_CipherDoFinal .The buffers srcData and destData SHALL be either completely disjoint or equal in their starting positions.The operation SHALL be in active state.

**Parameters**

| | |
|---|---|
| *operation* | Handle of a running Cipher operation |
| *srcData* | Input data buffer to be encrypted or decrypted |
| *srcLen* | length of input buffer |
| *destData* | output buffer |
| *destLen* | ouput buffer length. |

**Returns**

0 on success else

TEE_ERROR_SHORT_BUFFER If the output buffer is not large enough to contain the output. In this case, the input is not fed into the algorithm.

#### 4.9.1.51 TEE_CloseAndDeletePersistentObject()

```
void TEE_CloseAndDeletePersistentObject (
    TEE_ObjectHandle object )
```

#### 4.9.1.52 TEE_CloseAndDeletePersistentObject1()

```
TEE_Result TEE_CloseAndDeletePersistentObject1 (
    TEE_ObjectHandle object )
```

#### 4.9.1.53 TEE_CloseObject() `void TEE_CloseObject (`

`TEE_ObjectHandle object )`

Core Functions, Secure Storage Functions (data is isolated for each TA)

TEE_CloseObject() - Closes an opened object handle.

The object can be persistent or transient.For transient objects, TEE_CloseObject is equivalent to TEE_Free↩ TransientObject.

**Parameters**

| | |
|---|---|
| *object* | Handle of the object. |

**Returns**

TEE_SUCCESS if success else error occured.

TEE_CloseObject() - Function closes an opened object handle.

The object can be persistent or transient.For transient objects, TEE_CloseObject is equivalent to TEE_Free↩ TransientObject.

**Parameters**

| | |
|---|---|
| *object* | Handle of the object |

**Returns**

TEE_SUCCESS if success else error occured.

#### 4.9.1.54 TEE_CloseTASession() `void TEE_CloseTASession (`

`TEE_TASessionHandle session )`

#### 4.9.1.55 TEE_CopyObjectAttributes() `void TEE_CopyObjectAttributes (`

`TEE_ObjectHandle destObject,`
`TEE_ObjectHandle srcObject )`

#### 4.9.1.56 TEE_CopyObjectAttributes1() `TEE_Result TEE_CopyObjectAttributes1 (`

`TEE_ObjectHandle destObject,`
`TEE_ObjectHandle srcObject )`

#### 4.9.1.57 TEE_CopyOperation() void TEE_CopyOperation (

```
TEE_OperationHandle dstOperation,
TEE_OperationHandle srcOperation )
```

#### 4.9.1.58 TEE_CreatePersistentObject() TEE_Result TEE_CreatePersistentObject (

```
uint32_t storageID,
const void * objectID,
uint32_t objectIDLen,
uint32_t flags,
TEE_ObjectHandle attributes,
const void * initialData,
uint32_t initialDataLen,
TEE_ObjectHandle * object )
```

Core Functions, Secure Storage Functions (data is isolated for each TA)

TEE_CreatePersistentObject() - Creates a persistent object with initial attributes.

In this function an initial data stream content returns either a handle on the created object or TEE_HANDLE_NULL upon failure.

**Parameters**

| | |
|---|---|
| *storageID* | The storage to use. |
| *objectID* | The object identifier |
| *objectIDLen* | The object identifier |
| *flags* | The flags which determine the settings under which the object is opened. |
| *attributes* | A handle on a persistent object or an initialized transient object from which to take the persistent object attributes |
| *initialData* | The initial data content of the persistent object |
| *initialDataLen* | The initial data content of the persistent object |
| *object* | A pointer to the handle which contains the opened handle upon successful completion |

**Returns**

0 if success else error occured.

TEE_CreatePersistentObject() - Creates a persistent object with initial attributes.

An initial data stream content, and optionally returns either a handle on the created object, or TEE_HANDLE_NULL upon failure.

**Parameters**

| | |
|---|---|
| *storageID* | The storage to use. |
| *objectID* | The object identifier |
| *objectIDLen* | The object identifier |
| *flags* | The flags which determine the settings under which the object is opened. |

Paramter list continued on next page

| | |
|---|---|
| *attributes* | A handle on a persistent object or an initialized transient object from which to take the persistent object attributes |
| *initialData* | The initial data content of the persistent object |
| *initialDataLen* | The initial data content of the persistent object |
| *object* | A pointer to the handle, which contains the opened handle upon successful completion |

**Returns**

0 if success, else error occured.

#### 4.9.1.59 TEE_DeriveKey()

```
void TEE_DeriveKey (
    TEE_OperationHandle operation,
    const TEE_Attribute * params,
    uint32_t paramCount,
    TEE_ObjectHandle derivedKey )
```

#### 4.9.1.60 TEE_DigestDoFinal()

```
TEE_Result TEE_DigestDoFinal (
    TEE_OperationHandle operation,
    const void * chunk,
    uint32_t chunkLen,
    void * hash,
    uint32_t * hashLen )
```

TEE_DigestDoFinal() - Finalizes the message digest operation and produces the message hash.

This function finalizes the message digest operation and produces the message hash. Afterwards the Message Digest operation is reset to initial state and can be reused.

**Parameters**

| | |
|---|---|
| *operation* | Handle of a running Message Digest operation. |
| *chunk* | Chunk of data to be hashed. |
| *chunkLen* | size of the chunk. |
| *hash* | Output buffer filled with the message hash. |
| *hashLen* | lenth of the mesaage hash. |

**Returns**

0 on success

TEE_ERROR_SHORT_BUFFER If the output buffer is too small. In this case, the operation is not finalized.

#### 4.9.1.61 TEE_DigestUpdate() `void TEE_DigestUpdate (`

```
TEE_OperationHandle operation,
const void * chunk,
uint32_t chunkSize )
```

Crypto, Message Digest Functions.

TEE_DigestUpdate()- Accumulates message data for hashing.

This function describes the message does not have to be block aligned. Subsequent calls to this function are possible.The operation may be in either initial or active state and becomes active.

**Parameters**

| | |
|---|---|
| *operation* | Handle of a running Message Digest operation. |
| *chunk* | Chunk of data to be hashed |
| *chunkSize* | size of the chunk. |

#### 4.9.1.62 TEE_Free() `void TEE_Free (`

```
void * buffer )
```

TEE_Free() - causes the space pointed to by buffer to be deallocated;that is made available for further allocation.

This function describes if buffer is a NULL pointer, TEE_Free does nothing. Otherwise, it is a Programmer Error if the argument does not match a pointer previously returned by the TEE_Malloc or TEE_Realloc if the space has been deallocated by a call to TEE_Free or TEE_Realloc.

**Parameters**

| | |
|---|---|
| *buffer* | The pointer to the memory block to be freed. |

#### 4.9.1.63 TEE_FreeOperation() `void TEE_FreeOperation (`

```
TEE_OperationHandle operation )
```

Crypto, for all Crypto Functions.

TEE_FreeOperation() - Deallocates all resources associated with an operation handle.

This function deallocates all resources associated with an operation handle. After this function is called, the operation handle is no longer valid. All cryptographic material in the operation is destroyed. The function does nothing if operation is TEE_HANDLE_NULL.

**Parameters**

| | |
|---|---|
| *operation* | Reference to operation handle. |

**Returns**

nothing after the operation free.

#### 4.9.1.64 TEE_FreePersistentObjectEnumerator()

```
void TEE_FreePersistentObjectEnumerator (
    TEE_ObjectEnumHandle objectEnumerator )
```

#### 4.9.1.65 TEE_FreePropertyEnumerator()

```
void TEE_FreePropertyEnumerator (
    TEE_PropSetHandle enumerator )
```

#### 4.9.1.66 TEE_FreeTransientObject()

```
void TEE_FreeTransientObject (
    TEE_ObjectHandle object )
```

Crypto, Asymmetric key Verification Functions.

TEE_FreeTransientObject() - Deallocates a transient object previously allocated with TEE_AllocateTransientObject .

this function describes the object handle is no longer valid and all resources associated with the transient object shall have been reclaimed after the TEE_AllocateTransientObject() call.

**Parameters**

| | |
|---|---|
| *object* | Handle on the object to free. |

#### 4.9.1.67 TEE_GenerateKey()

```
TEE_Result TEE_GenerateKey (
    TEE_ObjectHandle object,
    uint32_t keySize,
    const TEE_Attribute * params,
    uint32_t paramCount )
```

Crypto, Asymmetric key Verification Functions.

TEE_GenerateKey () - Generates a random key or a key-pair and populates a transient key object with the generated key material.

The size of the desired key is passed in the keySize parameter and shall be less than or equal to the maximum key size specified when the transient object was created.

**Parameters**

| | |
|---|---|
| *object* | Handle on an uninitialized transient key to populate with the generated key. |
| *keySize* | Requested key size shall be less than or equal to the maximum key size specified when the object container was created |
| *params* | Parameters for the key generation. |
| *paramCount* | The values of all parameters are copied nto the object so that the params array and all the memory buffers it points to may be freed after this routine returns without affecting the object. |

**Returns**

0 on succes

TEE_ERROR_BAD_PARAMETERS If an incorrect or inconsistent attribute is detected. The checks that are performed depend on the implementation.

#### 4.9.1.68 TEE_GenerateRandom() `void TEE_GenerateRandom (`

```
void * randomBuffer,
uint32_t randomBufferLen )
```

Crypto, common.

TEE_GenerateRandom() - Generates random data.

This function generates random data of random buffer length and is stored in to random Buffer by calling wc_↩ RNG_GenerateBlock().If ret is not equal
to 0 then TEE_Panic is called.

**Parameters**

| | |
|---|---|
| *randomBuffer* | Reference to generated random data |
| *randomBufferLen* | Byte length of requested random data |

**Returns**

random data random data will be returned.

TEE_GenerateRandom() - Generates random data.

This function generates random data of random bufferlength and is stored in to randomBuffer by calling sgx_read↩ _rand().

**Parameters**

| | |
|---|---|
| *randomBuffer* | Reference to generated random data |
| *randomBufferLen* | Byte length of requested random data |

#### 4.9.1.69 TEE_GetCancellationFlag()

```
bool TEE_GetCancellationFlag (
    void  )
```

#### 4.9.1.70 TEE_GetInstanceData()

```
const void * TEE_GetInstanceData (
    void  )
```

#### 4.9.1.71 TEE_GetNextPersistentObject()

```
TEE_Result TEE_GetNextPersistentObject (
    TEE_ObjectEnumHandle objectEnumerator,
    TEE_ObjectInfo * objectInfo,
    void * objectID,
    uint32_t * objectIDLen )
```

#### 4.9.1.72 TEE_GetNextProperty()

```
TEE_Result TEE_GetNextProperty (
    TEE_PropSetHandle enumerator )
```

#### 4.9.1.73 TEE_GetObjectBufferAttribute()

```
TEE_Result TEE_GetObjectBufferAttribute (
    TEE_ObjectHandle object,
    uint32_t attributeID,
    void * buffer,
    uint32_t * size )
```

#### 4.9.1.74 TEE_GetObjectInfo()

```
void TEE_GetObjectInfo (
    TEE_ObjectHandle object,
    TEE_ObjectInfo * objectInfo )
```

#### 4.9.1.75 TEE_GetObjectInfo1()

```
TEE_Result TEE_GetObjectInfo1 (
    TEE_ObjectHandle object,
    TEE_ObjectInfo * objectInfo )
```

Core Functions, Secure Storage Functions (data is isolated for each TA)

TEE_GetObjectInfo1() - Returns the characteristics of an object.

This function returns a handle which can be used to access the object's attributes and data stream.

**Parameters**

| | |
|---|---|
| *objectInfo* | Pointer to a structure filled with the object information |
| *object* | Handle of the object |

**Returns**

0 if success else error occured.

TEE_GetObjectInfo1() - Function returns the characteristics of an object.

It returns a handle that can be used to access the object's attributes and data stream.

**Parameters**

| | |
|---|---|
| *objectInfo* | Pointer to a structure filled with the object information |
| *object* | Handle of the object |

**Returns**

0 if success else error occured.

#### 4.9.1.76 TEE_GetObjectValueAttribute()

```
TEE_Result TEE_GetObjectValueAttribute (
    TEE_ObjectHandle object,
    uint32_t attributeID,
    uint32_t * a,
    uint32_t * b )
```

#### 4.9.1.77 TEE_GetOperationInfo()

```
void TEE_GetOperationInfo (
    TEE_OperationHandle operation,
    TEE_OperationInfo * operationInfo )
```

#### 4.9.1.78 TEE_GetOperationInfoMultiple()

```
TEE_Result TEE_GetOperationInfoMultiple (
    TEE_OperationHandle operation,
    TEE_OperationInfoMultiple * operationInfoMultiple,
    uint32_t * operationSize )
```

#### 4.9.1.79 TEE_GetPropertyAsBinaryBlock()

```
TEE_Result TEE_GetPropertyAsBinaryBlock (
    TEE_PropSetHandle propsetOrEnumerator,
    const char * name,
    void * valueBuffer,
    uint32_t * valueBufferLen )
```

#### 4.9.1.80 TEE_GetPropertyAsBool()

```
TEE_Result TEE_GetPropertyAsBool (
    TEE_PropSetHandle propsetOrEnumerator,
    const char * name,
    bool * value )
```

#### 4.9.1.81 TEE_GetPropertyAsIdentity()

```
TEE_Result TEE_GetPropertyAsIdentity (
    TEE_PropSetHandle propsetOrEnumerator,
    const char * name,
    TEE_Identity * value )
```

#### 4.9.1.82 TEE_GetPropertyAsString()

```
TEE_Result TEE_GetPropertyAsString (
    TEE_PropSetHandle propsetOrEnumerator,
    const char * name,
    char * valueBuffer,
    uint32_t * valueBufferLen )
```

#### 4.9.1.83 TEE_GetPropertyAsU32()

```
TEE_Result TEE_GetPropertyAsU32 (
    TEE_PropSetHandle propsetOrEnumerator,
    const char * name,
    uint32_t * value )
```

#### 4.9.1.84 TEE_GetPropertyAsUUID()

```
TEE_Result TEE_GetPropertyAsUUID (
    TEE_PropSetHandle propsetOrEnumerator,
    const char * name,
    TEE_UUID * value )
```

#### 4.9.1.85 TEE_GetPropertyName()

```
TEE_Result TEE_GetPropertyName (
    TEE_PropSetHandle enumerator,
    void * nameBuffer,
    uint32_t * nameBufferLen )
```

#### 4.9.1.86 TEE_GetREETime()

```
void TEE_GetREETime (
    TEE_Time * time )
```

Core Functions, Time Functions.

TEE_GetREETime() - Retrieves the current REE system time.

This function retrieves the current time as seen from the point of view of the REE.

**Parameters**

| | |
|---|---|
| *time* | Filled with the number of seconds and milliseconds |

TEE_GetREETime() - Function retrieves the current REE system time.

This function retrieves the current time as seen from the point of view of the REE.

**Parameters**

| | |
|---|---|
| *time* | Filled with the number of seconds and milliseconds. |

#### 4.9.1.87 TEE_GetSystemTime() `void TEE_GetSystemTime ( TEE_Time * time )`

Core Functions, Time Functions.

TEE_GetSystemTime() - Retrieves the current system time.

This function describes the system time has an arbitrary implementation defined origin that can vary across TA instances. The minimum guarantee is that the system time shall be monotonic for a given TA instance.

**Parameters**

| | |
|---|---|
| *time* | Filled with the number of seconds and milliseconds |

TEE_GetSystemTime() - Retrieves the current system time.

The system time has an arbitrary implementation-defined origin that can vary across TA instances

**Parameters**

| | |
|---|---|
| *time* | Filled with the number of seconds and milliseconds. |

#### 4.9.1.88 TEE_GetTAPersistentTime() `TEE_Result TEE_GetTAPersistentTime ( TEE_Time * time )`

#### 4.9.1.89 TEE_InitRefAttribute()

```
void TEE_InitRefAttribute (
        TEE_Attribute * attr,
        uint32_t attributeID,
        const void * buffer,
        uint32_t length )
```

Crypto, Asymmetric key Verification Functions.

TEE_InitRefAttribute() - The helper function can be used to populate a single attribute either with a reference to a buffer or with integer values.

In TEE_InitRefAttribute () only the buffer pointer is copied, not the content of the buffer. This means that the attribute structure maintains a pointer back to the supplied buffer. It is the responsibility of the TA author to ensure that the contents of the buffer maintain their value until the attributes array is no longer in use.

**Parameters**

| | |
|---|---|
| *attr* | attribute structure to initialize. |
| *attributeID* | Identifier of the attribute to populate. |
| *buffer* | input buffer that holds the content of the attribute. |
| *length* | buffer length. |

#### 4.9.1.90 TEE_InitValueAttribute()

```
void TEE_InitValueAttribute (
        TEE_Attribute * attr,
        uint32_t attributeID,
        uint32_t a,
        uint32_t b )
```

Crypto, Asymmetric key Verification Functions.

TEE_InitValueAttribute() - The helper function can be used to populate a single attribute either with a reference to a buffer or with integer values.

**Parameters**

| | |
|---|---|
| *attr* | attribute structure to initialize. |
| *attributeID* | Identifier of the attribute to populate. |
| *a* | unsigned integer value to assign to the a member of the attribute structure. |
| *b* | unsigned integer value to assign to the b member of the attribute structure |

#### 4.9.1.91 TEE_InvokeTACommand()

```
TEE_Result TEE_InvokeTACommand (
        TEE_TASessionHandle session,
        uint32_t cancellationRequestTimeout,
        uint32_t commandID,
        uint32_t paramTypes,
```

```
    TEE_Param params[TEE_NUM_PARAMS],
    uint32_t * returnOrigin )
```

#### 4.9.1.92 TEE_IsAlgorithmSupported()

```
TEE_Result TEE_IsAlgorithmSupported (
    uint32_t algId,
    uint32_t element )
```

#### 4.9.1.93 TEE_MACCompareFinal()

```
TEE_Result TEE_MACCompareFinal (
    TEE_OperationHandle operation,
    const void * message,
    uint32_t messageLen,
    const void * mac,
    uint32_t macLen )
```

#### 4.9.1.94 TEE_MACComputeFinal()

```
TEE_Result TEE_MACComputeFinal (
    TEE_OperationHandle operation,
    const void * message,
    uint32_t messageLen,
    void * mac,
    uint32_t * macLen )
```

#### 4.9.1.95 TEE_MACInit()

```
void TEE_MACInit (
    TEE_OperationHandle operation,
    const void * IV,
    uint32_t IVLen )
```

#### 4.9.1.96 TEE_MACUpdate()

```
void TEE_MACUpdate (
    TEE_OperationHandle operation,
    const void * chunk,
    uint32_t chunkSize )
```

#### 4.9.1.97 TEE_Malloc()

```
void * TEE_Malloc (
    uint32_t size,
    uint32_t hint )
```

TEE_Malloc() - Allocates space for an object whose size in bytes is specified in the parameter size.

This function describes the pointer returned is guaranteed to be aligned
such that it may be assigned as a pointer to any basic C type.The valid hint values are a bitmask and can be independently set. This parameter allows Trusted Applications to refer to various pools of memory or to request special characteristics for the allocated memory by using an implementation-defined hint. Future versions of this specification may introduce additional standard hints.

**Parameters**

| | |
|---|---|
| *size* | The size of the buffer to be allocated. |
| *hint* | A hint to the allocator. |

**Returns**

Upon successful completion, with size not equal to zero, the function returns a pointer to the allocated space.

#### 4.9.1.98 TEE_MaskCancellation() `bool TEE_MaskCancellation (`

```
void  )
```

#### 4.9.1.99 TEE_MemCompare() `int32_t TEE_MemCompare (`

```
const void * buffer1,
const void * buffer2,
uint32_t size )
```

#### 4.9.1.100 TEE_MemFill() `void * TEE_MemFill (`

```
void * buff,
uint32_t x,
uint32_t size )
```

#### 4.9.1.101 TEE_MemMove() `void * TEE_MemMove (`

```
void * dest,
const void * src,
uint32_t size )
```

#### 4.9.1.102 TEE_OpenPersistentObject() `TEE_Result TEE_OpenPersistentObject (`

```
uint32_t storageID,
const void * objectID,
uint32_t objectIDLen,
uint32_t flags,
TEE_ObjectHandle * object )
```

Core Functions, Secure Storage Functions (data is isolated for each TA)

TEE_OpenPersistentObject() - Opens a handle on an existing persistent object.

This function returns a handle which can be used to access the object's attributes and data stream.

**Parameters**

| | |
|---|---|
| *storageID* | The storage to use |
| *objectID* | The object identifier |
| *objectIDLen* | The object identifier |
| *flags* | The flags which determine the settings under which the object is opened. |
| *object* | A pointer to the handle, which contains the opened handle upon successful completion |

**Returns**

0 if success else error occured.

TEE_OpenPersistentObject() - Opens a handle on an existing persistent object.

This function returns a handle that can be used to access the object's attributes and data stream.

**Parameters**

| | |
|---|---|
| *storageID* | The storage to use. |
| *objectID* | The object identifier |
| *objectIDLen* | The object identifier |
| *flags* | The flags which determine the settings under which the object is opened. |
| *object* | A pointer to the handle, which contains the opened handle upon successful completion |

**Returns**

0 if success, else error occured.

#### 4.9.1.103 TEE_OpenTASession()

```
TEE_Result TEE_OpenTASession (
    const TEE_UUID * destination,
    uint32_t cancellationRequestTimeout,
    uint32_t paramTypes,
    TEE_Param params[TEE_NUM_PARAMS],
    TEE_TASessionHandle * session,
    uint32_t * returnOrigin )
```

#### 4.9.1.104 TEE_Panic()

```
void TEE_Panic (
    TEE_Result panicCode )
```

#### 4.9.1.105 TEE_PopulateTransientObject()

```
TEE_Result TEE_PopulateTransientObject (
    TEE_ObjectHandle object,
    const TEE_Attribute * attrs,
    uint32_t attrCount )
```

#### 4.9.1.106 TEE_ReadObjectData()

```
TEE_Result TEE_ReadObjectData (
    TEE_ObjectHandle object,
    void * buffer,
    uint32_t size,
    uint32_t * count )
```

Core Functions, Secure Storage Functions (data is isolated for each TA)

TEE_ReadObjectData() - Attempts to read size bytes from the data stream associated with the object into the buffer pointed to by buffer.

The bytes are read starting at the position in the data stream currently stored in the object handle. The handle's position is incremented by the number of bytes actually read. On completion of TEE_ReadObjectData sets the number of bytes actually read in the "uint32_t" pointed to by count. The value written to ∗count may be less than size if the number of bytes until the end-of3067 stream is less than size. It is set to 0 if the position at the start of the read operation is at or beyond the end-of-stream. These are the only cases where ∗count may be less than size.

**Parameters**

| | |
|---|---|
| *object* | Handle of the object |
| *buffer* | The buffer containing the data to be written |
| *size* | The number of bytes to write |
| *count* | size of the buffer. |

**Returns**

TEE_SUCCESS if success else error occured.

TEE_ReadObjectData() - Attempts to read size bytes from the data stream associated with the object object into the buffer pointed to by buffer.

The bytes are read starting at the position in the data stream currently stored in the object handle. The handle's position is incremented by the number of bytes actually read. On completion TEE_ReadObjectData sets the number of bytes actually read in the uint32_t pointed to by count. The value written to ∗count may be less than size if the number of bytes until the end-of3067 stream is less than size. It is set to 0 if the position at the start of the read operation is at or beyond the end-of-stream. These are the only cases where ∗count may be less than size.

**Parameters**

| | |
|---|---|
| *object* | Handle of the object |
| *buffer* | The buffer containing the data to be written |
| *size* | The number of bytes to write |
| *count* | size of the buffer. |

**Returns**

TEE_SUCCESS if success, else error occured.

#### 4.9.1.107 TEE_Realloc()

```
void * TEE_Realloc (
    void * buffer,
    uint32_t newSize )
```

TEE_Realloc() - Changes the size of the memory object pointed to by buffer to the size specified by new size.

This function describes the content of the object remains unchanged up to the lesser of the new and old sizes. Space in excess of the old size contains unspecified content. If the new size of the memory object requires movement of the object, the space for the previous instantiation of the object is deallocated. If the space cannot be allocated, the original object remains allocated, and this function returns a NULL pointer.

**Parameters**

| | |
|---|---|
| *buffer* | The pointer to the object to be reallocated. |
| *newSize* | The new size required for the object |

**Returns**

Upon successful completion, TEE_Realloc returns a pointer to the (possibly moved) allocated space. If there is not enough available memory, TEE_Realloc returns a NULL pointer and the original buffer is still allocated and unchanged.

#### 4.9.1.108 TEE_RenamePersistentObject()

```
TEE_Result TEE_RenamePersistentObject (
    TEE_ObjectHandle object,
    const void * newObjectID,
    uint32_t newObjectIDLen )
```

#### 4.9.1.109 TEE_ResetOperation()

```
void TEE_ResetOperation (
    TEE_OperationHandle operation )
```

#### 4.9.1.110 TEE_ResetPersistentObjectEnumerator()

```
void TEE_ResetPersistentObjectEnumerator (
    TEE_ObjectEnumHandle objectEnumerator )
```

#### 4.9.1.111 TEE_ResetPropertyEnumerator()

```
void TEE_ResetPropertyEnumerator (
    TEE_PropSetHandle enumerator )
```

#### 4.9.1.112 TEE_ResetTransientObject()

```
void TEE_ResetTransientObject (
    TEE_ObjectHandle object )
```

#### 4.9.1.113 TEE_RestrictObjectUsage()

```
void TEE_RestrictObjectUsage (
    TEE_ObjectHandle object,
    uint32_t objectUsage )
```

#### 4.9.1.114 TEE_RestrictObjectUsage1()

```
TEE_Result TEE_RestrictObjectUsage1 (
    TEE_ObjectHandle object,
    uint32_t objectUsage )
```

#### 4.9.1.115 TEE_SeekObjectData()

```
TEE_Result TEE_SeekObjectData (
    TEE_ObjectHandle object,
    int32_t offset,
    TEE_Whence whence )
```

#### 4.9.1.116 TEE_SetInstanceData()

```
void TEE_SetInstanceData (
    const void * instanceData )
```

#### 4.9.1.117 TEE_SetOperationKey()

```
TEE_Result TEE_SetOperationKey (
    TEE_OperationHandle operation,
    TEE_ObjectHandle key )
```

Crypto, Authenticated Encryption with Symmetric key Verification Functions.

TEE_SetOperationKey() - Programs the key of an operation; that is, it associates an operation with a key.

The key material is copied from the key object handle into the operation. After the key has been set, there is no longer any link between the operation and the key object. The object handle can be closed or reset and this will not affect the operation. This copied material exists until the operation is freed using TEE_FreeOperation or another key is set into the operation.

**Parameters**

| | |
|---|---|
| *operation* | Operation handle. |
| *key* | A handle on a key object. |

**Returns**

0 on success return

TEE_ERROR_CORRUPT_OBJECT If the object is corrupt. The object handle is closed.

TEE_ERROR_STORAGE_NOT_AVAILABLE If the persistent object is stored in a storage area which is currently inaccessible.

#### 4.9.1.118 TEE_SetOperationKey2()

```
TEE_Result TEE_SetOperationKey2 (
    TEE_OperationHandle operation,
    TEE_ObjectHandle key1,
    TEE_ObjectHandle key2 )
```

#### 4.9.1.119 TEE_SetTAPersistentTime()

```
TEE_Result TEE_SetTAPersistentTime (
    const TEE_Time * time )
```

#### 4.9.1.120 TEE_StartPersistentObjectEnumerator()

```
TEE_Result TEE_StartPersistentObjectEnumerator
(
    TEE_ObjectEnumHandle objectEnumerator,
    uint32_t storageID )
```

#### 4.9.1.121 TEE_StartPropertyEnumerator()

```
void TEE_StartPropertyEnumerator (
    TEE_PropSetHandle enumerator,
    TEE_PropSetHandle propSet )
```

#### 4.9.1.122 TEE_TruncateObjectData()

```
TEE_Result TEE_TruncateObjectData (
    TEE_ObjectHandle object,
    uint32_t size )
```

#### 4.9.1.123 TEE_UnmaskCancellation()

```
bool TEE_UnmaskCancellation (
    void  )
```

#### 4.9.1.124 TEE_Wait()

```
TEE_Result TEE_Wait (
    uint32_t timeout )
```

#### 4.9.1.125 TEE_WriteObjectData()

```
TEE_Result TEE_WriteObjectData (
    TEE_ObjectHandle object,
    const void * buffer,
    uint32_t size )
```

Core Functions, Secure Storage Functions (data is isolated for each TA)

TEE_WriteObjectData() - Writes the buffer data in to persistent objects.

In this function it checks if object is present or not, the encryption/ decryption buffer is taken by calling mbedtls_aes↩_crypt_cbc() then that buffer data is encrypted and mapped to object.On the base of object creation TEE_SUCCESS appears else TEE_ERROR_GENERIC appears.

**Parameters**

| | |
|---|---|
| *object* | Handle of the object |
| *buffer* | The buffer containing the data to be written |
| *size* | The number of bytes to write |

**Returns**

TEE_SUCCESS if success else error occured.

TEE_WriteObjectData() - writes size bytes from the buffer pointed to by buffer to the data stream associated with the open object handle object.

If the current data position points before the end-of-stream, then size bytes are written to the data stream, overwriting bytes starting at the current data position. If the current data position points beyond the stream's end, then the data stream is first extended with zero bytes until the length indicated by the data position indicator is reached, and then size bytes are written to the stream.

**Parameters**

| | |
|---|---|
| *object* | Handle of the object |
| *buffer* | The buffer containing the data to be written |
| *size* | The number of bytes to write |

**Returns**

TEE_SUCCESS if success else error occured.

## 4.10 tee_api.h

Go to the documentation of this file.

```
1 /* SPDX-License-Identifier: BSD-2-Clause */
2 /*
3  * Copyright (c) 2014, STMicroelectronics International N.V.
4  */
5
6 /* Based on GP TEE Internal API Specification Version 1.1 */
7 #ifndef TEE_API_H
8 #define TEE_API_H
9
10 #include <stddef.h>
11 #include <compiler.h>
12 #include <tee_api_defines.h>
13 #include <tee_api_types.h>
14 #include <trace.h>
15
16 /* Property access functions */
17
18 TEE_Result TEE_GetPropertyAsString(TEE_PropSetHandle propsetOrEnumerator,
19                    const char *name, char *valueBuffer,
20                    uint32_t *valueBufferLen);
21
22 TEE_Result TEE_GetPropertyAsBool(TEE_PropSetHandle propsetOrEnumerator,
23                  const char *name, bool *value);
24
25 TEE_Result TEE_GetPropertyAsU32(TEE_PropSetHandle propsetOrEnumerator,
26                 const char *name, uint32_t *value);
27
28 TEE_Result TEE_GetPropertyAsBinaryBlock(TEE_PropSetHandle propsetOrEnumerator,
29                     const char *name, void *valueBuffer,
30                     uint32_t *valueBufferLen);
31
32 TEE_Result TEE_GetPropertyAsUUID(TEE_PropSetHandle propsetOrEnumerator,
33                  const char *name, TEE_UUID *value);
34
35 TEE_Result TEE_GetPropertyAsIdentity(TEE_PropSetHandle propsetOrEnumerator,
36                      const char *name, TEE_Identity *value);
37
38 TEE_Result TEE_AllocatePropertyEnumerator(TEE_PropSetHandle *enumerator);
39
40 void TEE_FreePropertyEnumerator(TEE_PropSetHandle enumerator);
41
42 void TEE_StartPropertyEnumerator(TEE_PropSetHandle enumerator,
43                  TEE_PropSetHandle propSet);
44
45 void TEE_ResetPropertyEnumerator(TEE_PropSetHandle enumerator);
46
47 TEE_Result TEE_GetPropertyName(TEE_PropSetHandle enumerator,
48                    void *nameBuffer, uint32_t *nameBufferLen);
49
50 TEE_Result TEE_GetNextProperty(TEE_PropSetHandle enumerator);
51
52 /* System API - Misc */
53
54 void TEE_Panic(TEE_Result panicCode);
55
56 /* System API - Internal Client API */
57
58 TEE_Result TEE_OpenTASession(const TEE_UUID *destination,
59                 uint32_t cancellationRequestTimeout,
60                 uint32_t paramTypes,
61                 TEE_Param params[TEE_NUM_PARAMS],
62                 TEE_TASessionHandle *session,
63                 uint32_t *returnOrigin);
64
65 void TEE_CloseTASession(TEE_TASessionHandle session);
66
67 TEE_Result TEE_InvokeTACommand(TEE_TASessionHandle session,
68                 uint32_t cancellationRequestTimeout,
69                 uint32_t commandID, uint32_t paramTypes,
70                 TEE_Param params[TEE_NUM_PARAMS],
71                 uint32_t *returnOrigin);
72
73 /* System API - Cancellations */
74
75 bool TEE_GetCancellationFlag(void);
76
77 bool TEE_UnmaskCancellation(void);
78
79 bool TEE_MaskCancellation(void);
80
81 /* System API - Memory Management */
82
83 TEE_Result TEE_CheckMemoryAccessRights(uint32_t accessFlags, void *buffer,
84                        uint32_t size);
85
```

```
86 void TEE_SetInstanceData(const void *instanceData);
87
88 const void *TEE_GetInstanceData(void);
89
90 void *TEE_Malloc(uint32_t size, uint32_t hint);
91
92 void *TEE_Realloc(void *buffer, uint32_t newSize);
93
94 void TEE_Free(void *buffer);
95
96 void *TEE_MemMove(void *dest, const void *src, uint32_t size);
97
98 /*
99  * Note: TEE_MemCompare() has a constant-time implementation (execution time
100  * does not depend on buffer content but only on buffer size). It is the main
101  * difference with memcmp().
102  */
103 int32_t TEE_MemCompare(const void *buffer1, const void *buffer2, uint32_t size);
104
105 void *TEE_MemFill(void *buff, uint32_t x, uint32_t size);
106
107 /* Data and Key Storage API  - Generic Object Functions */
108
109 void TEE_GetObjectInfo(TEE_ObjectHandle object, TEE_ObjectInfo *objectInfo);
110 TEE_Result TEE_GetObjectInfo1(TEE_ObjectHandle object, TEE_ObjectInfo *objectInfo);
111
112 void TEE_RestrictObjectUsage(TEE_ObjectHandle object, uint32_t objectUsage);
113 TEE_Result TEE_RestrictObjectUsage1(TEE_ObjectHandle object, uint32_t objectUsage);
114
115 TEE_Result TEE_GetObjectBufferAttribute(TEE_ObjectHandle object,
116                     uint32_t attributeID, void *buffer,
117                     uint32_t *size);
118
119 TEE_Result TEE_GetObjectValueAttribute(TEE_ObjectHandle object,
120                       uint32_t attributeID, uint32_t *a,
121                       uint32_t *b);
122
123 void TEE_CloseObject(TEE_ObjectHandle object);
124
125 /* Data and Key Storage API  - Transient Object Functions */
126
127 TEE_Result TEE_AllocateTransientObject(TEE_ObjectType objectType,
128                       uint32_t maxKeySize,
129                       TEE_ObjectHandle *object);
130
131 void TEE_FreeTransientObject(TEE_ObjectHandle object);
132
133 void TEE_ResetTransientObject(TEE_ObjectHandle object);
134
135 TEE_Result TEE_PopulateTransientObject(TEE_ObjectHandle object,
136                       const TEE_Attribute *attrs,
137                       uint32_t attrCount);
138
139 void TEE_InitRefAttribute(TEE_Attribute *attr, uint32_t attributeID,
140               const void *buffer, uint32_t length);
141
142 void TEE_InitValueAttribute(TEE_Attribute *attr, uint32_t attributeID,
143                 uint32_t a, uint32_t b);
144
145 void TEE_CopyObjectAttributes(TEE_ObjectHandle destObject,
146                   TEE_ObjectHandle srcObject);
147
148 TEE_Result TEE_CopyObjectAttributes1(TEE_ObjectHandle destObject,
149                   TEE_ObjectHandle srcObject);
150
151 TEE_Result TEE_GenerateKey(TEE_ObjectHandle object, uint32_t keySize,
152                const TEE_Attribute *params, uint32_t paramCount);
153
154 /* Data and Key Storage API  - Persistent Object Functions */
155
156 TEE_Result TEE_OpenPersistentObject(uint32_t storageID, const void *objectID,
157                     uint32_t objectIDLen, uint32_t flags,
158                     TEE_ObjectHandle *object);
159
160 TEE_Result TEE_CreatePersistentObject(uint32_t storageID, const void *objectID,
161                       uint32_t objectIDLen, uint32_t flags,
162                       TEE_ObjectHandle attributes,
163                       const void *initialData,
164                       uint32_t initialDataLen,
165                       TEE_ObjectHandle *object);
166
167 void TEE_CloseAndDeletePersistentObject(TEE_ObjectHandle object);
168
169 TEE_Result TEE_CloseAndDeletePersistentObject1(TEE_ObjectHandle object);
170
```

```
171 TEE_Result TEE_RenamePersistentObject(TEE_ObjectHandle object,
172                       const void *newObjectID,
173                       uint32_t newObjectIDLen);
174
175 TEE_Result TEE_AllocatePersistentObjectEnumerator(TEE_ObjectEnumHandle *
176                           objectEnumerator);
177
178 void TEE_FreePersistentObjectEnumerator(TEE_ObjectEnumHandle objectEnumerator);
179
180 void TEE_ResetPersistentObjectEnumerator(TEE_ObjectEnumHandle objectEnumerator);
181
182 TEE_Result TEE_StartPersistentObjectEnumerator(TEE_ObjectEnumHandle
183                            objectEnumerator,
184                            uint32_t storageID);
185
186 TEE_Result TEE_GetNextPersistentObject(TEE_ObjectEnumHandle objectEnumerator,
187                        TEE_ObjectInfo *objectInfo,
188                        void *objectID, uint32_t *objectIDLen);
189
190 /* Data and Key Storage API  - Data Stream Access Functions */
191
192 TEE_Result TEE_ReadObjectData(TEE_ObjectHandle object, void *buffer,
193                   uint32_t size, uint32_t *count);
194
195 TEE_Result TEE_WriteObjectData(TEE_ObjectHandle object, const void *buffer,
196                    uint32_t size);
197
198 TEE_Result TEE_TruncateObjectData(TEE_ObjectHandle object, uint32_t size);
199
200 TEE_Result TEE_SeekObjectData(TEE_ObjectHandle object, int32_t offset,
201                   TEE_Whence whence);
202
203 /* Cryptographic Operations API - Generic Operation Functions */
204
205 TEE_Result TEE_AllocateOperation(TEE_OperationHandle *operation,
206                  uint32_t algorithm, uint32_t mode,
207                  uint32_t maxKeySize);
208
209 void TEE_FreeOperation(TEE_OperationHandle operation);
210
211 void TEE_GetOperationInfo(TEE_OperationHandle operation,
212               TEE_OperationInfo *operationInfo);
213
214 TEE_Result TEE_GetOperationInfoMultiple(TEE_OperationHandle operation,
215               TEE_OperationInfoMultiple *operationInfoMultiple,
216               uint32_t *operationSize);
217
218 void TEE_ResetOperation(TEE_OperationHandle operation);
219
220 TEE_Result TEE_SetOperationKey(TEE_OperationHandle operation,
221                    TEE_ObjectHandle key);
222
223 TEE_Result TEE_SetOperationKey2(TEE_OperationHandle operation,
224                 TEE_ObjectHandle key1, TEE_ObjectHandle key2);
225
226 void TEE_CopyOperation(TEE_OperationHandle dstOperation,
227                TEE_OperationHandle srcOperation);
228
229 TEE_Result TEE_IsAlgorithmSupported(uint32_t algId, uint32_t element);
230
231 /* Cryptographic Operations API - Message Digest Functions */
232
233 void TEE_DigestUpdate(TEE_OperationHandle operation,
234               const void *chunk, uint32_t chunkSize);
235
236 TEE_Result TEE_DigestDoFinal(TEE_OperationHandle operation, const void *chunk,
237                  uint32_t chunkLen, void *hash, uint32_t *hashLen);
238
239 /* Cryptographic Operations API - Symmetric Cipher Functions */
240
241 void TEE_CipherInit(TEE_OperationHandle operation, const void *IV,
242             uint32_t IVLen);
243
244 TEE_Result TEE_CipherUpdate(TEE_OperationHandle operation, const void *srcData,
245                 uint32_t srcLen, void *destData, uint32_t *destLen);
246
247 TEE_Result TEE_CipherDoFinal(TEE_OperationHandle operation,
248                  const void *srcData, uint32_t srcLen,
249                  void *destData, uint32_t *destLen);
250
251 /* Cryptographic Operations API - MAC Functions */
252
253 void TEE_MACInit(TEE_OperationHandle operation, const void *IV,
254          uint32_t IVLen);
255
```

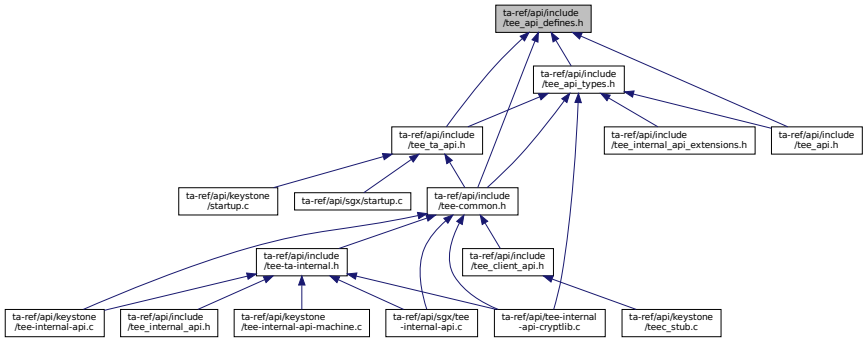
```
256 void TEE_MACUpdate(TEE_OperationHandle operation, const void *chunk,
257            uint32_t chunkSize);
258
259 TEE_Result TEE_MACComputeFinal(TEE_OperationHandle operation,
260                   const void *message, uint32_t messageLen,
261                   void *mac, uint32_t *macLen);
262
263 TEE_Result TEE_MACCompareFinal(TEE_OperationHandle operation,
264                   const void *message, uint32_t messageLen,
265                   const void *mac, uint32_t macLen);
266
267 /* Cryptographic Operations API - Authenticated Encryption Functions */
268
269 TEE_Result TEE_AEInit(TEE_OperationHandle operation, const void *nonce,
270              uint32_t nonceLen, uint32_t tagLen, uint32_t AADLen,
271              uint32_t payloadLen);
272
273 void TEE_AEUpdateAAD(TEE_OperationHandle operation, const void *AADdata,
274             uint32_t AADdataLen);
275
276 TEE_Result TEE_AEUpdate(TEE_OperationHandle operation, const void *srcData,
277            uint32_t srcLen, void *destData, uint32_t *destLen);
278
279 TEE_Result TEE_AEEncryptFinal(TEE_OperationHandle operation,
280                  const void *srcData, uint32_t srcLen,
281                  void *destData, uint32_t *destLen, void *tag,
282                  uint32_t *tagLen);
283
284 TEE_Result TEE_AEDecryptFinal(TEE_OperationHandle operation,
285                  const void *srcData, uint32_t srcLen,
286                  void *destData, uint32_t *destLen, void *tag,
287                  uint32_t tagLen);
288
289 /* Cryptographic Operations API - Asymmetric Functions */
290
291 TEE_Result TEE_AsymmetricEncrypt(TEE_OperationHandle operation,
292                 const TEE_Attribute *params,
293                 uint32_t paramCount, const void *srcData,
294                 uint32_t srcLen, void *destData,
295                 uint32_t *destLen);
296
297 TEE_Result TEE_AsymmetricDecrypt(TEE_OperationHandle operation,
298                 const TEE_Attribute *params,
299                 uint32_t paramCount, const void *srcData,
300                 uint32_t srcLen, void *destData,
301                 uint32_t *destLen);
302
303 TEE_Result TEE_AsymmetricSignDigest(TEE_OperationHandle operation,
304                    const TEE_Attribute *params,
305                    uint32_t paramCount, const void *digest,
306                    uint32_t digestLen, void *signature,
307                    uint32_t *signatureLen);
308
309 TEE_Result TEE_AsymmetricVerifyDigest(TEE_OperationHandle operation,
310                      const TEE_Attribute *params,
311                      uint32_t paramCount, const void *digest,
312                      uint32_t digestLen, const void *signature,
313                      uint32_t signatureLen);
314
315 /* Cryptographic Operations API - Key Derivation Functions */
316
317 void TEE_DeriveKey(TEE_OperationHandle operation,
318            const TEE_Attribute *params, uint32_t paramCount,
319            TEE_ObjectHandle derivedKey);
320
321 /* Cryptographic Operations API - Random Number Generation Functions */
322
323 void TEE_GenerateRandom(void *randomBuffer, uint32_t randomBufferLen);
324
325 /* Date & Time API */
326
327 void TEE_GetSystemTime(TEE_Time *time);
328
329 TEE_Result TEE_Wait(uint32_t timeout);
330
331 TEE_Result TEE_GetTAPersistentTime(TEE_Time *time);
332
333 TEE_Result TEE_SetTAPersistentTime(const TEE_Time *time);
334
335 void TEE_GetREETime(TEE_Time *time);
336
337 /* TEE Arithmetical API - Memory allocation and size of objects */
338
339 uint32_t TEE_BigIntFMMSizeInU32(uint32_t modulusSizeInBits);
340
```

```
341 uint32_t TEE_BigIntFMMContextSizeInU32(uint32_t modulusSizeInBits);
342
343 /* TEE Arithmetical API - Initialization functions */
344
345 void TEE_BigIntInit(TEE_BigInt *bigInt, uint32_t len);
346
347 void TEE_BigIntInitFMMContext(TEE_BigIntFMMContext *context, uint32_t len,
348                   const TEE_BigInt *modulus);
349
350 void TEE_BigIntInitFMM(TEE_BigIntFMM *bigIntFMM, uint32_t len);
351
352 /* TEE Arithmetical API - Converter functions */
353
354 TEE_Result TEE_BigIntConvertFromOctetString(TEE_BigInt *dest,
355                         const uint8_t *buffer,
356                         uint32_t bufferLen,
357                         int32_t sign);
358
359 TEE_Result TEE_BigIntConvertToOctetString(uint8_t *buffer, uint32_t *bufferLen,
360                       const TEE_BigInt *bigInt);
361
362 void TEE_BigIntConvertFromS32(TEE_BigInt *dest, int32_t shortVal);
363
364 TEE_Result TEE_BigIntConvertToS32(int32_t *dest, const TEE_BigInt *src);
365
366 /* TEE Arithmetical API - Logical operations */
367
368 int32_t TEE_BigIntCmp(const TEE_BigInt *op1, const TEE_BigInt *op2);
369
370 int32_t TEE_BigIntCmpS32(const TEE_BigInt *op, int32_t shortVal);
371
372 void TEE_BigIntShiftRight(TEE_BigInt *dest, const TEE_BigInt *op,
373               size_t bits);
374
375 bool TEE_BigIntGetBit(const TEE_BigInt *src, uint32_t bitIndex);
376
377 uint32_t TEE_BigIntGetBitCount(const TEE_BigInt *src);
378
379 void TEE_BigIntAdd(TEE_BigInt *dest, const TEE_BigInt *op1,
380            const TEE_BigInt *op2);
381
382 void TEE_BigIntSub(TEE_BigInt *dest, const TEE_BigInt *op1,
383            const TEE_BigInt *op2);
384
385 void TEE_BigIntNeg(TEE_BigInt *dest, const TEE_BigInt *op);
386
387 void TEE_BigIntMul(TEE_BigInt *dest, const TEE_BigInt *op1,
388            const TEE_BigInt *op2);
389
390 void TEE_BigIntSquare(TEE_BigInt *dest, const TEE_BigInt *op);
391
392 void TEE_BigIntDiv(TEE_BigInt *dest_q, TEE_BigInt *dest_r,
393            const TEE_BigInt *op1, const TEE_BigInt *op2);
394
395 /* TEE Arithmetical API - Modular arithmetic operations */
396
397 void TEE_BigIntMod(TEE_BigInt *dest, const TEE_BigInt *op,
398            const TEE_BigInt *n);
399
400 void TEE_BigIntAddMod(TEE_BigInt *dest, const TEE_BigInt *op1,
401               const TEE_BigInt *op2, const TEE_BigInt *n);
402
403 void TEE_BigIntSubMod(TEE_BigInt *dest, const TEE_BigInt *op1,
404               const TEE_BigInt *op2, const TEE_BigInt *n);
405
406 void TEE_BigIntMulMod(TEE_BigInt *dest, const  TEE_BigInt *op1,
407               const TEE_BigInt *op2, const TEE_BigInt *n);
408
409 void TEE_BigIntSquareMod(TEE_BigInt *dest, const TEE_BigInt *op,
410              const TEE_BigInt *n);
411
412 void TEE_BigIntInvMod(TEE_BigInt *dest, const TEE_BigInt *op,
413               const TEE_BigInt *n);
414
415 /* TEE Arithmetical API - Other arithmetic operations */
416
417 bool TEE_BigIntRelativePrime(const TEE_BigInt *op1, const TEE_BigInt *op2);
418
419 void TEE_BigIntComputeExtendedGcd(TEE_BigInt *gcd, TEE_BigInt *u,
420                   TEE_BigInt *v, const TEE_BigInt *op1,
421                   const TEE_BigInt *op2);
422
423 int32_t TEE_BigIntIsProbablePrime(const TEE_BigInt *op,
424                   uint32_t confidenceLevel);
425
```

```
426 /* TEE Arithmetical API - Fast modular multiplication operations */
427
428 void TEE_BigIntConvertToFMM(TEE_BigIntFMM *dest, const TEE_BigInt *src,
429                 const TEE_BigInt *n,
430                 const TEE_BigIntFMMContext *context);
431
432 void TEE_BigIntConvertFromFMM(TEE_BigInt *dest, const TEE_BigIntFMM *src,
433                   const TEE_BigInt *n,
434                   const TEE_BigIntFMMContext *context);
435
436 void TEE_BigIntFMMConvertToBigInt(TEE_BigInt *dest, const TEE_BigIntFMM *src,
437                   const TEE_BigInt *n,
438                   const TEE_BigIntFMMContext *context);
439
440 void TEE_BigIntComputeFMM(TEE_BigIntFMM *dest, const TEE_BigIntFMM *op1,
441               const TEE_BigIntFMM *op2, const TEE_BigInt *n,
442               const TEE_BigIntFMMContext *context);
443
444 #endif /* TEE_API_H */
```

## 4.11 ta-ref/api/include/tee_api_defines.h File Reference

This graph shows which files directly or indirectly include this file:




## 4.12 tee_api_defines.h

Go to the documentation of this file.

```
1 /*
2  * Copyright (c) 2014, STMicroelectronics International N.V.
3  * All rights reserved.
4  *
5  * Redistribution and use in source and binary forms, with or without
6  * modification, are permitted provided that the following conditions are met:
7  *
8  * 1. Redistributions of source code must retain the above copyright notice,
9  * this list of conditions and the following disclaimer.
10  *
11  * 2. Redistributions in binary form must reproduce the above copyright notice,
12  * this list of conditions and the following disclaimer in the documentation
13  * and/or other materials provided with the distribution.
14  *
15  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
16  * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
17  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
18  * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
19  * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
20  * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
21  * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
22  * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
23  * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
24  * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
```

```
 * POSSIBILITY OF SUCH DAMAGE.
 */

/* Based on GP TEE Internal Core API Specification Version 1.1 */

#ifndef TEE_API_DEFINES_H
#define TEE_API_DEFINES_H

#ifndef DOXYGEN_SHOULD_SKIP_THIS
#define TEE_INT_CORE_API_SPEC_VERSION     0x0000000A

#define TEE_HANDLE_NULL                   0

#define TEE_TIMEOUT_INFINITE              0xFFFFFFFF

/* API Error Codes */
#define TEE_SUCCESS                       0x00000000
#define TEE_ERROR_CORRUPT_OBJECT          0xF0100001
#define TEE_ERROR_CORRUPT_OBJECT_2        0xF0100002
#define TEE_ERROR_STORAGE_NOT_AVAILABLE   0xF0100003
#define TEE_ERROR_STORAGE_NOT_AVAILABLE_2 0xF0100004
#define TEE_ERROR_GENERIC                 0xFFFF0000
#define TEE_ERROR_ACCESS_DENIED           0xFFFF0001
#define TEE_ERROR_CANCEL                  0xFFFF0002
#define TEE_ERROR_ACCESS_CONFLICT         0xFFFF0003
#define TEE_ERROR_EXCESS_DATA             0xFFFF0004
#define TEE_ERROR_BAD_FORMAT              0xFFFF0005
#define TEE_ERROR_BAD_PARAMETERS          0xFFFF0006
#define TEE_ERROR_BAD_STATE               0xFFFF0007
#define TEE_ERROR_ITEM_NOT_FOUND          0xFFFF0008
#define TEE_ERROR_NOT_IMPLEMENTED         0xFFFF0009
#define TEE_ERROR_NOT_SUPPORTED           0xFFFF000A
#define TEE_ERROR_NO_DATA                 0xFFFF000B
#define TEE_ERROR_OUT_OF_MEMORY           0xFFFF000C
#define TEE_ERROR_BUSY                    0xFFFF000D
#define TEE_ERROR_COMMUNICATION           0xFFFF000E
#define TEE_ERROR_SECURITY                0xFFFF000F
#define TEE_ERROR_SHORT_BUFFER            0xFFFF0010
#define TEE_ERROR_EXTERNAL_CANCEL         0xFFFF0011
#define TEE_ERROR_OVERFLOW                0xFFFF300F
#define TEE_ERROR_TARGET_DEAD             0xFFFF3024
#define TEE_ERROR_STORAGE_NO_SPACE        0xFFFF3041
#define TEE_ERROR_MAC_INVALID             0xFFFF3071
#define TEE_ERROR_SIGNATURE_INVALID       0xFFFF3072
#define TEE_ERROR_TIME_NOT_SET            0xFFFF5000
#define TEE_ERROR_TIME_NEEDS_RESET        0xFFFF5001

/* Parameter Type Constants */
#define TEE_PARAM_TYPE_NONE             0
#define TEE_PARAM_TYPE_VALUE_INPUT      1
#define TEE_PARAM_TYPE_VALUE_OUTPUT     2
#define TEE_PARAM_TYPE_VALUE_INOUT      3
#define TEE_PARAM_TYPE_MEMREF_INPUT     5
#define TEE_PARAM_TYPE_MEMREF_OUTPUT    6
#define TEE_PARAM_TYPE_MEMREF_INOUT     7

/* Login Type Constants */
#define TEE_LOGIN_PUBLIC                0x00000000
#define TEE_LOGIN_USER                  0x00000001
#define TEE_LOGIN_GROUP                 0x00000002
#define TEE_LOGIN_APPLICATION           0x00000004
#define TEE_LOGIN_APPLICATION_USER      0x00000005
#define TEE_LOGIN_APPLICATION_GROUP     0x00000006
#define TEE_LOGIN_TRUSTED_APP           0xF0000000

/* Origin Code Constants */
#define TEE_ORIGIN_API                  0x00000001
#define TEE_ORIGIN_COMMS                0x00000002
#define TEE_ORIGIN_TEE                  0x00000003
#define TEE_ORIGIN_TRUSTED_APP          0x00000004

/* Property Sets pseudo handles */
#define TEE_PROPSET_TEE_IMPLEMENTATION  (TEE_PropSetHandle)0xFFFFFFFD
#define TEE_PROPSET_CURRENT_CLIENT      (TEE_PropSetHandle)0xFFFFFFFE
#define TEE_PROPSET_CURRENT_TA          (TEE_PropSetHandle)0xFFFFFFFF

/* Memory Access Rights Constants */
#define TEE_MEMORY_ACCESS_READ             0x00000001
#define TEE_MEMORY_ACCESS_WRITE            0x00000002
#define TEE_MEMORY_ACCESS_ANY_OWNER        0x00000004

/* Memory Management Constant */
#define TEE_MALLOC_FILL_ZERO               0x00000000

/* Other constants */
```

```
110 #define TEE_STORAGE_PRIVATE                0x00000001
111
112 #define TEE_DATA_FLAG_ACCESS_READ          0x00000001
113 #define TEE_DATA_FLAG_ACCESS_WRITE         0x00000002
114 #define TEE_DATA_FLAG_ACCESS_WRITE_META    0x00000004
115 #define TEE_DATA_FLAG_SHARE_READ           0x00000010
116 #define TEE_DATA_FLAG_SHARE_WRITE          0x00000020
117 #define TEE_DATA_FLAG_OVERWRITE            0x00000400
118 #define TEE_DATA_MAX_POSITION              0xFFFFFFFF
119 #define TEE_OBJECT_ID_MAX_LEN              64
120 #define TEE_USAGE_EXTRACTABLE              0x00000001
121 #define TEE_USAGE_ENCRYPT                  0x00000002
122 #define TEE_USAGE_DECRYPT                  0x00000004
123 #define TEE_USAGE_MAC                      0x00000008
124 #define TEE_USAGE_SIGN                     0x00000010
125 #define TEE_USAGE_VERIFY                   0x00000020
126 #define TEE_USAGE_DERIVE                   0x00000040
127 #define TEE_HANDLE_FLAG_PERSISTENT         0x00010000
128 #define TEE_HANDLE_FLAG_INITIALIZED        0x00020000
129 #define TEE_HANDLE_FLAG_KEY_SET            0x00040000
130 #define TEE_HANDLE_FLAG_EXPECT_TWO_KEYS    0x00080000
131 #define TEE_OPERATION_CIPHER               1
132 #define TEE_OPERATION_MAC                  3
133 #define TEE_OPERATION_AE                   4
134 #define TEE_OPERATION_DIGEST               5
135 #define TEE_OPERATION_ASYMMETRIC_CIPHER    6
136 #define TEE_OPERATION_ASYMMETRIC_SIGNATURE 7
137 #define TEE_OPERATION_KEY_DERIVATION       8
138 #define TEE_OPERATION_STATE_INITIAL        0x00000000
139 #define TEE_OPERATION_STATE_ACTIVE         0x00000001
140
141 /* Algorithm Identifiers */
142 #define TEE_ALG_AES_ECB_NOPAD                   0x10000010
143 #define TEE_ALG_AES_CBC_NOPAD                   0x10000110
144 #define TEE_ALG_AES_CTR                         0x10000210
145 #define TEE_ALG_AES_CTS                         0x10000310
146 #define TEE_ALG_AES_XTS                         0x10000410
147 #define TEE_ALG_AES_CBC_MAC_NOPAD               0x30000110
148 #define TEE_ALG_AES_CBC_MAC_PKCS5               0x30000510
149 #define TEE_ALG_AES_CMAC                        0x30000610
150 #define TEE_ALG_AES_CCM                         0x40000710
151 #define TEE_ALG_AES_GCM                         0x40000810
152 #define TEE_ALG_DES_ECB_NOPAD                   0x10000011
153 #define TEE_ALG_DES_CBC_NOPAD                   0x10000111
154 #define TEE_ALG_DES_CBC_MAC_NOPAD               0x30000111
155 #define TEE_ALG_DES_CBC_MAC_PKCS5               0x30000511
156 #define TEE_ALG_DES3_ECB_NOPAD                  0x10000013
157 #define TEE_ALG_DES3_CBC_NOPAD                  0x10000113
158 #define TEE_ALG_DES3_CBC_MAC_NOPAD              0x30000113
159 #define TEE_ALG_DES3_CBC_MAC_PKCS5              0x30000513
160 #define TEE_ALG_RSASSA_PKCS1_V1_5_MD5           0x70001830
161 #define TEE_ALG_RSASSA_PKCS1_V1_5_SHA1          0x70002830
162 #define TEE_ALG_RSASSA_PKCS1_V1_5_SHA224        0x70003830
163 #define TEE_ALG_RSASSA_PKCS1_V1_5_SHA256        0x70004830
164 #define TEE_ALG_RSASSA_PKCS1_V1_5_SHA384        0x70005830
165 #define TEE_ALG_RSASSA_PKCS1_V1_5_SHA512        0x70006830
166 #define TEE_ALG_RSASSA_PKCS1_V1_5_MD5SHA1       0x7000F830
167 #define TEE_ALG_RSASSA_PKCS1_PSS_MGF1_SHA1      0x70212930
168 #define TEE_ALG_RSASSA_PKCS1_PSS_MGF1_SHA224    0x70313930
169 #define TEE_ALG_RSASSA_PKCS1_PSS_MGF1_SHA256    0x70414930
170 #define TEE_ALG_RSASSA_PKCS1_PSS_MGF1_SHA384    0x70515930
171 #define TEE_ALG_RSASSA_PKCS1_PSS_MGF1_SHA512    0x70616930
172 #define TEE_ALG_RSAES_PKCS1_V1_5                0x60000130
173 #define TEE_ALG_RSAES_PKCS1_OAEP_MGF1_SHA1      0x60210230
174 #define TEE_ALG_RSAES_PKCS1_OAEP_MGF1_SHA224    0x60310230
175 #define TEE_ALG_RSAES_PKCS1_OAEP_MGF1_SHA256    0x60410230
176 #define TEE_ALG_RSAES_PKCS1_OAEP_MGF1_SHA384    0x60510230
177 #define TEE_ALG_RSAES_PKCS1_OAEP_MGF1_SHA512    0x60610230
178 #define TEE_ALG_RSA_NOPAD                       0x60000030
179 #define TEE_ALG_DSA_SHA1                        0x70002131
180 #define TEE_ALG_DSA_SHA224                      0x70003131
181 #define TEE_ALG_DSA_SHA256                      0x70004131
182 #define TEE_ALG_DH_DERIVE_SHARED_SECRET         0x80000032
183 #define TEE_ALG_MD5                             0x50000001
184 #define TEE_ALG_SHA1                            0x50000002
185 #define TEE_ALG_SHA224                          0x50000003
186 #define TEE_ALG_SHA256                          0x50000004
187 #define TEE_ALG_SHA384                          0x50000005
188 #define TEE_ALG_SHA512                          0x50000006
189 #define TEE_ALG_MD5SHA1                         0x5000000F
190 #define TEE_ALG_HMAC_MD5                        0x30000001
191 #define TEE_ALG_HMAC_SHA1                       0x30000002
192 #define TEE_ALG_HMAC_SHA224                     0x30000003
193 #define TEE_ALG_HMAC_SHA256                     0x30000004
194 #define TEE_ALG_HMAC_SHA384                     0x30000005
```

```
195 #define TEE_ALG_HMAC_SHA512                     0x30000006
196 /*
197  * Fix GP Internal Core API v1.1
198  *     "Table 6-12:  Structure of Algorithm Identifier"
199  *     indicates ECDSA have the algorithm "0x41" and ECDH "0x42"
200  * whereas
201  *     "Table 6-11:  List of Algorithm Identifiers" defines
202  *     TEE_ALG_ECDSA_P192 as 0x70001042
203  *
204  * We chose to define TEE_ALG_ECDSA_P192 as 0x70001041 (conform to table 6-12)
205  */
206 #define TEE_ALG_ECDSA_P192                      0x70001041
207 #define TEE_ALG_ECDSA_P224                      0x70002041
208 #define TEE_ALG_ECDSA_P256                      0x70003041
209 #define TEE_ALG_ECDSA_P384                      0x70004041
210 #define TEE_ALG_ECDSA_P521                      0x70005041
211 #define TEE_ALG_ECDH_P192                       0x80001042
212 #define TEE_ALG_ECDH_P224                       0x80002042
213 #define TEE_ALG_ECDH_P256                       0x80003042
214 #define TEE_ALG_ECDH_P384                       0x80004042
215 #define TEE_ALG_ECDH_P521                       0x80005042
216
217 /* Object Types */
218
219 #define TEE_TYPE_AES                          0xA0000010
220 #define TEE_TYPE_DES                          0xA0000011
221 #define TEE_TYPE_DES3                         0xA0000013
222 #define TEE_TYPE_HMAC_MD5                     0xA0000001
223 #define TEE_TYPE_HMAC_SHA1                    0xA0000002
224 #define TEE_TYPE_HMAC_SHA224                  0xA0000003
225 #define TEE_TYPE_HMAC_SHA256                  0xA0000004
226 #define TEE_TYPE_HMAC_SHA384                  0xA0000005
227 #define TEE_TYPE_HMAC_SHA512                  0xA0000006
228 #define TEE_TYPE_RSA_PUBLIC_KEY               0xA0000030
229 #define TEE_TYPE_RSA_KEYPAIR                  0xA1000030
230 #define TEE_TYPE_DSA_PUBLIC_KEY               0xA0000031
231 #define TEE_TYPE_DSA_KEYPAIR                  0xA1000031
232 #define TEE_TYPE_DH_KEYPAIR                   0xA1000032
233 #define TEE_TYPE_ECDSA_PUBLIC_KEY             0xA0000041
234 #define TEE_TYPE_ECDSA_KEYPAIR                0xA1000041
235 #define TEE_TYPE_ECDH_PUBLIC_KEY              0xA0000042
236 #define TEE_TYPE_ECDH_KEYPAIR                 0xA1000042
237 #define TEE_TYPE_GENERIC_SECRET               0xA0000000
238 #define TEE_TYPE_CORRUPTED_OBJECT             0xA00000BE
239 #define TEE_TYPE_DATA                         0xA00000BF
240
241 /* List of Object or Operation Attributes */
242
243 #define TEE_ATTR_SECRET_VALUE                 0xC0000000
244 #define TEE_ATTR_RSA_MODULUS                  0xD0000130
245 #define TEE_ATTR_RSA_PUBLIC_EXPONENT          0xD0000230
246 #define TEE_ATTR_RSA_PRIVATE_EXPONENT         0xC0000330
247 #define TEE_ATTR_RSA_PRIME1                   0xC0000430
248 #define TEE_ATTR_RSA_PRIME2                   0xC0000530
249 #define TEE_ATTR_RSA_EXPONENT1                0xC0000630
250 #define TEE_ATTR_RSA_EXPONENT2                0xC0000730
251 #define TEE_ATTR_RSA_COEFFICIENT              0xC0000830
252 #define TEE_ATTR_DSA_PRIME                    0xD0001031
253 #define TEE_ATTR_DSA_SUBPRIME                 0xD0001131
254 #define TEE_ATTR_DSA_BASE                     0xD0001231
255 #define TEE_ATTR_DSA_PUBLIC_VALUE             0xD0000131
256 #define TEE_ATTR_DSA_PRIVATE_VALUE            0xC0000231
257 #define TEE_ATTR_DH_PRIME                     0xD0001032
258 #define TEE_ATTR_DH_SUBPRIME                  0xD0001132
259 #define TEE_ATTR_DH_BASE                      0xD0001232
260 #define TEE_ATTR_DH_X_BITS                    0xF0001332
261 #define TEE_ATTR_DH_PUBLIC_VALUE              0xD0000132
262 #define TEE_ATTR_DH_PRIVATE_VALUE             0xC0000232
263 #define TEE_ATTR_RSA_OAEP_LABEL               0xD0000930
264 #define TEE_ATTR_RSA_PSS_SALT_LENGTH          0xF0000A30
265 #define TEE_ATTR_ECC_PUBLIC_VALUE_X           0xD0000141
266 #define TEE_ATTR_ECC_PUBLIC_VALUE_Y           0xD0000241
267 #define TEE_ATTR_ECC_PRIVATE_VALUE            0xC0000341
268 #define TEE_ATTR_ECC_CURVE                    0xF0000441
269
270 #define TEE_ATTR_BIT_PROTECTED      (1 « 28)
271 #define TEE_ATTR_BIT_VALUE      (1 « 29)
272
273 /* List of Supported ECC Curves */
274 #define TEE_ECC_CURVE_NIST_P192               0x00000001
275 #define TEE_ECC_CURVE_NIST_P224               0x00000002
276 #define TEE_ECC_CURVE_NIST_P256               0x00000003
277 #define TEE_ECC_CURVE_NIST_P384               0x00000004
278 #define TEE_ECC_CURVE_NIST_P521               0x00000005
279
```

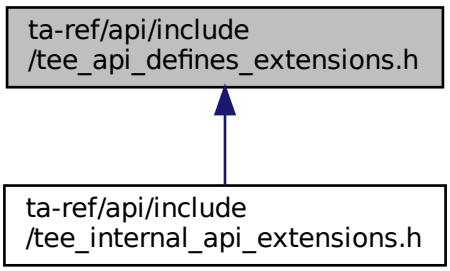
```
280
281 /* Panicked Functions Identification */
282 /* TA Interface */
283 #define TEE_PANIC_ID_TA_CLOSESESSIONENTRYPOINT      0x00000101
284 #define TEE_PANIC_ID_TA_CREATEENTRYPOINT            0x00000102
285 #define TEE_PANIC_ID_TA_DESTROYENTRYPOINT           0x00000103
286 #define TEE_PANIC_ID_TA_INVOKECOMMANDENTRYPOINT     0x00000104
287 #define TEE_PANIC_ID_TA_OPENSESSIONENTRYPOINT       0x00000105
288 /* Property Access */
289 #define TEE_PANIC_ID_TEE_ALLOCATEPROPERTYENUMERATOR 0x00000201
290 #define TEE_PANIC_ID_TEE_FREEPROPERTYENUMERATOR     0x00000202
291 #define TEE_PANIC_ID_TEE_GETNEXTPROPERTY            0x00000203
292 #define TEE_PANIC_ID_TEE_GETPROPERTYASBINARYBLOCK   0x00000204
293 #define TEE_PANIC_ID_TEE_GETPROPERTYASBOOL          0x00000205
294 #define TEE_PANIC_ID_TEE_GETPROPERTYASIDENTITY      0x00000206
295 #define TEE_PANIC_ID_TEE_GETPROPERTYASSTRING        0x00000207
296 #define TEE_PANIC_ID_TEE_GETPROPERTYASU32           0x00000208
297 #define TEE_PANIC_ID_TEE_GETPROPERTYASUUID          0x00000209
298 #define TEE_PANIC_ID_TEE_GETPROPERTYNAME            0x0000020A
299 #define TEE_PANIC_ID_TEE_RESETPROPERTYENUMERATOR    0x0000020B
300 #define TEE_PANIC_ID_TEE_STARTPROPERTYENUMERATOR    0x0000020C
301 /* Panic Function */
302 #define TEE_PANIC_ID_TEE_PANIC                      0x00000301
303 /* Internal Client API */
304 #define TEE_PANIC_ID_TEE_CLOSETASESSION             0x00000401
305 #define TEE_PANIC_ID_TEE_INVOKETACOMMAND            0x00000402
306 #define TEE_PANIC_ID_TEE_OPENTASESSION              0x00000403
307 /* Cancellation */
308 #define TEE_PANIC_ID_TEE_GETCANCELLATIONFLAG        0x00000501
309 #define TEE_PANIC_ID_TEE_MASKCANCELLATION           0x00000502
310 #define TEE_PANIC_ID_TEE_UNMASKCANCELLATION         0x00000503
311 /* Memory Management */
312 #define TEE_PANIC_ID_TEE_CHECKMEMORYACCESSRIGHTS    0x00000601
313 #define TEE_PANIC_ID_TEE_FREE                       0x00000602
314 #define TEE_PANIC_ID_TEE_GETINSTANCEDATA            0x00000603
315 #define TEE_PANIC_ID_TEE_MALLOC                     0x00000604
316 #define TEE_PANIC_ID_TEE_MEMCOMPARE                 0x00000605
317 #define TEE_PANIC_ID_TEE_MEMFILL                    0x00000606
318 #define TEE_PANIC_ID_TEE_MEMMOVE                    0x00000607
319 #define TEE_PANIC_ID_TEE_REALLOC                    0x00000608
320 #define TEE_PANIC_ID_TEE_SETINSTANCEDATA            0x00000609
321 /* Generic Object */
322 #define TEE_PANIC_ID_TEE_CLOSEOBJECT                0x00000701
323 #define TEE_PANIC_ID_TEE_GETOBJECTBUFFERATTRIBUTE   0x00000702
324 /* deprecated */
325 #define TEE_PANIC_ID_TEE_GETOBJECTINFO              0x00000703
326 #define TEE_PANIC_ID_TEE_GETOBJECTVALUEATTRIBUTE    0x00000704
327 /* deprecated */
328 #define TEE_PANIC_ID_TEE_RESTRICTOBJECTUSAGE        0x00000705
329 #define TEE_PANIC_ID_TEE_GETOBJECTINFO1             0x00000706
330 #define TEE_PANIC_ID_TEE_RESTRICTOBJECTUSAGE1       0x00000707
331 /* Transient Object */
332 #define TEE_PANIC_ID_TEE_ALLOCATETRANSIENTOBJECT    0x00000801
333 /* deprecated */
334 #define TEE_PANIC_ID_TEE_COPYOBJECTATTRIBUTES       0x00000802
335 #define TEE_PANIC_ID_TEE_FREETRANSIENTOBJECT        0x00000803
336 #define TEE_PANIC_ID_TEE_GENERATEKEY                0x00000804
337 #define TEE_PANIC_ID_TEE_INITREFATTRIBUTE           0x00000805
338 #define TEE_PANIC_ID_TEE_INITVALUEATTRIBUTE         0x00000806
339 #define TEE_PANIC_ID_TEE_POPULATETRANSIENTOBJECT    0x00000807
340 #define TEE_PANIC_ID_TEE_RESETTRANSIENTOBJECT       0x00000808
341 #define TEE_PANIC_ID_TEE_COPYOBJECTATTRIBUTES1      0x00000809
342 /* Persistent Object */
343 /* deprecated */
344 #define TEE_PANIC_ID_TEE_CLOSEANDDELETEPERSISTENTOBJECT  0x00000901
345 #define TEE_PANIC_ID_TEE_CREATEPERSISTENTOBJECT          0x00000902
346 #define TEE_PANIC_ID_TEE_OPENPERSISTENTOBJECT            0x00000903
347 #define TEE_PANIC_ID_TEE_RENAMEPERSISTENTOBJECT          0x00000904
348 #define TEE_PANIC_ID_TEE_CLOSEANDDELETEPERSISTENTOBJECT1 0x00000905
349 /* Persistent Object Enumeration */
350 #define TEE_PANIC_ID_TEE_ALLOCATEPERSISTENTOBJECTENUMERATOR 0x00000A01
351 #define TEE_PANIC_ID_TEE_FREEPERSISTENTOBJECTENUMERATOR     0x00000A02
352 #define TEE_PANIC_ID_TEE_GETNEXTPERSISTENTOBJECT            0x00000A03
353 #define TEE_PANIC_ID_TEE_RESETPERSISTENTOBJECTENUMERATOR    0x00000A04
354 #define TEE_PANIC_ID_TEE_STARTPERSISTENTOBJECTENUMERATOR    0x00000A05
355 /* Data Stream Access */
356 #define TEE_PANIC_ID_TEE_READOBJECTDATA             0x00000B01
357 #define TEE_PANIC_ID_TEE_SEEKOBJECTDATA             0x00000B02
358 #define TEE_PANIC_ID_TEE_TRUNCATEOBJECTDATA         0x00000B03
359 #define TEE_PANIC_ID_TEE_WRITEOBJECTDATA            0x00000B04
360 /* Generic Operation */
361 #define TEE_PANIC_ID_TEE_ALLOCATEOPERATION          0x00000C01
362 #define TEE_PANIC_ID_TEE_COPYOPERATION              0x00000C02
363 #define TEE_PANIC_ID_TEE_FREEOPERATION              0x00000C03
364 #define TEE_PANIC_ID_TEE_GETOPERATIONINFO           0x00000C04
```

```
365 #define TEE_PANIC_ID_TEE_RESETOPERATION             0x00000C05
366 #define TEE_PANIC_ID_TEE_SETOPERATIONKEY            0x00000C06
367 #define TEE_PANIC_ID_TEE_SETOPERATIONKEY2           0x00000C07
368 #define TEE_PANIC_ID_TEE_GETOPERATIONINFOMULTIPLE   0x00000C08
369 /* Message Digest */
370 #define TEE_PANIC_ID_TEE_DIGESTDOFINAL              0x00000D01
371 #define TEE_PANIC_ID_TEE_DIGESTUPDATE               0x00000D02
372 /* Symmetric Cipher */
373 #define TEE_PANIC_ID_TEE_CIPHERDOFINAL              0x00000E01
374 #define TEE_PANIC_ID_TEE_CIPHERINIT                 0x00000E02
375 #define TEE_PANIC_ID_TEE_CIPHERUPDATE               0x00000E03
376 /* MAC */
377 #define TEE_PANIC_ID_TEE_MACCOMPAREFINAL            0x00000F01
378 #define TEE_PANIC_ID_TEE_MACCOMPUTEFINAL            0x00000F02
379 #define TEE_PANIC_ID_TEE_MACINIT                    0x00000F03
380 #define TEE_PANIC_ID_TEE_MACUPDATE                  0x00000F04
381 /* Authenticated Encryption */
382 #define TEE_PANIC_ID_TEE_AEDECRYPTFINAL             0x00001001
383 #define TEE_PANIC_ID_TEE_AEENCRYPTFINAL             0x00001002
384 #define TEE_PANIC_ID_TEE_AEINIT                     0x00001003
385 #define TEE_PANIC_ID_TEE_AEUPDATE                   0x00001004
386 #define TEE_PANIC_ID_TEE_AEUPDATEAAD                0x00001005
387 /* Asymmetric */
388 #define TEE_PANIC_ID_TEE_ASYMMETRICDECRYPT          0x00001101
389 #define TEE_PANIC_ID_TEE_ASYMMETRICENCRYPT          0x00001102
390 #define TEE_PANIC_ID_TEE_ASYMMETRICSIGNDIGEST       0x00001103
391 #define TEE_PANIC_ID_TEE_ASYMMETRICVERIFYDIGEST     0x00001104
392 /* Key Derivation */
393 #define TEE_PANIC_ID_TEE_DERIVEKEY                  0x00001201
394 /* Random Data Generation */
395 #define TEE_PANIC_ID_TEE_GENERATERANDOM             0x00001301
396 /* Time */
397 #define TEE_PANIC_ID_TEE_GETREETIME                 0x00001401
398 #define TEE_PANIC_ID_TEE_GETSYSTEMTIME              0x00001402
399 #define TEE_PANIC_ID_TEE_GETTAPERSISTENTTIME        0x00001403
400 #define TEE_PANIC_ID_TEE_SETTAPERSISTENTTIME        0x00001404
401 #define TEE_PANIC_ID_TEE_WAIT                       0x00001405
402 /* Memory Allocation and Size of Objects */
403 #define TEE_PANIC_ID_TEE_BIGINTFMMCONTEXTSIZEINU32  0x00001501
404 #define TEE_PANIC_ID_TEE_BIGINTFMMSIZEINU32         0x00001502
405 /* Initialization */
406 #define TEE_PANIC_ID_TEE_BIGINTINIT                 0x00001601
407 #define TEE_PANIC_ID_TEE_BIGINTINITFMM              0x00001602
408 #define TEE_PANIC_ID_TEE_BIGINTINITFMMCONTEXT       0x00001603
409 /* Converter */
410 #define TEE_PANIC_ID_TEE_BIGINTCONVERTFROMOCTETSTRING 0x00001701
411 #define TEE_PANIC_ID_TEE_BIGINTCONVERTFROMS32         0x00001702
412 #define TEE_PANIC_ID_TEE_BIGINTCONVERTTOOCTETSTRING   0x00001703
413 #define TEE_PANIC_ID_TEE_BIGINTCONVERTTOS32           0x00001704
414 /* Logical Operation */
415 #define TEE_PANIC_ID_TEE_BIGINTCMP                  0x00001801
416 #define TEE_PANIC_ID_TEE_BIGINTCMPS32               0x00001802
417 #define TEE_PANIC_ID_TEE_BIGINTGETBIT               0x00001803
418 #define TEE_PANIC_ID_TEE_BIGINTGETBITCOUNT          0x00001804
419 #define TEE_PANIC_ID_TEE_BIGINTSHIFTRIGHT           0x00001805
420 /* Basic Arithmetic */
421 #define TEE_PANIC_ID_TEE_BIGINTADD                  0x00001901
422 #define TEE_PANIC_ID_TEE_BIGINTDIV                  0x00001902
423 #define TEE_PANIC_ID_TEE_BIGINTMUL                  0x00001903
424 #define TEE_PANIC_ID_TEE_BIGINTNEG                  0x00001904
425 #define TEE_PANIC_ID_TEE_BIGINTSQUARE               0x00001905
426 #define TEE_PANIC_ID_TEE_BIGINTSUB                  0x00001906
427 /* Modular Arithmetic */
428 #define TEE_PANIC_ID_TEE_BIGINTADDMOD               0x00001A01
429 #define TEE_PANIC_ID_TEE_BIGINTINVMOD               0x00001A02
430 #define TEE_PANIC_ID_TEE_BIGINTMOD                  0x00001A03
431 #define TEE_PANIC_ID_TEE_BIGINTMULMOD               0x00001A04
432 #define TEE_PANIC_ID_TEE_BIGINTSQUAREMOD            0x00001A05
433 #define TEE_PANIC_ID_TEE_BIGINTSUBMOD               0x00001A06
434 /* Other Arithmetic */
435 #define TEE_PANIC_ID_TEE_BIGINTCOMPUTEEXTENDEDGCD   0x00001B01
436 #define TEE_PANIC_ID_TEE_BIGINTISPROBABLEPRIME      0x00001B02
437 #define TEE_PANIC_ID_TEE_BIGINTRELATIVEPRIME        0x00001B03
438 /* Fast Modular Multiplication */
439 #define TEE_PANIC_ID_TEE_BIGINTCOMPUTEFMM           0x00001C01
440 #define TEE_PANIC_ID_TEE_BIGINTCONVERTFROMFMM       0x00001C02
441 #define TEE_PANIC_ID_TEE_BIGINTCONVERTTOFMM         0x00001C03
442
443 /*
444  * The macro TEE_PARAM_TYPES can be used to construct a value that you can
445  * compare against an incoming paramTypes to check the type of all the
446  * parameters in one comparison, like in the following example:
447  * if (paramTypes != TEE_PARAM_TYPES(TEE_PARAM_TYPE_MEMREF_INPUT,
448  *                                   TEE_PARAM_TYPE_MEMREF_OUPUT,
449  *                                   TEE_PARAM_TYPE_NONE, TEE_PARAM_TYPE_NONE)) {
```

```
450  *      return TEE_ERROR_BAD_PARAMETERS;
451  *  }
452  */
453 #define TEE_PARAM_TYPES(t0,t1,t2,t3) \
454    ((t0) | ((t1) « 4) | ((t2) « 8) | ((t3) « 12))
455
456 /*
457  * The macro TEE_PARAM_TYPE_GET can be used to extract the type of a given
458  * parameter from paramTypes if you need more fine-grained type checking.
459  */
460 #define TEE_PARAM_TYPE_GET(t, i) ((((uint32_t)t) » ((i)*4)) & 0xF)
461
462 /*
463  * The macro TEE_PARAM_TYPE_SET can be used to load the type of a given
464  * parameter from paramTypes without specifying all types (TEE_PARAM_TYPES)
465  */
466 #define TEE_PARAM_TYPE_SET(t, i) (((uint32_t)(t) & 0xF) « ((i)*4))
467
468 /* Not specified in the standard */
469 #define TEE_NUM_PARAMS  4
470
471 /* TEE Arithmetical APIs */
472
473 #define TEE_BigIntSizeInU32(n) ((((n)+31)/32)+2)
474
475 #endif /*DOXYGEN_SHOULD_SKIP_THIS*/
476 #endif /* TEE_API_DEFINES_H */
```

## 4.13 ta-ref/api/include/tee_api_defines_extensions.h File Reference

This graph shows which files directly or indirectly include this file:




## 4.14 tee_api_defines_extensions.h

Go to the documentation of this file.

```
1 /*
2  * Copyright (c) 2014, Linaro Limited
3  * All rights reserved.
4  *
5  * Redistribution and use in source and binary forms, with or without
6  * modification, are permitted provided that the following conditions are met:
7  *
8  * 1. Redistributions of source code must retain the above copyright notice,
9  * this list of conditions and the following disclaimer.
10  *
11  * 2. Redistributions in binary form must reproduce the above copyright notice,
12  * this list of conditions and the following disclaimer in the documentation
13  * and/or other materials provided with the distribution.
```

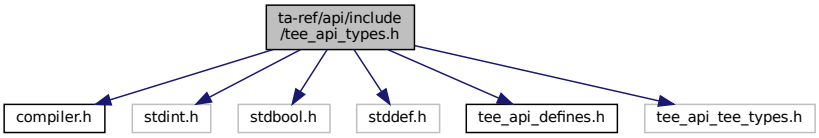
```
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

#ifndef TEE_API_DEFINES_EXTENSIONS_H
#define TEE_API_DEFINES_EXTENSIONS_H
#ifndef DOXYGEN_SHOULD_SKIP_THIS

/*
 * HMAC-based Extract-and-Expand Key Derivation Function (HKDF)
 */

#define TEE_ALG_HKDF_MD5_DERIVE_KEY     0x800010C0
#define TEE_ALG_HKDF_SHA1_DERIVE_KEY    0x800020C0
#define TEE_ALG_HKDF_SHA224_DERIVE_KEY  0x800030C0
#define TEE_ALG_HKDF_SHA256_DERIVE_KEY  0x800040C0
#define TEE_ALG_HKDF_SHA384_DERIVE_KEY  0x800050C0
#define TEE_ALG_HKDF_SHA512_DERIVE_KEY  0x800060C0

#define TEE_TYPE_HKDF_IKM               0xA10000C0

#define TEE_ATTR_HKDF_IKM               0xC00001C0
#define TEE_ATTR_HKDF_SALT              0xD00002C0
#define TEE_ATTR_HKDF_INFO              0xD00003C0
#define TEE_ATTR_HKDF_OKM_LENGTH        0xF00004C0

/*
 * Concatenation Key Derivation Function (Concat KDF)
 * NIST SP 800-56A section 5.8.1
 */

#define TEE_ALG_CONCAT_KDF_SHA1_DERIVE_KEY    0x800020C1
#define TEE_ALG_CONCAT_KDF_SHA224_DERIVE_KEY  0x800030C1
#define TEE_ALG_CONCAT_KDF_SHA256_DERIVE_KEY  0x800040C1
#define TEE_ALG_CONCAT_KDF_SHA384_DERIVE_KEY  0x800050C1
#define TEE_ALG_CONCAT_KDF_SHA512_DERIVE_KEY  0x800060C1

#define TEE_TYPE_CONCAT_KDF_Z                 0xA10000C1

#define TEE_ATTR_CONCAT_KDF_Z                 0xC00001C1
#define TEE_ATTR_CONCAT_KDF_OTHER_INFO        0xD00002C1
#define TEE_ATTR_CONCAT_KDF_DKM_LENGTH        0xF00003C1

/*
 * PKCS #5 v2.0 Key Derivation Function 2 (PBKDF2)
 * RFC 2898 section 5.2
 * https://www.ietf.org/rfc/rfc2898.txt
 */

#define TEE_ALG_PBKDF2_HMAC_SHA1_DERIVE_KEY 0x800020C2

#define TEE_TYPE_PBKDF2_PASSWORD            0xA10000C2

#define TEE_ATTR_PBKDF2_PASSWORD            0xC00001C2
#define TEE_ATTR_PBKDF2_SALT                0xD00002C2
#define TEE_ATTR_PBKDF2_ITERATION_COUNT     0xF00003C2
#define TEE_ATTR_PBKDF2_DKM_LENGTH          0xF00004C2

/*
 * Implementation-specific object storage constants
 */

/* Storage is provided by the Rich Execution Environment (REE) */
#define TEE_STORAGE_PRIVATE_REE  0x80000000
/* Storage is the Replay Protected Memory Block partition of an eMMC device */
#define TEE_STORAGE_PRIVATE_RPMB 0x80000100
/* Was TEE_STORAGE_PRIVATE_SQL, which isn't supported any longer */
#define TEE_STORAGE_PRIVATE_SQL_RESERVED  0x80000200

/*
 * Extension of "Memory Access Rights Constants"
 * #define TEE_MEMORY_ACCESS_READ             0x00000001
 * #define TEE_MEMORY_ACCESS_WRITE            0x00000002
 * #define TEE_MEMORY_ACCESS_ANY_OWNER        0x00000004
 *
```

```
99  * TEE_MEMORY_ACCESS_NONSECURE : if set TEE_CheckMemoryAccessRights()
100  * successfully returns only if target vmem range is mapped non-secure.
101  *
102  * TEE_MEMORY_ACCESS_SECURE : if set TEE_CheckMemoryAccessRights()
103  * successfully returns only if target vmem range is mapped secure.
104
105  */
106 #define TEE_MEMORY_ACCESS_NONSECURE          0x10000000
107 #define TEE_MEMORY_ACCESS_SECURE             0x20000000
108
109 #endif /*DOXYGEN_SHOULD_SKIP_THIS*/
110 #endif /* TEE_API_DEFINES_EXTENSIONS_H */
```

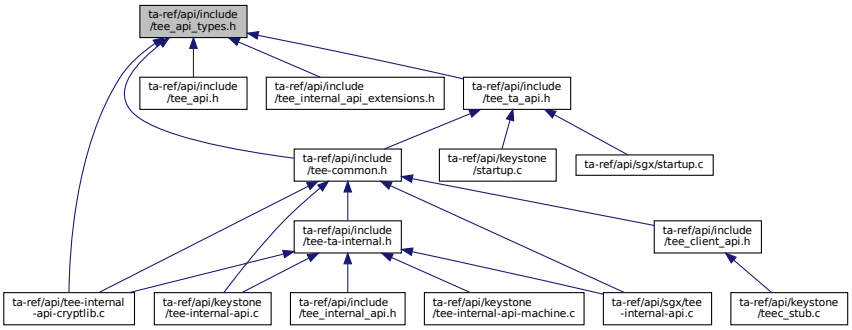
## 4.15 ta-ref/api/include/tee_api_types.h File Reference

```
#include <compiler.h>
#include <stdint.h>
#include <stdbool.h>
#include <stddef.h>
#include <tee_api_defines.h>
#include "tee_api_tee_types.h"
```

Include dependency graph for tee_api_types.h:




This graph shows which files directly or indirectly include this file:




### Classes

- struct TEE_UUID
- struct TEE_Identity
- union TEE_Param
- struct TEE_ObjectInfo
- struct TEE_Attribute

- struct TEE_OperationInfo
- struct TEE_OperationInfoKey
- struct TEE_OperationInfoMultiple
- struct TEE_Time
- struct TEE_SEReaderProperties
- struct TEE_SEAID
- struct pollfd
- struct addrinfo

### Typedefs

- typedef uint32_t TEE_Result
- typedef struct __TEE_TASessionHandle * TEE_TASessionHandle
- typedef struct __TEE_PropSetHandle * TEE_PropSetHandle
- typedef struct __TEE_ObjectHandle * TEE_ObjectHandle
- typedef struct __TEE_ObjectEnumHandle * TEE_ObjectEnumHandle
- typedef struct __TEE_OperationHandle * TEE_OperationHandle
- typedef uint32_t TEE_ObjectType
- typedef uint32_t TEE_BigInt
- typedef uint32_t TEE_BigIntFMM
- typedef uint32_t TEE_BigIntFMMContext __aligned(__alignof__(void *))
- typedef struct __TEE_SEServiceHandle * TEE_SEServiceHandle
- typedef struct __TEE_SEReaderHandle * TEE_SEReaderHandle
- typedef struct __TEE_SESessionHandle * TEE_SESessionHandle
- typedef struct __TEE_SEChannelHandle * TEE_SEChannelHandle
- typedef uint32_t TEE_ErrorOrigin
- typedef void * TEE_Session
- typedef unsigned long int nfds_t
- typedef uint32_t socklen_t

### Enumerations

- enum TEE_Whence { TEE_DATA_SEEK_SET = 0 , TEE_DATA_SEEK_CUR = 1 , TEE_DATA_SEEK_END = 2 }
- enum TEE_OperationMode {
  TEE_MODE_ENCRYPT = 0 , TEE_MODE_DECRYPT = 1 , TEE_MODE_SIGN = 2 , TEE_MODE_VERIFY = 3 ,
  TEE_MODE_MAC = 4 , TEE_MODE_DIGEST = 5 , TEE_MODE_DERIVE = 6 }

### 4.15.1 Typedef Documentation

#### 4.15.1.1 __aligned `typedef uint32_t TEE_BigIntFMMContext __aligned(__alignof__(void *))`

#### 4.15.1.2 nfds_t `typedef unsigned long int nfds_t`

#### 4.15.1.3 socklen_t

```
typedef uint32_t socklen_t
```

#### 4.15.1.4 TEE_BigInt

```
typedef uint32_t TEE_BigInt
```

#### 4.15.1.5 TEE_BigIntFMM

```
typedef uint32_t TEE_BigIntFMM
```

#### 4.15.1.6 TEE_ErrorOrigin

```
typedef uint32_t TEE_ErrorOrigin
```

#### 4.15.1.7 TEE_ObjectEnumHandle

```
typedef struct __TEE_ObjectEnumHandle* TEE_ObjectEnumHandle
```

#### 4.15.1.8 TEE_ObjectHandle

```
typedef struct __TEE_ObjectHandle* TEE_ObjectHandle
```

#### 4.15.1.9 TEE_ObjectType

```
typedef uint32_t TEE_ObjectType
```

#### 4.15.1.10 TEE_OperationHandle

```
typedef struct __TEE_OperationHandle* TEE_OperationHandle
```

#### 4.15.1.11 TEE_PropSetHandle

```
typedef struct __TEE_PropSetHandle* TEE_PropSetHandle
```

#### 4.15.1.12 TEE_Result

```
typedef uint32_t TEE_Result
```

#### 4.15.1.13 TEE_SEChannelHandle

```
typedef struct __TEE_SEChannelHandle* TEE_SEChannelHandle
```

#### 4.15.1.14 TEE_SEReaderHandle `typedef struct __TEE_SEReaderHandle* TEE_SEReaderHandle`

#### 4.15.1.15 TEE_SEServiceHandle `typedef struct __TEE_SEServiceHandle* TEE_SEServiceHandle`

#### 4.15.1.16 TEE_SESessionHandle `typedef struct __TEE_SESessionHandle* TEE_SESessionHandle`

#### 4.15.1.17 TEE_Session `typedef void* TEE_Session`

#### 4.15.1.18 TEE_TASessionHandle `typedef struct __TEE_TASessionHandle* TEE_TASessionHandle`

### 4.15.2 Enumeration Type Documentation

#### 4.15.2.1 TEE_OperationMode `enum TEE_OperationMode`

**Enumerator**

| | |
|---|---|
| TEE_MODE_ENCRYPT | |
| TEE_MODE_DECRYPT | |
| TEE_MODE_SIGN | |
| TEE_MODE_VERIFY | |
| TEE_MODE_MAC | |
| TEE_MODE_DIGEST | |
| TEE_MODE_DERIVE | |

#### 4.15.2.2 TEE_Whence `enum TEE_Whence`

**Enumerator**

| | |
|---|---|
| TEE_DATA_SEEK_SET | |
| TEE_DATA_SEEK_CUR | |
| TEE_DATA_SEEK_END | |

## 4.16 tee_api_types.h

Go to the documentation of this file.

```
1 /*
2  * Copyright (c) 2014, STMicroelectronics International N.V.
3  * All rights reserved.
4  *
5  * Redistribution and use in source and binary forms, with or without
6  * modification, are permitted provided that the following conditions are met:
7  *
8  * 1. Redistributions of source code must retain the above copyright notice,
9  * this list of conditions and the following disclaimer.
10  *
11  * 2. Redistributions in binary form must reproduce the above copyright notice,
12  * this list of conditions and the following disclaimer in the documentation
13  * and/or other materials provided with the distribution.
14  *
15  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
16  * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
17  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
18  * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
19  * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
20  * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
21  * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
22  * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
23  * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
24  * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
25  * POSSIBILITY OF SUCH DAMAGE.
26  */
27
28 /* Based on GP TEE Internal API Specification Version 0.11 */
29 #ifndef TEE_API_TYPES_H
30 #define TEE_API_TYPES_H
31
32 #include <compiler.h>
33 #include <stdint.h>
34 #include <stdbool.h>
35 #include <stddef.h>
36 #include <tee_api_defines.h>
37 #include "tee_api_tee_types.h"
38
39 /*
40  * Common Definitions
41  */
42
43 typedef uint32_t TEE_Result;
44
45 typedef struct {
46     uint32_t timeLow;
47     uint16_t timeMid;
48     uint16_t timeHiAndVersion;
49     uint8_t clockSeqAndNode[8];
50 } TEE_UUID;
51
52 /*
53  * The TEE_Identity structure defines the full identity of a Client:
54  * - login is one of the TEE_LOGIN_XXX constants
55  * - uuid contains the client UUID or Nil if not applicable
56  */
57 typedef struct {
58     uint32_t login;
59     TEE_UUID uuid;
60 } TEE_Identity;
61
62 /*
63  * This union describes one parameter passed by the Trusted Core Framework
64  * to the entry points TA_OpenSessionEntryPoint or
65  * TA_InvokeCommandEntryPoint or by the TA to the functions
66  * TEE_OpenTASession or TEE_InvokeTACommand.
67  *
68  * Which of the field value or memref to select is determined by the
69  * parameter type specified in the argument paramTypes passed to the entry
70  * point.
71 */
72 typedef union {
73     struct {
74         void *buffer;
75         uint32_t size;
76     } memref;
77     struct {
78         uint32_t a;
79         uint32_t b;
80     } value;
81 } TEE_Param;
```

```
82
83 /*
84  * The type of opaque handles on TA Session. These handles are returned by
85  * the function TEE_OpenTASession.
86  */
87 typedef struct __TEE_TASessionHandle *TEE_TASessionHandle;
88
89 /*
90  * The type of opaque handles on property sets or enumerators. These
91  * handles are either one of the pseudo handles TEE_PROPSET_XXX or are
92  * returned by the function TEE_AllocatePropertyEnumerator.
93 */
94 typedef struct __TEE_PropSetHandle *TEE_PropSetHandle;
95
96 typedef struct __TEE_ObjectHandle *TEE_ObjectHandle;
97 typedef struct __TEE_ObjectEnumHandle *TEE_ObjectEnumHandle;
98 typedef struct __TEE_OperationHandle *TEE_OperationHandle;
99
100 /*
101  * Storage Definitions
102  */
103
104 typedef uint32_t TEE_ObjectType;
105
106 typedef struct {
107     uint32_t objectType;
108     __extension__ union {
109         uint32_t keySize;    /* used in 1.1 spec */
110         uint32_t objectSize;  /* used in 1.1.1 spec */
111     };
112     __extension__ union {
113         uint32_t maxKeySize;  /* used in 1.1 spec */
114         uint32_t maxObjectSize;    /* used in 1.1.1 spec */
115     };
116     uint32_t objectUsage;
117     uint32_t dataSize;
118     uint32_t dataPosition;
119     uint32_t handleFlags;
120 } TEE_ObjectInfo;
121
122 typedef enum {
123     TEE_DATA_SEEK_SET = 0,
124     TEE_DATA_SEEK_CUR = 1,
125     TEE_DATA_SEEK_END = 2
126 } TEE_Whence;
127
128 typedef struct {
129     uint32_t attributeID;
130     union {
131         struct {
132             void *buffer;
133             uint32_t length;
134         } ref;
135         struct {
136             uint32_t a, b;
137         } value;
138     } content;
139 } TEE_Attribute;
140
141 #ifndef DOXYGEN_SHOULD_SKIP_THIS
142 #define DMREQ_FINISH 0
143 #define DMREQ_WRITE 1
144 #endif /*DOXYGEN_SHOULD_SKIP_THIS*/
145
146 /* Cryptographic Operations API */
147
148 typedef enum {
149     TEE_MODE_ENCRYPT = 0,
150     TEE_MODE_DECRYPT = 1,
151     TEE_MODE_SIGN = 2,
152     TEE_MODE_VERIFY = 3,
153     TEE_MODE_MAC = 4,
154     TEE_MODE_DIGEST = 5,
155     TEE_MODE_DERIVE = 6
156 } TEE_OperationMode;
157
158 typedef struct {
159     uint32_t algorithm;
160     uint32_t operationClass;
161     uint32_t mode;
162     uint32_t digestLength;
163     uint32_t maxKeySize;
164     uint32_t keySize;
165     uint32_t requiredKeyUsage;
166     uint32_t handleState;
```

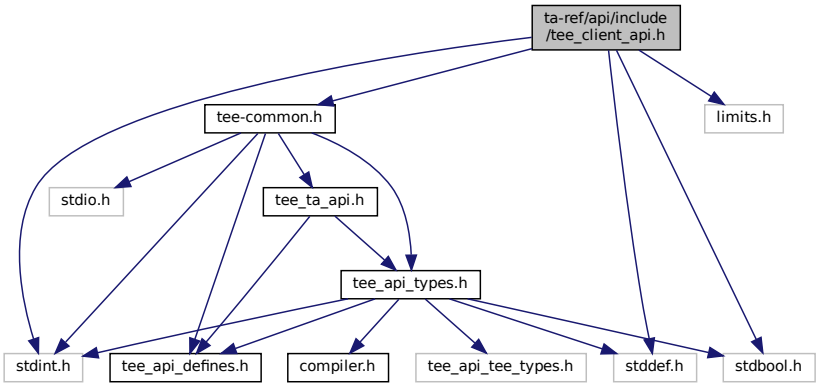
```
167 } TEE_OperationInfo;
168
169 typedef struct {
170     uint32_t keySize;
171     uint32_t requiredKeyUsage;
172 } TEE_OperationInfoKey;
173
174 typedef struct {
175     uint32_t algorithm;
176     uint32_t operationClass;
177     uint32_t mode;
178     uint32_t digestLength;
179     uint32_t maxKeySize;
180     uint32_t handleState;
181     uint32_t operationState;
182     uint32_t numberOfKeys;
183     TEE_OperationInfoKey keyInformation[];
184 } TEE_OperationInfoMultiple;
185
186 /* Time & Date API */
187
188 typedef struct {
189     uint32_t seconds;
190     uint32_t millis;
191 } TEE_Time;
192
193 /* TEE Arithmetical APIs */
194
195 typedef uint32_t TEE_BigInt;
196
197 typedef uint32_t TEE_BigIntFMM;
198
199 typedef uint32_t TEE_BigIntFMMContext __aligned(__alignof__(void *));
200
201 /* Tee Secure Element APIs */
202
203 typedef struct __TEE_SEServiceHandle *TEE_SEServiceHandle;
204 typedef struct __TEE_SEReaderHandle *TEE_SEReaderHandle;
205 typedef struct __TEE_SESessionHandle *TEE_SESessionHandle;
206 typedef struct __TEE_SEChannelHandle *TEE_SEChannelHandle;
207
208 typedef struct {
209     bool sePresent;
210     bool teeOnly;
211     bool selectResponseEnable;
212 } TEE_SEReaderProperties;
213
214 typedef struct {
215     uint8_t *buffer;
216     size_t bufferLen;
217 } TEE_SEAID;
218
219 /* Other definitions */
220 typedef uint32_t TEE_ErrorOrigin;
221 typedef void *TEE_Session;
222
223 #ifndef DOXYGEN_SHOULD_SKIP_THIS
224 #define TEE_MEM_INPUT   0x00000001
225 #define TEE_MEM_OUTPUT  0x00000002
226
227 #define TEE_MEMREF_0_USED  0x00000001
228 #define TEE_MEMREF_1_USED  0x00000002
229 #define TEE_MEMREF_2_USED  0x00000004
230 #define TEE_MEMREF_3_USED  0x00000008
231
232 #define TEE_SE_READER_NAME_MAX  20
233 #endif /*DOXYGEN_SHOULD_SKIP_THIS*/
234
235 #ifndef PLAT_KEYSTONE
236 // TODO: ???
237
238 typedef unsigned long int nfds_t;
239
240 struct pollfd
241 {
242         int fd;                     /* File descriptor to poll.  */
243     short int events;           /* Types of events poller cares about.  */
244     short int revents;          /* Types of events that actually occurred.  */
245 };
246
247 typedef uint32_t socklen_t;
248
249 struct addrinfo {
250     int              ai_flags;
251     int              ai_family;
```

```
252     int              ai_socktype;
253     int              ai_protocol;
254     socklen_t        ai_addrlen;
255     struct sockaddr *ai_addr;
256     char            *ai_canonname;
257     struct addrinfo *ai_next;
258 };
259
260 #endif /* !PLAT_KEYSTONE */
261
262 #endif /* TEE_API_TYPES_H */
```

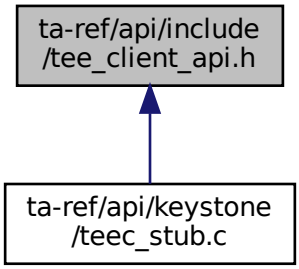
## 4.17 ta-ref/api/include/tee_client_api.h File Reference

```
#include <stdint.h>
#include <stddef.h>
#include <stdbool.h>
#include <limits.h>
#include "tee-common.h"
```

Include dependency graph for tee_client_api.h:




This graph shows which files directly or indirectly include this file:



### Classes

- struct TEEC_Context
- struct TEEC_UUID
- struct TEEC_SharedMemory
- struct TEEC_TempMemoryReference
- struct TEEC_RegisteredMemoryReference
- struct TEEC_Value
- union TEEC_Parameter
- struct TEEC_Session
- struct TEEC_Operation

### Typedefs

- typedef uint32_t TEEC_Result

### Functions

- TEEC_Result TEEC_InitializeContext (const char *name, TEEC_Context *context)
- void TEEC_FinalizeContext (TEEC_Context *context)
- TEEC_Result TEEC_OpenSession (TEEC_Context *context, TEEC_Session *session, const TEEC_UUID *destination, uint32_t connectionMethod, const void *connectionData, TEEC_Operation *operation, uint32_t *returnOrigin)
- void TEEC_CloseSession (TEEC_Session *session)
- TEEC_Result TEEC_InvokeCommand (TEEC_Session *session, uint32_t commandID, TEEC_Operation *operation, uint32_t *returnOrigin)
- TEEC_Result TEEC_RegisterSharedMemory (TEEC_Context *context, TEEC_SharedMemory *sharedMem)
- TEEC_Result TEEC_AllocateSharedMemory (TEEC_Context *context, TEEC_SharedMemory *sharedMem)
- void TEEC_ReleaseSharedMemory (TEEC_SharedMemory *sharedMemory)
- void TEEC_RequestCancellation (TEEC_Operation *operation)

## 4.17.1 Typedef Documentation

### 4.17.1.1 TEEC_Result `typedef uint32_t TEEC_Result`

## 4.17.2 Function Documentation

### 4.17.2.1 TEEC_AllocateSharedMemory()

```
TEEC_Result TEEC_AllocateSharedMemory (
        TEEC_Context * context,
        TEEC_SharedMemory * sharedMem )
```

TEEC_AllocateSharedMemory() - Allocate shared memory for TEE.

**Parameters**

| | |
|---|---|
| *context* | The initialized TEE context structure in which scope to open the session. |
| *sharedMem* | Pointer to the allocated shared memory. |

**Returns**

TEEC_SUCCESS The registration was successful.

TEEC_ERROR_OUT_OF_MEMORY Memory exhaustion.

TEEC_Result Something failed.

#### 4.17.2.2 TEEC_CloseSession() `void TEEC_CloseSession (`

`TEEC_Session * session )`

TEEC_CloseSession() - Closes the session which has been opened with the specific trusted application.

**Parameters**

| | |
|---|---|
| *session* | The opened session to close. |

#### 4.17.2.3 TEEC_FinalizeContext() `void TEEC_FinalizeContext (`

`TEEC_Context * context )`

TEEC_FinalizeContext() - Destroys a context holding connection information on the specific TEE.

This function destroys an initialized TEE context, closing the connection between the client application and the TEE. This function must only be called when all sessions related to this TEE context have been closed and all shared memory blocks have been released.

**Parameters**

| | |
|---|---|
| *context* | The context to be destroyed. |

TEEC_FinalizeContext() - Destroys a context holding connection information on the specific TEE.

This function finalizes an initialized TEE context, closing the connection between the client application and the TEE. This function must only be called when all sessions related to this TEE context have been closed and all shared memory blocks have been released.

**Parameters**

| | |
|---|---|
| *context* | The context to be finalized. |

#### 4.17.2.4 TEEC_InitializeContext()

```
TEEC_Result TEEC_InitializeContext (
    const char * name,
    TEEC_Context * context )
```

TEEC_InitializeContext() - Initializes a context holding connection information on the specific TEE, designated by the name string.

**Parameters**

| | |
|---|---|
| *name* | A zero-terminated string identifying the TEE to connect to. If name is set to NULL, the default TEE is connected to. NULL is the only supported value in this version of the API implementation. |
| *context* | The context structure which is to be initialized. |

**Returns**

TEEC_SUCCESS The initialization was successful.

TEEC_Result Something failed.

#### 4.17.2.5 TEEC_InvokeCommand()

```
TEEC_Result TEEC_InvokeCommand (
    TEEC_Session * session,
    uint32_t commandID,
    TEEC_Operation * operation,
    uint32_t * returnOrigin )
```

TEEC_InvokeCommand() - Executes a command in the specified trusted application.

**Parameters**

| | |
|---|---|
| *session* | A handle to an open connection to the trusted application. |
| *commandID* | Identifier of the command in the trusted application to invoke. |
| *operation* | An operation structure to use in the invoke command. May be set to NULL to signify no operation structure needed. |
| *returnOrigin* | A parameter which will hold the error origin if this function returns any value other than TEEC_SUCCESS. |

**Returns**

TEEC_SUCCESS OpenSession successfully opened a new session.

TEEC_Result Something failed.

#### 4.17.2.6 TEEC_OpenSession()

```
TEEC_Result TEEC_OpenSession (
    TEEC_Context * context,
    TEEC_Session * session,
    const TEEC_UUID * destination,
    uint32_t connectionMethod,
    const void * connectionData,
    TEEC_Operation * operation,
    uint32_t * returnOrigin )
```

TEEC_OpenSession() - Opens a new session with the specified trusted application.

**Parameters**

| | |
|---|---|
| *context* | The initialized TEE context structure in which scope to open the session. |
| *session* | The session to initialize. |
| *destination* | A structure identifying the trusted application with which to open a session. |
| *connectionMethod* | The connection method to use. |
| *connectionData* | Any data necessary to connect with the chosen connection method. Not supported, should be set to NULL. |
| *operation* | An operation structure to use in the session. May be set to NULL to signify no operation structure needed. |
| *returnOrigin* | A parameter which will hold the error origin if this function returns any value other than TEEC_SUCCESS. |

**Returns**

TEEC_SUCCESS OpenSession successfully opened a new session.

TEEC_Result Something failed.

#### 4.17.2.7 TEEC_RegisterSharedMemory()

```
TEEC_Result TEEC_RegisterSharedMemory (
    TEEC_Context * context,
    TEEC_SharedMemory * sharedMem )
```

TEEC_RegisterSharedMemory() - Register a block of existing memory as a shared block within the scope of the specified context.

**Parameters**

| | |
|---|---|
| *context* | The initialized TEE context structure in which scope to open the session. |
| *sharedMem* | pointer to the shared memory structure to register. |

**Returns**

TEEC_SUCCESS The registration was successful.

TEEC_ERROR_OUT_OF_MEMORY Memory exhaustion.

TEEC_Result Something failed.

#### 4.17.2.8 TEEC_ReleaseSharedMemory() `void TEEC_ReleaseSharedMemory ( TEEC_SharedMemory * sharedMemory )`

TEEC_ReleaseSharedMemory() - Free or deregister the shared memory.

**Parameters**

| | |
|---|---|
| *sharedMem* | Pointer to the shared memory to be freed. |

#### 4.17.2.9 TEEC_RequestCancellation() `void TEEC_RequestCancellation ( TEEC_Operation * operation )`

TEEC_RequestCancellation() - Request the cancellation of a pending open session or command invocation.

**Parameters**

| | |
|---|---|
| *operation* | Pointer to an operation previously passed to open session or invoke. |

## 4.18 tee_client_api.h

Go to the documentation of this file.

```
1 /*
2  * Copyright (c) 2014, STMicroelectronics International N.V.
3  * All rights reserved.
4  * Copyright (c) 2015, Linaro Limited
5  * All rights reserved.
6  *
7  * Redistribution and use in source and binary forms, with or without
8  * modification, are permitted provided that the following conditions are met:
9  *
10  * 1. Redistributions of source code must retain the above copyright notice,
11  * this list of conditions and the following disclaimer.
12  *
13  * 2. Redistributions in binary form must reproduce the above copyright notice,
14  * this list of conditions and the following disclaimer in the documentation
15  * and/or other materials provided with the distribution.
16  *
17  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
18  * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
19  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
20  * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
21  * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
22  * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
23  * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
```

```
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */
#ifndef TEE_CLIENT_API_H
#define TEE_CLIENT_API_H

#ifdef __cplusplus
extern "C" {
#endif

#include <stdint.h>
#include <stddef.h>
#include <stdbool.h>
#include <limits.h>
#include "tee-common.h"

#ifndef DOXYGEN_SHOULD_SKIP_THIS
/*
 * Defines the number of available memory references in an open session or
 * invoke command operation payload.
 */
#define TEEC_CONFIG_PAYLOAD_REF_COUNT 4

#define TEEC_CONFIG_SHAREDMEM_MAX_SIZE ULONG_MAX

#define TEEC_NONE                   0x00000000
#define TEEC_VALUE_INPUT            0x00000001
#define TEEC_VALUE_OUTPUT           0x00000002
#define TEEC_VALUE_INOUT            0x00000003
#define TEEC_MEMREF_TEMP_INPUT      0x00000005
#define TEEC_MEMREF_TEMP_OUTPUT     0x00000006
#define TEEC_MEMREF_TEMP_INOUT      0x00000007
#define TEEC_MEMREF_WHOLE           0x0000000C
#define TEEC_MEMREF_PARTIAL_INPUT   0x0000000D
#define TEEC_MEMREF_PARTIAL_OUTPUT  0x0000000E
#define TEEC_MEMREF_PARTIAL_INOUT   0x0000000F

#define TEEC_MEM_INPUT   0x00000001
#define TEEC_MEM_OUTPUT  0x00000002

#define TEEC_SUCCESS                0x00000000
#define TEEC_ERROR_GENERIC          0xFFFF0000
#define TEEC_ERROR_ACCESS_DENIED    0xFFFF0001
#define TEEC_ERROR_CANCEL           0xFFFF0002
#define TEEC_ERROR_ACCESS_CONFLICT  0xFFFF0003
#define TEEC_ERROR_EXCESS_DATA      0xFFFF0004
#define TEEC_ERROR_BAD_FORMAT       0xFFFF0005
#define TEEC_ERROR_BAD_PARAMETERS   0xFFFF0006
#define TEEC_ERROR_BAD_STATE        0xFFFF0007
#define TEEC_ERROR_ITEM_NOT_FOUND   0xFFFF0008
#define TEEC_ERROR_NOT_IMPLEMENTED  0xFFFF0009
#define TEEC_ERROR_NOT_SUPPORTED    0xFFFF000A
#define TEEC_ERROR_NO_DATA          0xFFFF000B
#define TEEC_ERROR_OUT_OF_MEMORY    0xFFFF000C
#define TEEC_ERROR_BUSY             0xFFFF000D
#define TEEC_ERROR_COMMUNICATION    0xFFFF000E
#define TEEC_ERROR_SECURITY         0xFFFF000F
#define TEEC_ERROR_SHORT_BUFFER     0xFFFF0010
#define TEEC_ERROR_EXTERNAL_CANCEL  0xFFFF0011
#define TEEC_ERROR_TARGET_DEAD      0xFFFF3024

#define TEEC_ORIGIN_API          0x00000001
#define TEEC_ORIGIN_COMMS        0x00000002
#define TEEC_ORIGIN_TEE          0x00000003
#define TEEC_ORIGIN_TRUSTED_APP  0x00000004

#define TEEC_LOGIN_PUBLIC       0x00000000
#define TEEC_LOGIN_USER         0x00000001
#define TEEC_LOGIN_GROUP        0x00000002
#define TEEC_LOGIN_APPLICATION  0x00000004
#define TEEC_LOGIN_USER_APPLICATION  0x00000005
#define TEEC_LOGIN_GROUP_APPLICATION  0x00000006

#define TEEC_PARAM_TYPES(p0, p1, p2, p3) \
    ((p0) | ((p1) << 4) | ((p2) << 8) | ((p3) << 12))

#define TEEC_PARAM_TYPE_GET(p, i) (((p) >> (i * 4)) & 0xF)
#endif /*DOXYGEN_SHOULD_SKIP_THIS*/

typedef uint32_t TEEC_Result;

typedef struct {
    /* Implementation defined */
```

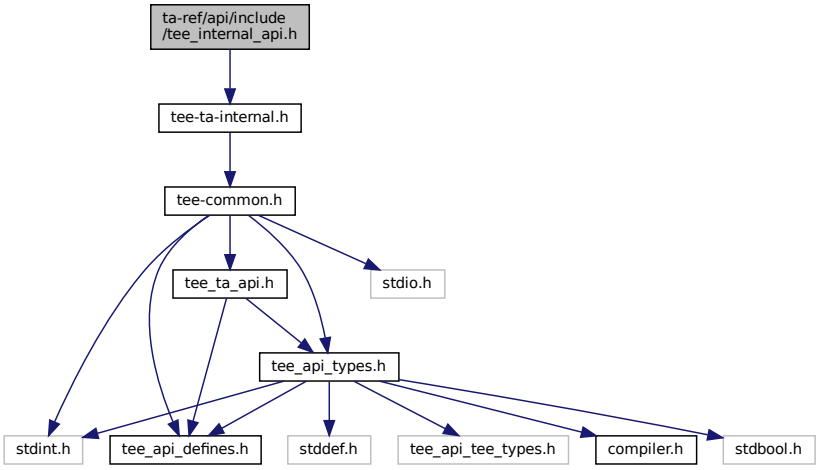
```
259     int fd;
260     bool reg_mem;
261 } TEEC_Context;
262
268 typedef struct {
269     uint32_t timeLow;
270     uint16_t timeMid;
271     uint16_t timeHiAndVersion;
272     uint8_t clockSeqAndNode[8];
273 } TEEC_UUID;
274
291 typedef struct {
292     void *buffer;
293     size_t size;
294     uint32_t flags;
295     /*
296      * Implementation-Defined
297      */
298     int id;
299     size_t alloced_size;
300     void *shadow_buffer;
301     int registered_fd;
302     bool buffer_allocated;
303 } TEEC_SharedMemory;
304
317 typedef struct {
318     void *buffer;
319     size_t size;
320 } TEEC_TempMemoryReference;
321
337 typedef struct {
338     TEEC_SharedMemory *parent;
339     size_t size;
340     size_t offset;
341 } TEEC_RegisteredMemoryReference;
342
353 typedef struct {
354     uint32_t a;
355     uint32_t b;
356 } TEEC_Value;
357
372 typedef union {
373     TEEC_TempMemoryReference tmpref;
374     TEEC_RegisteredMemoryReference memref;
375     TEEC_Value value;
376 } TEEC_Parameter;
377
382 typedef struct {
383     /* Implementation defined */
384     TEEC_Context *ctx;
385     uint32_t session_id;
386 } TEEC_Session;
387
402 typedef struct {
403     uint32_t started;
404     uint32_t paramTypes;
405     TEEC_Parameter params[TEEC_CONFIG_PAYLOAD_REF_COUNT];
406     /* Implementation-Defined */
407     TEEC_Session *session;
408 } TEEC_Operation;
409
424 TEEC_Result TEEC_InitializeContext(const char *name, TEEC_Context *context);
425
437 void TEEC_FinalizeContext(TEEC_Context *context);
438
465 TEEC_Result TEEC_OpenSession(TEEC_Context *context,
466                 TEEC_Session *session,
467                 const TEEC_UUID *destination,
468                 uint32_t connectionMethod,
469                 const void *connectionData,
470                 TEEC_Operation *operation,
471                 uint32_t *returnOrigin);
472
479 void TEEC_CloseSession(TEEC_Session *session);
480
498 TEEC_Result TEEC_InvokeCommand(TEEC_Session *session,
499                   uint32_t commandID,
500                   TEEC_Operation *operation,
501                   uint32_t *returnOrigin);
502
515 TEEC_Result TEEC_RegisterSharedMemory(TEEC_Context *context,
516                       TEEC_SharedMemory *sharedMem);
517
529 TEEC_Result TEEC_AllocateSharedMemory(TEEC_Context *context,
530                       TEEC_SharedMemory *sharedMem);
```

```
531
537 void TEEC_ReleaseSharedMemory(TEEC_SharedMemory *sharedMemory);
538
546 void TEEC_RequestCancellation(TEEC_Operation *operation);
547
548 #ifdef __cplusplus
549 }
550 #endif
551
552 #endif
```

## 4.19 ta-ref/api/include/tee_internal_api.h File Reference

```
#include "tee-ta-internal.h"
```

Include dependency graph for tee_internal_api.h:




## 4.20 tee_internal_api.h

Go to the documentation of this file.

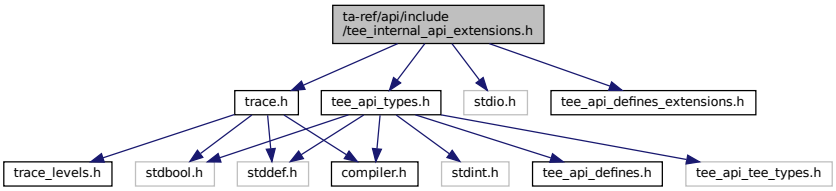
```
1 #include "tee-ta-internal.h"
```

## 4.21 ta-ref/api/include/tee_internal_api_extensions.h File Reference

```
#include <trace.h>
#include <stdio.h>
#include <tee_api_defines_extensions.h>
```

```
#include <tee_api_types.h>
```

Include dependency graph for tee_internal_api_extensions.h:




## Functions

- void tee_user_mem_mark_heap (void)
- size_t tee_user_mem_check_heap (void)
- TEE_Result TEE_CacheClean (char ∗buf, size_t len)
- TEE_Result TEE_CacheFlush (char ∗buf, size_t len)
- TEE_Result TEE_CacheInvalidate (char ∗buf, size_t len)
- void ∗ tee_map_zi (size_t len, uint32_t flags)
- TEE_Result tee_unmap (void ∗buf, size_t len)
- TEE_Result tee_uuid_from_str (TEE_UUID ∗uuid, const char ∗s)

### 4.21.1 Function Documentation

#### 4.21.1.1 TEE_CacheClean()

```
TEE_Result TEE_CacheClean (
        char * buf,
        size_t len )
```

#### 4.21.1.2 TEE_CacheFlush()

```
TEE_Result TEE_CacheFlush (
        char * buf,
        size_t len )
```

#### 4.21.1.3 TEE_CacheInvalidate()

```
TEE_Result TEE_CacheInvalidate (
        char * buf,
        size_t len )
```

#### 4.21.1.4 tee_map_zi()

```
void * tee_map_zi (
        size_t len,
        uint32_t flags )
```

#### 4.21.1.5 tee_unmap()

```
TEE_Result tee_unmap (
        void * buf,
        size_t len )
```

#### 4.21.1.6 tee_user_mem_check_heap()

```
size_t tee_user_mem_check_heap (
        void  )
```

#### 4.21.1.7 tee_user_mem_mark_heap()

```
void tee_user_mem_mark_heap (
        void  )
```

#### 4.21.1.8 tee_uuid_from_str()

```
TEE_Result tee_uuid_from_str (
        TEE_UUID * uuid,
        const char * s )
```

## 4.22 tee_internal_api_extensions.h

Go to the documentation of this file.

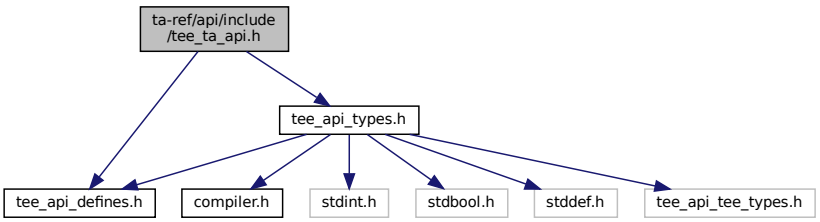
```
1 /* SPDX-License-Identifier: BSD-2-Clause */
2 /*
3  * Copyright (c) 2014, STMicroelectronics International N.V.
4  */
5
6 #ifndef TEE_INTERNAL_API_EXTENSIONS_H
7 #define TEE_INTERNAL_API_EXTENSIONS_H
8
9 /* trace support */
10 #include <trace.h>
11 #include <stdio.h>
12 #include <tee_api_defines_extensions.h>
13 #include <tee_api_types.h>
14
15 void tee_user_mem_mark_heap(void);
16 size_t tee_user_mem_check_heap(void);
17 /* Hint implementation defines */
18
19 #ifndef DOXYGEN_SHOULD_SKIP_THIS
20 #define TEE_USER_MEM_HINT_NO_FILL_ZERO       0x80000000
21 #endif /*DOXYGEN_SHOULD_SKIP_THIS*/
22
23 /*
24  * Cache maintenance support (TA requires the CACHE_MAINTENANCE property)
25  *
26  * TEE_CacheClean() Write back to memory any dirty data cache lines. The line
27  *                  is marked as not dirty. The valid bit is unchanged.
28  *
29  * TEE_CacheFlush() Purges any valid data cache lines. Any dirty cache lines
30  *                  are first written back to memory, then the cache line is
31  *                  invalidated.
32  *
33  * TEE_CacheInvalidate() Invalidate any valid data cache lines. Any dirty line
34  *                       are not written back to memory.
35  */
36 TEE_Result TEE_CacheClean(char *buf, size_t len);
37 TEE_Result TEE_CacheFlush(char *buf, size_t len);
38 TEE_Result TEE_CacheInvalidate(char *buf, size_t len);
39
40 /*
41  * tee_map_zi() - Map zero initialized memory
42  * @len:    Number of bytes
```

```
43  * @flags:  0 or TEE_MEMORY_ACCESS_ANY_OWNER to allow sharing with other TAs
44  *
45  * Returns valid pointer on success or NULL on error.
46  */
47 void *tee_map_zi(size_t len, uint32_t flags);
48
49 /*
50  * tee_unmap() - Unmap previously mapped memory
51  * @buf:    Buffer
52  * @len:    Number of bytes
53  *
54  * Note that supplied @buf and @len has to match exactly what has
55  * previously been returned by tee_map_zi().
56  *
57  * Return TEE_SUCCESS on success or TEE_ERRROR_* on failure.
58  */
59 TEE_Result tee_unmap(void *buf, size_t len);
60
61 /*
62  * Convert a UUID string @s into a TEE_UUID @uuid
63  * Expected format for @s is: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
64  * 'x' being any hexadecimal digit (0-9a-fA-F)
65  */
66 TEE_Result tee_uuid_from_str(TEE_UUID *uuid, const char *s);
67
68 #endif
```

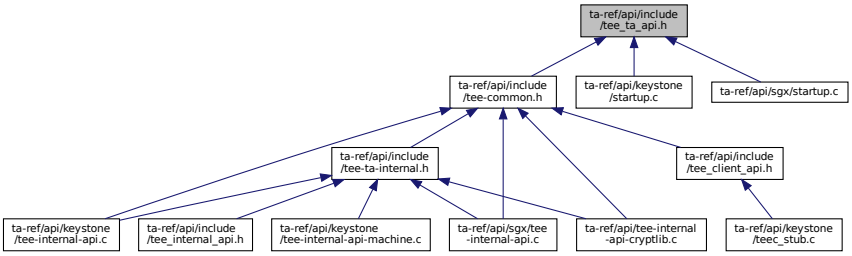
## 4.23 ta-ref/api/include/tee_ta_api.h File Reference

```
#include <tee_api_defines.h>
#include <tee_api_types.h>
```

Include dependency graph for tee_ta_api.h:




This graph shows which files directly or indirectly include this file:



### Functions

- TEE_Result TA_EXPORT TA_CreateEntryPoint (void)
- void TA_EXPORT TA_DestroyEntryPoint (void)
- TEE_Result TA_EXPORT TA_OpenSessionEntryPoint (uint32_t paramTypes, TEE_Param params[TEE_↩ NUM_PARAMS], void ∗∗sessionContext)
- void TA_EXPORT TA_CloseSessionEntryPoint (void ∗sessionContext)
- TEE_Result TA_EXPORT TA_InvokeCommandEntryPoint (void ∗sessionContext, uint32_t commandID, uint32_t paramTypes, TEE_Param params[TEE_NUM_PARAMS])

#### 4.23.1 Function Documentation

##### 4.23.1.1 TA_CloseSessionEntryPoint()

```
void TA_EXPORT TA_CloseSessionEntryPoint (
    void * sessionContext )
```

##### 4.23.1.2 TA_CreateEntryPoint()

```
TEE_Result TA_EXPORT TA_CreateEntryPoint (
    void  )
```

##### 4.23.1.3 TA_DestroyEntryPoint()

```
void TA_EXPORT TA_DestroyEntryPoint (
    void  )
```

##### 4.23.1.4 TA_InvokeCommandEntryPoint()

```
TEE_Result TA_EXPORT TA_InvokeCommandEntryPoint (
    void * sessionContext,
    uint32_t commandID,
    uint32_t paramTypes,
    TEE_Param params[TEE_NUM_PARAMS] )
```

##### 4.23.1.5 TA_OpenSessionEntryPoint()

```
TEE_Result TA_EXPORT TA_OpenSessionEntryPoint (
    uint32_t paramTypes,
    TEE_Param params[TEE_NUM_PARAMS],
    void ** sessionContext )
```

## 4.24 tee_ta_api.h

Go to the documentation of this file.

```c
/*
 * Copyright (c) 2014, STMicroelectronics International N.V.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 * this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 * this list of conditions and the following disclaimer in the documentation
 * and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/* Based on GP TEE Internal API Specification Version 0.22 */
#ifndef TEE_TA_API_H
#define TEE_TA_API_H

#include <tee_api_defines.h>
#include <tee_api_types.h>

#ifndef DOXYGEN_SHOULD_SKIP_THIS
/* This is a null define in STE TEE environment */
#define TA_EXPORT
#endif /*DOXYGEN_SHOULD_SKIP_THIS*/

/*
 * TA Interface
 *
 * Each Trusted Application must provide the Implementation with a number
 * of functions, collectively called the "TA interface". These functions
 * are the entry points called by the Trusted Core Framework to create the
 * instance, notify the instance that a new client is connecting, notify
 * the instance when the client invokes a command, etc.
 *
 * Trusted Application Entry Points:
 */

/*
 * The function TA_CreateEntryPoint is the Trusted Application's
 * constructor, which the Framework calls when it creates a new instance of
 * the Trusted Application. To register instance data, the implementation
 * of this constructor can use either global variables or the function
 * TEE_InstanceSetData.
 *
 * Return Value:
 * - TEE_SUCCESS: if the instance is successfully created, the function
 *   must return TEE_SUCCESS.
 * - Any other value: if any other code is returned the instance is not
 *   created, and no other entry points of this instance will be called.
 *   The Framework MUST reclaim all resources and dereference all objects
 *   related to the creation of the instance.
 *
 *   If this entry point was called as a result of a client opening a
 *   session, the error code is returned to the client and the session is
 *   not opened.
 */
TEE_Result TA_EXPORT TA_CreateEntryPoint(void);

/*
 * The function TA_DestroyEntryPoint is the Trusted Applications
 * destructor, which the Framework calls when the instance is being
 * destroyed.
 *
 * When the function TA_DestroyEntryPoint is called, the Framework
 * guarantees that no client session is currently open. Once the call to
 * TA_DestroyEntryPoint has been completed, no other entry point of this
 * instance will ever be called.
 *
 * Note that when this function is called, all resources opened by the
 * instance are still available. It is only after the function returns that
 * the Implementation MUST start automatically reclaiming resources left
```

```
 * opened.
 *
 * Return Value:
 * This function can return no success or error code. After this function
 * returns the Implementation MUST consider the instance destroyed and
 * reclaims all resources left open by the instance.
 */
void TA_EXPORT TA_DestroyEntryPoint(void);

/*
 * The Framework calls the function TA_OpenSessionEntryPoint when a client
 * requests to open a session with the Trusted Application. The open
 * session request may result in a new Trusted Application instance being
 * created as defined in section 4.5.
 *
 * The client can specify parameters in an open operation which are passed
 * to the Trusted Application instance in the arguments paramTypes and
 * params. These arguments can also be used by the Trusted Application
 * instance to transfer response data back to the client. See section 4.3.6
 * for a specification of how to handle the operation parameters.
 *
 * If this function returns TEE_SUCCESS, the client is connected to a
 * Trusted Application instance and can invoke Trusted Application
 * commands. When the client disconnects, the Framework will eventually
 * call the TA_CloseSessionEntryPoint entry point.
 *
 * If the function returns any error, the Framework rejects the connection
 * and returns the error code and the current content of the parameters the
 * client. The return origin is then set to TEE_ORIGIN_TRUSTED_APP.
 *
 * The Trusted Application instance can register a session data pointer by
 * setting *psessionContext. The value of this pointer is not interpreted
 * by the Framework, and is simply passed back to other TA_ functions
 * within this session. Note that *sessionContext may be set with a pointer
 * to a memory allocated by the Trusted Application instance or with
 * anything else, like an integer, a handle etc. The Framework will not
 * automatically free *sessionContext when the session is closed; the
 * Trusted Application instance is responsible for freeing memory if
 * required.
 *
 * During the call to TA_OpenSessionEntryPoint the client may request to
 * cancel the operation. See section 4.10 for more details on
 * cancellations. If the call to TA_OpenSessionEntryPoint returns
 * TEE_SUCCESS, the client must consider the session as successfully opened
 * and explicitly close it if necessary.
 *
 * Parameters:
 * - paramTypes: the types of the four parameters.
 * - params: a pointer to an array of four parameters.
 * - sessionContext: A pointer to a variable that can be filled by the
 *   Trusted Application instance with an opaque void* data pointer
 *
 * Return Value:
 * - TEE_SUCCESS if the session is successfully opened.
 * - Any other value if the session could not be open.
 *   o The error code may be one of the pre-defined codes, or may be a new
 *     error code defined by the Trusted Application implementation itself.
 */
TEE_Result TA_EXPORT TA_OpenSessionEntryPoint(uint32_t paramTypes,
                TEE_Param params[TEE_NUM_PARAMS],
                void **sessionContext);

/*
 * The Framework calls this function to close a client session. During the
 * call to this function the implementation can use any session functions.
 *
 * The Trusted Application implementation is responsible for freeing any
 * resources consumed by the session being closed. Note that the Trusted
 * Application cannot refuse to close a session, but can hold the closing
 * until it returns from TA_CloseSessionEntryPoint. This is why this
 * function cannot return an error code.
 *
 * Parameters:
 * - sessionContext: The value of the void* opaque data pointer set by the
 *   Trusted Application in the function TA_OpenSessionEntryPoint for this
 *   session.
 */
void TA_EXPORT TA_CloseSessionEntryPoint(void *sessionContext);

/*
 * The Framework calls this function when the client invokes a command
 * within the given session.
 *
 * The Trusted Application can access the parameters sent by the client
 * through the paramTypes and params arguments. It can also use these
```

```
171  * arguments to transfer response data back to the client.
172  *
173  * During the call to TA_InvokeCommandEntryPoint the client may request to
174  * cancel the operation.
175  *
176  * A command is always invoked within the context of a client session.
177  * Thus, any session function  can be called by the command implementation.
178  *
179  * Parameter:
180  * - sessionContext: The value of the void* opaque data pointer set by the
181  *   Trusted Application in the function TA_OpenSessionEntryPoint.
182  * - commandID: A Trusted Application-specific code that identifies the
183  *   command to be invoked.
184  * - paramTypes: the types of the four parameters.
185  * - params: a pointer to an array of four parameters.
186  *
187  * Return Value:
188  * - TEE_SUCCESS: if the command is successfully executed, the function
189  *   must return this value.
190  * - Any other value: if the invocation of the command fails for any
191  *   reason.
192  *   o The error code may be one of the pre-defined codes, or may be a new
193  *     error code defined by the Trusted Application implementation itself.
194  */
195
196 TEE_Result TA_EXPORT TA_InvokeCommandEntryPoint(void *sessionContext,
197             uint32_t commandID,
198             uint32_t paramTypes,
199             TEE_Param params[TEE_NUM_PARAMS]);
200
201 /*
202  * Correspondance Client Functions <--> TA Functions
203  *
204  * TEE_OpenSession or TEE_OpenTASession:
205  * If a new Trusted Application instance is needed to handle the session,
206  * TA_CreateEntryPoint is called.
207  * Then, TA_OpenSessionEntryPoint is called.
208  *
209  *
210  * TEE_InvokeCommand or TEE_InvokeTACommand:
211  * TA_InvokeCommandEntryPoint is called.
212  *
213  *
214  * TEE_CloseSession or TEE_CloseTASession:
215  * TA_CloseSessionEntryPoint is called.
216  * For a multi-instance TA or for a single-instance, non keep-alive TA, if
217  * the session closed was the last session on the instance, then
218  * TA_DestroyEntryPoint is called. Otherwise, the instance is kept until
219  * the TEE shuts down.
220  *
221  */
222
223 #endif
```

## 4.25 ta-ref/api/include/test_dev_key.h File Reference

### Variables

- static const unsigned char _sanctum_dev_secret_key [ ]
- static const size_t _sanctum_dev_secret_key_len = 64
- static const unsigned char _sanctum_dev_public_key [ ]
- static const size_t _sanctum_dev_public_key_len = 32

### 4.25.1 Variable Documentation

#### 4.25.1.1 _sanctum_dev_public_key `const unsigned char _sanctum_dev_public_key[ ] [static]`

**Initial value:**

```
= {
  0x0f, 0xaa, 0xd4, 0xff, 0x01, 0x17, 0x85, 0x83, 0xba, 0xa5, 0x88, 0x96,
  0x6f, 0x7c, 0x1f, 0xf3, 0x25, 0x64, 0xdd, 0x17, 0xd7, 0xdc, 0x2b, 0x46,
  0xcb, 0x50, 0xa8, 0x4a, 0x69, 0x27, 0x0b, 0x4c
}
```

#### 4.25.1.2 _sanctum_dev_public_key_len `const size_t _sanctum_dev_public_key_len = 32 [static]`

#### 4.25.1.3 _sanctum_dev_secret_key `const unsigned char _sanctum_dev_secret_key[] [static]`

**Initial value:**

```
= {
  0x40, 0xa0, 0x99, 0x47, 0x8c, 0xce, 0xfa, 0x3a, 0x06, 0x63, 0xab, 0xc9,
  0x5e, 0x7a, 0x1e, 0xc9, 0x54, 0xb4, 0xf5, 0xf6, 0x45, 0xba, 0xd8, 0x04,
  0xdb, 0x13, 0xe7, 0xd7, 0x82, 0x6c, 0x70, 0x73, 0x57, 0x6a, 0x9a, 0xb6,
  0x21, 0x60, 0xd9, 0xd1, 0xc6, 0xae, 0xdc, 0x29, 0x85, 0x2f, 0xb9, 0x60,
  0xee, 0x51, 0x32, 0x83, 0x5a, 0x16, 0x89, 0xec, 0x06, 0xa8, 0x72, 0x34,
  0x51, 0xaa, 0x0e, 0x4a
}
```

#### 4.25.1.4 _sanctum_dev_secret_key_len `const size_t _sanctum_dev_secret_key_len = 64 [static]`

## 4.26 test_dev_key.h

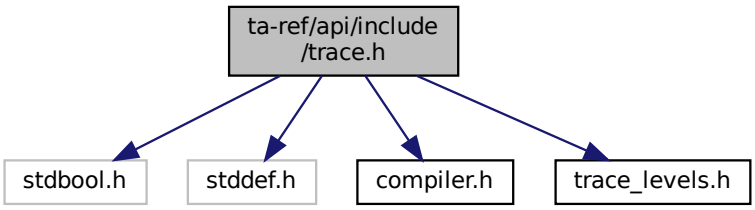
Go to the documentation of this file.

```
1 /* These are known device TESTING keys, use them for testing on platforms/qemu */
2
3 #warning Using TEST device root key. No integrity guarantee.
4 static const unsigned char _sanctum_dev_secret_key[] = {
5   0x40, 0xa0, 0x99, 0x47, 0x8c, 0xce, 0xfa, 0x3a, 0x06, 0x63, 0xab, 0xc9,
6   0x5e, 0x7a, 0x1e, 0xc9, 0x54, 0xb4, 0xf5, 0xf6, 0x45, 0xba, 0xd8, 0x04,
7   0xdb, 0x13, 0xe7, 0xd7, 0x82, 0x6c, 0x70, 0x73, 0x57, 0x6a, 0x9a, 0xb6,
8   0x21, 0x60, 0xd9, 0xd1, 0xc6, 0xae, 0xdc, 0x29, 0x85, 0x2f, 0xb9, 0x60,
9   0xee, 0x51, 0x32, 0x83, 0x5a, 0x16, 0x89, 0xec, 0x06, 0xa8, 0x72, 0x34,
10   0x51, 0xaa, 0x0e, 0x4a
11 };
12 static const size_t _sanctum_dev_secret_key_len = 64;
13
14 static const unsigned char _sanctum_dev_public_key[] = {
15   0x0f, 0xaa, 0xd4, 0xff, 0x01, 0x17, 0x85, 0x83, 0xba, 0xa5, 0x88, 0x96,
16   0x6f, 0x7c, 0x1f, 0xf3, 0x25, 0x64, 0xdd, 0x17, 0xd7, 0xdc, 0x2b, 0x46,
17   0xcb, 0x50, 0xa8, 0x4a, 0x69, 0x27, 0x0b, 0x4c
18 };
19 static const size_t _sanctum_dev_public_key_len = 32;
```

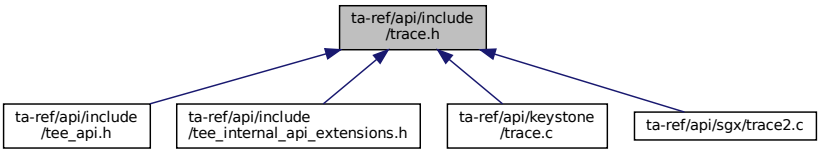
## 4.27 ta-ref/api/include/trace.h File Reference

```
#include <stdbool.h>
#include <stddef.h>
#include <compiler.h>
#include <trace_levels.h>
```

Include dependency graph for trace.h:




This graph shows which files directly or indirectly include this file:




### Functions

- void trace_ext_puts (const char ∗str)
- int trace_ext_get_thread_id (void)
- void trace_set_level (int level)
- int trace_get_level (void)
- void trace_printf (const char ∗func, int line, int level, bool level_ok, const char ∗fmt,...) __printf(5
- void void dhex_dump (const char ∗function, int line, int level, const void ∗buf, int len)

### Variables

- int trace_level
- const char trace_ext_prefix [ ]

#### 4.27.1 Function Documentation

#### 4.27.1.1 dhex_dump() void void dhex_dump (

```
const char * function,
int line,
int level,
const void * buf,
int len )
```

#### 4.27.1.2 trace_ext_get_thread_id() int trace_ext_get_thread_id (

```
void  )
```

#### 4.27.1.3 trace_ext_puts() void trace_ext_puts (

```
const char * str )
```

#### 4.27.1.4 trace_get_level() int trace_get_level (

```
void  )
```

#### 4.27.1.5 trace_printf() void trace_printf (

```
const char * func,
int line,
int level,
bool level_ok,
const char * fmt,
 ...  )
```

#### 4.27.1.6 trace_set_level() void trace_set_level (

```
int level )
```

### 4.27.2 Variable Documentation

#### 4.27.2.1 trace_ext_prefix const char trace_ext_prefix[] [extern]

#### 4.27.2.2 trace_level int trace_level [extern]

## 4.28 trace.h

Go to the documentation of this file.

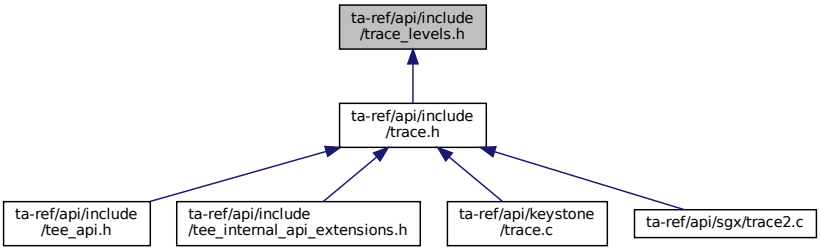
```
/*
 * Copyright (c) 2014, STMicroelectronics International N.V.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * 1. Redistributions of source code must retain the above copyright notice,
 * this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright notice,
 * this list of conditions and the following disclaimer in the documentation
 * and/or other materials provided with the distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */
#ifndef TRACE_H
#define TRACE_H

#include <stdbool.h>
#include <stddef.h>
#include <compiler.h>
#include <trace_levels.h>

#ifndef DOXYGEN_SHOULD_SKIP_THIS
#define MAX_PRINT_SIZE      256
#define MAX_FUNC_PRINT_SIZE 32

#ifndef TRACE_LEVEL
#define TRACE_LEVEL TRACE_MAX
#endif
#endif /*DOXYGEN_SHOULD_SKIP_THIS*/

/*
 * Symbols provided by the entity that uses this API.
 */
extern int trace_level;
extern const char trace_ext_prefix[];
void trace_ext_puts(const char *str);
int trace_ext_get_thread_id(void);
void trace_set_level(int level);
int trace_get_level(void);

/* Internal functions used by the macros below */
void trace_printf(const char *func, int line, int level, bool level_ok,
          const char *fmt, ...) __printf(5, 6);

#ifndef DOXYGEN_SHOULD_SKIP_THIS
#define trace_printf_helper(level, level_ok, ...) \
    trace_printf(__func__, __LINE__, (level), (level_ok), \
             __VA_ARGS__)

/* Formatted trace tagged with level independent */
#if (TRACE_LEVEL <= 0)
#define MSG(...)    (void)0
#else
#define MSG(...)    trace_printf_helper(0, false, __VA_ARGS__)
#endif

/* Formatted trace tagged with TRACE_ERROR level */
#if (TRACE_LEVEL < TRACE_ERROR)
#define EMSG(...)    (void)0
#else
#define EMSG(...)    trace_printf_helper(TRACE_ERROR, true, __VA_ARGS__)
#endif

/* Formatted trace tagged with TRACE_INFO level */
#if (TRACE_LEVEL < TRACE_INFO)
#define IMSG(...)    (void)0
#else
#define IMSG(...)    trace_printf_helper(TRACE_INFO, true, __VA_ARGS__)
#endif

/* Formatted trace tagged with TRACE_DEBUG level */
#if (TRACE_LEVEL < TRACE_DEBUG)
```

```
86 #define DMSG(...)    (void)0
87 #else
88 #define DMSG(...)    trace_printf_helper(TRACE_DEBUG, true, __VA_ARGS__)
89 #endif
90
91 /* Formatted trace tagged with TRACE_FLOW level */
92 #if (TRACE_LEVEL < TRACE_FLOW)
93 #define FMSG(...)    (void)0
94 #else
95 #define FMSG(...)    trace_printf_helper(TRACE_FLOW, true, __VA_ARGS__)
96 #endif
97
98 /* Formatted trace tagged with TRACE_FLOW level and prefix with '> ' */
99 #define INMSG(...)     FMSG("> " __VA_ARGS__)
100 /* Formatted trace tagged with TRACE_FLOW level and prefix with '< ' */
101 #define OUTMSG(...)    FMSG("< " __VA_ARGS__)
102 /* Formatted trace tagged with TRACE_FLOW level and prefix with '< ' and print
103  * an error message if r != 0 */
104 #define OUTRMSG(r)          \
105     do {                \
106         OUTMSG("r=[%x]", r);    \
107         return r;       \
108     } while (0)
109
110 #endif /*DOXYGEN_SHOULD_SKIP_THIS*/
111
112 void dhex_dump(const char *function, int line, int level,
113            const void *buf, int len);
114
115
116 #ifndef DOXYGEN_SHOULD_SKIP_THIS
117 #if (TRACE_LEVEL < TRACE_DEBUG)
118 #define DHEXDUMP(buf, len) (void)0
119 #else
120 #define DHEXDUMP(buf, len) dhex_dump(__func__, __LINE__, TRACE_DEBUG, \
121                      buf, len)
122 #endif
123
124
125 /* Trace api without trace formatting */
126
127 #define trace_printf_helper_raw(level, level_ok, ...) \
128     trace_printf(NULL, 0, (level), (level_ok), __VA_ARGS__)
129
130 /* No formatted trace tagged with level independent */
131 #if (TRACE_LEVEL <= 0)
132 #define MSG_RAW(...)    (void)0
133 #else
134 #define MSG_RAW(...)    trace_printf_helper_raw(0, false, __VA_ARGS__)
135 #endif
136
137 /* No formatted trace tagged with TRACE_ERROR level */
138 #if (TRACE_LEVEL < TRACE_ERROR)
139 #define EMSG_RAW(...)    (void)0
140 #else
141 #define EMSG_RAW(...)    trace_printf_helper_raw(TRACE_ERROR, true, __VA_ARGS__)
142 #endif
143
144 /* No formatted trace tagged with TRACE_INFO level */
145 #if (TRACE_LEVEL < TRACE_INFO)
146 #define IMSG_RAW(...)    (void)0
147 #else
148 #define IMSG_RAW(...)    trace_printf_helper_raw(TRACE_INFO, true, __VA_ARGS__)
149 #endif
150
151 /* No formatted trace tagged with TRACE_DEBUG level */
152 #if (TRACE_LEVEL < TRACE_DEBUG)
153 #define DMSG_RAW(...)    (void)0
154 #else
155 #define DMSG_RAW(...)    trace_printf_helper_raw(TRACE_DEBUG, true, __VA_ARGS__)
156 #endif
157
158 /* No formatted trace tagged with TRACE_FLOW level */
159 #if (TRACE_LEVEL < TRACE_FLOW)
160 #define FMSG_RAW(...)    (void)0
161 #else
162 #define FMSG_RAW(...)    trace_printf_helper_raw(TRACE_FLOW, true, __VA_ARGS__)
163 #endif
164
165 #if (TRACE_LEVEL <= 0)
166 #define SMSG(...)    (void)0
167 #else
168 /*
169  * Synchronised flushed trace, an Always message straight to HW trace IP.
170  * Current only supported inside OP-TEE kernel, will be just like an EMSG()
```

```
171  * in another context.
172  */
173 #define SMSG(...)   \
174     trace_printf(__func__, __LINE__, TRACE_ERROR, true, __VA_ARGS__)
175
176 #endif /* TRACE_LEVEL */
177
178 #if defined(__KERNEL__) && defined(CFG_UNWIND)
179 #include <kernel/unwind.h>
180 #define _PRINT_STACK
181 #endif
182
183 #if defined(_PRINT_STACK) && (TRACE_LEVEL >= TRACE_ERROR)
184 #define EPRINT_STACK() print_kernel_stack(TRACE_ERROR)
185 #else
186 #define EPRINT_STACK() (void)0
187 #endif
188
189 #if defined(_PRINT_STACK) && (TRACE_LEVEL >= TRACE_INFO)
190 #define IPRINT_STACK() print_kernel_stack(TRACE_INFO)
191 #else
192 #define IPRINT_STACK() (void)0
193 #endif
194
195 #if defined(_PRINT_STACK) && (TRACE_LEVEL >= TRACE_DEBUG)
196 #define DPRINT_STACK() print_kernel_stack(TRACE_DEBUG)
197 #else
198 #define DPRINT_STACK() (void)0
199 #endif
200
201 #if defined(_PRINT_STACK) && (TRACE_LEVEL >= TRACE_FLOW)
202 #define FPRINT_STACK() print_kernel_stack(TRACE_FLOW)
203 #else
204 #define FPRINT_STACK() (void)0
205 #endif
206
207 #if defined(__KERNEL__) && defined(CFG_UNWIND)
208 #undef _PRINT_STACK
209 #endif
210
211 #endif /*DOXYGEN_SHOULD_SKIP_THIS*/
212 #endif /* TRACE_H */
```

## 4.29 ta-ref/api/include/trace_levels.h File Reference

This graph shows which files directly or indirectly include this file:




## 4.30 trace_levels.h

Go to the documentation of this file.

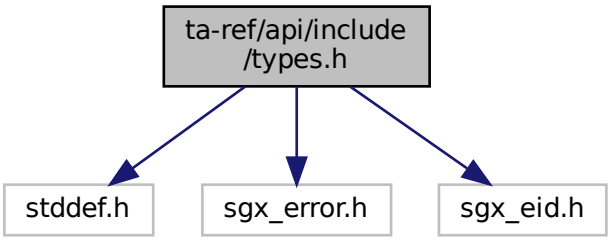
```
1 /*
2  * Copyright (c) 2014, STMicroelectronics International N.V.
3  * All rights reserved.
```

```
4  *
5  * Redistribution and use in source and binary forms, with or without
6  * modification, are permitted provided that the following conditions are met:
7  *
8  * 1. Redistributions of source code must retain the above copyright notice,
9  * this list of conditions and the following disclaimer.
10  *
11  * 2. Redistributions in binary form must reproduce the above copyright notice,
12  * this list of conditions and the following disclaimer in the documentation
13  * and/or other materials provided with the distribution.
14  *
15  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
16  * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
17  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
18  * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
19  * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
20  * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
21  * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
22  * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
23  * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
24  * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
25  * POSSIBILITY OF SUCH DAMAGE.
26  */
27 #ifndef TRACE_LEVELS_H
28 #define TRACE_LEVELS_H
29
30 /*
31  * Trace levels.
32  *
33  * ALWAYS is used when you always want a print to be seen, but it is not always
34  * an error.
35  *
36  * ERROR is used when some kind of error has happened, this is most likely the
37  * print you will use most of the time when you report some kind of error.
38  *
39  * INFO is used when you want to print some 'normal' text to the user.
40  * This is the default level.
41  *
42  * DEBUG is used to print extra information to enter deeply in the module.
43  *
44  * FLOW is used to print the execution flox, typically the in/out of functions.
45  *
46  */
47
48 #ifndef DOXYGEN_SHOULD_SKIP_THIS
49 #define TRACE_MIN       1
50 #define TRACE_ERROR     TRACE_MIN
51 #define TRACE_INFO      2
52 #define TRACE_DEBUG     3
53 #define TRACE_FLOW      4
54 #define TRACE_MAX       TRACE_FLOW
55
56 /* Trace level of the casual printf */
57 #define TRACE_PRINTF_LEVEL TRACE_ERROR
58
59 #endif /*DOXYGEN_SHOULD_SKIP_THIS*/
60 #endif /*TRACE_LEVELS_H*/
61
```

## 4.31 ta-ref/api/include/types.h File Reference

```
#include <stddef.h>
#include "sgx_error.h"
#include "sgx_eid.h"
```

Include dependency graph for types.h:




### Classes

- struct _sgx_errlist_t

### Typedefs

- typedef struct _sgx_errlist_t sgx_errlist_t

### Variables

- sgx_enclave_id_t global_eid = 0
- static sgx_errlist_t sgx_errlist [ ]

#### 4.31.1 Typedef Documentation

##### 4.31.1.1 sgx_errlist_t `typedef struct _sgx_errlist_t sgx_errlist_t`

#### 4.31.2 Variable Documentation

##### 4.31.2.1 global_eid `sgx_enclave_id_t global_eid = 0`

##### 4.31.2.2 sgx_errlist `sgx_errlist_t sgx_errlist[] [static]`

## 4.32 types.h

Go to the documentation of this file.

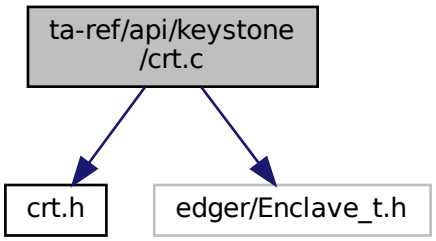
```
#pragma once
#include <stddef.h>
#include "sgx_error.h"       /* sgx_status_t */
#include "sgx_eid.h"     /* sgx_enclave_id_t */

/* Global EID shared by multiple threads */
sgx_enclave_id_t global_eid = 0;

typedef struct _sgx_errlist_t {
    sgx_status_t err;
    const char *msg;
    const char *sug; /* Suggestion */
} sgx_errlist_t;

/* Error code returned by sgx_create_enclave */
static sgx_errlist_t sgx_errlist[] = {
    {
        SGX_ERROR_UNEXPECTED,
        "Unexpected error occurred.",
        NULL
    },
    {
        SGX_ERROR_INVALID_PARAMETER,
        "Invalid parameter.",
        NULL
    },
    {
        SGX_ERROR_OUT_OF_MEMORY,
        "Out of memory.",
        NULL
    },
    {
        SGX_ERROR_ENCLAVE_LOST,
        "Power transition occurred.",
        "Please refer to the sample \"PowerTransition\" for details."
    },
    {
        SGX_ERROR_INVALID_ENCLAVE,
        "Invalid enclave image.",
        NULL
    },
    {
        SGX_ERROR_INVALID_ENCLAVE_ID,
        "Invalid enclave identification.",
        NULL
    },
    {
        SGX_ERROR_INVALID_SIGNATURE,
        "Invalid enclave signature.",
        NULL
    },
    {
        SGX_ERROR_OUT_OF_EPC,
        "Out of EPC memory.",
        NULL
    },
    {
        SGX_ERROR_NO_DEVICE,
        "Invalid SGX device.",
        "Please make sure SGX module is enabled in the BIOS, and install SGX driver afterwards."
    },
    {
        SGX_ERROR_MEMORY_MAP_CONFLICT,
        "Memory map conflicted.",
        NULL
    },
    {
        SGX_ERROR_INVALID_METADATA,
        "Invalid enclave metadata.",
        NULL
    },
    {
        SGX_ERROR_DEVICE_BUSY,
        "SGX device was busy.",
        NULL
    },
    {
        SGX_ERROR_INVALID_VERSION,
        "Enclave version was invalid.",
        NULL
    },
    {
        SGX_ERROR_INVALID_ATTRIBUTE,
        "Enclave was not authorized.",
        NULL
```

```
86     },
87     {
88         SGX_ERROR_ENCLAVE_FILE_ACCESS,
89         "Can't open enclave file.",
90         NULL
91     },
92 };
```

## 4.33 ta-ref/api/keystone/crt.c File Reference

```
#include "crt.h"
#include "edger/Enclave_t.h"
```

Include dependency graph for crt.c:




### Functions

- void crt_end (void)

### Variables

- static void(∗const init_array [ ])() __attribute__((section(".init_array")
- static void(∗const aligned [ ])(sizeof(void ∗)))
- static void(∗const fini_array [ ])() __attribute__((section(".fini_array")
- void(∗ __init_array_start [ ])(void)

### 4.33.1 Function Documentation

#### 4.33.1.1 crt_end() `void crt_end (`

```
    void  )
```

crt_end() - Ends the certification.

It compares __fini_array_start and __fini_array_end; and then it the loops through the file pointer.

### 4.33.2 Variable Documentation

#### 4.33.2.1 __init_array_start `void(* __init_array_start[])(void) ( void ) [extern]`

crt_begin() - Commences the certification.

It compares __init_array_start and __init_array_end; and then it the loops through the file pointer.

#### 4.33.2.2 aligned `void(*const aligned[])(sizeof(void *))) ( sizeof(void *) )`

**Initial value:**

```
= {
  }
```

#### 4.33.2.3 fini_array `void(*const fini_array[])() __attribute__((section(".fini_array") ( ) [static]`

Termination array for the executable.

This section holds an array of function pointers that contributes to a single termination array for the executable or shared object containing the section and if defined is PERF_ENABLE then unmapping the profiler information.

**Parameters**

| | |
|---|---|
| *fini_array[ ]* | constant array. |

#### 4.33.2.4 init_array `void(*const init_array[])() __attribute__((section(".init_array") ( ) [static]`

Initialization array for the executable.

This section holds an array of function pointers that contributes to a single initialization array for the executable or shared object containing the section if defined is PERF_ENABLE then mapping the profiler information.

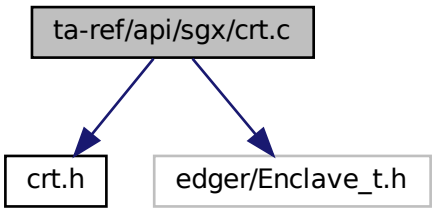
**Parameters**

| | |
|---|---|
| *init_array[ ]* | constant array. |

## 4.34 ta-ref/api/sgx/crt.c File Reference

```
#include "crt.h"
#include "edger/Enclave_t.h"
```

Include dependency graph for crt.c:




### Functions

- void crt_end (void)

### Variables

- static void(∗const init_array [])() __attribute__((section(".init_array")
- static void(∗const aligned [])(sizeof(void ∗))))
- static void(∗const fini_array [])() __attribute__((section(".fini_array")
- void(∗ __init_array_start [])(void)

### 4.34.1 Function Documentation

#### 4.34.1.1 crt_end() `void crt_end ( void )`

crt_end() - Ends the certification.

It compares __fini_array_start and __fini_array_end; and then it the loops through the file pointer.

### 4.34.2 Variable Documentation

#### 4.34.2.1 __init_array_start `void(* __init_array_start[])(void) ( void ) [extern]`

crt_begin() - Commences the certification.

It compares __init_array_start and __init_array_end; and then it the loops through the file pointer.

#### 4.34.2.2 aligned `void(*const aligned[])(sizeof(void *))) ( sizeof(void *) )`

**Initial value:**

```
= {
  }
```

#### 4.34.2.3 fini_array `void(*const fini_array[])() __attribute__((section(".fini_array") ( ) [static]`

Termination array for the executable.

This section holds an array of function pointers that contributes to a single termination array for the executable or shared object containing the section and if defined is PERF_ENABLE then unmapping the profiler information.

**Parameters**

| | |
|---|---|
| *fini_array[ ]* | constant array. |

#### 4.34.2.4 init_array `void(*const init_array[])() __attribute__((section(".init_array") ( ) [static]`

Initialization array for the executable.

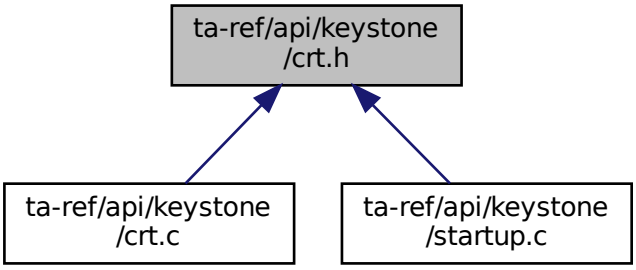
This section holds an array of function pointers that contributes to a single initialization array for the executable or shared object containing the section if defined is PERF_ENABLE then mapping the profiler information.

**Parameters**

| | |
|---|---|
| *init_array[ ]* | constant array. |

## 4.35 ta-ref/api/keystone/crt.h File Reference

This graph shows which files directly or indirectly include this file:




### Functions

- void crt_begin (void)
- void crt_end (void)
- int main (void)

### 4.35.1 Function Documentation

#### 4.35.1.1 crt_begin() `void crt_begin (`

```
void )
```

#### 4.35.1.2 crt_end() `void crt_end (`

```
void )
```

crt_end() - Ends the certification.

It compares __fini_array_start and __fini_array_end; and then it the loops through the file pointer.

#### 4.35.1.3 main() `int main (`

```
void )
```

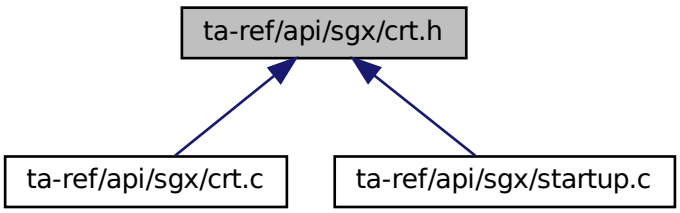
## 4.36 crt.h

Go to the documentation of this file.

```
1 void crt_begin(void);
2 void crt_end(void);
3 int main(void);
```

## 4.37 ta-ref/api/sgx/crt.h File Reference

This graph shows which files directly or indirectly include this file:




### Functions

- void crt_begin (void)
- void crt_end (void)
- int main (void)

### 4.37.1 Function Documentation

#### 4.37.1.1 crt_begin() `void crt_begin (`

`void )`

#### 4.37.1.2 crt_end() `void crt_end (`

`void )`

crt_end() - Ends the certification.

It compares __fini_array_start and __fini_array_end; and then it the loops through the file pointer.

#### 4.37.1.3 main() `int main (`

`void )`

## 4.38 crt.h

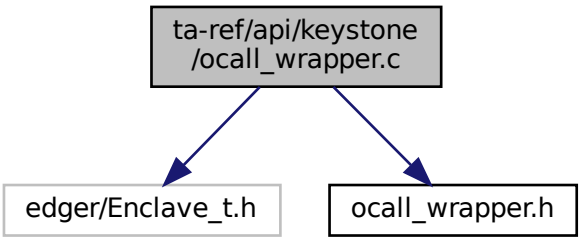
Go to the documentation of this file.

```
1 void crt_begin(void);
2 void crt_end(void);
3 int main(void);
```

## 4.39 ta-ref/api/keystone/ocall_wrapper.c File Reference

```
#include "edger/Enclave_t.h"
#include "ocall_wrapper.h"
```

Include dependency graph for ocall_wrapper.c:




### Functions

- unsigned int ocall_print_string_wrapper (const char *str)

### 4.39.1 Function Documentation

#### 4.39.1.1 ocall_print_string_wrapper() `unsigned int ocall_print_string_wrapper ( const char * str )`

ocall_print_string_wrapper() - To print the argument string

This function invokes ocall_print_string() to print the string.

**Parameters**

| | |
|---|---|
| *str* | The string value for print. |

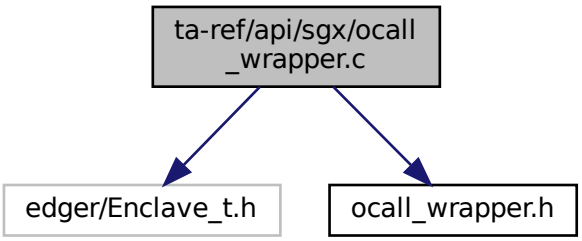
**Returns**

string It prints the value of str by calling ocall_print_string().

## 4.40 ta-ref/api/sgx/ocall_wrapper.c File Reference

```
#include "edger/Enclave_t.h"
#include "ocall_wrapper.h"
```

Include dependency graph for ocall_wrapper.c:




### Functions

- unsigned int ocall_print_string_wrapper (const char *str)

### 4.40.1 Function Documentation

#### 4.40.1.1 ocall_print_string_wrapper() `unsigned int ocall_print_string_wrapper ( const char * str )`

ocall_print_string_wrapper() - To print the argument string

This function invokes ocall_print_string() to print the string.

**Parameters**

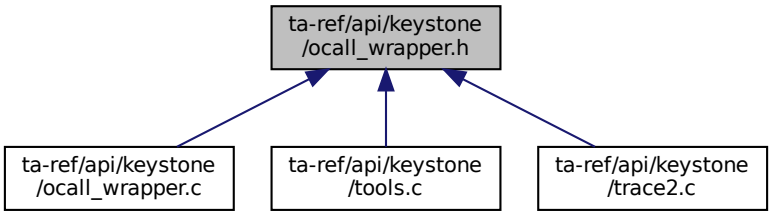
| | |
|---|---|
| *str* | The string value for print. |

**Returns**

retval Its prints the value of str by calling ocall_print_string().

## 4.41 ta-ref/api/keystone/ocall_wrapper.h File Reference

This graph shows which files directly or indirectly include this file:




### Functions

- unsigned int ocall_print_string_wrapper (const char ∗str)

### 4.41.1 Function Documentation

#### 4.41.1.1 ocall_print_string_wrapper() `unsigned int ocall_print_string_wrapper ( const char * str )`

ocall_print_string_wrapper() - To print the argument string

This function invokes ocall_print_string() to print the string.

**Parameters**

| | |
|---|---|
| *str* | The string value for print. |

**Returns**

string It prints the value of str by calling ocall_print_string().

ocall_print_string_wrapper() - To print the argument string

This function invokes ocall_print_string() to print the string.

**Parameters**

| | |
|---|---|
| *str* | The string value for print. |

**Returns**

retval Its prints the value of str by calling ocall_print_string().

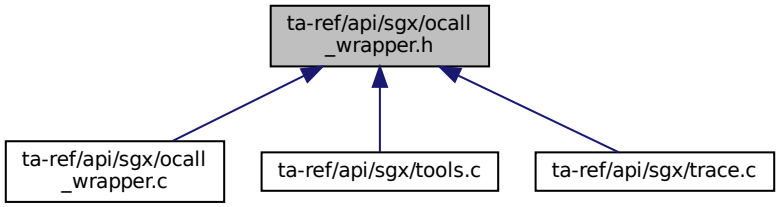
## 4.42 ocall_wrapper.h

Go to the documentation of this file.

```
1 #pragma once
2 unsigned int ocall_print_string_wrapper(const char* str);
```

## 4.43 ta-ref/api/sgx/ocall_wrapper.h File Reference

This graph shows which files directly or indirectly include this file:




### Functions

- unsigned int ocall_print_string_wrapper (const char ∗str)

### 4.43.1 Function Documentation

#### 4.43.1.1 ocall_print_string_wrapper() `unsigned int ocall_print_string_wrapper ( const char * str )`

ocall_print_string_wrapper() - To print the argument string

This function invokes ocall_print_string() to print the string.

**Parameters**

| | |
|---|---|
| *str* | The string value for print. |

**Returns**

string It prints the value of str by calling ocall_print_string().

ocall_print_string_wrapper() - To print the argument string

This function invokes ocall_print_string() to print the string.

**Parameters**

| | |
|---|---|
| *str* | The string value for print. |

**Returns**

retval Its prints the value of str by calling ocall_print_string().

## 4.44 ocall_wrapper.h

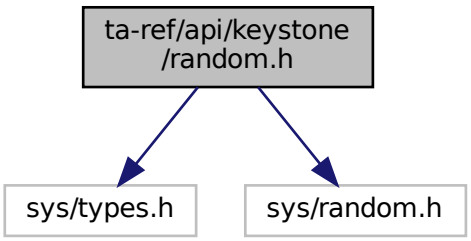
Go to the documentation of this file.

```
1 #pragma once
2 unsigned int ocall_print_string_wrapper(const char* str);
```

## 4.45 ta-ref/api/keystone/random.h File Reference

```
#include <sys/types.h>
#include <sys/random.h>
```

Include dependency graph for random.h:




## 4.46 random.h

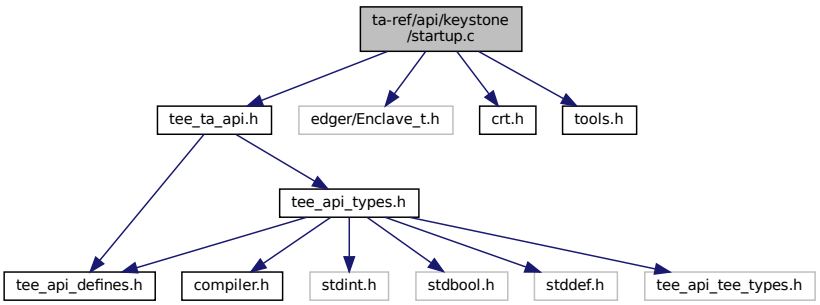
Go to the documentation of this file.

```
1 #include <sys/types.h>
2 // for keystone-enclave v0.4 we saw the getrandom(2) function freeze, so we use srandom/random
      instead when we set 'SEED' value.
3 #ifdef SEED
4 #include <stdlib.h>
5 #define getrandom seed_random
6 static ssize_t seed_random(void *buf, size_t buflen, unsigned int flags) {
7     (flags); // not used
8     const ssize_t ss = sizeof(unsigned int);
9     unsigned int retval;
10     unsigned int *b = (unsigned int*)buf;
11     ssize_t idx = 0;
12     srandom((unsigned int)SEED);
13     while(buflen) {
14         retval = random();
15         buflen -= ss;
16         b[idx++] = retval;
17     }
18     return idx*ss;
19 }
20 #else
21 #include <sys/random.h>
22 #endif
```

## 4.47 ta-ref/api/keystone/startup.c File Reference

```
#include "tee_ta_api.h"
#include "edger/Enclave_t.h"
#include "crt.h"
#include "tools.h"
```

Include dependency graph for startup.c:




### Functions

- TEE_Result TA_InvokeCommandEntryPoint (void *sess_ctx, uint32_t cmd_id, uint32_t param_types, TEE_Param params[4])
- void EAPP_ENTRY eapp_entry ()

### 4.47.1 Function Documentation

#### 4.47.1.1 eapp_entry() `void EAPP_ENTRY eapp_entry ( )`

The eapp_entry() - It contains enclave verbose and invokes main function.

This function invokes crt_begin() if defined macro is ENCLAVE_VERBOSE then prints the main start and invokes main(). Once main() is completed prints the main end and invokes the crt_end().

**Returns**

It will return EAPP_RETURN(0).

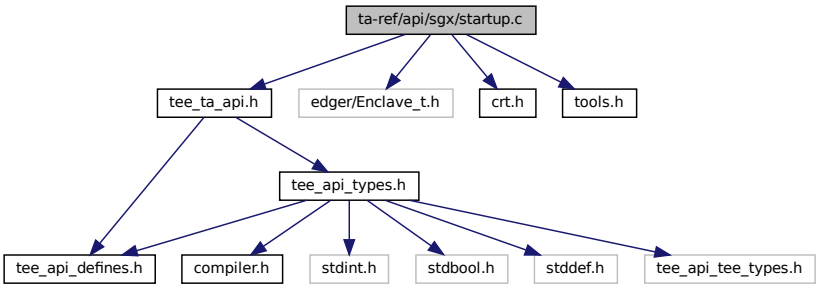
#### 4.47.1.2 TA_InvokeCommandEntryPoint()

```
TEE_Result TA_InvokeCommandEntryPoint (
        void * sess_ctx,
        uint32_t cmd_id,
        uint32_t param_types,
        TEE_Param params[4] )
```

## 4.48 ta-ref/api/sgx/startup.c File Reference

```
#include "tee_ta_api.h"
#include "edger/Enclave_t.h"
#include "crt.h"
#include "tools.h"
```

Include dependency graph for startup.c:




### Functions

- TEE_Result TA_InvokeCommandEntryPoint (void *sess_ctx, uint32_t cmd_id, uint32_t param_types, TEE_Param params[4])
- void ecall_ta_main (uint32_t command)

### 4.48.1 Function Documentation

#### 4.48.1.1 ecall_ta_main()

```
void ecall_ta_main (
        uint32_t command )
```

The eapp_entry() - It contains enclave verbose and invokes the main function.

This function invokes crt_begin() if defined macro is ENCLAVE_VERBOSE then prints the main start and invokes main(). Once main() is completed, it prints the main end and invokes the crt_end().

**Returns**

It will return EAPP_RETURN(0).

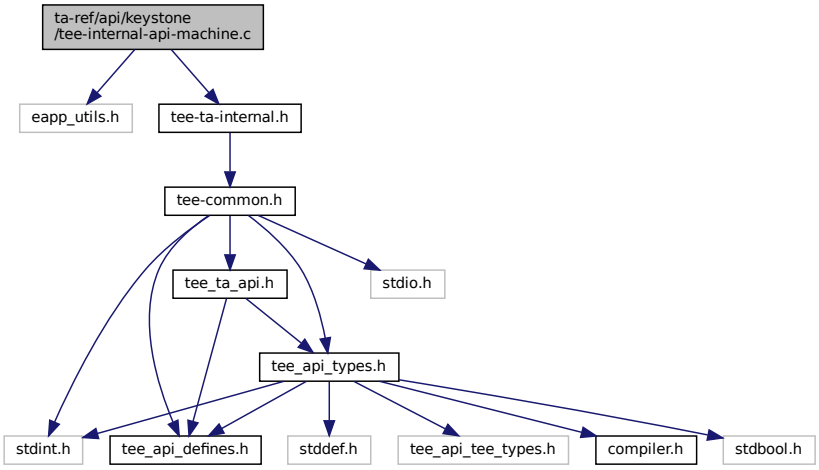
#### 4.48.1.2 TA_InvokeCommandEntryPoint()

```
TEE_Result TA_InvokeCommandEntryPoint (
        void * sess_ctx,
        uint32_t cmd_id,
        uint32_t param_types,
        TEE_Param params[4] )
```

## 4.49 ta-ref/api/keystone/tee-internal-api-machine.c File Reference

```
#include "eapp_utils.h"
#include "tee-ta-internal.h"
```

Include dependency graph for tee-internal-api-machine.c:




### Functions

- void __attribute__ ((noreturn))

### 4.49.1 Function Documentation

#### 4.49.1.1 __attribute__() void __attribute__ ( (noreturn) )

TEE_Panic() - Raises a panic in the Trusted Application instance.

When a Trusted Application calls the TEE_Panic function, the current instance shall be destroyed and all the resources opened by the instance shall be reclaimed. All sessions opened from the panicking instance on another TA shall be gracefully closed and all cryptographic objects and operations shall be closed properly.

**Parameters**

| | |
|---|---|
| *code* | An informative panic code defined by the TA. |

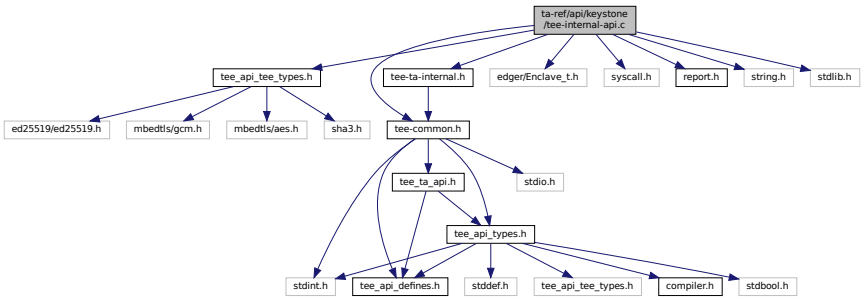
**Returns**

panic code will be returned.

## 4.50 ta-ref/api/keystone/tee-internal-api.c File Reference

```
#include "tee_api_tee_types.h"
#include "tee-common.h"
#include "tee-ta-internal.h"
#include "edger/Enclave_t.h"
#include "syscall.h"
#include "report.h"
#include <string.h>
#include <stdlib.h>
```

Include dependency graph for tee-internal-api.c:




### Functions

- void * TEE_Malloc (uint32_t size, uint32_t hint)
- void * TEE_Realloc (void *buffer, uint32_t newSize)
- void TEE_Free (void *buffer)
- void TEE_GetREETime (TEE_Time *time)

  *Core Functions, Time Functions.*
- void TEE_GetSystemTime (TEE_Time *time)

  *Core Functions, Time Functions.*

- TEE_Result GetRelTimeStart (uint64_t start)

  *Core Functions, Time Functions.*
- TEE_Result GetRelTimeEnd (uint64_t end)

  *Core Functions, Time Functions.*
- static int flags2flags (int flags)
- static int set_object_key (void *id, unsigned int idlen, TEE_ObjectHandle object)
- static TEE_Result OpenPersistentObject (uint32_t storageID, const void *objectID, uint32_t objectIDLen, uint32_t flags, TEE_ObjectHandle *object, int ocreat)
- TEE_Result TEE_CreatePersistentObject (uint32_t storageID, const void *objectID, uint32_t objectIDLen, uint32_t flags, TEE_ObjectHandle attributes, const void *initialData, uint32_t initialDataLen, TEE_ObjectHandle *object)

  *Core Functions, Secure Storage Functions (data is isolated for each TA)*
- TEE_Result TEE_OpenPersistentObject (uint32_t storageID, const void *objectID, uint32_t objectIDLen, uint32_t flags, TEE_ObjectHandle *object)

  *Core Functions, Secure Storage Functions (data is isolated for each TA)*
- TEE_Result TEE_GetObjectInfo1 (TEE_ObjectHandle object, TEE_ObjectInfo *objectInfo)

  *Core Functions, Secure Storage Functions (data is isolated for each TA)*
- TEE_Result TEE_WriteObjectData (TEE_ObjectHandle object, const void *buffer, uint32_t size)

  *Core Functions, Secure Storage Functions (data is isolated for each TA)*
- TEE_Result TEE_ReadObjectData (TEE_ObjectHandle object, void *buffer, uint32_t size, uint32_t *count)

  *Core Functions, Secure Storage Functions (data is isolated for each TA)*
- void TEE_CloseObject (TEE_ObjectHandle object)

  *Core Functions, Secure Storage Functions (data is isolated for each TA)*
- WC_RNG * get_wc_rng (void)
- int wc_ocall_genseed (void *nonce, uint32_t len)
- void TEE_GenerateRandom (void *randomBuffer, uint32_t randomBufferLen)

  *Crypto, common.*

### Variables

- static int wc_rng_init = 0
- static WC_RNG rngstr

## 4.50.1 Function Documentation

#### 4.50.1.1 flags2flags() `static int flags2flags ( int flags ) [inline], [static]`

flags2flags() - Checks the status for reading or writing of the file operational.

This function is used to check the status for reading or writing of the file operational.

**Parameters**

| | |
|---|---|
| *flags* | Flags of the referencing node. |

**Returns**

ret if success.

#### 4.50.1.2 get_wc_rng() `WC_RNG * get_wc_rng ( void )`

get_wc_rng() - Gets the seed (from OS) and key cipher for rng (random number genertor).

This function returns the random number or unique number of "rngstr".

**Returns**

random number if success else error occured.

#### 4.50.1.3 GetRelTimeEnd() `TEE_Result GetRelTimeEnd ( uint64_t end )`

Core Functions, Time Functions.

GetRelTimeEnd() - finds the real time of the end timing.

This function prints the ending time.

**Parameters**

| | |
|---|---|
| *end* | End timing |

**Returns**

0 If success

#### 4.50.1.4 GetRelTimeStart() `TEE_Result GetRelTimeStart ( uint64_t start )`

Core Functions, Time Functions.

GetRelTimeStart() - Gets the real time of the start timing.

This function prints the starting time.

**Parameters**

| | |
|---|---|
| *start* | Start timing |

**Returns**

0 on success

#### 4.50.1.5 OpenPersistentObject()

```
static TEE_Result OpenPersistentObject (
    uint32_t storageID,
    const void * objectID,
    uint32_t objectIDLen,
    uint32_t flags,
    TEE_ObjectHandle * object,
    int ocreat )  [static]
```

OpenPersistentObject() - Opens a handle on an existing persistent object.

The flags parameter is a set of flags that controls the access rights and sharing permissions with which the object handle is opened. The value of the flags parameter is constructed by a bitwise-inclusive OR of flags TEE_DATA↩ _FLAG_ACCESS_READ, the object is opened with the read access right. This allows the Trusted Application to call the function TEE_ReadObjectData. TEE_DATA_FLAG_ACCESS_WRITE, the object is opened with the write access right. TEE_DATA_FLAG_ACCESS_WRITE_META, the object is opened with the write-meta access right.

**Parameters**

| | |
|---|---|
| *storageID* | The storage to use. |
| *objectID* | The object identifier |
| *objectIDLen* | length of the identifier |
| *flags* | The flags which determine the settings under which the object is opened. |
| *object* | A pointer to the handle, which contains the opened handle upon successful completion. |

**Returns**

0 if success else error occured.

#### 4.50.1.6 set_object_key()

```
static int set_object_key (
    void * id,
    unsigned int idlen,
    TEE_ObjectHandle object )  [static]
```

set_object_key() - Initialize report and then attest enclave with file.

This function describes the intialization of report, attest the enclave with file id and its length then assigned to ret. Based on "mbedtls" key encryption and decryption position of the object will be copied. Finally ret value returns on success else signature too short error will appear on failure.

**Parameters**

| | |
|---|---|
| *id* | id of the object. |
| *idlen* | length of the id. |
| *object* | TEE_ObjectHandle type handle. |

**Returns**

ret if success.

#### 4.50.1.7 TEE_CloseObject()

```
void TEE_CloseObject (
        TEE_ObjectHandle object )
```

Core Functions, Secure Storage Functions (data is isolated for each TA)

TEE_CloseObject() - Closes an opened object handle.

The object can be persistent or transient.For transient objects, TEE_CloseObject is equivalent to TEE_Free↩ TransientObject.

**Parameters**

| | |
|---|---|
| *object* | Handle of the object. |

**Returns**

TEE_SUCCESS if success else error occured.

#### 4.50.1.8 TEE_CreatePersistentObject()

```
TEE_Result TEE_CreatePersistentObject (
        uint32_t storageID,
        const void * objectID,
        uint32_t objectIDLen,
        uint32_t flags,
        TEE_ObjectHandle attributes,
        const void * initialData,
        uint32_t initialDataLen,
        TEE_ObjectHandle * object )
```

Core Functions, Secure Storage Functions (data is isolated for each TA)

TEE_CreatePersistentObject() - Creates a persistent object with initial attributes.

In this function an initial data stream content returns either a handle on the created object or TEE_HANDLE_NULL upon failure.

**Parameters**

| | |
|---|---|
| *storageID* | The storage to use. |
| *objectID* | The object identifier |
| *objectIDLen* | The object identifier |
| *flags* | The flags which determine the settings under which the object is opened. |
| *attributes* | A handle on a persistent object or an initialized transient object from which to take the persistent object attributes |
| *initialData* | The initial data content of the persistent object |
| *initialDataLen* | The initial data content of the persistent object |
| *object* | A pointer to the handle which contains the opened handle upon successful completion |

**Returns**

0 if success else error occured.

#### 4.50.1.9 TEE_Free() `void TEE_Free (`

```
void * buffer )
```

TEE_Free() - causes the space pointed to by buffer to be deallocated;that is made available for further allocation.

This function describes if buffer is a NULL pointer, TEE_Free does nothing. Otherwise, it is a Programmer Error if the argument does not match a pointer previously returned by the TEE_Malloc or TEE_Realloc if the space has been deallocated by a call to TEE_Free or TEE_Realloc.

**Parameters**

| | |
|---|---|
| *buffer* | The pointer to the memory block to be freed. |

#### 4.50.1.10 TEE_GenerateRandom() `void TEE_GenerateRandom (`

```
void * randomBuffer,
uint32_t randomBufferLen )
```

Crypto, common.

TEE_GenerateRandom() - Generates random data.

This function generates random data of random buffer length and is stored in to random Buffer by calling wc_↩ RNG_GenerateBlock().If ret is not equal
to 0 then TEE_Panic is called.

**Parameters**

| | |
|---|---|
| *randomBuffer* | Reference to generated random data |
| *randomBufferLen* | Byte length of requested random data |

**Returns**

random data random data will be returned.

#### 4.50.1.11 TEE_GetObjectInfo1()

```
TEE_Result TEE_GetObjectInfo1 (
    TEE_ObjectHandle object,
    TEE_ObjectInfo * objectInfo )
```

Core Functions, Secure Storage Functions (data is isolated for each TA)

TEE_GetObjectInfo1() - Returns the characteristics of an object.

This function returns a handle which can be used to access the object's attributes and data stream.

**Parameters**

| | |
|---|---|
| *objectInfo* | Pointer to a structure filled with the object information |
| *object* | Handle of the object |

**Returns**

0 if success else error occured.

#### 4.50.1.12 TEE_GetREETime()

```
void TEE_GetREETime (
    TEE_Time * time )
```

Core Functions, Time Functions.

TEE_GetREETime() - Retrieves the current REE system time.

This function retrieves the current time as seen from the point of view of the REE.

**Parameters**

| | |
|---|---|
| *time* | Filled with the number of seconds and milliseconds |

#### 4.50.1.13 TEE_GetSystemTime() `void TEE_GetSystemTime (`

```
TEE_Time * time )
```

Core Functions, Time Functions.

TEE_GetSystemTime() - Retrieves the current system time.

This function describes the system time has an arbitrary implementation
defined origin that can vary across TA instances. The minimum guarantee
is that the system time shall be monotonic for a given TA instance.

**Parameters**

| | |
|---|---|
| *time* | Filled with the number of seconds and milliseconds |

#### 4.50.1.14 TEE_Malloc() `void * TEE_Malloc (`

```
uint32_t size,
uint32_t hint )
```

TEE_Malloc() - Allocates space for an object whose size in bytes is specified in the parameter size.

This function describes the pointer returned is guaranteed to be aligned
such that it may be assigned as a pointer to any basic C type.The valid hint values are a bitmask and can be independently set. This parameter allows Trusted Applications to refer to various pools of memory or to
request special characteristics for the allocated memory by using an
implementation-defined hint. Future versions of this specification may introduce additional standard hints.

**Parameters**

| | |
|---|---|
| *size* | The size of the buffer to be allocated. |
| *hint* | A hint to the allocator. |

**Returns**

Upon successful completion, with size not equal to zero, the function returns a pointer to the allocated space.

#### 4.50.1.15 TEE_OpenPersistentObject() `TEE_Result TEE_OpenPersistentObject (`

```
uint32_t storageID,
const void * objectID,
uint32_t objectIDLen,
uint32_t flags,
TEE_ObjectHandle * object )
```

Core Functions, Secure Storage Functions (data is isolated for each TA)

TEE_OpenPersistentObject() - Opens a handle on an existing persistent object.

This function returns a handle which can be used to access the object's attributes and data stream.

**Parameters**

| | |
|---|---|
| *storageID* | The storage to use |
| *objectID* | The object identifier |
| *objectIDLen* | The object identifier |
| *flags* | The flags which determine the settings under which the object is opened. |
| *object* | A pointer to the handle, which contains the opened handle upon successful completion |

**Returns**

0 if success else error occured.

#### 4.50.1.16 TEE_ReadObjectData()

```
TEE_Result TEE_ReadObjectData (
    TEE_ObjectHandle object,
    void * buffer,
    uint32_t size,
    uint32_t * count )
```

Core Functions, Secure Storage Functions (data is isolated for each TA)

TEE_ReadObjectData() - Attempts to read size bytes from the data stream associated with the object into the buffer pointed to by buffer.

The bytes are read starting at the position in the data stream currently stored in the object handle. The handle's position is incremented by the number of bytes actually read. On completion of TEE_ReadObjectData sets the number of bytes actually read in the "uint32_t" pointed to by count. The value written to ∗count may be less than size if the number of bytes until the end-of3067 stream is less than size. It is set to 0 if the position at the start of the read operation is at or beyond the end-of-stream. These are the only cases where ∗count may be less than size.

**Parameters**

| | |
|---|---|
| *object* | Handle of the object |
| *buffer* | The buffer containing the data to be written |
| *size* | The number of bytes to write |
| *count* | size of the buffer. |

**Returns**

TEE_SUCCESS if success else error occured.

#### 4.50.1.17 TEE_Realloc()

```
void * TEE_Realloc (
    void * buffer,
    uint32_t newSize )
```

TEE_Realloc() - Changes the size of the memory object pointed to by buffer to the size specified by new size.

This function describes the content of the object remains unchanged up to the lesser of the new and old sizes. Space in excess of the old size contains unspecified content. If the new size of the memory object requires movement of the object, the space for the previous instantiation of the object is deallocated. If the space cannot be allocated, the original object remains allocated, and this function returns a NULL pointer.

**Parameters**

| | |
|---|---|
| *buffer* | The pointer to the object to be reallocated. |
| *newSize* | The new size required for the object |

**Returns**

Upon successful completion, TEE_Realloc returns a pointer to the (possibly moved) allocated space. If there is not enough available memory, TEE_Realloc returns a NULL pointer and the original buffer is still allocated and unchanged.

#### 4.50.1.18 TEE_WriteObjectData()

```
TEE_Result TEE_WriteObjectData (
    TEE_ObjectHandle object,
    const void * buffer,
    uint32_t size )
```

Core Functions, Secure Storage Functions (data is isolated for each TA)

TEE_WriteObjectData() - Writes the buffer data in to persistent objects.

In this function it checks if object is present or not, the encryption/ decryption buffer is taken by calling mbedtls_aes↩ _crypt_cbc() then that buffer data is encrypted and mapped to object.On the base of object creation TEE_SUCCESS appears else TEE_ERROR_GENERIC appears.

**Parameters**

| | |
|---|---|
| *object* | Handle of the object |
| *buffer* | The buffer containing the data to be written |
| *size* | The number of bytes to write |

**Returns**

TEE_SUCCESS if success else error occured.

#### 4.50.1.19 wc_ocall_genseed() `int wc_ocall_genseed (`

```
    void * nonce,
    uint32_t len )
```

wc_ocall_genseed() To generate random data.

This function describes the return value of random generated data. if generated random value is not equal to length of buffer then panic
reason occurs.

**Parameters**

| | |
|---|---|
| *nonce* | pointer of buffer |
| *len* | length of the buffer. |

**Returns**

0 on success else error will occur based on panic raised inside trusted application.

### 4.50.2 Variable Documentation

#### 4.50.2.1 rngstr `WC_RNG rngstr [static]`

#### 4.50.2.2 wc_rng_init `int wc_rng_init = 0 [static]`

ocall_getrandom() - For getting random data.

This function describes that the retval is returned based on the size of buffer by calling the functions ocall_↩ getrandom196 and ocall_getrandom16

**Parameters**

| | |
|---|---|
| *buf* | character type buffer |
| *len* | size of the buffer |
| *flags* | unassigned integer flag |

**Returns**

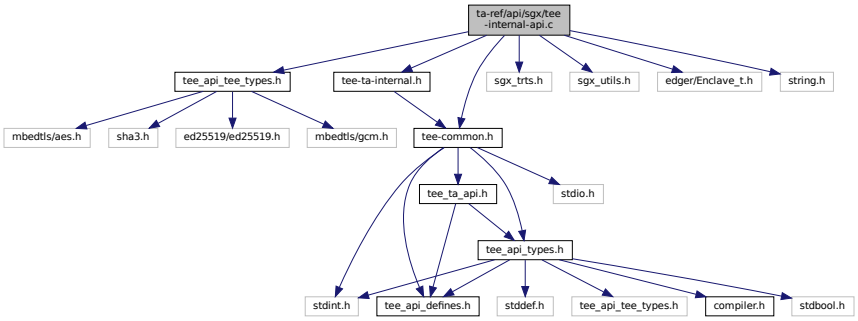
retval value will be returned based on length of buffer.

## 4.51 ta-ref/api/sgx/tee-internal-api.c File Reference

```
#include "tee_api_tee_types.h"
#include "tee-common.h"
```

```
#include "tee-ta-internal.h"
#include "sgx_trts.h"
#include "sgx_utils.h"
#include "edger/Enclave_t.h"
#include <string.h>
```

Include dependency graph for tee-internal-api.c:




## Functions

- void __attribute__ ((noreturn))
- void TEE_GetREETime (TEE_Time *time)

  *Core Functions, Time Functions.*
- void TEE_GetSystemTime (TEE_Time *time)

  *Core Functions, Time Functions.*
- TEE_Result GetRelTimeStart (uint64_t start)

  *Core Functions, Time Functions.*
- TEE_Result GetRelTimeEnd (uint64_t end)

  *Core Functions, Time Functions.*
- static int flags2flags (int flags)
- static int set_object_key (const void *id, unsigned int idlen, TEE_ObjectHandle object)
- static TEE_Result OpenPersistentObject (uint32_t storageID, const void *objectID, uint32_t objectIDLen, uint32_t flags, TEE_ObjectHandle *object, int ocreat)
- TEE_Result TEE_CreatePersistentObject (uint32_t storageID, const void *objectID, uint32_t objectIDLen, uint32_t flags, TEE_ObjectHandle attributes, const void *initialData, uint32_t initialDataLen, TEE_ObjectHandle *object)

  *Core Functions, Secure Storage Functions (data is isolated for each TA)*
- TEE_Result TEE_OpenPersistentObject (uint32_t storageID, const void *objectID, uint32_t objectIDLen, uint32_t flags, TEE_ObjectHandle *object)

  *Core Functions, Secure Storage Functions (data is isolated for each TA)*
- TEE_Result TEE_GetObjectInfo1 (TEE_ObjectHandle object, TEE_ObjectInfo *objectInfo)

  *Core Functions, Secure Storage Functions (data is isolated for each TA)*
- TEE_Result TEE_WriteObjectData (TEE_ObjectHandle object, const void *buffer, uint32_t size)

  *Core Functions, Secure Storage Functions (data is isolated for each TA)*
- TEE_Result TEE_ReadObjectData (TEE_ObjectHandle object, void *buffer, uint32_t size, uint32_t *count)

  *Core Functions, Secure Storage Functions (data is isolated for each TA)*
- void TEE_CloseObject (TEE_ObjectHandle object)

  *Core Functions, Secure Storage Functions (data is isolated for each TA)*
- void TEE_GenerateRandom (void *randomBuffer, uint32_t randomBufferLen)

  *Crypto, common.*
- static WC_RNG * get_wc_rng (void)

### Variables

- static int wc_rng_init = 0
- static WC_RNG rngstr

## 4.51.1 Function Documentation

#### 4.51.1.1 __attribute__() `void __attribute__ ( (noreturn) )`

TEE_Panic() - Raises a Panic in the Trusted Application instance

When a Trusted Application calls the TEE_Panic function, the current instance shall be destroyed and all the resources opened by the instance shall be reclaimed.

**Parameters**

| | |
|---|---|
| *ec* | An informative panic code defined by the TA. May be displayed in traces if traces are available. |

#### 4.51.1.2 flags2flags() `static int flags2flags ( int flags ) [inline], [static]`

flags2flags() - Checks the status for reading or writing of the file operational.

This function is to check the status for reading or writing of the file operational.

**Parameters**

| | |
|---|---|
| *flags* | Flags of the referencing node. |

**Returns**

0 if success else error occured.

#### 4.51.1.3 get_wc_rng() `static WC_RNG * get_wc_rng ( void ) [static]`

get_wc_rng() - Gets the seed (from OS) and key cipher for rng(random number genertor).

This function returns the random number or unique number of "rngstr".

**Returns**

random number if success else error occured.

#### 4.51.1.4 GetRelTimeEnd() `TEE_Result GetRelTimeEnd (`

```
uint64_t end )
```

Core Functions, Time Functions.

GetRelTimeStart() - find the real time of the end timing.

This function prints the End timing.

**Parameters**

| | |
|---|---|
| *end* | End timing |

**Returns**

0 if success else error occured

#### 4.51.1.5 GetRelTimeStart() `TEE_Result GetRelTimeStart (`

```
uint64_t start )
```

Core Functions, Time Functions.

GetRelTimeStart() - Gets the real time of the start timing.

Ths function prints the start timing.

**Parameters**

| | |
|---|---|
| *start* | start timing |

**Returns**

0 if success else error occured.

#### 4.51.1.6 OpenPersistentObject() `static TEE_Result OpenPersistentObject (`

```
uint32_t storageID,
const void * objectID,
```

```
uint32_t objectIDLen,
uint32_t flags,
TEE_ObjectHandle * object,
int ocreat )  [static]
```

OpenPersistentObject() - Opens a handle on an existing persistent object.

The flags parameter is a set of flags that controls the access rights and sharing permissions with which the object handle is opened. The value of the flags parameter is constructed by a bitwise-inclusive OR of flags TEE_DATA↩ _FLAG_ACCESS_READ, the object is opened with the read access right. This allows the Trusted Application to call the function TEE_ReadObjectData. TEE_DATA_FLAG_ACCESS_WRITE, the object is opened with the write access right. TEE_DATA_FLAG_ACCESS_WRITE_META, the object is opened with the write-meta access right.

**Parameters**

| | |
|---|---|
| *storageID* | The storage to use. |
| *objectID* | The object identifier |
| *objectIDLen* | length of the identifier |
| *flags* | The flags which determine the settings under which the object is opened. |
| *object* | A pointer to the handle, which contains the opened handle upon successful completion. |

**Returns**

0 if success else error occured.

#### 4.51.1.7 set_object_key()

```
static int set_object_key (
const void * id,
unsigned int idlen,
TEE_ObjectHandle object )  [static]
```

set_object_key - To initalize report and then attest enclave with file.

This function describes objectID as key_id to make the key dependent on it sgx report key is 128-bit. Fill another 128-bit with seal key. seal key doesn't change with enclave. Better than nothing, though. random nonce can not use for AES here because of persistency. the digest of attestation report and objectID as the last resort has been used.

**Parameters**

| | |
|---|---|
| *id* | id of the object. |
| *idlen* | length of the id. |
| *object* | TEE_ObjectHandle type handle. |

**Returns**

0 if success else error occured.

#### 4.51.1.8 TEE_CloseObject() `void TEE_CloseObject (`

```
TEE_ObjectHandle object )
```

Core Functions, Secure Storage Functions (data is isolated for each TA)

TEE_CloseObject() - Function closes an opened object handle.

The object can be persistent or transient.For transient objects, TEE_CloseObject is equivalent to TEE_Free↩ TransientObject.

**Parameters**

| | |
|---|---|
| *object* | Handle of the object |

**Returns**

TEE_SUCCESS if success else error occured.

#### 4.51.1.9 TEE_CreatePersistentObject() `TEE_Result TEE_CreatePersistentObject (`

```
uint32_t storageID,
const void * objectID,
uint32_t objectIDLen,
uint32_t flags,
TEE_ObjectHandle attributes,
const void * initialData,
uint32_t initialDataLen,
TEE_ObjectHandle * object )
```

Core Functions, Secure Storage Functions (data is isolated for each TA)

TEE_CreatePersistentObject() - Creates a persistent object with initial attributes.

An initial data stream content, and optionally returns either a handle on the created object, or TEE_HANDLE_NULL upon failure.

**Parameters**

| | |
|---|---|
| *storageID* | The storage to use. |
| *objectID* | The object identifier |
| *objectIDLen* | The object identifier |
| *flags* | The flags which determine the settings under which the object is opened. |
| *attributes* | A handle on a persistent object or an initialized transient object from which to take the persistent object attributes |
| *initialData* | The initial data content of the persistent object |
| *initialDataLen* | The initial data content of the persistent object |
| *object* | A pointer to the handle, which contains the opened handle upon successful completion |

**Returns**

0 if success, else error occured.

#### 4.51.1.10 TEE_GenerateRandom()

```
void TEE_GenerateRandom (
    void * randomBuffer,
    uint32_t randomBufferLen )
```

Crypto, common.

TEE_GenerateRandom() - Generates random data.

This function generates random data of random bufferlength and is stored in to randomBuffer by calling sgx_read↩ _rand().

**Parameters**

| | |
|---|---|
| *randomBuffer* | Reference to generated random data |
| *randomBufferLen* | Byte length of requested random data |

#### 4.51.1.11 TEE_GetObjectInfo1()

```
TEE_Result TEE_GetObjectInfo1 (
    TEE_ObjectHandle object,
    TEE_ObjectInfo * objectInfo )
```

Core Functions, Secure Storage Functions (data is isolated for each TA)

TEE_GetObjectInfo1() - Function returns the characteristics of an object.

It returns a handle that can be used to access the object's attributes and data stream.

**Parameters**

| | |
|---|---|
| *objectInfo* | Pointer to a structure filled with the object information |
| *object* | Handle of the object |

**Returns**

0 if success else error occured.

#### 4.51.1.12 TEE_GetREETime()

```
void TEE_GetREETime (
    TEE_Time * time )
```

Core Functions, Time Functions.

TEE_GetREETime() - Function retrieves the current REE system time.

This function retrieves the current time as seen from the point of view of the REE.

**Parameters**

| | |
|---|---|
| *time* | Filled with the number of seconds and milliseconds. |

#### 4.51.1.13 TEE_GetSystemTime()

```
void TEE_GetSystemTime (
    TEE_Time * time )
```

Core Functions, Time Functions.

TEE_GetSystemTime() - Retrieves the current system time.

The system time has an arbitrary implementation-defined origin that can vary across TA instances

**Parameters**

| | |
|---|---|
| *time* | Filled with the number of seconds and milliseconds. |

#### 4.51.1.14 TEE_OpenPersistentObject()

```
TEE_Result TEE_OpenPersistentObject (
    uint32_t storageID,
    const void * objectID,
    uint32_t objectIDLen,
    uint32_t flags,
    TEE_ObjectHandle * object )
```

Core Functions, Secure Storage Functions (data is isolated for each TA)

TEE_OpenPersistentObject() - Opens a handle on an existing persistent object.

This function returns a handle that can be used to access the object's attributes and data stream.

**Parameters**

| | |
|---|---|
| *storageID* | The storage to use. |
| *objectID* | The object identifier |
| *objectIDLen* | The object identifier |
| *flags* | The flags which determine the settings under which the object is opened. |
| *object* | A pointer to the handle, which contains the opened handle upon successful completion |

**Returns**

0 if success, else error occured.

#### 4.51.1.15 TEE_ReadObjectData()

```
TEE_Result TEE_ReadObjectData (
    TEE_ObjectHandle object,
    void * buffer,
    uint32_t size,
    uint32_t * count )
```

Core Functions, Secure Storage Functions (data is isolated for each TA)

TEE_ReadObjectData() - Attempts to read size bytes from the data stream associated with the object object into the buffer pointed to by buffer.

The bytes are read starting at the position in the data stream currently stored in the object handle. The handle's position is incremented by the number of bytes actually read. On completion TEE_ReadObjectData sets the number of bytes actually read in the uint32_t pointed to by count. The value written to ∗count may be less than size if the number of bytes until the end-of3067 stream is less than size. It is set to 0 if the position at the start of the read operation is at or beyond the end-of-stream. These are the only cases where ∗count may be less than size.

**Parameters**

| | |
|---|---|
| *object* | Handle of the object |
| *buffer* | The buffer containing the data to be written |
| *size* | The number of bytes to write |
| *count* | size of the buffer. |

**Returns**

TEE_SUCCESS if success, else error occured.

#### 4.51.1.16 TEE_WriteObjectData()

```
TEE_Result TEE_WriteObjectData (
    TEE_ObjectHandle object,
    const void * buffer,
    uint32_t size )
```

Core Functions, Secure Storage Functions (data is isolated for each TA)

TEE_WriteObjectData() - writes size bytes from the buffer pointed to by buffer to the data stream associated with the open object handle object.

If the current data position points before the end-of-stream, then size bytes are written to the data stream, overwriting bytes starting at the current data position. If the current data position points beyond the stream's end, then the data stream is first extended with zero bytes until the length indicated by the data position indicator is reached, and then size bytes are written to the stream.

**Parameters**

| | |
|---|---|
| *object* | Handle of the object |
| *buffer* | The buffer containing the data to be written |
| *size* | The number of bytes to write |

**Returns**

TEE_SUCCESS if success else error occured.

### 4.51.2 Variable Documentation

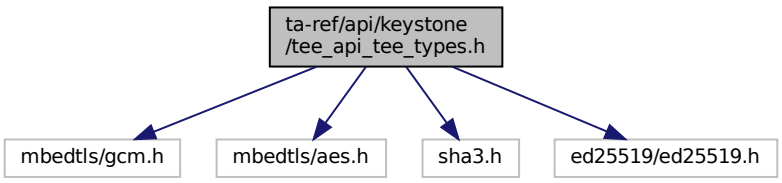
#### 4.51.2.1 rngstr `WC_RNG rngstr [static]`

#### 4.51.2.2 wc_rng_init `int wc_rng_init = 0 [static]`

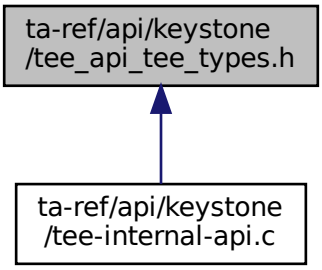
## 4.52 ta-ref/api/keystone/tee_api_tee_types.h File Reference

```
#include "mbedtls/gcm.h"
#include "mbedtls/aes.h"
#include "sha3.h"
#include "ed25519/ed25519.h"
```

Include dependency graph for tee_api_tee_types.h:



This graph shows which files directly or indirectly include this file:




### Classes

- struct __TEE_OperationHandle
- struct __TEE_ObjectHandle

## 4.53 tee_api_tee_types.h

Go to the documentation of this file.

```
1 /*
2  * SPDX-License-Identifier: BSD-2-Clause
3  *
4  * Copyright (C) 2019 National Institute of Advanced Industrial Science
5  *                          and Technology (AIST)
6  * All rights reserved.
7  *
8  * Redistribution and use in source and binary forms, with or without
9  * modification, are permitted provided that the following conditions are met:
10  *
11  * 1. Redistributions of source code must retain the above copyright notice,
12  * this list of conditions and the following disclaimer.
13  *
14  * 2. Redistributions in binary form must reproduce the above copyright notice,
15  * this list of conditions and the following disclaimer in the documentation
16  * and/or other materials provided with the distribution.
17  *
18  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
19  * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
20  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
21  * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
22  * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
23  * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
24  * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
25  * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
26  * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
27  * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
28  * POSSIBILITY OF SUCH DAMAGE.
29  */
30
31 #ifndef TEE_API_TYPES_KEYSTONE_H
32 #define TEE_API_TYPES_KEYSTONE_H
33
34 #ifndef DOXYGEN_SHOULD_SKIP_THIS
35 #define MBEDCRYPT 1
36 #define WOLFCRYPT 2
37 #endif /*DOXYGEN_SHOULD_SKIP_THIS*/
38
39 #if CRYPTLIB==MBEDCRYPT
40 #ifndef DOXYGEN_SHOULD_SKIP_THIS
```

```
41 # define MBEDTLS_CONFIG_FILE "mbed-crypto-config.h"
42 # define AES256 1
43 #endif /*DOXYGEN_SHOULD_SKIP_THIS*/
44 # include "mbedtls/gcm.h"
45 # include "mbedtls/aes.h"
46 # include "sha3.h"
47 # include "ed25519/ed25519.h"
48 #elif CRYPTLIB==WOLFCRYPT
49 #ifndef DOXYGEN_SHOULD_SKIP_THIS
50 # define HAVE_AESGCM 1
51 # define HAVE_AES_CBC 1
52 # define HAVE_AES_DECRYPT 1
53 # define HAVE_FIPS 1
54 # define HAVE_FIPS_VERSION 2
55 # define HAVE_ED25519 1
56 # define HAVE_ED25519_SIGN 1
57 # define HAVE_ED25519_VERIFY 1
58 # define WOLFSSL_SHA512 1
59 # define WOLFSSL_SHA3 1
60 # define WOLFSSL_SHA3_SMALL 1
61 # define WOLFCRYPT_ONLY 1
62 # define WOLF_CRYPT_PORT_H
63 #endif /*DOXYGEN_SHOULD_SKIP_THIS*/
64 # include "wolfssl/wolfcrypt/sha3.h"
65 # include "wolfssl/wolfcrypt/aes.h"
66 # include "wolfssl/wolfcrypt/sha512.h"
67 # include "wolfssl/wolfcrypt/ed25519.h"
68 #else
69 # include "sha3.h"
70 # include "ed25519/ed25519.h"
71 # include "tiny_AES_c/aes.h"
72 #ifndef DOXYGEN_SHOULD_SKIP_THIS
73 # define AES256 1
74 #endif /*DOXYGEN_SHOULD_SKIP_THIS*/
75 #endif
76
77 #ifndef DOXYGEN_SHOULD_SKIP_THIS
78 #define SHA_LENGTH (256/8)
79 #define TEE_OBJECT_NONCE_SIZE 16
80 #define TEE_OBJECT_KEY_SIZE 32
81 #define TEE_OBJECT_SKEY_SIZE 64
82 #define TEE_OBJECT_AAD_SIZE 16
83 #define TEE_OBJECT_TAG_SIZE 16
84 #endif /*DOXYGEN_SHOULD_SKIP_THIS*/
85
86 struct __TEE_OperationHandle
87 {
88   int mode;
89   int flags;
90   int alg;
91 #if CRYPTLIB==MBEDCRYPT
92   sha3_ctx_t ctx;
93   mbedtls_aes_context aectx;
94   mbedtls_gcm_context aegcmctx;
95 #elif CRYPTLIB==WOLFCRYPT
96   wc_Sha3 ctx;
97   Aes aectx;
98   Aes aegcmctx;
99   unsigned int aegcm_aadsz;
100    unsigned char aegcm_aad[TEE_OBJECT_AAD_SIZE];
101    ed25519_key key;
102 #else
103    sha3_ctx_t ctx;
104    struct AES_ctx aectx;
105 #endif
106    int aegcm_state;
107    unsigned char aeiv[TEE_OBJECT_NONCE_SIZE];
108    unsigned char aekey[32];
109    unsigned char pubkey[TEE_OBJECT_KEY_SIZE];
110    unsigned char prikey[TEE_OBJECT_SKEY_SIZE];
111 };
112
113 struct __TEE_ObjectHandle
114 {
115    unsigned int type;
116    int flags;
117    int desc;
118 #if CRYPTLIB==MBEDCRYPT
119   mbedtls_aes_context persist_ctx;
120   unsigned char persist_iv[TEE_OBJECT_NONCE_SIZE];
121 #elif CRYPTLIB==WOLFCRYPT
122   Aes persist_ctx;
123   unsigned char persist_iv[TEE_OBJECT_NONCE_SIZE];
124   ed25519_key key;
125 #else
```

```
126   struct AES_ctx persist_ctx;
127 #endif
128   unsigned char public_key[TEE_OBJECT_KEY_SIZE];
129   unsigned char private_key[TEE_OBJECT_SKEY_SIZE];
130 };
131
132 // defined in tee_api_defines.h
133 // enum Data_Flag_Constants {
134 //   TEE_DATA_FLAG_ACCESS_READ = 0x00000001,
135 //   TEE_DATA_FLAG_ACCESS_WRITE = 0x00000002,
136 //   //TEE_DATA_FLAG_ACCESS_WRITE_META = 0x00000004,
137 //   //TEE_DATA_FLAG_SHARE_READ = 0x00000010,
138 //   //TEE_DATA_FLAG_SHARE_WRITE = 0x00000020,
139 //   TEE_DATA_FLAG_OVERWRITE = 0x00000400
140 // };
141 // enum Data_Flag_Constants {
142 //   TEE_DATA_FLAG_ACCESS_READ = 0x00000001,
143 //   TEE_DATA_FLAG_ACCESS_WRITE = 0x00000002,
144 //   //TEE_DATA_FLAG_ACCESS_WRITE_META = 0x00000004,
145 //   //TEE_DATA_FLAG_SHARE_READ = 0x00000010,
146 //   //TEE_DATA_FLAG_SHARE_WRITE = 0x00000020,
147 //   TEE_DATA_FLAG_OVERWRITE = 0x00000400
148 // };
149 #endif
```

## 4.54 ta-ref/api/optee/tee_api_tee_types.h File Reference

## 4.55 tee_api_tee_types.h

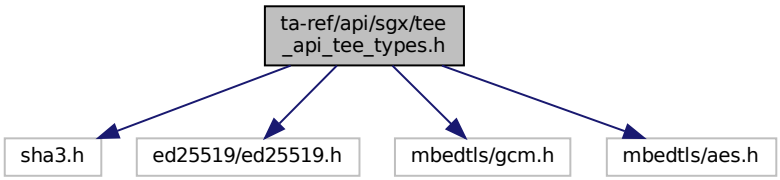
Go to the documentation of this file.

```
1 // empty
```

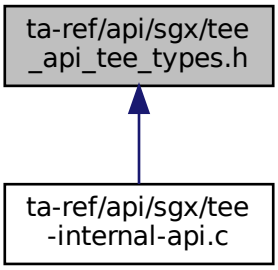
## 4.56 ta-ref/api/sgx/tee_api_tee_types.h File Reference

```
#include "sha3.h"
#include "ed25519/ed25519.h"
#include "mbedtls/gcm.h"
#include "mbedtls/aes.h"
```

Include dependency graph for tee_api_tee_types.h:



This graph shows which files directly or indirectly include this file:




### Classes

- struct __TEE_OperationHandle
- struct __TEE_ObjectHandle

## 4.57 tee_api_tee_types.h

Go to the documentation of this file.

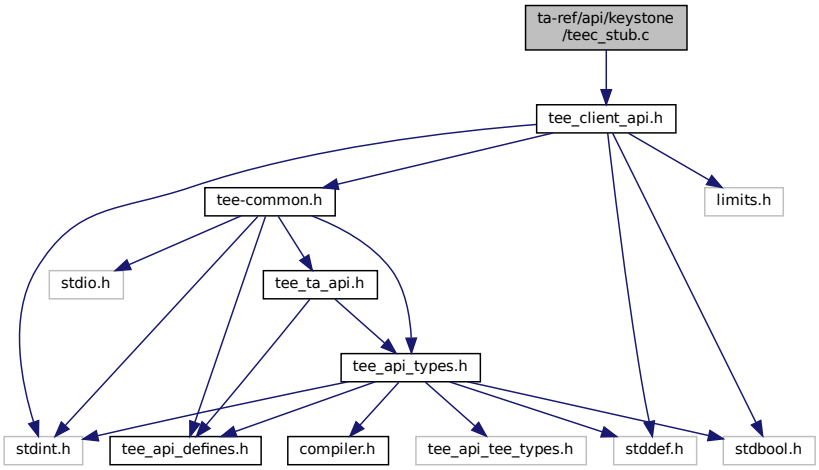
```
1 /*
2  * SPDX-License-Identifier: BSD-2-Clause
3  *
4  * Copyright (C) 2019 National Institute of Advanced Industrial Science
5  *                          and Technology (AIST)
6  * All rights reserved.
7  *
8  * Redistribution and use in source and binary forms, with or without
9  * modification, are permitted provided that the following conditions are met:
10  *
11  * 1. Redistributions of source code must retain the above copyright notice,
12  * this list of conditions and the following disclaimer.
13  *
14  * 2. Redistributions in binary form must reproduce the above copyright notice,
15  * this list of conditions and the following disclaimer in the documentation
16  * and/or other materials provided with the distribution.
17  *
18  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
19  * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
20  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
21  * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
22  * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
23  * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
24  * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
25  * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
26  * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
27  * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
28  * POSSIBILITY OF SUCH DAMAGE.
29  */
30
31 #ifndef TEE_API_TYPES_KEYSTONE_H
32 #define TEE_API_TYPES_KEYSTONE_H
33
34 #ifndef DOXYGEN_SHOULD_SKIP_THIS
35 #define MBEDCRYPT 1
36 #define WOLFCRYPT 2
37 #define SHA_LENGTH (256/8)
38 #define AES256 1
39 #endif /*DOXYGEN_SHOULD_SKIP_THIS*/
40
```

```
41 #include "sha3.h"
42 #include "ed25519/ed25519.h"
43
44 #if CRYPTLIB==MBEDCRYPT
45 #ifndef DOXYGEN_SHOULD_SKIP_THIS
46 # define MBEDTLS_CONFIG_FILE "mbed-crypto-config.h"
47 #endif /*DOXYGEN_SHOULD_SKIP_THIS*/
48 # include "mbedtls/gcm.h"
49 # include "mbedtls/aes.h"
50 #elif CRYPTLIB==WOLFCRYPT
51 #ifndef DOXYGEN_SHOULD_SKIP_THIS
52 # define HAVE_AESGCM 1
53 # define HAVE_AES_CBC 1
54 # define HAVE_AES_DECRYPT 1
55 # define HAVE_FIPS 1
56 # define HAVE_FIPS_VERSION 2
57 # define HAVE_ED25519 1
58 # define HAVE_ED25519_SIGN 1
59 # define HAVE_ED25519_VERIFY 1
60 # define WOLFSSL_SHA3 1
61 # define WOLF_CRYPT_PORT_H
62 #endif /*DOXYGEN_SHOULD_SKIP_THIS*/
63 # include "wolfssl/wolfcrypt/sha3.h"
64 # include "wolfssl/wolfcrypt/aes.h"
65 # include "wolfssl/wolfcrypt/sha512.h"
66 # include "wolfssl/wolfcrypt/ed25519.h"
67 #else
68 # include "tiny_AES_c/aes.h"
69 #endif
70
71 #ifndef DOXYGEN_SHOULD_SKIP_THIS
72 #define TEE_OBJECT_NONCE_SIZE 16
73 #define TEE_OBJECT_KEY_SIZE 32
74 #define TEE_OBJECT_SKEY_SIZE 64
75 #define TEE_OBJECT_AAD_SIZE 16
76 #define TEE_OBJECT_TAG_SIZE 16
77 #endif /*DOXYGEN_SHOULD_SKIP_THIS*/
78
79 struct __TEE_OperationHandle
80 {
81   int mode;
82   int flags;
83   int alg;
84 #if CRYPTLIB==MBEDCRYPT
85   sha3_ctx_t ctx;
86   mbedtls_aes_context aectx;
87   mbedtls_gcm_context aegcmctx;
88 #elif CRYPTLIB==WOLFCRYPT
89   wc_Sha3 ctx;
90   Aes aectx;
91   Aes aegcmctx;
92   unsigned int aegcm_aadsz;
93   unsigned char aegcm_aad[TEE_OBJECT_AAD_SIZE];
94   ed25519_key key;
95 #else
96   sha3_ctx_t ctx;
97   struct AES_ctx aectx;
98 #endif
99   int aegcm_state;
100   unsigned char aeiv[TEE_OBJECT_NONCE_SIZE];
101   unsigned char aekey[32];
102   unsigned char pubkey[TEE_OBJECT_KEY_SIZE];
103   unsigned char prikey[TEE_OBJECT_SKEY_SIZE];
104 };
105
106 struct __TEE_ObjectHandle
107 {
108   unsigned int type;
109   int flags;
110   int desc;
111 #if CRYPTLIB==MBEDCRYPT
112   mbedtls_aes_context persist_ctx;
113   unsigned char persist_iv[TEE_OBJECT_NONCE_SIZE];
114 #elif CRYPTLIB==WOLFCRYPT
115   Aes persist_ctx;
116   unsigned char persist_iv[TEE_OBJECT_NONCE_SIZE];
117   ed25519_key key;
118 #else
119   struct AES_ctx persist_ctx;
120 #endif
121   unsigned char public_key[TEE_OBJECT_KEY_SIZE];
122   unsigned char private_key[TEE_OBJECT_SKEY_SIZE];
123 };
124
125 // Minimal constant definitions
```

```
126 #ifndef DOXYGEN_SHOULD_SKIP_THIS
127 #define TEE_HANDLE_NULL 0
128 #endif /*DOXYGEN_SHOULD_SKIP_THIS*/
129
130 #endif
```

## 4.58 ta-ref/api/keystone/teec_stub.c File Reference

`#include <tee_client_api.h>`
Include dependency graph for teec_stub.c:




### Functions

- TEEC_Result TEEC_InitializeContext (const char *name, TEEC_Context *context)
- void TEEC_FinalizeContext (TEEC_Context *context)
- TEEC_Result TEEC_OpenSession (TEEC_Context *context, TEEC_Session *session, const TEEC_UUID *destination, uint32_t connectionMethod, const void *connectionData, TEEC_Operation *operation, uint32_t *returnOrigin)
- void TEEC_CloseSession (TEEC_Session *session)
- TEEC_Result TEEC_RegisterSharedMemory (TEEC_Context *context, TEEC_SharedMemory *sharedMem)
- TEEC_Result TEEC_AllocateSharedMemory (TEEC_Context *context, TEEC_SharedMemory *sharedMem)
- void TEEC_ReleaseSharedMemory (TEEC_SharedMemory *sharedMemory)
- void TEEC_RequestCancellation (TEEC_Operation *operation)

### 4.58.1 Function Documentation

#### 4.58.1.1 TEEC_AllocateSharedMemory()

```
TEEC_Result TEEC_AllocateSharedMemory (
        TEEC_Context * context,
        TEEC_SharedMemory * sharedMem )
```

TEEC_AllocateSharedMemory() - Allocate shared memory for TEE.

**Parameters**

| | |
|---|---|
| *context* | The initialized TEE context structure in which scope to open the session. |
| *sharedMem* | Pointer to the allocated shared memory. |

**Returns**

TEEC_SUCCESS The registration was successful.

TEEC_ERROR_OUT_OF_MEMORY Memory exhaustion.

TEEC_Result Something failed.

#### 4.58.1.2 TEEC_CloseSession() `void TEEC_CloseSession (`

```
TEEC_Session * session )
```

TEEC_CloseSession() - Closes the session which has been opened with the specific trusted application.

**Parameters**

| | |
|---|---|
| *session* | The opened session to close. |

#### 4.58.1.3 TEEC_FinalizeContext() `void TEEC_FinalizeContext (`

```
TEEC_Context * context )
```

TEEC_FinalizeContext() - Destroys a context holding connection information on the specific TEE.

This function finalizes an initialized TEE context, closing the connection between the client application and the TEE. This function must only be called when all sessions related to this TEE context have been closed and all shared memory blocks have been released.

**Parameters**

| | |
|---|---|
| *context* | The context to be finalized. |

#### 4.58.1.4 TEEC_InitializeContext() `TEEC_Result TEEC_InitializeContext (`

```
const char * name,
TEEC_Context * context )
```

TEEC_InitializeContext() - Initializes a context holding connection information on the specific TEE, designated by the name string.

**Parameters**

| | |
|---|---|
| *name* | A zero-terminated string identifying the TEE to connect to. If name is set to NULL, the default TEE is connected to. NULL is the only supported value in this version of the API implementation. |
| *context* | The context structure which is to be initialized. |

**Returns**

TEEC_SUCCESS The initialization was successful.

TEEC_Result Something failed.

#### 4.58.1.5 TEEC_OpenSession()

```
TEEC_Result TEEC_OpenSession (
    TEEC_Context * context,
    TEEC_Session * session,
    const TEEC_UUID * destination,
    uint32_t connectionMethod,
    const void * connectionData,
    TEEC_Operation * operation,
    uint32_t * returnOrigin )
```

TEEC_OpenSession() - Opens a new session with the specified trusted application.

**Parameters**

| | |
|---|---|
| *context* | The initialized TEE context structure in which scope to open the session. |
| *session* | The session to initialize. |
| *destination* | A structure identifying the trusted application with which to open a session. |
| *connectionMethod* | The connection method to use. |
| *connectionData* | Any data necessary to connect with the chosen connection method. Not supported, should be set to NULL. |
| *operation* | An operation structure to use in the session. May be set to NULL to signify no operation structure needed. |
| *returnOrigin* | A parameter which will hold the error origin if this function returns any value other than TEEC_SUCCESS. |

**Returns**

TEEC_SUCCESS OpenSession successfully opened a new session.

TEEC_Result Something failed.

#### 4.58.1.6 TEEC_RegisterSharedMemory()

```
TEEC_Result TEEC_RegisterSharedMemory (
    TEEC_Context * context,
    TEEC_SharedMemory * sharedMem )
```

TEEC_RegisterSharedMemory() - Register a block of existing memory as a shared block within the scope of the specified context.

**Parameters**

| | |
|---|---|
| *context* | The initialized TEE context structure in which scope to open the session. |
| *sharedMem* | pointer to the shared memory structure to register. |

**Returns**

TEEC_SUCCESS The registration was successful.

TEEC_ERROR_OUT_OF_MEMORY Memory exhaustion.

TEEC_Result Something failed.

#### 4.58.1.7 TEEC_ReleaseSharedMemory() `void TEEC_ReleaseSharedMemory ( TEEC_SharedMemory * sharedMemory )`

TEEC_ReleaseSharedMemory() - Free or deregister the shared memory.

**Parameters**

| | |
|---|---|
| *sharedMem* | Pointer to the shared memory to be freed. |

#### 4.58.1.8 TEEC_RequestCancellation() `void TEEC_RequestCancellation ( TEEC_Operation * operation )`

TEEC_RequestCancellation() - Request the cancellation of a pending open session or command invocation.

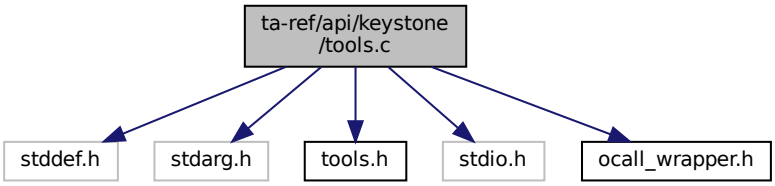
**Parameters**

| | |
|---|---|
| *operation* | Pointer to an operation previously passed to open session or invoke. |

## 4.59 ta-ref/api/keystone/tools.c File Reference

```
#include <stddef.h>
#include <stdarg.h>
#include "tools.h"
#include <stdio.h>
#include "ocall_wrapper.h"
```

Include dependency graph for tools.c:




### Functions

- static unsigned int _strlen (const char ∗str)
- int puts (const char ∗s)
- int putchar (int c)
- int printf (const char ∗fmt,...)

### 4.59.1 Function Documentation

#### 4.59.1.1 _strlen()

```
static unsigned int _strlen (
    const char * str )  [inline], [static]
```

#### 4.59.1.2 printf()

```
int printf (
    const char * fmt,
     ...  )
```

printf() - Function sends formatted output to stdout.

format can optionally contain embedded format tags that are replaced by the values specified in subsequent additional arguments and formatted as requested.

**Parameters**

| | |
|---|---|
| *fm* | This is the string that contains the text to be written to stdout. |

**Returns**

string length If success.

0 Error occured.

#### 4.59.1.3 putchar() `int putchar ( int c )`

putchar() - Function writes a character (an unsigned char) specified by the argument char to stdout.

This function returns the character written as an unsigned char cast to an int or EOF on error.

**Parameters**

| | |
|---|---|
| *c* | This is the character to be written. This is passed as its int promotion. |

**Returns**

size If success.

0 Error occured.

#### 4.59.1.4 puts() `int puts ( const char * s )`

puts() - Function writes a string to stdout up to but not including the null character.

A newline character is appended to the output by calling putchar(). Compiler may replace simple printf to puts and putchar.

**Parameters**

| | |
|---|---|
| *s* | This is the C string to be written |

**Returns**

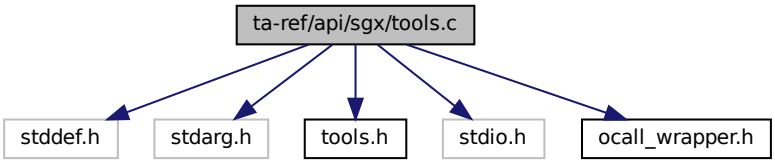
size If success.

0 Error occured.

## 4.60 ta-ref/api/sgx/tools.c File Reference

```
#include <stddef.h>
#include <stdarg.h>
#include "tools.h"
#include <stdio.h>
#include "ocall_wrapper.h"
```

Include dependency graph for tools.c:




### Functions

- static unsigned int _strlen (const char *str)
- int puts (const char *s)
- int putchar (int c)
- int printf (const char *fmt,...)

#### 4.60.1 Function Documentation

##### 4.60.1.1 _strlen()

```
static unsigned int _strlen (
    const char * str )  [inline], [static]
```

##### 4.60.1.2 printf()

```
int printf (
    const char * fmt,
     ...  )
```

printf() - Function sends formatted output to stdout.

format can optionally contain embedded format tags that are replaced by the values specified in subsequent additional arguments and formatted as requested.

**Parameters**

| | |
|---|---|
| *fm* | This is the string that contains the text to be written to stdout. |

**Returns**

string length If success.

0 Error occured.

#### 4.60.1.3 putchar() `int putchar ( int c )`

putchar() - Function writes a character (an unsigned char) specified by the argument char to stdout.

This function returns the character written as an unsigned char cast to an int or EOF on error.

**Parameters**

| | |
|---|---|
| *c* | This is the character to be written. This is passed as its int promotion. |

**Returns**

size If success.

0 Error occured.

#### 4.60.1.4 puts() `int puts ( const char * s )`

puts() - Function writes a string to stdout up to but not including the null character.

A newline character is appended to the output by calling putchar(). Compiler may replace simple printf to puts and putchar.

**Parameters**

| | |
|---|---|
| *s* | This is the C string to be written |

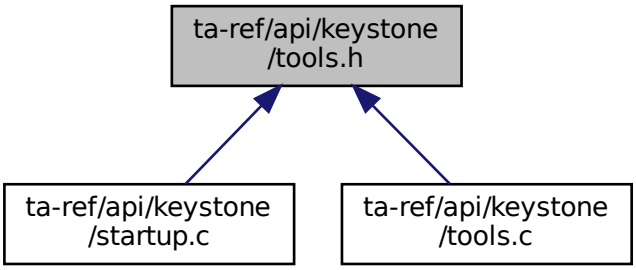
**Returns**

size If success.

0 Error occured.

## 4.61 ta-ref/api/keystone/tools.h File Reference

This graph shows which files directly or indirectly include this file:




### Functions

- int puts (const char ∗s)
- int putchar (int c)
- int printf (const char ∗fmt,...)

### 4.61.1 Function Documentation

#### 4.61.1.1 printf() `int printf ( const char * fmt, ... )`

printf() - Function sends formatted output to stdout.

format can optionally contain embedded format tags that are replaced by the values specified in subsequent additional arguments and formatted as requested.

**Parameters**

| | |
|---|---|
| *fm* | This is the string that contains the text to be written to stdout. |

**Returns**

string length If success.

0 Error occured.

#### 4.61.1.2 putchar() int putchar ( int c )

putchar() - Function writes a character (an unsigned char) specified by the argument char to stdout.

This function returns the character written as an unsigned char cast to an int or EOF on error.

**Parameters**

| | |
|---|---|
| *c* | This is the character to be written. This is passed as its int promotion. |

**Returns**

size If success.

0 Error occured.

#### 4.61.1.3 puts() int puts ( const char * s )

puts() - Function writes a string to stdout up to but not including the null character.

A newline character is appended to the output by calling putchar(). Compiler may replace simple printf to puts and putchar.

**Parameters**

| | |
|---|---|
| *s* | This is the C string to be written |

**Returns**

size If success.

0 Error occured.

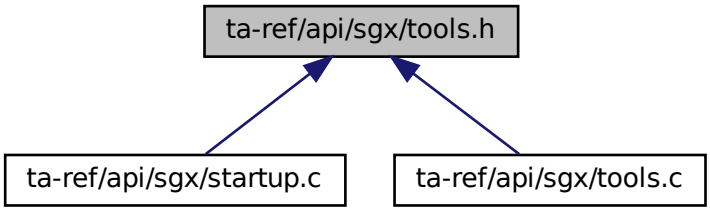
## 4.62 tools.h

Go to the documentation of this file.

```
1 int puts(const char *s);
2 int putchar(int c);
3 int printf(const char* fmt, ...);
```

## 4.63 ta-ref/api/sgx/tools.h File Reference

This graph shows which files directly or indirectly include this file:




### Functions

- int puts (const char ∗s)
- int putchar (int c)
- int printf (const char ∗fmt,...)

### 4.63.1 Function Documentation

#### 4.63.1.1 printf()

```
int printf (
        const char * fmt,
         ...  )
```

printf() - Function sends formatted output to stdout.

format can optionally contain embedded format tags that are replaced by the values specified in subsequent additional arguments and formatted as requested.

**Parameters**

| | |
|---|---|
| *fm* | This is the string that contains the text to be written to stdout. |

**Returns**

string length If success.

0 Error occured.

#### 4.63.1.2 putchar() int putchar ( int c )

putchar() - Function writes a character (an unsigned char) specified by the argument char to stdout.

This function returns the character written as an unsigned char cast to an int or EOF on error.

**Parameters**

| | |
|---|---|
| *c* | This is the character to be written. This is passed as its int promotion. |

**Returns**

size If success.

0 Error occured.

#### 4.63.1.3 puts() int puts ( const char * s )

puts() - Function writes a string to stdout up to but not including the null character.

A newline character is appended to the output by calling putchar(). Compiler may replace simple printf to puts and putchar.

**Parameters**

| | |
|---|---|
| *s* | This is the C string to be written |

**Returns**

size If success.

0 Error occured.

## 4.64 tools.h

Go to the documentation of this file.

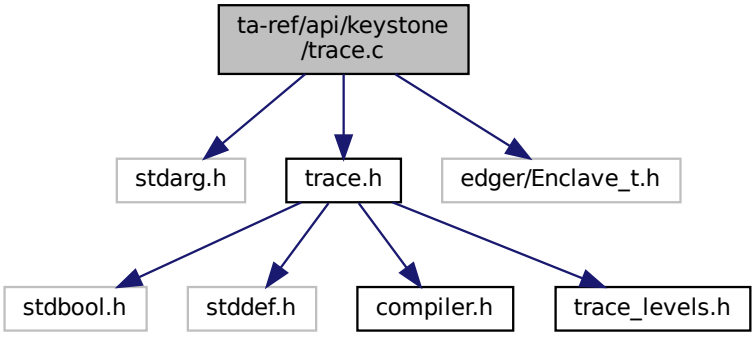
```
1 int puts(const char *s);
2 int putchar(int c);
3 int printf(const char* fmt, ...);
```

## 4.65 ta-ref/api/keystone/trace.c File Reference

```
#include <stdarg.h>
#include "trace.h"
```

```
#include "edger/Enclave_t.h"
```

Include dependency graph for trace.c:




### Functions

- void trace_vprintf (const char ∗func, int line, int level, bool level_ok, const char ∗fmt, va_list ap)
- void trace_printf (const char ∗func, int line, int level, bool level_ok, const char ∗fmt,...)

### 4.65.1 Function Documentation

#### 4.65.1.1 trace_printf()

```
void trace_printf (
        const char * func,
        int line,
        int level,
        bool level_ok,
        const char * fmt,
         ...  )
```

trace_printf() - Prints the formatted data to stdout.

This function returns the value of ap by calling va_end().

**Parameters**

| | |
|---|---|
| *func* | Pointer to a buffer where the resulting C-string is stored. |
| *line* | integer type of line |
| *level_ok* | boolen value |
| *fmt* | C string that contains a format string |
| *ap* | A value identifying a variable arguments list |

**Returns**

Total number of characters is returned.

#### 4.65.1.2 trace_vprintf()

```
void trace_vprintf (
        const char * func,
        int line,
        int level,
        bool level_ok,
        const char * fmt,
        va_list ap )
```

trace_vprintf() - Writes the formatted data from variable argument list to sized buffer.

This function returns the buffer character by calling ocall_print_string()

**Parameters**

| | |
|---|---|
| *func* | Pointer to a buffer where the resulting C-string is stored. |
| *line* | integer type of line |
| *level_ok* | boolen value |
| *fmt* | C string that contains a format string |
| *ap* | A value identifying a variable arguments list |

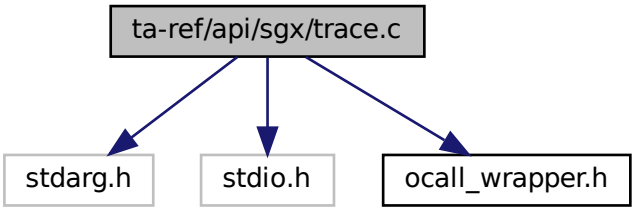
**Returns**

buf The total number of characters written is returned.

## 4.66 ta-ref/api/sgx/trace.c File Reference

```
#include <stdarg.h>
#include <stdio.h>
#include "ocall_wrapper.h"
```

Include dependency graph for trace.c:



### Functions

- static unsigned int _strlen (const char ∗str)
- int tee_printf (const char ∗fmt,...)

#### 4.66.1 Function Documentation

##### 4.66.1.1 _strlen() `static unsigned int _strlen ( const char * str ) [inline], [static]`

_strlen() - calculate the length of characters in a str.

**Parameters**

| | |
|---|---|
| *str* | str is an argument of type pointer. |

**Returns**

string length on success.

##### 4.66.1.2 tee_printf() `int tee_printf ( const char * fmt, ... )`

tee_printf() - For tracing GP API.

Initializes ap variable. Formats data under control of the format control string and stores the result in buf and ends the processing of ap. Finally print the buffer value.

**Parameters**

| | |
|---|---|
| *fmt* | fmt is a constant character argument of type pointer. |

**Returns**

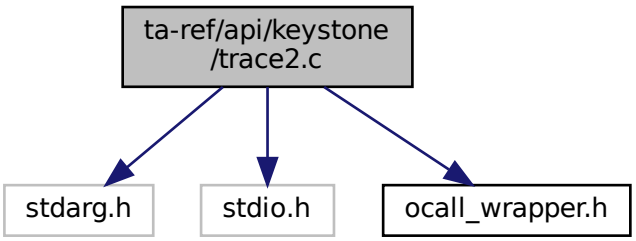
buffer If successfully executed, else error occured.

## 4.67 ta-ref/api/keystone/trace2.c File Reference

```
#include <stdarg.h>
#include <stdio.h>
```

```
#include "ocall_wrapper.h"
```

Include dependency graph for trace2.c:




### Functions

- static unsigned int _strlen (const char ∗str)
- int tee_printf (const char ∗fmt,...)

### 4.67.1 Function Documentation

#### 4.67.1.1 _strlen() `static unsigned int _strlen ( const char * str ) [inline], [static]`

_strlen() - calculate the length of characters in str.

**Parameters**

| | |
|---|---|
| *str* | str is argument of type pointer. |

**Returns**

string string length.

#### 4.67.1.2 tee_printf() `int tee_printf ( const char * fmt, ... )`

tee_printf() - For trace GP API.

Initializes ap variable. Formats data under control of the format control string and stores the result in buf and ends the processing of ap. Finally prints the buffer value.

**Parameters**

| | |
|---|---|
| *fmt* | fmt is constant character argument of type pointer. |

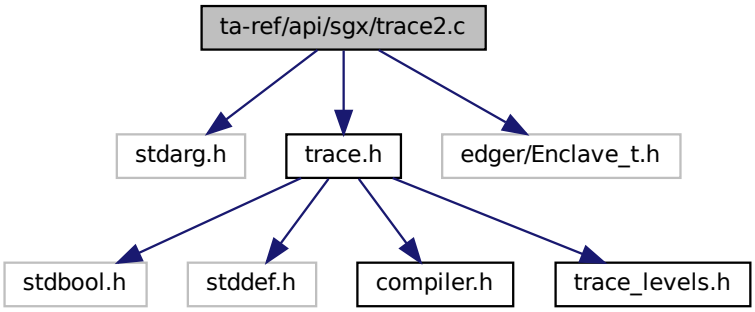
**Returns**

res Based on the condition check it will return string length else returns 0.

## 4.68 ta-ref/api/sgx/trace2.c File Reference

```
#include <stdarg.h>
#include "trace.h"
#include "edger/Enclave_t.h"
```

Include dependency graph for trace2.c:




### Functions

- void trace_vprintf (const char *func, int line, int level, bool level_ok, const char *fmt, va_list ap)
- void trace_printf (const char *func, int line, int level, bool level_ok, const char *fmt,...)

### 4.68.1 Function Documentation

#### 4.68.1.1 trace_printf()

```
void trace_printf (
        const char * func,
        int line,
        int level,
        bool level_ok,
        const char * fmt,
         ...  )
```

trace_printf() - Prints the formatted data to stdout.

This function returns the value of ap by calling va_end().

**Parameters**

| | |
|---|---|
| *func* | Pointer to a buffer where the resulting C-string is stored. |
| *line* | integer type of line |
| *level_ok* | boolen value |
| *fmt* | C string that contains a format string |
| *ap* | A value identifying a variable arguments list |

**Returns**

Total number of characters is returned.

#### 4.68.1.2 trace_vprintf()

```
void trace_vprintf (
        const char * func,
        int line,
        int level,
        bool level_ok,
        const char * fmt,
        va_list ap )
```

trace_vprintf() - Writes the formatted data from variable argument list to sized buffer.

This function returns the buffer character by calling ocall_print_string()

**Parameters**

| | |
|---|---|
| *func* | Pointer to a buffer where the resulting C-string is stored. |
| *line* | integer type of line |
| *level_ok* | boolen value |
| *fmt* | C string that contains a format string |
| *ap* | A value identifying a variable arguments list |

**Returns**

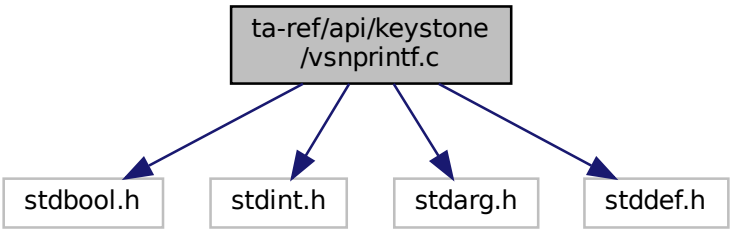
buf The total number of characters written is returned.

## 4.69 ta-ref/api/keystone/vsnprintf.c File Reference

```
#include <stdbool.h>
#include <stdint.h>
#include <stdarg.h>
```

```
#include <stddef.h>
```

Include dependency graph for vsnprintf.c:




### Classes

- struct out_fct_wrap_type

### Typedefs

- typedef void(∗ out_fct_type) (char character, void ∗buffer, size_t idx, size_t maxlen)

### Functions

- static void _out_buffer (char character, void ∗buffer, size_t idx, size_t maxlen)
- static void _out_null (char character, void ∗buffer, size_t idx, size_t maxlen)
- static void _out_char (char character, void ∗buffer, size_t idx, size_t maxlen)
- static void _out_fct (char character, void ∗buffer, size_t idx, size_t maxlen)
- static unsigned int _strlen (const char ∗str)
- static bool _is_digit (char ch)
- static unsigned int _atoi (const char ∗∗str)
- static size_t _ntoa_format (out_fct_type out, char ∗buffer, size_t idx, size_t maxlen, char ∗buf, size_t len, bool negative, unsigned int base, unsigned int prec, unsigned int width, unsigned int flags)
- static size_t _ntoa_long (out_fct_type out, char ∗buffer, size_t idx, size_t maxlen, unsigned long value, bool negative, unsigned long base, unsigned int prec, unsigned int width, unsigned int flags)
- static int _vsnprintf (out_fct_type out, char ∗buffer, const size_t maxlen, const char ∗format, va_list va)
- int sprintf (char ∗buffer, const char ∗format,...)
- int snprintf (char ∗buffer, size_t count, const char ∗format,...)
- int vsnprintf (char ∗buffer, size_t count, const char ∗format, va_list va)
- int fctprintf (void(∗out)(char character, void ∗arg), void ∗arg, const char ∗format,...)

#### 4.69.1 Typedef Documentation

##### 4.69.1.1 out_fct_type `typedef void(* out_fct_type) (char character, void *buffer, size_t idx, size_t maxlen)`

### 4.69.2 Function Documentation

#### 4.69.2.1 _atoi()

```
static unsigned int _atoi (
    const char ** str )  [static]
```

#### 4.69.2.2 _is_digit()

```
static bool _is_digit (
    char ch )  [inline], [static]
```

#### 4.69.2.3 _ntoa_format()

```
static size_t _ntoa_format (
    out_fct_type out,
    char * buffer,
    size_t idx,
    size_t maxlen,
    char * buf,
    size_t len,
    bool negative,
    unsigned int base,
    unsigned int prec,
    unsigned int width,
    unsigned int flags )  [static]
```

#### 4.69.2.4 _ntoa_long()

```
static size_t _ntoa_long (
    out_fct_type out,
    char * buffer,
    size_t idx,
    size_t maxlen,
    unsigned long value,
    bool negative,
    unsigned long base,
    unsigned int prec,
    unsigned int width,
    unsigned int flags )  [static]
```

#### 4.69.2.5 _out_buffer()

```
static void _out_buffer (
    char character,
    void * buffer,
    size_t idx,
    size_t maxlen )  [inline], [static]
```

#### 4.69.2.6 _out_char()

```
static void _out_char (
    char character,
    void * buffer,
    size_t idx,
    size_t maxlen )  [inline], [static]
```

#### 4.69.2.7 _out_fct()

```
static void _out_fct (
    char character,
    void * buffer,
    size_t idx,
    size_t maxlen )  [inline], [static]
```

#### 4.69.2.8 _out_null()

```
static void _out_null (
    char character,
    void * buffer,
    size_t idx,
    size_t maxlen )  [inline], [static]
```

#### 4.69.2.9 _strlen()

```
static unsigned int _strlen (
    const char * str )  [inline], [static]
```

#### 4.69.2.10 _vsnprintf()

```
static int _vsnprintf (
    out_fct_type out,
    char * buffer,
    const size_t maxlen,
    const char * format,
    va_list va )  [static]
```

#### 4.69.2.11 fctprintf()

```
int fctprintf (
    void(*)(char character, void *arg) out,
    void * arg,
    const char * format,
     ...  )
```

#### 4.69.2.12 snprintf()

```
int snprintf (
    char * buffer,
    size_t count,
    const char * format,
     ...  )
```

#### 4.69.2.13 sprintf()

```
int sprintf (
    char * buffer,
    const char * format,
    ... )
```

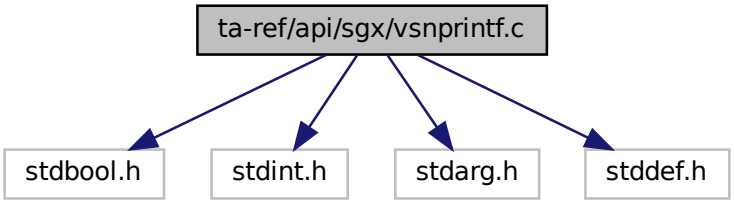
#### 4.69.2.14 vsnprintf()

```
int vsnprintf (
    char * buffer,
    size_t count,
    const char * format,
    va_list va )
```

## 4.70 ta-ref/api/sgx/vsnprintf.c File Reference

```
#include <stdbool.h>
#include <stdint.h>
#include <stdarg.h>
#include <stddef.h>
```

Include dependency graph for vsnprintf.c:




### Classes

- struct out_fct_wrap_type

### Macros

- #define PRINTF_NTOA_BUFFER_SIZE 32U
- #define PRINTF_FTOA_BUFFER_SIZE 32U
- #define PRINTF_SUPPORT_FLOAT
- #define PRINTF_SUPPORT_LONG_LONG
- #define PRINTF_SUPPORT_PTRDIFF_T
- #define FLAGS_ZEROPAD (1U << 0U)
- #define FLAGS_LEFT (1U << 1U)
- #define FLAGS_PLUS (1U << 2U)
- #define FLAGS_SPACE (1U << 3U)
- #define FLAGS_HASH (1U << 4U)

- #define FLAGS_UPPERCASE (1U << 5U)
- #define FLAGS_CHAR (1U << 6U)
- #define FLAGS_SHORT (1U << 7U)
- #define FLAGS_LONG (1U << 8U)
- #define FLAGS_LONG_LONG (1U << 9U)
- #define FLAGS_PRECISION (1U << 10U)
- #define _putchar putchar

### Typedefs

- typedef void(* out_fct_type) (char character, void *buffer, size_t idx, size_t maxlen)

### Functions

- int putchar (char ch)
- static void _out_buffer (char character, void *buffer, size_t idx, size_t maxlen)
- static void _out_null (char character, void *buffer, size_t idx, size_t maxlen)
- static void _out_char (char character, void *buffer, size_t idx, size_t maxlen)
- static void _out_fct (char character, void *buffer, size_t idx, size_t maxlen)
- static unsigned int _strlen (const char *str)
- static bool _is_digit (char ch)
- static unsigned int _atoi (const char **str)
- static size_t _ntoa_format (out_fct_type out, char *buffer, size_t idx, size_t maxlen, char *buf, size_t len, bool negative, unsigned int base, unsigned int prec, unsigned int width, unsigned int flags)
- static size_t _ntoa_long (out_fct_type out, char *buffer, size_t idx, size_t maxlen, unsigned long value, bool negative, unsigned long base, unsigned int prec, unsigned int width, unsigned int flags)
- static size_t _ntoa_long_long (out_fct_type out, char *buffer, size_t idx, size_t maxlen, unsigned long long value, bool negative, unsigned long long base, unsigned int prec, unsigned int width, unsigned int flags)
- static size_t _ftoa (out_fct_type out, char *buffer, size_t idx, size_t maxlen, double value, unsigned int prec, unsigned int width, unsigned int flags)
- static int _vsnprintf (out_fct_type out, char *buffer, const size_t maxlen, const char *format, va_list va)
- int sprintf (char *buffer, const char *format,...)
- int snprintf (char *buffer, size_t count, const char *format,...)
- int vsnprintf (char *buffer, size_t count, const char *format, va_list va)
- int fctprintf (void(*out)(char character, void *arg), void *arg, const char *format,...)

#### 4.70.1 Macro Definition Documentation

##### 4.70.1.1 _putchar `#define _putchar putchar`

##### 4.70.1.2 FLAGS_CHAR `#define FLAGS_CHAR (1U << 6U)`

#### 4.70.1.3 FLAGS_HASH

```
#define FLAGS_HASH (1U << 4U)
```

#### 4.70.1.4 FLAGS_LEFT

```
#define FLAGS_LEFT (1U << 1U)
```

#### 4.70.1.5 FLAGS_LONG

```
#define FLAGS_LONG (1U << 8U)
```

#### 4.70.1.6 FLAGS_LONG_LONG

```
#define FLAGS_LONG_LONG (1U << 9U)
```

#### 4.70.1.7 FLAGS_PLUS

```
#define FLAGS_PLUS (1U << 2U)
```

#### 4.70.1.8 FLAGS_PRECISION

```
#define FLAGS_PRECISION (1U << 10U)
```

#### 4.70.1.9 FLAGS_SHORT

```
#define FLAGS_SHORT (1U << 7U)
```

#### 4.70.1.10 FLAGS_SPACE

```
#define FLAGS_SPACE (1U << 3U)
```

#### 4.70.1.11 FLAGS_UPPERCASE

```
#define FLAGS_UPPERCASE (1U << 5U)
```

#### 4.70.1.12 FLAGS_ZEROPAD

```
#define FLAGS_ZEROPAD (1U << 0U)
```

#### 4.70.1.13 PRINTF_FTOA_BUFFER_SIZE

```
#define PRINTF_FTOA_BUFFER_SIZE 32U
```

#### 4.70.1.14 PRINTF_NTOA_BUFFER_SIZE `#define PRINTF_NTOA_BUFFER_SIZE 32U`

#### 4.70.1.15 PRINTF_SUPPORT_FLOAT `#define PRINTF_SUPPORT_FLOAT`

#### 4.70.1.16 PRINTF_SUPPORT_LONG_LONG `#define PRINTF_SUPPORT_LONG_LONG`

#### 4.70.1.17 PRINTF_SUPPORT_PTRDIFF_T `#define PRINTF_SUPPORT_PTRDIFF_T`

### 4.70.2 Typedef Documentation

#### 4.70.2.1 out_fct_type `typedef void(* out_fct_type) (char character, void *buffer, size_t idx, size_t maxlen)`

### 4.70.3 Function Documentation

#### 4.70.3.1 _atoi() `static unsigned int _atoi ( const char ** str ) [static]`

_atoi() - Converts the internal ASCII string into an unsigned integer.

This function is to convert the internal ASCII string into unsigned integer.

**Parameters**

| | |
|---|---|
| *str* | string representation of an integral number. |

**Returns**

i unsigned integer value.

#### 4.70.3.2 _ftoa()

```
static size_t _ftoa (
    out_fct_type out,
    char * buffer,
    size_t idx,
    size_t maxlen,
    double value,
    unsigned int prec,
    unsigned int width,
    unsigned int flags )  [static]
```

_ftoa() - Converts a given floating-point number or a double to a string with the use of standard library functions.

This function checks whether the value is negative or not, then it checks with if condtion default precision to 6, if it not set it will set explicitly. Using the while loop it limits the precision to 9,because it causes a overflow error when precision crosses above 10. Using the if condition rollover or round If the precsion value is greater than 0.5 up the precision value.it round up to

1. Using the while_loop condition adding extra zeros and append decimal value to the lenghth. Finally using the conditional statement executes pad leading zeros, handling the hash value, padding spaces up to given width and reverses the string.

**Parameters**

| | |
|---|---|
| *out* | type of out_fct_type |
| *buffer* | Pointer to a character string to write the result. |
| *idx* | idx bytes of size_t |
| *maxlen* | Maximum number of characters to write. |
| *negative* | boolean type |
| *base* | an unsigned long data type |
| *prec* | an unsigned integral data type |
| *width* | an unsigned integral data type |
| *flags* | an unsigned integral data type |

**Returns**

non integer value if success else error occur

#### 4.70.3.3 _is_digit()

```
static bool _is_digit (
    char ch )  [inline], [static]
```

_is_digit() - Is for the internal test if char is a digit from 0 to 9

**Parameters**

| | |
|---|---|
| *ch* | This is the character to be checked. |

**Returns**

true if char is a digit and internal test if char is a digit from 0 to 9

#### 4.70.3.4 _ntoa_format()

```
static size_t _ntoa_format (
    out_fct_type out,
    char * buffer,
    size_t idx,
    size_t maxlen,
    char * buf,
    size_t len,
    bool negative,
    unsigned int base,
    unsigned int prec,
    unsigned int width,
    unsigned int flags )  [static]
```

_ntoa_format() - Converts the string into the defined format structure.

This function uses the while condition for padding the leading zeroes and also applies the if conditions to handle the hash. Using the if condtion pad spaces up to given width what specifies in that. It reverse the string and again append pad spaces up to given width.

**Parameters**

| | |
|---|---|
| *out* | type of out_fct_type |
| *buffer* | Pointer to a character string to write the result. |
| *idx* | idx bytes of size_t |
| *maxlen* | Maximum number of characters to write. |
| *negative* | boolean type |
| *base* | an unsigned long data type |
| *prec* | an unsigned integer data type |
| *width* | an unsigned integer data type |
| *flags* | an unsigned integer data type |

**Returns**

idx non integer value if success else error occur.

#### 4.70.3.5 _ntoa_long()

```
static size_t _ntoa_long (
    out_fct_type out,
    char * buffer,
    size_t idx,
    size_t maxlen,
    unsigned long value,
    bool negative,
    unsigned long base,
```

```
unsigned int prec,
unsigned int width,
unsigned int flags )  [static]
```

_ntoa_long() - Converts string into long value.

This function begins with an if condition value then it assigns ∼FLAGS_HASH into flags & value. Later it uses the if condition and do while write if precision not equal to zero and value is not equals to zero.

**Parameters**

| | |
|---|---|
| *out* | type of out_fct_type |
| *buffer* | Pointer to a character string to write the result. |
| *id* | idx bytes of size_t |
| *maxlen* | Maximum number of characters to write. |
| *negative* | boolean type |
| *base* | an unsigned long data type |
| *prec* | an unsigned integral data type |
| *width* | an unsigned integral data type |
| *flags* | an unsigned integral data type |

**Returns**

idx non integer value if success else error occur.

#### 4.70.3.6 _ntoa_long_long() static size_t _ntoa_long_long (

```
out_fct_type out,
char * buffer,
size_t idx,
size_t maxlen,
unsigned long long value,
bool negative,
unsigned long long base,
unsigned int prec,
unsigned int width,
unsigned int flags )  [static]
```

_ntoa_long_long() - Function to convert string to long value.

This function begins with an if condition then it assigns ∼FLAGS_HASH into flags & value. Later it uses the if condition and do while write if precision not equal to zero and value is not equals to zero.

**Parameters**

| | |
|---|---|
| *out* | type of out_fct_type |
| *buffer* | Pointer to a character string to write the result. |
| *idx* | idx bytes of size_t |
| *maxlen* | Maximum number of characters to write. |
| | Paramter list continued on next page |

| | |
|---|---|
| *negative* | boolean type |
| *base* | an unsigned long data type |
| *prec* | an unsigned integral data type |
| *width* | an unsigned integral data type |
| *flag* | an unsigned integral data type |

**Returns**

idx non integer value if success else error occur.

#### 4.70.3.7 _out_buffer()

```
static void _out_buffer (
    char character,
    void * buffer,
    size_t idx,
    size_t maxlen )  [inline], [static]
```

_out_buffer() - Internal buffer output

This function compares the idx and maxlen, If "idx" is less than "maxlen" then it will assign "character" value into the typecasting char "buffer[idx]"

**Parameters**

| | |
|---|---|
| *character* | character type string |
| *buffer* | Pointer to a character string to write the result. |
| *idx* | bytes of size_t |
| *maxlen* | Maximum number of characters to write. |

#### 4.70.3.8 _out_char()

```
static void _out_char (
    char character,
    void * buffer,
    size_t idx,
    size_t maxlen )  [inline], [static]
```

_out_char() - Internal putchar wrapper

The typecasting of arguments with void is to avoid unused variable warnings in some compilers. Checks the character value once the if condtion is success then putchar() writes a character into stdout.

**Parameters**

| | |
|---|---|
| *character* | character type string |

Paramter list continued on next page

| | |
|---|---|
| *buffer* | Pointer to a character string to write the result. |
| *idx* | bytes of size_t |
| *maxlen* | Maximum number of characters to write. |

#### 4.70.3.9 _out_fct()

```
static void _out_fct (
    char character,
    void * buffer,
    size_t idx,
    size_t maxlen )  [inline], [static]
```

_out_fct() - Internal output function wrapper

This function typecasting idx and maxlen arguments is to avoid compiler error. And then output function wrapper and the buffer is the output fct pointer.

**Parameters**

| | |
|---|---|
| *character* | character type string |
| *buffer* | Pointer to a character string to write the result. |
| *idx* | bytes of size_t |
| *maxlen* | Maximum number of characters to write. |

#### 4.70.3.10 _out_null()

```
static void _out_null (
    char character,
    void * buffer,
    size_t idx,
    size_t maxlen )  [inline], [static]
```

_out_null() - Internal null output.

The typecasting of arguments with void is applied to avoid unused variable warnings in some compilers.

**Parameters**

| | |
|---|---|
| *character* | character type string |
| *buffer* | Pointer to a character string to write the result. |
| *idx* | bytes of size_t |
| *maxlen* | Maximum number of characters to write. |

#### 4.70.3.11 _strlen() static unsigned int _strlen ( const char * *str* ) [inline], [static]

_strlen() - calculates the length of the string.

**Parameters**

| | |
|---|---|
| *str* | str is an argument of type pointer. |

**Returns**

string length if successfully executed,else error occured.

#### 4.70.3.12 _vsnprintf() static int _vsnprintf ( out_fct_type *out*, char * *buffer*, const size_t *maxlen*, const char * *format*, va_list *va* ) [static]

_vsnprintf() - Function writes formatted output to a character array, up to a maximum number of characters.

The _vsnprintf fucntion firstly initializes the varibles of format specifers like flags, width, precsion in this they evaluate all the specifiers invidually.First it checks the buffer equal to zero or not for null output function. After that flags evaluation will start using the switch case, then width field evalucation take process using if condition.

**Parameters**

| | |
|---|---|
| *out* | type of out_fct_type. |
| *buffer* | pointer to the buffer where you want to function to store the formatted string. |
| *maxlen* | maximum number of characters to store in the buffer. |
| *format* | string that specifies the format of the output. |
| *va* | variable-argument list of the additional argument. |

**Returns**

Its return the typecasted int of idx if success otherwise error occured.

#### 4.70.3.13 fctprintf() int fctprintf ( void(*)(char character, void *arg) *out*, void * *arg*, const char * *format*, ... )

fctprintf() - Function is using the libary macros of variable aruguments like vastart and vaend.

This function initializes the va_list variable and invokes the va_start(). Invokes _vsnprintf function and stores the value into ret. It applies the functions va_start and va_end on va and returns ret.

**Parameters**

| | |
|---|---|
| *out* | An output function which takes one character and an argument pointer. |
| *arg* | An argument pointer for user data passed to output function. |
| *format* | A string that specifies the format of the output. |

**Returns**

The number of characters that are sent to the output function, not counting the terminating null character.

#### 4.70.3.14 putchar()

```
int putchar (
    char ch )
```

#### 4.70.3.15 snprintf()

```
int snprintf (
    char * buffer,
    size_t count,
    const char * format,
     ... )
```

snprintf() - Places the generated output into the character array pointed to by buf, instead of writing it to a file

This function initializes the va_list variable and invokes the va_start(). Invokes _vsnprintf function and stores the value into ret. It applies the functions va_start and va_end on va and returns ret.

**Parameters**

| | |
|---|---|
| *buffer* | pointer to buffer where you want to function to store the formatted string. |
| *count* | maximum number of characters to store in the buffer. |
| *format* | string that specifies the format of the output. |

**Returns**

ret returns the ret value as an integer type.

#### 4.70.3.16 sprintf()

```
int sprintf (
    char * buffer,
    const char * format,
     ... )
```

sprintf() - Sends formatted output to a string pointed to by the argument buffer.

This function initialize the va_list variable and invokes the va_start(). Invokes _vsnprintf function and store the value into ret. It applies the functions va_start and va_end on va and returns ret.

**Parameters**

| | |
|---|---|
| *buffer* | pointer to an array of char elements resulting string will store. |
| *format* | string that contains the text to be written to buffer. |

**Returns**

ret It returns the ret value as an integer type.

#### 4.70.3.17 vsnprintf()

```
int vsnprintf (
    char * buffer,
    size_t count,
    const char * format,
    va_list va )
```

vsnprintf() - Invokes another function called _vsnprintf(). with some arguments.

**Parameters**

| | |
|---|---|
| *buffer* | Pointer to the buffer where you want to function to store the formatted string. |
| *count* | maximum number of characters to store in the buffer. |
| *format* | string that specifies the format of the output. |

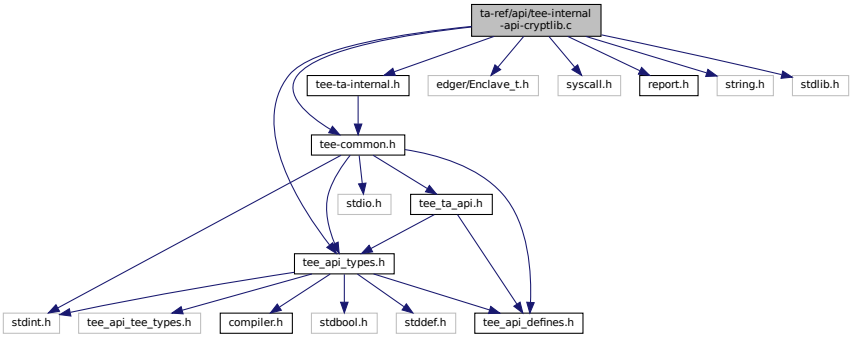
**Returns**

Its return the typecasted int of idx if success otherwise error occured.

## 4.71 ta-ref/api/tee-internal-api-cryptlib.c File Reference

```
#include "tee_api_types.h"
#include "tee-common.h"
#include "tee-ta-internal.h"
#include "edger/Enclave_t.h"
#include "syscall.h"
#include "report.h"
#include <string.h>
#include <stdlib.h>
```

Include dependency graph for tee-internal-api-cryptlib.c:




## Functions

- void wolfSSL_Free (void *p)
- void * wolfSSL_Malloc (size_t n)
- TEE_Result TEE_AllocateOperation (TEE_OperationHandle *operation, uint32_t algorithm, uint32_t mode, uint32_t maxKeySize)

  *Crypto, for all Crypto Functions.*
- void TEE_FreeOperation (TEE_OperationHandle operation)

  *Crypto, for all Crypto Functions.*
- void TEE_DigestUpdate (TEE_OperationHandle operation, const void *chunk, uint32_t chunkSize)

  *Crypto, Message Digest Functions.*
- TEE_Result TEE_DigestDoFinal (TEE_OperationHandle operation, const void *chunk, uint32_t chunkLen, void *hash, uint32_t *hashLen)
- TEE_Result TEE_SetOperationKey (TEE_OperationHandle operation, TEE_ObjectHandle key)

  *Crypto, Authenticated Encryption with Symmetric key Verification Functions.*
- TEE_Result TEE_AEInit (TEE_OperationHandle operation, const void *nonce, uint32_t nonceLen, uint32_t tagLen, uint32_t AADLen, uint32_t payloadLen)

  *Crypto, Authenticated Encryption with Symmetric key Verification Functions.*
- void TEE_AEUpdateAAD (TEE_OperationHandle operation, const void *AADdata, uint32_t AADdataLen)

  *Crypto, Authenticated Encryption with Symmetric key Verification Functions.*
- TEE_Result TEE_AEUpdate (TEE_OperationHandle operation, const void *srcData, uint32_t srcLen, void *destData, uint32_t *destLen)

  *Crypto, Authenticated Encryption with Symmetric key Verification Functions.*
- TEE_Result TEE_AEEncryptFinal (TEE_OperationHandle operation, const void *srcData, uint32_t srcLen, void *destData, uint32_t *destLen, void *tag, uint32_t *tagLen)

  *Crypto, Authenticated Encryption with Symmetric key Verification Functions.*
- TEE_Result TEE_AEDecryptFinal (TEE_OperationHandle operation, const void *srcData, uint32_t srcLen, void *destData, uint32_t *destLen, void *tag, uint32_t tagLen)

  *Crypto, Authenticated Encryption with Symmetric key Verification Functions.*
- void TEE_CipherInit (TEE_OperationHandle operation, const void *nonce, uint32_t nonceLen)

  *Crypto, Authenticated Encryption with Symmetric key Verification Functions.*
- TEE_Result TEE_CipherUpdate (TEE_OperationHandle operation, const void *srcData, uint32_t srcLen, void *destData, uint32_t *destLen)

  *Crypto, Authenticated Encryption with Symmetric key Verification Functions.*
- TEE_Result TEE_CipherDoFinal (TEE_OperationHandle operation, const void *srcData, uint32_t srcLen, void *destData, uint32_t *destLen)

- TEE_Result TEE_GenerateKey (TEE_ObjectHandle object, uint32_t keySize, const TEE_Attribute ∗params, uint32_t paramCount)

  *Crypto, Asymmetric key Verification Functions.*
- TEE_Result TEE_AllocateTransientObject (TEE_ObjectType objectType, uint32_t maxKeySize, TEE_ObjectHandle ∗object)

  *Crypto, Asymmetric key Verification Functions.*
- void TEE_InitRefAttribute (TEE_Attribute ∗attr, uint32_t attributeID, const void ∗buffer, uint32_t length)

  *Crypto, Asymmetric key Verification Functions.*
- void TEE_InitValueAttribute (TEE_Attribute ∗attr, uint32_t attributeID, uint32_t a, uint32_t b)

  *Crypto, Asymmetric key Verification Functions.*
- void TEE_FreeTransientObject (TEE_ObjectHandle object)

  *Crypto, Asymmetric key Verification Functions.*
- TEE_Result TEE_AsymmetricSignDigest (TEE_OperationHandle operation, const TEE_Attribute ∗params, uint32_t paramCount, const void ∗digest, uint32_t digestLen, void ∗signature, uint32_t ∗signatureLen)

  *Crypto, Asymmetric key Verification Functions.*
- TEE_Result TEE_AsymmetricVerifyDigest (TEE_OperationHandle operation, const TEE_Attribute ∗params, uint32_t paramCount, const void ∗digest, uint32_t digestLen, const void ∗signature, uint32_t signatureLen)

  *Crypto, Asymmetric key Verification Functions.*

### 4.71.1 Function Documentation

#### 4.71.1.1 TEE_AEDecryptFinal()

```
TEE_Result TEE_AEDecryptFinal (
    TEE_OperationHandle operation,
    const void * srcData,
    uint32_t srcLen,
    void * destData,
    uint32_t * destLen,
    void * tag,
    uint32_t tagLen )
```

Crypto, Authenticated Encryption with Symmetric key Verification Functions.

TEE_AEDecryptFinal() - Processes data that has not been processed by previous calls to TEE_AEUpdate as well as data supplied in srcData.

This function completes the AE operation and compares the computed tag with the tag supplied in the parameter tag .The operation handle can be reused or newly initialized.The buffers srcData and destData shall be either completely disjoint or equal in their starting positions.The operation may be in either initial or active state and enters initial state afterwards.

**Parameters**

| | |
|---|---|
| *operation* | Handle of a running AE operation |
| *srcData* | Reference to final chunk of input data to be encrypted |
| *srcLen* | length of the input data |
| *destData* | Output buffer. Can be omitted if the output is to be discarded. |
| *destLen* | length of the buffer. |
| *tag* | Output buffer filled with the computed tag |
| *tagLen* | length of the tag. |

**Returns**

0 on success.

TEE_ERROR_SHORT_BUFFER If the output buffer is not large enough to contain the output

TEE_ERROR_MAC_INVALID If the computed tag does not match the supplied tag

#### 4.71.1.2 TEE_AEEncryptFinal()

```
TEE_Result TEE_AEEncryptFinal (
        TEE_OperationHandle operation,
        const void * srcData,
        uint32_t srcLen,
        void * destData,
        uint32_t * destLen,
        void * tag,
        uint32_t * tagLen )
```

Crypto, Authenticated Encryption with Symmetric key Verification Functions.

TEE_AEEncryptFinal() - processes data that has not been processed by previous calls to TEE_AEUpdate as well as data supplied in srcData .

TEE_AEEncryptFinal completes the AE operation and computes the tag. The operation handle can be reused or newly initialized. The buffers srcData and destData SHALL be either completely disjoint or equal in their starting positions.The operation may be in either initial or active state and enters initial state afterwards.

**Parameters**

| | |
|---|---|
| *operation* | Handle of a running AE operation |
| *srcData* | Reference to final chunk of input data to be encrypted |
| *srcLen* | length of the input data |
| *destData* | Output buffer. Can be omitted if the output is to be discarded. |
| *destLen* | length of the buffer. |
| *tag* | Output buffer filled with the computed tag |
| *tagLen* | length of the tag. |

**Returns**

0 on success.

TEE_ERROR_SHORT_BUFFER If the output or tag buffer is not large enoughto contain the output.

#### 4.71.1.3 TEE_AEInit()

```
TEE_Result TEE_AEInit (
        TEE_OperationHandle operation,
        const void * nonce,
        uint32_t nonceLen,
        uint32_t tagLen,
        uint32_t AADLen,
        uint32_t payloadLen )
```

Crypto, Authenticated Encryption with Symmetric key Verification Functions.

TEE_AEInit() - Initializes an Authentication Encryption operation.

The operation must be in initial state and remains in the initial state afterwards.

**Parameters**

| | |
|---|---|
| *operation* | A handle on the operation. |
| *nonce* | The operation nonce or IV |
| *nonceLen* | length of nonce |
| *tagLen* | Size in bits of the tag |
| *AADLen* | Length in bytes of the AAD |
| *payloadLen* | Length in bytes of the payload. |

**Returns**

0 on success.

TEE_ERROR_NOT_SUPPORTED If the tag length is not supported by the algorithm.

#### 4.71.1.4 TEE_AEUpdate()

```
TEE_Result TEE_AEUpdate (
        TEE_OperationHandle operation,
        const void * srcData,
        uint32_t srcLen,
        void * destData,
        uint32_t * destLen )
```

Crypto, Authenticated Encryption with Symmetric key Verification Functions.

TEE_AEUpdate() - Accumulates data for an Authentication Encryption operation

This function describes Input data does not have to be a multiple of block size. Subsequent calls to this function are possible. Unless one or more calls of this function have supplied sufficient input data,no output is generated. when using this routine to decrypt the returned data may be corrupt since the integrity check is not performed until all the data has been processed. If this is a concern then only use the TEE_AEDecryptFinal routine.

**Parameters**

| | |
|---|---|
| *operation* | Handle of a running AE operation. |
| *srcData* | Input data buffer to be encrypted or decrypted |
| *srcLen* | length of the input buffer. |
| *destData* | Output buffer |
| *destLen* | length of the out put buffer. |

**Returns**

0 on success.

TEE_ERROR_SHORT_BUFFER if the output buffer is not large enough to contain the output.

#### 4.71.1.5 TEE_AEUpdateAAD()

```
void TEE_AEUpdateAAD (
        TEE_OperationHandle operation,
        const void * AADdata,
        uint32_t AADdataLen )
```

Crypto, Authenticated Encryption with Symmetric key Verification Functions.

TEE_AEUpdateAAD() - Feeds a new chunk of Additional Authentication Data (AAD) to the AE operation. Subsequent calls to this function are possible.

The TEE_AEUpdateAAD function feeds a new chunk of Additional Authentication Data (AAD) to the AE operation. Subsequent calls to this function are possible.The buffers srcData and destData shall be either completely disjoint or equal in their starting positions.The operation SHALL be in initial state and remains in initial state afterwards.

**Parameters**

| | |
|---|---|
| *operation* | Handle on the AE operation |
| *AADdata* | Input buffer containing the chunk of AAD |
| *AADdataLen* | length of the chunk of AAD. |

#### 4.71.1.6 TEE_AllocateOperation()

```
TEE_Result TEE_AllocateOperation (
        TEE_OperationHandle * operation,
        uint32_t algorithm,
        uint32_t mode,
        uint32_t maxKeySize )
```

Crypto, for all Crypto Functions.

TEE_AllocateOperation() - Allocates a handle for a new cryptographic operation and sets the mode and algorithm type.

If this function does not return with TEE_SUCCESS then there is no valid handle value.Once a cryptographic operation has been created, the implementation shall guarantee that all resources necessary for the operation are allocated and that any operation with a key of at most maxKeySize bits can be performed. For algorithms that take multiple keys, for example the AES XTS algorithm, the maxKeySize parameter specifies the size of the largest key. It is up to the implementation to properly allocate space for multiple keys if the algorithm so requires.

**Parameters**

| | |
|---|---|
| *operation* | reference to generated operation handle. |
| *algorithm* | One of the cipher algorithms. |
| *mode* | The operation mode. |
| *maxKeySize* | Maximum key size in bits for the operation. |

**Returns**

0 in case of success

TEE_ERROR_OUT_OF_MEMORY If there are not enough resources to allocate the operation.

TEE_ERROR_NOT_SUPPORTED If the mode is not compatible with the algorithm or key size or if the algorithm is not one of the listed algorithms or if maxKeySize is not appropriate for the algorithm.

#### 4.71.1.7 TEE_AllocateTransientObject()

```
TEE_Result TEE_AllocateTransientObject (
        TEE_ObjectType objectType,
        uint32_t maxKeySize,
        TEE_ObjectHandle * object )
```

Crypto, Asymmetric key Verification Functions.

TEE_AllocateTransientObject() - Allocates an uninitialized transient object. Transient objects are used to hold a cryptographic object (key or key-pair).

The value TEE_KEYSIZE_NO_KEY should be used for maxObjectSize for object types that do not require a key so that all the container resources can be pre-allocated. As allocated, the container is uninitialized. It can be initialized by subsequently importing the object material,generating an object, deriving an object, or loading an object from the Trusted Storage.

**Parameters**

| | |
|---|---|
| *objectType* | Type of uninitialized object container to be created |
| *maxKeySize* | Key Size of the object. |
| *object* | Filled with a handle on the newly created key container. |

**Returns**

0 on success

TEE_ERROR_OUT_OF_MEMORY If not enough resources are available to allocate the object handle.

TEE_ERROR_NOT_SUPPORTED If the key size is not supported or the object type is not supported.

#### 4.71.1.8 TEE_AsymmetricSignDigest()

```
TEE_Result TEE_AsymmetricSignDigest (
        TEE_OperationHandle operation,
        const TEE_Attribute * params,
        uint32_t paramCount,
        const void * digest,
        uint32_t digestLen,
        void * signature,
        uint32_t * signatureLen )
```

Crypto, Asymmetric key Verification Functions.

TEE_AsymmetricSignDigest() - Signs a message digest within an asymmetric operation.

**Parameters**

| | |
|---|---|
| *operation* | Handle on the operation, which SHALL have been suitably set up with an operation key. |
| *params* | Optional operation parameters |
| *paramCount* | size of param |
| *digest* | Input buffer containing the input message digest |
| *digestLen* | length of input buffer. |
| *signature* | Output buffer written with the signature of the digest |
| *signatureLen* | length of output buffer. |

**Returns**

0 on sccess

TEE_ERROR_SHORT_BUFFER If the signature buffer is not large enough to hold the result

#### 4.71.1.9 TEE_AsymmetricVerifyDigest()

```
TEE_Result TEE_AsymmetricVerifyDigest (
    TEE_OperationHandle operation,
    const TEE_Attribute * params,
    uint32_t paramCount,
    const void * digest,
    uint32_t digestLen,
    const void * signature,
    uint32_t signatureLen )
```

Crypto, Asymmetric key Verification Functions.

TEE_AsymmetricVerifyDigest() - verifies a message digest signature within an asymmetric operation.

This function describes the message digest signature verify by calling ed25519_verify().

**Parameters**

| | |
|---|---|
| *operation* | Handle on the operation, which SHALL have been suitably set up with an operation key. |
| *params* | Optional operation parameters |
| *paramCount* | size of param. |
| *digest* | Input buffer containing the input message digest |
| *digestLen* | length of input buffer. |
| *signature* | Output buffer written with the signature of the digest |
| *signatureLen* | length of output buffer. |

**Returns**

TEE_SUCCESS on success

TEE_ERROR_SIGNATURE_INVALID if the signature is invalid.

#### 4.71.1.10 TEE_CipherDoFinal()

```
TEE_Result TEE_CipherDoFinal (
    TEE_OperationHandle operation,
    const void * srcData,
    uint32_t srcLen,
    void * destData,
    uint32_t * destLen )
```

TEE_CipherDoFinal() - Finalizes the cipher operation, processing data that has not been processed by previous calls to TEE_CipherUpdate as well as data supplied in srcData .

This function describes The operation handle can be reused or re-initialized. The buffers srcData and destData shall be either completely disjoint or equal in their starting positions.The operation SHALL be in active state and is set to initial state afterwards.

**Parameters**

| | |
|---|---|
| *operation* | Handle of a running Cipher operation |
| *srcData* | Input data buffer to be encrypted or decrypted |
| *srcLen* | length of input buffer |
| *destData* | output buffer |
| *destLen* | ouput buffer length. |

**Returns**

0 on success

TEE_ERROR_SHORT_BUFFER If the output buffer is not large enough to contain the output

#### 4.71.1.11 TEE_CipherInit()

```
void TEE_CipherInit (
    TEE_OperationHandle operation,
    const void * nonce,
    uint32_t nonceLen )
```

Crypto, Authenticated Encryption with Symmetric key Verification Functions.

TEE_CipherInit() - starts the symmetric cipher operation.

The operation shall have been associated with a key. If the operation is in active state, it is reset and then initialized. If the operation is in initial state, it is moved to active state.

**Parameters**

| | |
|---|---|
| *operation* | A handle on an opened cipher operation setup with a key |
| *nonce* | Buffer containing the operation Initialization Vector as appropriate. |
| *nonceLen* | length of the buffer |

#### 4.71.1.12 TEE_CipherUpdate()

```
TEE_Result TEE_CipherUpdate (
    TEE_OperationHandle operation,
    const void * srcData,
    uint32_t srcLen,
    void * destData,
    uint32_t * destLen )
```

Crypto, Authenticated Encryption with Symmetric key Verification Functions.

TEE_CipherUpdate() - encrypts or decrypts input data.

Input data does not have to be a multiple of block size. Subsequent calls to this function are possible. Unless one or more calls of this function have supplied sufficient input data, no output is generated. The cipher operation is finalized with a call to TEE_CipherDoFinal .The buffers srcData and destData SHALL be either completely disjoint or equal in their starting positions.The operation SHALL be in active state.

**Parameters**

| | |
|---|---|
| *operation* | Handle of a running Cipher operation |
| *srcData* | Input data buffer to be encrypted or decrypted |
| *srcLen* | length of input buffer |
| *destData* | output buffer |
| *destLen* | ouput buffer length. |

**Returns**

0 on success else

TEE_ERROR_SHORT_BUFFER If the output buffer is not large enough to contain the output. In this case, the input is not fed into the algorithm.

#### 4.71.1.13 TEE_DigestDoFinal()

```
TEE_Result TEE_DigestDoFinal (
    TEE_OperationHandle operation,
    const void * chunk,
    uint32_t chunkLen,
    void * hash,
    uint32_t * hashLen )
```

TEE_DigestDoFinal() - Finalizes the message digest operation and produces the message hash.

This function finalizes the message digest operation and produces the message hash. Afterwards the Message Digest operation is reset to initial state and can be reused.

**Parameters**

| | |
|---|---|
| *operation* | Handle of a running Message Digest operation. |
| *chunk* | Chunk of data to be hashed. |
| *chunkLen* | size of the chunk. |
| *hash* | Output buffer filled with the message hash. |
| *hashLen* | lenth of the mesaage hash. |

**Returns**

0 on success

TEE_ERROR_SHORT_BUFFER If the output buffer is too small. In this case, the operation is not finalized.

#### 4.71.1.14 TEE_DigestUpdate() `void TEE_DigestUpdate (`

```
TEE_OperationHandle operation,
const void * chunk,
uint32_t chunkSize )
```

Crypto, Message Digest Functions.

TEE_DigestUpdate()- Accumulates message data for hashing.

This function describes the message does not have to be block aligned. Subsequent calls to this function are possible.The operation may be in either initial or active state and becomes active.

**Parameters**

| | |
|---|---|
| *operation* | Handle of a running Message Digest operation. |
| *chunk* | Chunk of data to be hashed |
| *chunkSize* | size of the chunk. |

#### 4.71.1.15 TEE_FreeOperation() `void TEE_FreeOperation (`

```
TEE_OperationHandle operation )
```

Crypto, for all Crypto Functions.

TEE_FreeOperation() - Deallocates all resources associated with an operation handle.

This function deallocates all resources associated with an operation handle. After this function is called, the operation handle is no longer valid. All cryptographic material in the operation is destroyed. The function does nothing if operation is TEE_HANDLE_NULL.

**Parameters**

| | |
|---|---|
| *operation* | Reference to operation handle. |

**Returns**

nothing after the operation free.

#### 4.71.1.16 TEE_FreeTransientObject() `void TEE_FreeTransientObject (`

```
TEE_ObjectHandle object )
```

Crypto, Asymmetric key Verification Functions.

TEE_FreeTransientObject() - Deallocates a transient object previously allocated with TEE_AllocateTransientObject .

this function describes the object handle is no longer valid and all resources associated with the transient object shall have been reclaimed after the TEE_AllocateTransientObject() call.

**Parameters**

| | |
|---|---|
| *object* | Handle on the object to free. |

#### 4.71.1.17 TEE_GenerateKey() `TEE_Result TEE_GenerateKey (`

```
TEE_ObjectHandle object,
uint32_t keySize,
const TEE_Attribute * params,
uint32_t paramCount )
```

Crypto, Asymmetric key Verification Functions.

TEE_GenerateKey () - Generates a random key or a key-pair and populates a transient key object with the generated key material.

The size of the desired key is passed in the keySize parameter and shall be less than or equal to the maximum key size specified when the transient object was created.

**Parameters**

| | |
|---|---|
| *object* | Handle on an uninitialized transient key to populate with the generated key. |
| *keySize* | Requested key size shall be less than or equal to the maximum key size specified when the object container was created |
| *params* | Parameters for the key generation. |
| *paramCount* | The values of all parameters are copied nto the object so that the params array and all the memory buffers it points to may be freed after this routine returns without affecting the object. |

**Returns**

0 on succes

TEE_ERROR_BAD_PARAMETERS If an incorrect or inconsistent attribute is detected. The checks that are performed depend on the implementation.

#### 4.71.1.18 TEE_InitRefAttribute() `void TEE_InitRefAttribute (`

```
TEE_Attribute * attr,
uint32_t attributeID,
const void * buffer,
uint32_t length )
```

Crypto, Asymmetric key Verification Functions.

TEE_InitRefAttribute() - The helper function can be used to populate a single attribute either with a reference to a buffer or with integer values.

In TEE_InitRefAttribute () only the buffer pointer is copied, not the content of the buffer. This means that the attribute structure maintains a pointer back to the supplied buffer. It is the responsibility of the TA author to ensure that the contents of the buffer maintain their value until the attributes array is no longer in use.

**Parameters**

| | |
|---|---|
| *attr* | attribute structure to initialize. |
| *attributeID* | Identifier of the attribute to populate. |
| *buffer* | input buffer that holds the content of the attribute. |
| *length* | buffer length. |

#### 4.71.1.19 TEE_InitValueAttribute() `void TEE_InitValueAttribute (`

```
TEE_Attribute * attr,
uint32_t attributeID,
uint32_t a,
uint32_t b )
```

Crypto, Asymmetric key Verification Functions.

TEE_InitValueAttribute() - The helper function can be used to populate a single attribute either with a reference to a buffer or with integer values.

**Parameters**

| | |
|---|---|
| *attr* | attribute structure to initialize. |
| *attributeID* | Identifier of the attribute to populate. |
| *a* | unsigned integer value to assign to the a member of the attribute structure. |
| *b* | unsigned integer value to assign to the b member of the attribute structure |

#### 4.71.1.20 TEE_SetOperationKey() `TEE_Result TEE_SetOperationKey (`

```
TEE_OperationHandle operation,
TEE_ObjectHandle key )
```

Crypto, Authenticated Encryption with Symmetric key Verification Functions.

TEE_SetOperationKey() - Programs the key of an operation; that is, it associates an operation with a key.

The key material is copied from the key object handle into the operation. After the key has been set, there is no longer any link between the operation and the key object. The object handle can be closed or reset and this will not affect the operation. This copied material exists until the operation is freed using TEE_FreeOperation or another key is set into the operation.

**Parameters**

| | |
|---|---|
| *operation* | Operation handle. |
| *key* | A handle on a key object. |

**Returns**

0 on success return

TEE_ERROR_CORRUPT_OBJECT If the object is corrupt. The object handle is closed.

TEE_ERROR_STORAGE_NOT_AVAILABLE If the persistent object is stored in a storage area which is currently inaccessible.

#### 4.71.1.21 wolfSSL_Free() `void wolfSSL_Free ( void * p )`

wolfSSL_Free() - Deallocates the memory which allocated previously.

**Parameters**

| | |
|---|---|
| *p* | This is the pointer to a memory block. |

#### 4.71.1.22 wolfSSL_Malloc() `void * wolfSSL_Malloc ( size_t n )`

wolfSSL_Malloc() - Allocates the requested memory and returns a pointer to it.

**Parameters**

| | |
|---|---|
| *n* | size of the memory block. |

# Index