1 Main Page

QTEE TA Software Developers Kit

User Guide

80-PF777-58 A

January 23, 2025

Qualcomm Technologies, Inc.
5775 Morehouse Drive
San Diego, CA 92121
U.S.A.

2 Revision History

Revision History

Revision	Date	Description
A	May 2019	Initial version of document.

Contents

List of Figures

List of Tables

3 Introduction

3.1 Purpose

The software that runs within the ARM TrustZone (TZ) environment on Qualcomm Technologies, Inc. (QTI) chipsets, is known as the Qualcomm Trusted Execution Environment (QTEE). This document describes the QTEE architecture, feature set, and extensions that are added to QTEE and provides details on how to write trusted applications on top of QTEE.

Certain features are part of the hardware-supported TZ architecture to provide system security configuration and allow the customer to tailor parts of this configuration to meet their requirements. For more support, see Section 3.3. This document is applicable to SDM845 and any future chipsets that use QTEE 5.0 architecture.

3.2 Conventions

Function declarations, function names, type declarations, and code samples appear in a different font. For example, `#include`.

Code variables appear in angle brackets. For example, <number>.

Commands and command variables appear in a different font. For example, **{copy a:*.* b:}**

3.3 Technical Assistance

For assistance or clarification on information in this document, submit a case to Qualcomm Technologies, Inc. (QTI) at https://createpoint.qti.qualcomm.com/.

If you do not have access to the CDMATech Support website, register for access or send email to support.cdmatech@qti.qualcomm.com

4 QTEE Overview

4.1 Hardware

The TZ technology was pioneered by ARM in their v6 architecture and was redesigned and formalized for ARMv7. In ARMv8, the TZ technology is further extended and enhanced, and has made security an integral part of the architecture. This technology provides a security framework that enables a device to counter many security threats from both a software and hardware level.

The hardware solution provided by ARM enables the design and implementation, in software, of applications or services that run in a trusted environment. This trusted environment is an isolated execution unit that establishes hardware separation from rich execution environments. TZ software is enabled on all MSM8960 and later chipsets.

4.2 Software

QTEE is the collection of privileged and nonprivileged software that runs in the QTI's trusted execution environment. On cold boot, QTEE is responsible for performing secure configuration of the SoC.

The QTEE privileged kernel offers run-time services to the REE, such as power collapse, secure PIL, content protection. As well, the kernel supports the execution of trusted nonprivileged user applications within the TZ environment. QTEE runs from pIMEM or potentially OCIMEM on high tier MSM/ APQ/MPQ chipsets and from DDR on MDMs and mid/low tier MSM/APQ/MPQ chipsets.

4.3 Architecture

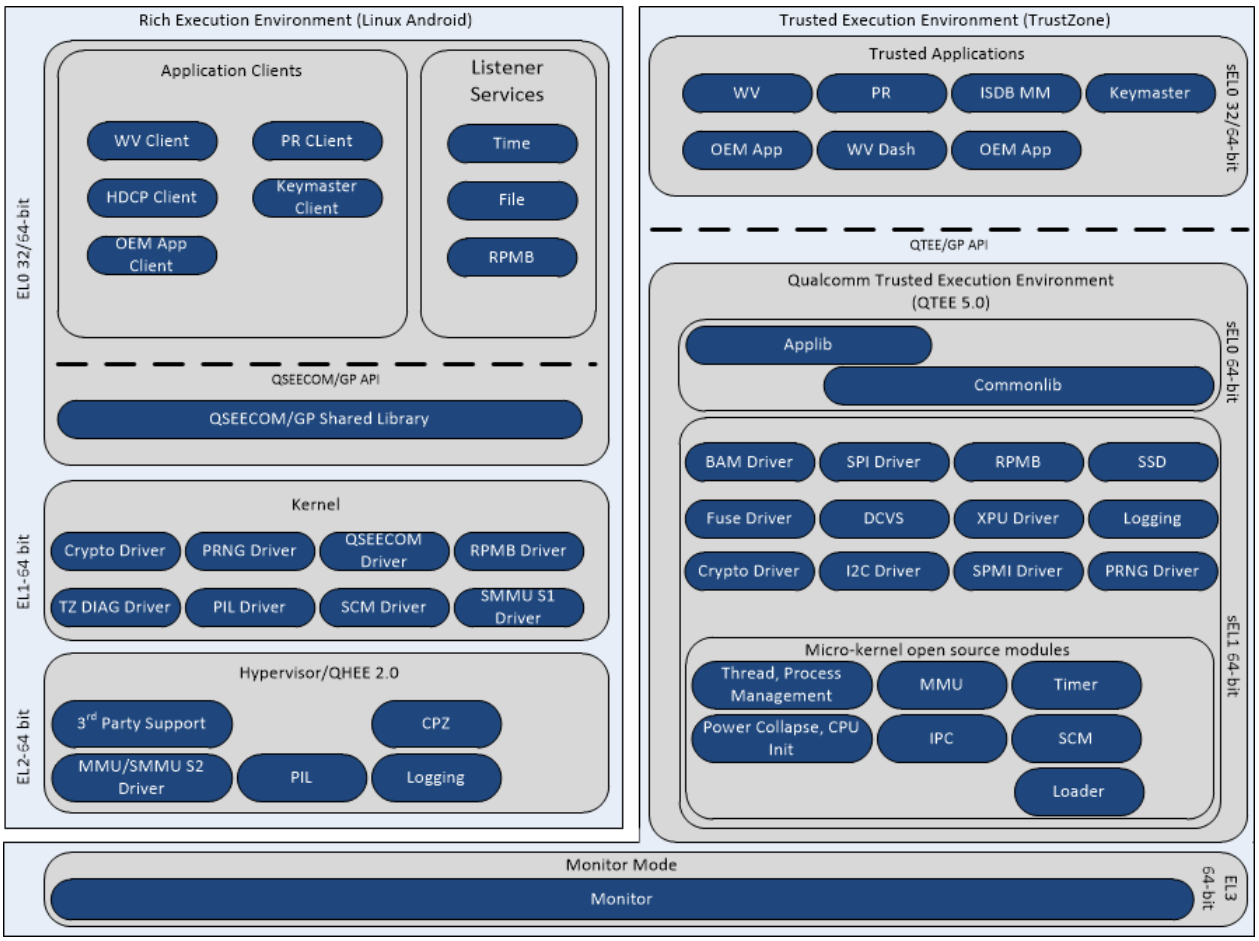


Figure 4-1 QTEE 5.0 architecture for premium-tier chipsets

4.4 Feature Set

QTEE 5.0 supports the following features:

- Access control - Support for three security domains
- QTEE runs from pIMEM or DDR
- Power collapse of security blocks
- Secure peripheral image loader (PIL)
- Subsystem restart
- Access control for OCIMEM
- Content protection
- Rollback protection of subsystem images
- Support for running trusted applications (both 32 bits and 64 bits)
- Sandboxing for trusted applications

- Position-independent trusted application loading
- Message passing between trusted applications
- Secure file system
- Rollback protection of secure file system
- Fuse management
- MMU enablement
- Enabling debug on secure boot-enabled devices based on chip serial number

5 Mini-Kernel (MINK)

5.1 MINK Architecture

5.1.1 Overview

In a MINK-based system, software executes in one or more communicating domains. Domains differ in their ability to access memory and other system resources.

One of these domains executes at sEL1, and is referred to as the kernel domain. The kernel domain has complete reign over the resources of the system.

Other domains, called user domains or processes, execute at sEL0, and have restricted access to system resources. MINK allows precise control over what is granted to a process.

Communication between the domains is enabled by the IPC primitives of MINK, which can be described briefly as an object capability-based synchronous message passing facility. It allows code executing in one domain to invoke objects running in other domains.

Figure 5-1 illustrates multiple execution domains. Within each domain are drawn objects that are "owned" by the respective domain, that is, the data of the object is manipulated only by the software executing in that domain. The connecting arrows represent object references, which allow a domain to invoke the referenced object in another domain.

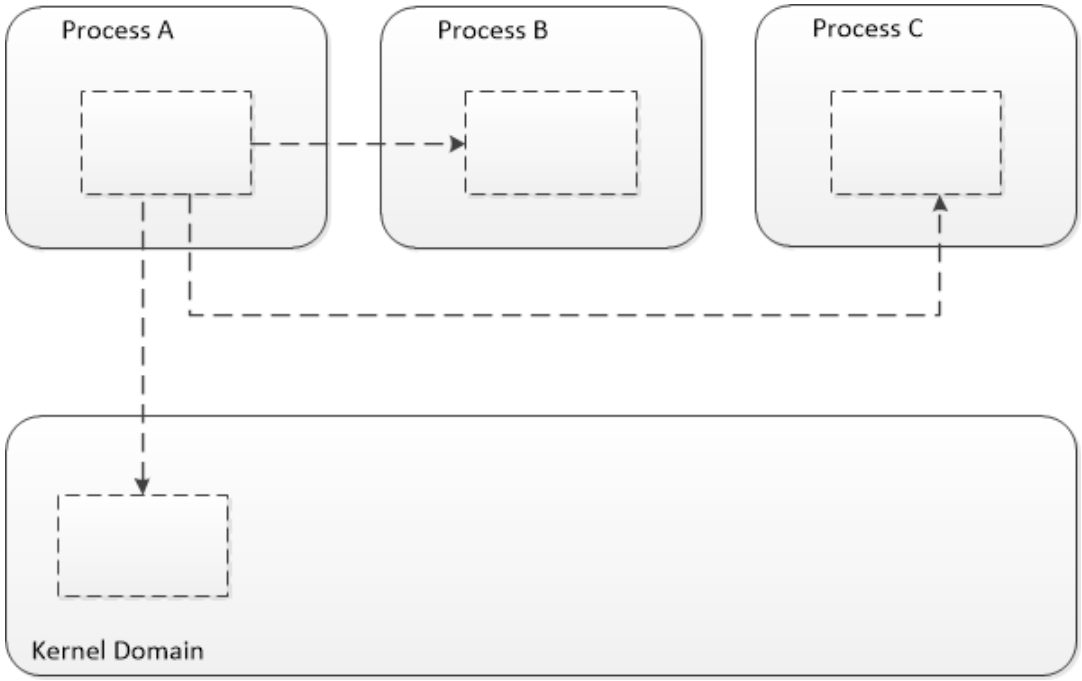


Figure 5-1 Multiple execution domains

MINK is like a microkernel in that it implements access control mechanisms, but not policies. The creator of a process can grant memory access rights and objects. MINK does not automatically grant memory access rights or any other authority to the process. It enforces no restrictions except for those inherent in the capability model.

MINK differs from many microkernels in that system services can be implemented either in the kernel domain or in processes. When a process invokes a kernel-resident object, it triggers an operation similar to a syscall in popular "monolithic" kernels: a "trap" or "software interrupt" instruction transfers control to the kernel domain where the invocation handler of the object is called, and when that function returns execution resumes in user mode. When a process invokes an object that resides in another process, it triggers an operation similar to the synchronous IPC in microkernels like L4 or Mach.

5.1.2 Kernel domain

The kernel domain is the domain in which the MINK kernel software executes. The kernel software provides C APIs for creating processes, threads, and memory regions. These APIs are callable by any code running in the kernel domain.

Alongside the MINK kernel software, there is other software that builds on MINK. For example, the following responsibilities are not handled by the MINK kernel software, but are typically handled by other software in the kernel image:

- Loading an ELF file and constructing a process in which it executes
- Controlling hardware other than the CPU and MMU

5.1.3 Processes

A process is a protection domain in which software executes in the user mode of the CPU. Some processes may contain QTEE applications, and some may contain drivers or other system services.

Each process has an associated address space that dictates the memory access rights of the process. An address space associates virtual addresses (the addresses used by code executing in the process) with physical addresses and access rights (read, write, execute).

Each process also has an object table, which is analogous to a file descriptor table in POSIX-based operating systems. The object table is an array of objects that the process is allowed to invoke. When the process invokes an external object (one that resides in another domain), it identifies the object by its index in the object table and executes a trap instruction. In the kernel, MINK validates that the index is within the bounds of the array and that the entry is not vacant before dispatching the call to the specified object.

5.1.4 Threads

Processes are single-threaded. There are no APIs callable in user mode for thread creation, management, or synchronization. There is a simple threading API callable in the kernel domain, but this API is considered private to the MINK kernel software.

The primary means of communication between threads is object invocation. Invocation into a process will rendezvous with the thread of that process, which processes the request and, when done, awaken the invoking thread. If the invoked process is busy, the invocation will be queued and the invoking thread blocks until the invoked process handles the invocation.

The kernel environment is multithreaded but non-preemptive.

For each process, there is an associated "task" that can execute in either kernel mode or supervisor mode.

Each process has an associated "thread", or to be specific, an instance of KThread. This thread begins execution in user mode on a stack that has been allocated for that process. When a process performs an invocation, it transitions from user mode to kernel mode and a kernel-resident stack is initialized. While executing in the kernel, the thread executes as part of the kernel protection domain. On completion of kernel-side execution, control returns to user mode, executing as part of the original process.

While there is one instance of KThread - one entity known to the scheduler - associated with a user process, from the point of view of any "thread-local" storage mechanisms, the kernel-side, and user-side execution modes appear as different "threads". Code executing within the process will not be given access to the kernel-side stack or other kernel data.

5.1.5 Objects

A MINK object consists of a function pointer and a context value.

```
struct Object{
  int (*invoke)(ObjectCxt h,
                ObjectOp op,
                ObjectArg * args,
                ObjectCounts counts);
  void *context; //context data to pass to the invoke function
};
```

When an object is invoked, its invoke function is called, and its context value is passed as its first argument. The Object_invoke macro captures the following:

```
static inline int Object_invoke(Object o,
                                ObjectOp op,
```

```
                                ObjectArg *args,
                                ObjectCounts k)
{
  return o.invoke(o.context, op, args, k);
}
```

Context values typically hold pointers to heap nodes that have been allocated to hold the state of the object. In rare cases, in which the objects have no mutable internal state, a pointer to const memory can be used, or the context value can be ignored entirely by the invoke function of the object.

5.1.6 IPC

When the object and its client reside in different domains, the IPC of the MINK mechanism works.

In these cases, the Object structures held by the client are proxy objects that are responsible for delivering the invocation to the target object.

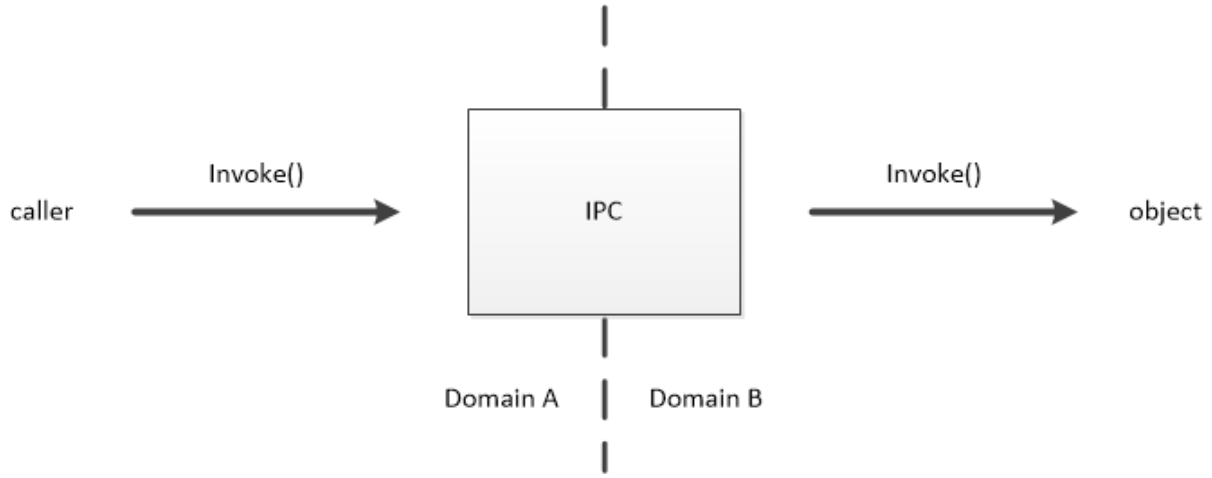


Figure 5-2 Proxy objects

The MINK kernel software decomposes the IPC problem into two cases:

- User-to-kernel invocation
- Kernel-to-user invocation

In the user-to-kernel case, a client running in a process invokes an object that resides in the kernel. In this case, the proxy object is called an outbound proxy. Outbound proxies have the same invoke function pointer, a function named minkIPC. The context value of each outbound object holds a number that is an index into the object table of the calling process. minkIPC executes a trap instruction after populating CPU registers with the object index and other invoke parameters. The trap instruction transitions to the kernel, where the trap handler locates a stack on which to execute and proceeds to validate arguments and call the kernel object's invoke function.

In the kernel-to-user case, the kernel invokes an object hosted by a process. In this case, the proxy is known as an inbound proxy. Their context value holds a pointer to a kernel data structure that identifies the host process and the 'Object' to invoke in that process. The invoke function for inbound objects interacts with the thread of the host domain and provides it with the information required to invoke the object, and suspends the calling thread until the host process responds with the return value.

We can also identify another scenario, user-to-user, in which a process invokes an object that resides in

another process. MINK does not create a special proxy for this case. Instead, it chains together an inbound proxy and an outbound proxy. The client process has an outbound proxy that forwards to a kernel object, which is in turn an inbound proxy that forwards to a process- hosted object.

The role of IPC in MINK is to convey Object_invoke across domain boundaries, so the Object_invoke function, discussed further in Section 5.2, serves as a specification for MINK IPC. The salient features that can be used to compare and contrast IPC of the MINK with other operating systems are as follows:

- Synchronous operation - Invoke is like POSIX system calls in that in addition to accepting inputs it can provide outputs. The invoke call blocks the caller until the operation has completed.
- Explicit parameters - The caller of Object_invoke explicitly declares all parameters and indicates whether they are input parameters or output parameters. Listing a buffer as an input (or output) parameter indicates that the invoked object should be allowed to read (or write) the contents of that memory during the invocation.

Due to this property, MINK can easily and securely convey invocations across domain boundaries without having explicit knowledge of the semantics of each call.

In this respect Object_invoke is like most POSIX system calls, but unlike ioctl that in general relies on the ability of the driver implementation to inspect and manipulate the address space of the calling process at its own discretion.

- Discontiguous buffers - Input and output data can reside in multiple discontiguous buffers. Each buffer is a sequence of bytes located anywhere in the address space of the process; there is no requirement for page alignment. Restricting inputs or outputs to a single contiguous buffer or for page alignment can introduce a need to copy the data to/from the original source or final destination.
- Caller-allocated output buffers - The caller of invoke provides memory to hold response data. The invoked object may fill only a portion of the buffer, but it may not return more data than was allocated by the caller.
- Object-based addressing - Messages are directed to individual objects (e.g. "sockets", files, or devices). This can be contrasted with systems in which addressing is based on protection domains or threads.
- Object-based access control - The kernel allows each process to invoke only the objects that have been granted to it.
- Object parameters - Objects can be passed as inputs or outputs in an invocation. This is the mechanism by which addresses are communicated and by which access is granted. Due to the access control implications, objects are clearly identified as such and distinguished from "plain old data". A process can pass an object to another process only when it holds a legitimate reference to the object.
- Errors - An integer return value indicates error or success conditions. When the reference capability cannot be invoked - for example, the process in which it resides has terminated - the kernel returns one of a number of predefined error codes. In the case of a successfully delivered invocation, holds the value returned by the invoked capability.

5.1.7 TA to TA IPC

Assumptions:

- TA1 has following properties:
  1. Entry points implemented:
     (a) tz_module_open: When TA service is invoked using qsee_open
  2. Has no registered UUID.
- TA2 & TA3 has following properties:
  1. Entry points implemented:
     (a) TA_OpenSessionEntryPoint: When TA session is opened via TEE_OpenTASession
     (b) TA_InvokeCommandEntryPoint: When TA is invoked for a service via TEE_InvokeTACommand
     (c) TA_CloseSessionEntryPoint: When TA session is closed via TEE_CloseTASession
  2. Has registered UUID

IPC scenarios:

- TA1 opening session to TA2
  - Steps:
    1. Open session to TA2 using TEE_OpenTASession from TA1
    2. Invoke GP TA command using TEE_InvokeTACommand from TA1
    3. Close session to TA2 using TEE_CloseTASession from TA1
  - Return status:
    1. TEE_SUCCESS (0x00000000)
- TA2 acquiring TA1's exposed service Object
  - Steps:
    1. Acquire TA1's exposed service object using qsee_open
    2. Invoke TA1's service
    3. Release TA1's exposed service object using object_release
  - Return status:
    1. SUCCESS (0)
- TA2 opening session to another TA3
  - Steps:
    1. Open session to TA3 using TEE_OpenTASession from TA2
    2. Invoke TA3 command using TEE_InvokeTACommand from TA2
    3. Close session to TA3 using TEE_CloseTASession from TA2
  - Return status:
    1. TEE_SUCCESS (0x00000000)
- TA2 opening session to TA1
  - Steps:
    1. Open session to TA1 using TEE_OpenTASession from TA2
  - Return status:

1. TZBSP_MINK_GPSVC_OPENSSN_NOT_FOUND: TA1 doesn't have UUID registered, hence cannot be found.
2. TEE_ERROR_GENERIC(0xFFFF0000): TA1 doesn't have TA_OpenSessionEntryPoint implemented, hence generic error is returned.

- Observation:
  - If TA2 was accessed via the GP stack, and then it accesses TA3 using the GP stack (i.e. TEE_OpenTASession and TEE_InvokeTACommand), then the call while in TA3 is going to have the same cancellation masking state that it had while executing in TA2 and is considered part of the same operation initiated by the client. Therefore a TEEC_RequestCancellation() on the operation initiated in TA2 will affect the operation while active in TA3, on TA3 defined cancellation points. TA3 can though mask the cancellation as per standard.
  - If instead TA1 was not accessed via the GP stack, then the operations initiated by TA1 when calling into TA2 via TEE_OpenTASession() and TEE_InvokeTACommand() are not cancellable, regardless of the cancellation masking state of TA2 in its cancellation points.

5.1.8 Capabilities

When a process holds a reference to an object that lives in another domain that object reference is a capability. In other words, it has the following security properties:

- Object references are unforgeable. A process may not access an object unless it has been granted a reference to the object. In MINK, an object can be granted to another domain by passing it as an object input argument, or by returning it as an object output argument.
- Object references embody the authority to invoke the object. A process can invoke an object if and only if it holds a reference to it.
- Object references may be granted. If a process holds a reference, it can grant it to another process by passing it as a parameter to an invocation.
- Information about the invoking domain is not leaked. Invoking an object conveys only the information explicitly provided by the caller. The invoked object cannot inspect properties of the calling process, or access memory of the calling process (outside of the inputs and outputs).
- Information about the invoked domain is not leaked. Holding an object reference does not allow the holder to obtain information about the object - such as which process it resides in - unless the object chooses to make that information available.

In these respects, MINK object references are similar to "file descriptors" in POSIX, "ports" in Mach and "objects" in Binder IPC.

5.1.8.1 Process termination

During an invocation into a process, the called process may terminate due to an exception. In this case, the invocation immediately returns to the caller with an appropriate error code.

During an invocation from a process, the calling process may be terminated. This results in cancellation of the invocation. If the invocation remains queued at the time of cancellation, it is dequeued. If the target process has already accepted the invocation and has begun handling the invocation, the invocation runs to completion in the target process, and the return value and any output data will be discarded.

5.1.8.2 Parameter validation

When a process invokes an external object, the kernel validates that the following is true:

- The object reference is valid.
- Input buffers are in memory that is readable by the calling process.
- Output buffers are in memory that is writable by the calling process.

When these conditions are violated, an exception is generated in the calling process, which results in termination of the process. This is a "fail-safe" mode of operation since invalid values are indicative of a serious problem in the process. Invoking a local capability is equivalent to a C function call, and in those cases invalid pointers may or may not corrupt process memory or generate exceptions

5.1.8.3 Memory regions

A memory region is a MINK object that embodies the authority to map memory. Each region identifies a set of physical memory addresses that appear contiguously in the virtual address space. Each region also identifies a set of permissions (read, write, execute) that are allowed.

A process can grant access to memory by passing the appropriate memory region to another process (as an object argument). When a process receives a memory region, it can then add it to its memory space by invoking its memory space object. There are a few different kinds of memory regions that differ by how they are created.

- Allocated regions - A process can create a newly allocated memory region by specifying a size and a set of permissions. The memory to satisfy this request is pulled from a pool of available virtual memory pages. When the last reference to the region is released, the underlying virtual memory will be freed.
- Window regions - The kernel can create regions that map arbitrary physical pages into virtual memory. This is a powerful API that, when misused, can violate the security guarantees provided by the kernel. Care must be taken to ensure that the memory is owned by the kernel and does not overlap other regions in use.
- Sharing regions - Sharing regions authorize restricted levels of access to other pre-existing regions. For example, a process might create an allocated regionwith read and write access, and then create a sharing region that authorizes only read access to that memory, which it can then grant to another process.

5.1.8.4 Mapping

Before accessing a memory region, a process must first map it, which modifies the address space of the process, mapping contiguous virtual addresses to the physical pages denoted by the region.

The mapping operation returns information about where in the address space the memory has been mapped (an address and a size). This provides a guarantee to the calling process that the memory within that region is accessible according to the permissions listed in perms.

```
method map(in interface region,
           in int32 perms,
           out uint64 address,
           out uint64 size,
           out interface memmap);
```

Address is returned as a uint64, which the caller must eventually convert into a pointer in order to be able to access the memory. (On 32-bit platforms, the top 32-bits may be safely ignored.)

On success, the returns address range is guaranteed to be accessible in that memory space for the duration of the mapping. The memmap object governs the lifetime of the mapping. When it is deleted, the memory is unmapped.

Due to its nature, the map operation must be trusted by the caller. Since regions may be provided by untrusted parties, we cannot rely on the region objects themselves to implement this method. Instead, each client calls its own memory space object, which implements IMemSpace.

5.1.8.5 Summary of memory objects

The summary is as follows:

- Memory space - Controls the virtual address space (mapping and permissions) seen by code executing within a domain.
- Memory region - Designates physical memory and access rights and confers the authority to map that memory with those rights.
- Memory map - Acts as a reference count on an address space mapping. When deleted, the memory access guarantees provided by mapRegion are invalidated.
- Memory title - Designates a physical memory resource, and confers the authority to claim it (obtain exclusive authority to map the memory).

5.1.8.6 Limitations

The memory APIs in Section 5.1.8.4 describe a protocol or set of conventions that could apply to any capability-based distributed system. They could be used in many different scenarios involving clients running in different domains, whether they be in the MINK kernel, a MINK process, or in other domains running in other processors or VMs atop a hypervisor. Practical considerations, however, limit the applicability.

Two domains connected via a TCP link is not able to share memory. However, two MINK processes should be able to share any region mappable in either process.

When memory regions allocated in the REE are passed to a MINK kernel running in TZ, the contiguity and size of the regions may be a factor in whether certain operations can be supported. Guarantees of exclusivity (as with IMemSpace::claim) requires an available xPU partition. The region might not even be mappable due to the amount of memory required for translation table entries.

5.1.9 Modules and services

5.1.9.1 Services

Each execution domain (process or kernel) has an environment object that it can use to request functionality from other domains. Environments are kernel- resident objects. They implement the IEnv interface, and provide a method called open that allows callers to request services.

In order to route these requests and do so securely, environment objects keep track of the credentials associated with each execution domain. When a process is constructed from an ELF file, signed data contains metadata associated with the code. The metadata kept in the environment includes the following:

- A privilege set, which is a set of IDs. When a service is requested via open, the environment ensures that the service ID is present in the privilege set. If not, the environment returns an error code.
- A set of services hosted by the domain. When other parts of the system request these services, they will be routed to this domain.

The kernel also has an environment object. It is considered to have a universal privilege set - anything it requests will be honored.

5.1.9.2 Modules

Each domain provides a module object to the system to allow the system to access services hosted in the domain. Module object implements the IModule interface:

```
interface IModule {
    method opens (in ServiceID id,
                  in interface credentials,
                  out interface obj);
};
```

An open method of a module is called by the kernel. The credentials parameter can be thought of as describing who is requesting the service. MINK is a capability-based system, though, so it is more accurate to say that the credentials parameter describes the authority on which the request is based. A typical sequence of events looks like this:

1. Software executing within a process calls IEnv_open to request a service, invoking the environment object of that process.
2. The environment object checks its privilege set, and refuses to handle the request if the corresponding privilege ID is not present, otherwise it continues.
3. The environment object locates a module that hosts the requested service.
4. The environment object calls the IModule_open method of the hosting module, and passes a facet of itself as the credentials object.
5. The hosting module dispatches the open request to the appropriate object constructor function.
6. The object construct may inspect the credentials

5.1.9.3 Dynamic Modules

Processes can be constructed around ELF files. A MINK Object structure is constructed by using the entry point of the ELF file as the invoke member and NULL as the context member. This object is expected to implement the IElfFile interface. The ELF file is initialized by invoking the init method of this object, inside the newly created process. During this call, global variables can be initialized.

```
interface IElfFile {
    method init (in IEnv env,
                 in IElfFile_LibInfo libInfo,
                 out IModule module); };
```

The init method returns a module object for the newly constructed process. This is used to obtain any further functionality from the module.

5.1.9.4 App Loading Example

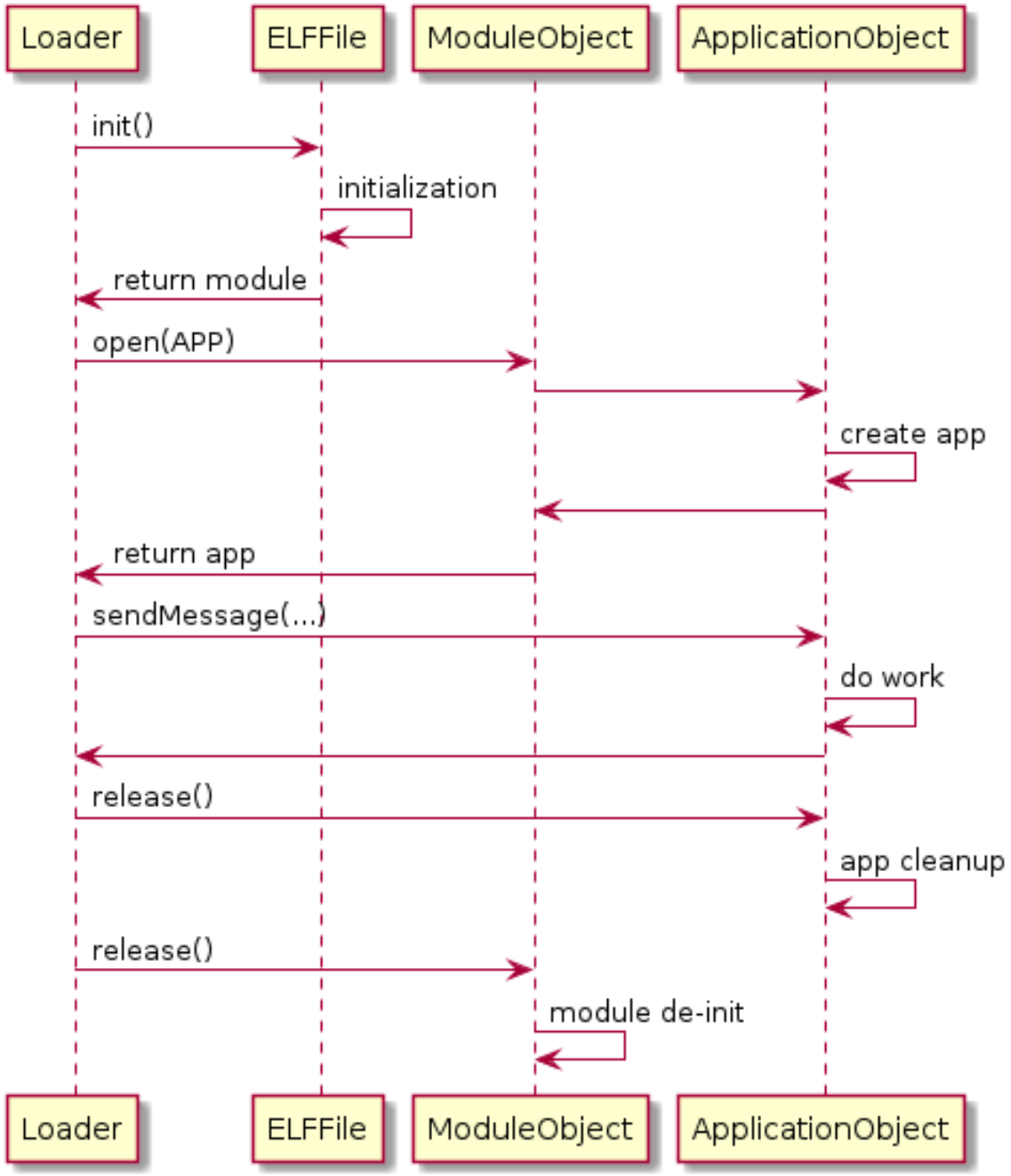


Figure 5-3 App loading example

Referring to Figure 5-3

1. An ELF file is loaded and a process is constructed to host it.
2. The init method of the ELF file is called, providing it with an IEnv instance, and returning from it an IModule instance.
3. The open method of the module is invoked to instantiate an application.

4. Applications are objects that respond to one or more messages from their host (the external entity that has launched them).
5. When there are no more messages, the application object will be released, and before the module is unloaded the module object is released.

Note: The release() operations are shown as one-way operations because nothing outside of the process waits on a response from them. They do have the opportunity to complete within their thread, however.

5.1.10 Module metadata

Metadata, or information about the module, is stored in the ELF file of the module with its code. Metadata is divided into two parts: ELF-specific information, and properties explained in Section 6.2.1.

5.1.10.1 ELF-specific Items

These items describe values that will be used in loading and ELF and initializing its execution environment. This information describes the built ELF file, and locations within it, and therefore must be resolved at link time.

This information does not grant authority to apps, so therefore it does not need to be auditable, and we could trust the application code to provide these values early in the app start procedure.

- stack address and size
- accept buffer address and size
- initial invoke - Address of the invoke function for IELFFile
- version - Indicates the version of the platform for which the binary was built.

  This is composed of a "major" and "minor" fields. A major version mismatch indicates a compatibility break, and the system must not load any binary unless it knows that it supports that major version. The minor version can be safely ignored by the system loading the ELF. Minor versions indicate optional features supported by the ELF. Any minor version N $>$ 1 implies support for all of the features of version N-1.

5.2 Object invocation

5.2.1 Introduction

This section describes the Object_invoke (herein invoke) interface in detail. It identifies requirements that the caller and callee must meet, including security requirements.

Object_invoke defines a generic transport layer for a remote method invocation system. It does not define the semantics of invocations, but it describes inputs and outputs only to the extent that is required to communicate them across domain boundaries. MINK IDL (Section 5.3) provides a mechanism for describing the semantics of objects compatible with invoke.

5.2.2 API

The C interface is defined in object.h A short overview is provided here. Clients invoke an object by calling the following function:

```
int Object_invoke(Object o,
                  ObjectOp op,
                  ObjectArg args [],
                  ObjectCounts k)
```

The Object o parameter is a value that identifies the object being invoked.

It takes the form of a structure of two values: a function pointer invokes and a void pointer context.

```
typedef struct Object {
   ObjectInvoke invoke ;
   ObjectCxt context ;
} Object ;
```

To invoke the object, the client calls its invoke function, passing its paired context value as its first argument. The Object_invoke inline function is a shorthand for this.

To invoke the object, the client calls its invoke function, passing its paired context value as its first argument. The Object_invoke inline function is a shorthand for this.

5.2.2.1 Operation

Each ObjectOp value contains two values stored in bit fields: modifier bits and the method.

Modifier bits are reserved for transport-level semantics, described in Section 5.2.2.3. Callers other than transports should leave these bits zero. Modifier bits are the higher-order bits, so callers can simply pass a method ID as the operation code.

Callees must extract the method ID using ObjectOp_methodID(op).

Method IDs from 0 to ObjectOp_METHOD_USERMAX and from ObjectOp_LOCAL to ObjectOp_LOCAL + ObjectOp_METHOD_USERMAX (inclusive) are "user" values: objects can choose any meanings for them. Other values are reserved for use by object.h. Currently two such methods IDs are defined: Object_OP_release and Object_OP_retain.

Objects must safely ignore reserved methods they do not support and return Object_ERROR_INVALID.

5.2.2.2 Arguments and Counts

The args parameter is an array containing the arguments, which describes buffers (contiguous byte arrays in memory) or objects.

```
typedef union ObjectArg{
    ObjectBuf b;
    Object o;
} ObjectArg;
```

Each argument, whether buffer or object, is classified as either an input argument or an output argument. Arguments are ordered according to their classification: input buffers are followed by output buffers, then input objects, and finally output objects.

The ObjectCounts k parameter specifies the number of each kind of argument. It is constructed with the following macro:

```
ObjectCounts_pack (nBuffersIn, nBuffersOut, nObjectsIn, nObjectsOut)
```

For example, ObjectCounts_pack(2, 0, 0, 1) indicates that args[0] and args[1] contain ObjectBuf values used as inputs, and that args[2] holds an ObjectArg value used as an output.

- Buffer arguments

  Each buffer is described by a pointer and a size. Buffers are sequences of bytes that are logically passed by value. The size field of each ObjectBuf structure must be set by the caller to the amount of memory allocated. The contents of input buffers must be initialized by the caller. When returning Object_OK, a callee must set the size field of each output buffer to the number of bytes (starting at offset 0) that have been filled with data. In error cases, the output data is not marshaled and the caller

must ignore the size value.

- Object arguments

  Object arguments are simply Object values - the same values used to invoke objects, plus a special value, Object_NULL.

  The caller must initialize all input objects to valid objects or Object_ NULL. The callee must initialize all output objects, on success, to valid objects or Object_NULL. In the event of an error return value, output objects must be ignored by the caller.

- Reference counting

  Each Object value (aside from Object_NULL) constitutes a reference to some object. It is the responsibility of the programmer to keep track of references held by the software. The retain and release methods increment and decrement an internal reference count in the object.

  Each object returned from invoke - as a non-NULL output object argument returned from a successful invoke - constitutes a new reference held by the caller, which must be eventually released.

  When an object is invoked, the input object arguments are available for its use during the invocation. The callee may copy out the object argument and holds it beyond the duration of that call, but if so it must call retain on the object to account for the newly created reference, and then call release when that reference is destroyed.

In this example, args and k describe invocation arguments consisting of two input buffers, one output buffer, one input object, and one output object.

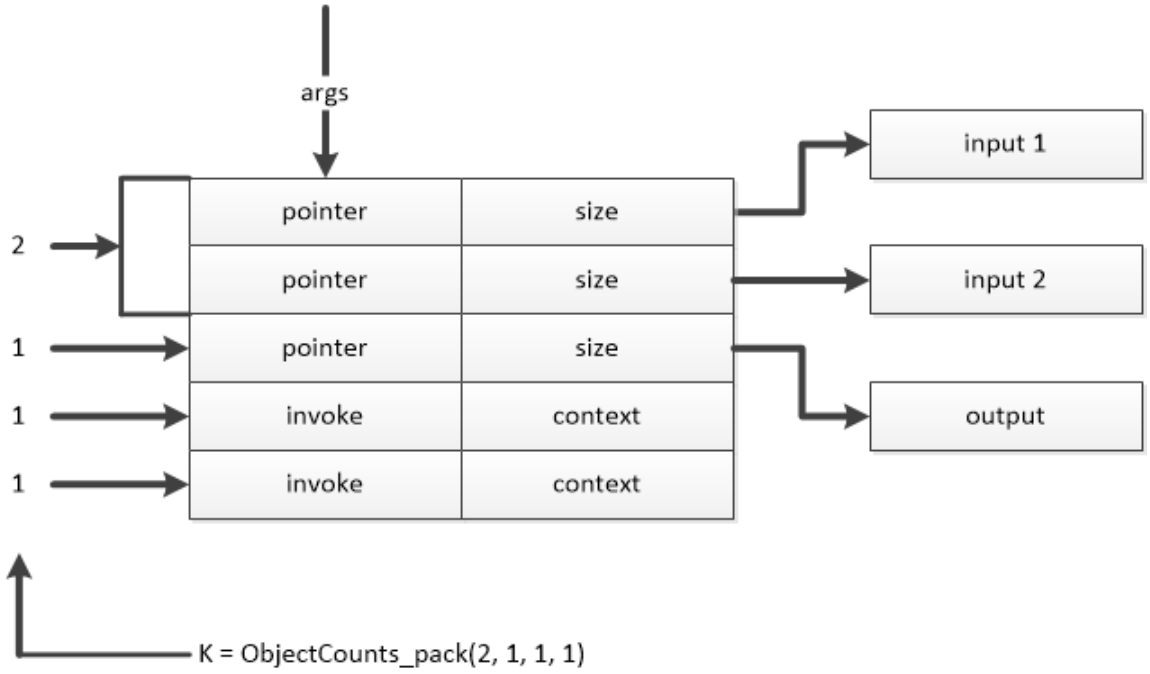


Figure 5-4 Example: arguments and counts

5.2.2.3 Transport implications

Given a domain and a transport that allows communication with objects outside of that domain, we can classify objects as "local" or "transport" objects.

- A "transport" object is an instance of a class that is provided by the transport. Each transport object identifies a destination object in some other domain. When it is invoked, the transport forwards the invocation to another domain.
- All other objects are "local" as far as that transport is concerned. When these are invoked from another domain, the transport forwards the invocations. When these are called by other objects within the domain, the transport is not involved at all.

We can visualize transport objects as "wrappers" that construct a chain to the destination object. Wrapping occurs when local objects are passed to a different domain. This happens when we pass a local object as an input to a transport object, or when a transport object returns another transport object. Unwrapping occurs when object references are passed "back downstream": when a transport object is passed to a transport object, or when a local object is returned from a transport object. In these cases, the transport recovers the referenced object from the wrapper.

- Operation

  The modifier bit ObjectOp_REMOTE_BUFS is set by transports when the memory location and alignment were provided by an untrusted domain. Implementations may skip alignment validation and/or copying when they receive an invocation with this bit clear.

  Method values are partitioned into local and remotable ranges. Local methods - ObjectOp_isLocal() can be used to test for these - are not forwarded by transports across domain boundaries. Objects receiving local methods can safely assume that the caller is in the same domain and can trust other arguments.

- Buffer arguments

  Buffer pointers are not preserved across a boundary. Destination objects may be provided buffers that have different addresses than those provided by the caller.

  All or part of the memory in a client-provided buffer may be shared with the invoked object. If not shared, the contents of input buffers will be copied before the destination object is called, and the contents of output buffers will be copied after the destination object returns (if it indicates success).

  Although the pointer value is not preserved, alignment is preserved, at least at 2-, 4-, and 8-byte boundaries. When a client provides a buffer that is N-byte aligned (where N is 2, 4, or 8), the destination object receives a corresponding buffer that is at least N-byte aligned. Transports that copy buffers can easily comply with this by copying into 8-byte-aligned buffers.

- Return values

  A zero return value indicates success, while non-zero values indicate errors. Errors indicate exceptional conditions in which the invocation or its response were not successfully delivered. This could involve a failure in marshaling, and it could also result from a case in which an invoked object did not support the requested method.

  When transports receive an error result they do not marshal data or object outputs; instead, they set output sizes to zero and set output objects to Object_NULL.

- Generic errors

  Generic errors are as follows:
  – Object_ERROR - This is a generic error code that does not have any further implications.

  - Object_ERROR_INVALID - The invoked object does not understand the request. This could be a result of an unknown op ID or an unknown configuration of arguments. This should not be returned in cases where the call has had any side effects.

    One potential use for this error code is inheritance. An object's invoke function can delegate requests to a base class invoke and consider any result other than Object_ERROR_INVALID an indication that the request was "handled". In other cases, it can proceed to test for other op values or delegate to other invoke functions.
  - Object_ERROR_MEM - This error code indicates a failure in memory allocation.

- Transport errors

  Transport errors are returned by a transport in the event of a problem in forwarding the invocation to the destination object. These can provide diagnostic value to the developer, and may suggest a potential recourse.
  - Object_ERROR_DEFUNCT - The object is inoperable or unreachable. All future invocations of the object fail. This can easily be caused by errors in reference count handling.

    This MINK IPC mechanism returns this code when the process hosting the invoked object been terminated.
  - Object_ERROR_BADOBJ - An invalid object has been provided to a transport. minkIPC return this code when the caller invokes an object with an invalid handle (object.context) or passes such an object as an input parameter.
  - Object_ERROR_MAXDATA - The args array is too large to be marshaled into the destination domain.

    This may occur when invoking, from inside a process, any object that lives outside the process. The args size limit is fixed for a given build of MINK.
  - Object_ERROR_MAXDATA - The payload is too large to be marshaled into the destination domain.

    This may occur when invoking into a process. The payload size includes the size of the args[ ] array and all of the input and output buffers after each is rounded up to the next multiple of eight bytes. The payload limit is configurable per process.
  - Object_ERROR_NOSLOTS - The calling process cannot hold any more references to external objects.

    This may occur when an invocation out of a process returns an object that does not reside in the calling process.
  - Object_ERROR_UNAVAIL - This error is an indication of that the destination process cannot fulfill the request at the current time.

    This may be a result of resource constraints, such as exhaustion of the object table in the destination process. Retrying the operation in the future might result in success.
  - Object_ERROR_KMEM - The invocation failed due to the failure to allocate kernel memory.

    This may occur when marshaling objects. It may also occur when passing strings or other buffers that must be copied for security reasons in the destination domain.
  - Object_ERROR_REMOTE - This error will be returned by a transport when local operations are requested on remote objects. Transports do not forward local operations.
  - Object_ERROR_BUSY - This error indicates that the target domain or process is busy and cannot currently accept an invocation.

– Object_ERROR_AUTH - Failure to authenticate a message.

When the transport is authenticated (ie, Tunnel Invoke), authentication errors may occur. This typically indicates the message has been tampered with before arriving at the destination domain.

– Object_ERROR_REPLAY - The message is being replayed.

When the transport is replay protected (ie, Tunnel Invoke), replay errors may occur. This indicates the destination domain has already received a message with the same counter value, implying this message is being replayed.

– Object_ERROR_MAXREPLAY - The invocation failed because a replay counter cannot be incremented.

When the transport is replay protected (ie, Tunnel Invoke), message replay counters are used to enforce the replay protection. These are incremented regularly, and may eventually reach their maximum value. At this point, replay protection can no longer be enforced.

– Object_ERROR_TIMEOUT - This error can be returned by the transport when the target of invocation takes too long to respond.

Support for this error code is specific to the implementation of each remote domain, since the transport doesn't mandate timing out of incoming calls. When supported, how the timeout is specified is also specific to the remote domain itself.

5.2.3 Security requirements

This section summarizes requirements that must be met by callers and callees (objects implementing invoke). When met, these requirements guarantee safety in all Object_invoke scenarios (local, remoted, etc.). In some specific scenarios - e.g. using local method IDs or when the caller has specific knowledge of the callee - these requirements could potentially be safely violated. In general, however, failure to meet these may compromise security or functionality.

5.2.4 Definitions

- Fault - A domain is said to fault when it performs an illegal action - such as a memory access violation - results in termination of the process
- Compromise - A domain is said to be compromised when its internal state - in-memory data structures, for example - becomes undefined. This may result in a fault, or it may result in the process continuing to execute in an unpredictable manner. If the kernel domain is corrupted, we can infer that the entire system is compromised.
- Corrupt buffers are said to be corrupted when their contents are not predictable.

5.2.4.1 Callers of invoke

Requirements for callers are as follows:

- The caller must have read and write access to the args array. Otherwise, the calling domain may fault.
- During the invocation, the caller must not modify the args array. Otherwise, the calling domain may be compromised.
- Alignment of args[ ] must be that of ObjectArg. Otherwise, the calling domain might fault.
- When specifying an input buffer, a caller must provide memory that is readable by the caller. Otherwise, the domain of the caller may fault.
- When specifying an output buffer, a caller must provide memory that is writable by the caller. Otherwise, the domain of the caller may fault.

- During the invocation, the supplied buffers must not be modified by the caller. Modifying input buffers during the invocation may result in unpredictable semantics, and modifying output buffers may corrupt the output buffers.

5.2.4.2 Implementations of invoke

Requirements for implementations are as follows:

- The invoked object must not modify the contents of the argument array except for output objects and the size field of output buffers. Otherwise, its host domain may fault.
- The arguments pointer and buffer arguments describe memory that is accessed only during the invocation. Keeping a pointer into this memory and using it after the invocation may generate a fault in the host domain.
- Input buffers describe memory that is readable by the callee. Writing to this memory during the invocation may generate a fault.
- Output buffers describe memory that is writable by the callee during the invocation. Reading from this memory may generate a fault.
- The implementation may modify the size field of an output buffer argument, setting it to a value less than or equal to its initial value. Setting it to a value larger than its initial value may generate a fault in the host domain.
- On return, data copied into an output buffer will be made available to the caller. If the callee does not overwrite all of the contents of the buffer, the caller receives unpredictable data.
- Reference counting: when external reference to an object is created and destroyed, the object's retain and release methods are called. As long as external references exist, the object may be invoked. Objects must therefore maintain a "retain count" to track the number of external references, or ensure that they remain invokable until program exit (e.g., objects without mutable context, or objects that are never freed).
- Memory referenced by buffer arguments may be subject to modification by an untrusted client domain at any time. Implementations must guard against TOCTOU (time of check, time of use) bugs, taking countermeasures where appropriate (e.g. copying out memory that must be read after validation).
- If an implementation relies on any alignment restrictions it must check alignment of the pointers it receives, since untrusted domains can in general cause the object to receive nonaligned pointers.

The ObjectOp_REMOTE_BUFS bit in the operation code, when clear, provides assurance that the memory is under the exclusive control of the host domain, allowing the last two requirements to be relaxed. Note, however, that this is generally known only at run time.

5.2.4.3 Transports

- Local operations

  Local operations shall not cross protection domain boundaries. This provides an assurance to the caller that such invocations do not leave its domain, and provides an assurance to the invoked object that the invocation did not originate in another domain.

  The reference counting operations retain and release are local operations. Although a release in one domain may trigger a release in another domain, the first release technically terminates at the transport layer (the process boundary). The kernel does not blindly forward reference counting requests as it does other invocations. It will, however, track retain and release requests arriving at the boundary to keep track of liveness, and once the holder releases its last reference, it calls release on

the object

- Synchronous re-entry

  When a thread calls an invoke that crosses a process boundary - either an outbound object from a process or a kernel thread invoking into a process - it can expect that no other local code will be called during the invoke except for calls to retain and/or release on local objects that were passed as inputs or received as outputs. (Note that any such retain cannot reduce the reference count to zero, according to reference counting rules.)

- Other security requirements

  Invoke should not leak information about the destination domain other than what is explicitly returned by the callee. Error codes can leak some information about the destination domain, or perhaps about another transport between the caller and the destination domain:
  - Object_ERROR_DEFUNCT
  - Object_ERROR_MAXDATA
  - Object_ERROR_UNAVAIL

5.3 MINK IDL

MINK IDL is used to describe programming interfaces that can be remoted over the Object_invoke IPC mechanism of the MINK. Once an interface is described in an IDL source file, the MINK IDL compiler, minkidl, can be used to generate type definitions and code for proxy functions that shield clients and implementations from the details of calling invoke directly. The client- side proxies are called stubs, and the implementation-side proxies are called skeletons.

MINK IDL defines its own type system independent of any particular target language. MINK IDL is designed to map efficiently to C, yet also to be compatible with many other languages.

Since MINK invocation is object-based, MINK IDL is object-oriented. Remoted function calls take the form of method invocations. Methods have zero or more input and output parameters. Methods are grouped into interfaces, which define types of object references.

MINK IDL is a lightweight IDL. It withholds language features that lead to surprising performance penalties, and meets the following performance goals for C proxies:

- Each stub and skeleton uses a fixed (and predictable) amount of stack.
- Stubs and skeletons never allocate from the heap.
- Stub and skeletons perform only constant-time operations.

It can be contrasted with other approaches, such as CORBA IDL, that presents a rich and powerful type system. Experience has shown that highly dynamic IDL types - e.g. compositions of unions, strings, and sequences, each of which can take different forms at runtime - can result in complex C APIs and expensive marshaling and un-marshaling operations.

Instead of defining yet another general-purpose serialization format, MINK IDL limits itself to data types that can simply be copied "as-is". Where highly dynamic data structures are required, programmers can pass serialized data through MINK IDL as an opaque buffer. This allows the programmer to select the serialization format (e.g. JSON) most appropriate for the problem domain, and to decide when, where, and whether serialization and deserialization are done.

5.3.1 Types

Types describe sets of values that may be passed as inputs to an invocation or returned as outputs from an invocation. We can group types broadly into "data" types and "object" types. Data values convey information stored in one or more bytes of memory. Object types convey references to objects.

Data types can further be divided into fixed-size and variable-size types. Fixed-size types include numeric types and structures. All other data types

- buffers, and arrays - are variable-size types, which means that their sizes are known only at runtime.

5.3.1.1 Built-in numeric types

The following types include numeric values:

- int8, int16, int32, int64
- uint8, uint16, uint32, uint64
- float32, float64

The numeric suffix in each type name gives the number of bits in the value.

The uint and int prefixes identify unsigned and signed (two's complement) integer values. The float prefix identifies floating point values.

5.3.1.2 Built-in buffer types

The keyword buffer identifies the buffer type. A buffer value is a sequence of bytes whose length is known at runtime.

Buffers are similar to arrays of int8 or uint8 values except that the contents are untyped, and do not necessarily describe a sequence of numbers.

5.3.1.3 Built-in object types

The keyword interface identifies the generic object reference type. All object references are a member of this type.

The same keyword, in a different context, is used to introduce user-defined object reference types (see Section 5.3.1.6).

5.3.1.4 Built-in array types

An array contains an ordered collection of fixed-size values of a given type, called the member type. The syntax for an array type consists of the fixed-size member type name suffixed with square brackets:

```
TYPE[]
```

When used as an in parameter, the caller of the method provides the array contents. When used as an out parameter, the caller provides a number that specifies maximum length of the array that may be returned, and depending on the programming language also provides allocated storage for the result. See Section 5.3.4.2 for a description of how this manifests in C. At most one input and/or output object array argument is supported per method.

5.3.1.5 User-defined structure types

Structures are user-defined types that contain one or more values of fixed- size type. A structure definition assigns a name to the type and describes all of its members. Structures may contain fixed-size arrays (the "[n]" is appended to the type name, not the field name). Structures may contain other structures as long as alignment rules are obeyed. Finally, structures may contain objects, but there is some overhead in doing so,

especially in languages other than C. Structures containing Objects are not supported as variable size array arguments, but a fixed size array of said is supported. Note that total argument limit and the total objects in/out limit must be complied with, no matter what.

```
struct NAME   MEMBER...   ;
```

Each MEMBER of the structure has the following syntax:

```
TYPE NAME;
```

Members of structures must adhere to alignment restrictions. Every fixed- size data type in MINK IDL has a well-defined size, which determines how many bytes it occupies in a structure, and an alignment, which is a number that determines where within a structure it can appear. These are given by the following rules:

- The alignment of each numeric type is the same as its size.
- The size of a structure is the sum of the sizes of members.
- The alignment of a structure is the largest of the alignments of its members.
- Sizes and alignments constrain structure contents according to the following rules:
- No internal padding: Each member of a structure must appear at an offset that is an integral multiple of its alignment. An offset of a memner is the sum of the sizes of the members that precede it.
- No padding at the end: The size of a structure must be a multiple of its alignment.

Structure definitions that violate these rules result in a compile-time error.

For example:

- int32 has a size of 4 and an alignment of 4.
- struct Foo { int32 a; int16 b; int16 c; }; has a size of 8 and an alignment of 4.
- struct Foo { int16 a; int32 b;}; is illegal because b has an alignment of 4 and appears at an offset of 2.
- struct Foo { int32 a; int16 b;}; is illegal because it has an alignment of 4 (due to a) and size of 6, which is not a multiple of 4.

The purpose of the alignment restrictions is to prevent compiler-generated padding. If padding was present, assigning a structure value or initializing all of its members individually could leave padding bytes in unspecified state. This could lead to unintentional leakage of information when a structure value is sent to another domain, or alternatively would require costly counter-measures in the IDL-generated proxies, neither of which are preferable.

5.3.1.6 User-defined interfaces

Interface values are specialized object references.

Interface typese are declared or defined. A declaration makes the interface usable as a type in subsequent method definitions. A definition declares an interface and also provides a description of all of the members of an interface. A declaration or definition must occur before its name is used as a type name.

Interface types are declared using the following syntax:

```
interface NAME;
```

Interface types are defined using the following syntax:

```
interface NAME [:   PARENT]   IDECL...   ;
```

An IDECL is one of the following:

- error NAME;

- method NAME(ARGS...)

5.3.1.7 User-defined typedefs

Typedefs introduce aliases or synonyms for other types. These can be assigned to enhance the self-documenting nature of some method definitions.

- typedef TYPE NAME;

5.3.2 Other syntax elements

5.3.2.1 Comments and whitespace

Whitespace may appear between any tokens. Comments may appear anywhere whitespace may appear. A comment is equivalent to a space character.

5.3.2.2 File

An IDL file consists of a sequence of the following:

- Include directives
- Type definitions or declarations

5.3.2.3 Include directives

```
include FILENAME
```

FILENAME is a string literal. Include directives name another file to be processes. Types defined in the included file are available for use by definitions that follow later in the including file. Include directives are idempotent. Including a file more than once is legal and has the same effect as including it once.

5.3.2.4 String literals

String literals are enclosed in ASCII double-quote characters, and the following escape sequences are supported:

- \\ represents a single backslash "\" character
- \n is a newline (ASCII 10)
- \r is a newline (ASCII 13)
- \t is a newline (ASCII 9)
- \n, \nn or \nnn represents a single byte whose numeric index is given by decimal digits following "\".

5.3.2.5 Keywords

Identifiers that are keywords in MINK IDL or in C/C++ are reserved.

5.3.3 Security considerations

Callers and implementers of IDL-generated APIs must do the following:

- Provide valid pointers (and lengths, in the case of arrays).
- Ensure that memory passed as inputs has been initialized, and ensure that outputs are properly initialized except when an error code is returned. A hazard is that C treats reads of uninitialized data as "undefined behavior", so when invoking an object in the same domain (and when in the same compilation unit or even when linked together) the compiler may generate unpredictable code. Another hazard is that uninitialized inputs can leak information to other domains. Using an allocator that provides zero-initialized memory is recommended.

- Be wary of external modifications to memory in any parameter passed by reference - this includes all array and structure parameters.

Stub code (generated by minkidl) must ensure the following:

- No leakage of information that is not explicitly passed to the method. If a stub constructs a temporary structure ("bundle") to convey discrete parameters, it should ensure that there are no padding holes in the structure, or that they are zero-initialized, or that any holes are gaps are not transmitted across domains.

Skeleton code (generated by minkidl) must assume that the caller is potentially malicious and must therefore validate all aspects of the invocation that could invalidate the guarantees implicit in the IDL-generated API before they call the implementation method. This includes the following:

- Ensuring proper alignment of input and output values.
- Ensuring adequate size of the input values.

5.3.4 C Language mapping

5.3.4.1 Type names

The following named IDL types map to a C type:

- Numeric types intXX and uintXX manifest as the C99 <stdint.h>d types intXX _t and uintXX_t.
- float32 is float.
- float64 is double.
- Names for struct and typedef types defined in IDL take on the same name in C.
- Every interface manifests as an Object.

This means that C API provides no means for statically type-checking object references. Note, however, that skeletons perform dynamic checks on argument sizes and alignment, ensuring memory safety when object references are misused.

5.3.4.2 Parameters

Each IDL parameter manifests as one or more C parameters. Otherwise, ordering of parameters is the same as in IDL.

All numeric parameters manifest as a single C parameter. Input parameters are passed by value, and output parameters by reference (a pointer type is used).

Each structure parameter, whether input or output, manifests in C as a pointer to the corresponding C type.

Input structure pointers are const-qualified. Input array parameters manifest as two C parameters: a const-qualified pointer to the underlying type and a size_t array length. For example:

```
IDL: method write (in int8 [] a);
C: int32_t iface_write(Object o,
                       const int8_t *a_ptr ,
                       size_t a_len );
```

Output array parameters manifest as three C parameters: a pointer to the underlying type, a size_t length, indicating the maximum number of elements that may be written to by the callee, and a size_t $*$ pointing to a value that, on return, hold the number of elements returned by the callee. The object implementation may leave this value unmodified or set it to a number less than or equal to the array length. Any returned elements are at the beginning of the array. For example:

```
IDL: method read (out int8 [] a);
C: int32_t iface_read(Object o,
                      int8_t *a_ptr,
                      size_t a_len,
                      size_t * a_lenout );
```

For each method invocation, the maximum number of objects that can be transported is limited. This fixed limit includes all objects in the invocation, whether they are parameters, array parameters, or objects embedded in any structures.

All data values in IDL are logically passed by value, so even when a C input value is passed by reference, callees may not overwrite the value.

Parameters of type buffer are treated like arrays, except that the unqualified pointer type is void *.

5.3.4.3 Header files

The minkidl tool generates one C header file for each IDL file. This C header contains C types, constants, and proxy code derived from the definitions that appear in the source IDL file.

For each include statement in the source IDL there are a corresponding # include in the generated header, in which the .idl extension is replaced with .h.

For each typedef and struct in the IDL file, a correspond C type is defined in the generated header.

For each interface definition in the IDL file, the header contains stubs and a skeleton. Stubs take the form of static inline functions that call Object_ invoke. The skeleton is a macro that expands to a function that implements invoke. Each generated header file is as follows:

- Self-contained - They include all required header dependencies using # include so they fill compile successfully as individual C sources. It is assumed that for each IDL file name.idl included, a corresponding name.h is available for inclusion by the generated header.
- Include-guarded - Each generated header uses include guards (#ifndef NAME / #define NAME / ... / #endif) to avoid compiler errors when the file is included twice.

5.3.4.4 Architecture assumptions

In-memory layouts of structures are designed to match the "wire" format, minimizing the amount of work required by proxies, while avoiding padding "holes" in structures that may unwittingly leak information. Achieving this requires making some assumptions about the C ABI and architecture.

- Little-endian
- Floating point representation is IEEE-754 compliant
- CHAR_BIT == 8
- alignof(T) <= sizeof(T) for all numeric types T

Support for alien architectures - e.g. 9-bit characters or big-endian integer layout - would not be impossible, but it would introduce some performance overhead in the stubs and skeletons for such a platform.

5.3.5 C++ Language mapping

5.3.5.1 Parameters

Each IDL parameter manifests as one or more C++ parameters. Otherwise, ordering of parameters is the same as in IDL.

All numeric parameters manifest as a single C++ parameter. Input parameters are passed by value, output parameters are passed by pointer and interface parameters are passed by reference of type ProxyBase.

Each structure parameter, whether input or output, manifests in C++ as pass by pointer of corresponding C++ type.

Input structure pointers are const-qualified. Input array parameters manifest as two C++ parameters: a const-qualified pointer to the underlying type and a size_t array length. For example:

```
IDL: method write (in int8 [] a);
C++: virtual int32_t write(const int8_t *a_ptr , size_t a_len ) = 0;
```

Output array parameters manifest as three C++ parameters: a pointer to the underlying type, a size_t length, indicating the maximum number of elements that may be written to by the callee, and a size_t * pointing to a value that, on return, hold the number of elements returned by the callee. The object implementation may leave this value unmodified or set it to a number less than or equal to the array length. Any returned elements are at the beginning of the array. For example:

```
IDL: method read (out int8 [] a);
C++: virtual int32_t read(int8_t *a_ptr, size_t a_len, size_t * a_lenout) = 0;
```

All data values in IDL are logically passed by value, so even when a C++ input value is passed by reference, callees may not overwrite the value.

Parameters of type buffer are treated like arrays, except that the unqualified pointer type is void *.

5.3.5.2 Header files

The minkidl tool generates IExample.hpp, IExample_invoke.hpp C++ header files for an IExample.idl file and CExample.hpp C++ header file for a CExample.idl file.

For each include statement in the source IDL, corresponding #include statements are created in the generated header, in which the .idl extension is replaced with .hpp.

For each typedef and struct in the IDL file, a corresponding C++ type is defined in the generated header.

Interface backend classes:

To implement an invocation mechanism between different domains, two helper classes are provided: ProxyBase and ImplBase.

ProxyBase class: It manages MINK objects similar to the std::shared_ptr class.

Implementation details

- Member Functions
  - It has an assignment operator for ProxyBase object.
  - It has get and consume methods, which return and consume raw MINK objects. These methods behave like std::shared_ptr::get() and std::shared_ptr::reset().
  - It has an extract method, which is used when internal MINK object needs not be managed by the current ProxyBase object. Conceptually, this stops managing a ProxyBase Object similar to a smart pointer stopping management of a resource. This should not be used directly, it is typically used by auto-generated interface headers.

  - isNull() : To check whether the given MINK object is Object_NULL.
  - It has a protected member function "Invoke", which will be inherited by derived classes.

ImplBase class: It is the base class for the C++ skeleton code.

Implementation Details:

- Member Functions
  - It implements the actual "invoke" function. Using the op argument of invoke, the corresponding function of the transport object is called.
  - Protected virtual member functions Object_release and Object_retains should only be overridden for non-standard reference counting semantics. For example, a static / singleton object.
  - Protected virtual member function invoke will be overridden by MINK IDL interfaces for invoking the client specific methods.

For each interface definition in the IDL file, the header contains stubs and a skeleton.

Stub has two classes.

- class1 IIExample has methods defined as pure virtual functions. These methods are defined as virtual, since user will override them via derived class.
- class2 IExample inherits class IIExample and class ProxyBase as public. Inheriting ProxyBase class is needed to perform operations on the local MINK object.
- Object of class IExample will be used for the corresponding method Object_invoke.

skeleton has one class.

- class3 IExampleImplBase inherits class1 IIExample as public and class ImplBase defined in impl_base.hpp as proctected. This class overrides the virtual invoke function declared in the ImplBase.
- class IExampleImplBase will be inherited as public by user defined class CExample to override the methods defined in IExampleImplBase.

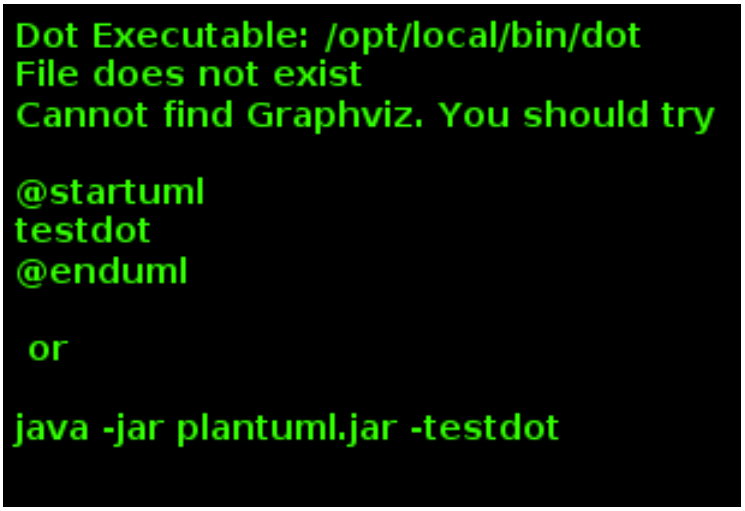


Figure 5-5 Class Diagram

Each generated header file is as follows:

- Self-contained - They include all required header dependencies using #include so they compile successfully as individual C++ sources. It is assumed that for each included IDL file name.idl, a corresponding name.hpp is available for inclusion by the generated header.
- Include-guarded - Each generated header uses include guards (#pragma once) to avoid compiler errors when the file is included twice.

5.3.5.3 Architecture assumptions

- Functionally, a C++ interface will be the same as a C interface.
- A C++ TA developer should use the C++ interface headers for MINK IPC and shoudn't mix with C-style interface headers.
- C++ interface headers have a dependency on proxy_base.hpp & impl_base.hpp
- MINK objects are consumed in the C++ ProxyBase class
- C++ clients will use ProxyBase regardless of whether the object implementing the interface is written in C, C++, or JAVA.
- Class CExample should inherit IExampleImplBase as public.
- The file including the skeleton header should expose the interface object as {ImplBase::invoke, me}

```
int32_t CExample_open(Object *objOut)
{
CExample *me = new (std::nothrow) CExample();
if (!me) {
    qsee_log(QSEE_LOG_MSG_ERROR, "Memory allocation for CExample failed!");
    return Object_ERROR_MEM;
}

*objOut = (Object){ImplBase::invoke, me};
return Object_OK;
}
```

5.3.6 Invoke transport mapping

5.3.6.1 Layering

The Object_invoke mechanism of MINK provides the transport layer used by MINK IDL. Invoke is concerned only with moving data and object arguments from one place to another, and not with the semantics of those arguments. MINK IDL defines the protocol used to send MINK IDL method invocations over Object_invoke. This protocol describes the values for operation IDs and how arguments are formatted. The stubs and skeletons generated by minkidl embody this protocol.

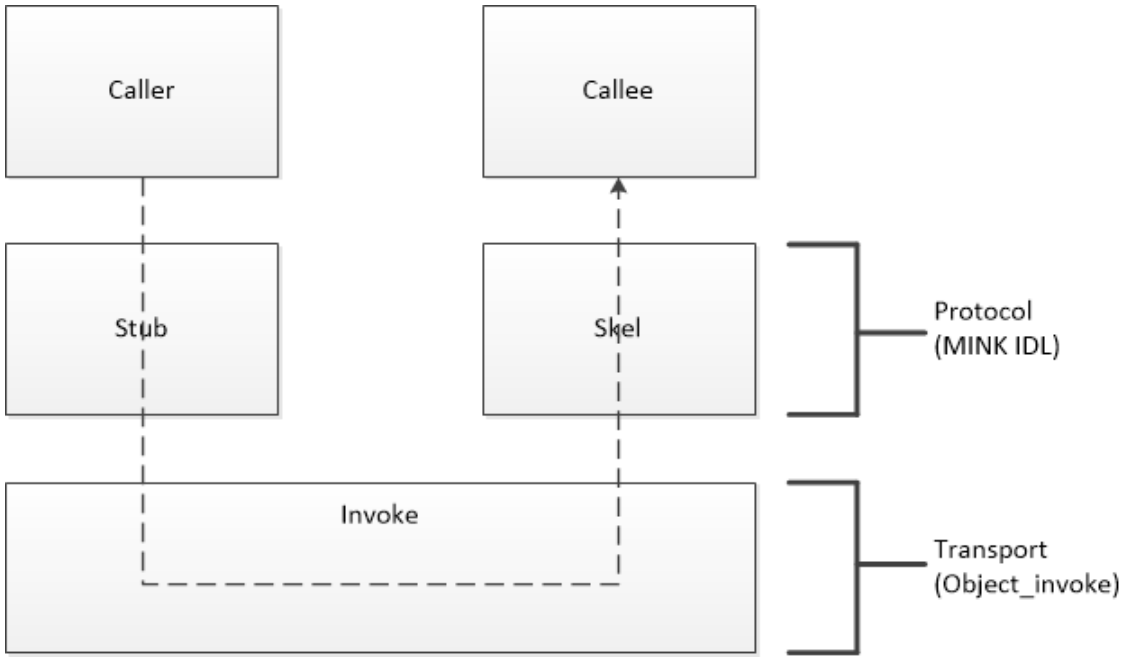


Figure 5-6 Layering

5.3.6.2 Layout

Each non-object type has a well-defined size, alignment, and transport layout (the byte-by-byte description of its content). The transport layout describes how a value appears within a buffer argument.

Numeric types have a layout consistent with their in-memory representation on a little-endian, two's complement, 8-bit-per-byte CPU architecture. The alignment of a numeric type is the same as its size.

The size of a structure is the sum of the sizes of its members. The alignment of a structure is the largest of the alignments of its members.

The layout of a structure consists of all its members placed sequentially in memory, in the order they are declared, with no intervening padding.

The size of an array is its length multiplied by the size of the member type. The alignment of an array is the alignment of its member type. The layout of an array consists of all of its members placed sequentially in memory with no intervening padding.

5.3.6.3 Class UIDs

MINK IPC services are identified by UIDs. These UIDs are not generated automatically but are chosen by the developer. As such, care must be taken to prevent collisions between service UIDs. To prevent collisions between Qualcomm and OEM services, Qualcomm has defined the following ranges:

Qualcomm: 0x0 - 0xFFFFFF (0 - 16,777,215)

OEM: 0x1000000 - 0x1FFFFFF (16,777,216 - 33,554,431)

It is recommended OEMs enforce new UIDs lie in the OEM range.

5.3.6.4 Operation IDs

The operation ID parameter of Object_invoke is a number that identifies the method being invoked. IDs are allocated to methods in the order they are defined in IDL source files, starting at 0 and incrementing by one.

When an interface inherits from other interfaces, IDs are allocated to the inherited methods first.

5.3.6.5 Invoke arguments

Recall that each invoke call accepts an array of ObjectArg values. We refer to these values as arguments herein (not to be confused with the parameters. Each argument is either a buffer (pointer and size) or an object, and each is either input or output. Arguments are arranged in groups: first all input buffers, followed by all output buffers, then input objects, and then output objects.

The ordering of arguments within a group matches the ordering of the parameters in the method declaration.

Except for bundling (see Section 5.3.6.6), every IDL parameter corresponds to one argument, as follows:

- For object parameters, the o member of the argument holds an Object value (initialized before the call to invoke in the case of input arguments, and copied out after invoke in the case of output arguments).
- For data parameters, the b.ptr of the arguement points to memory containing the value in its transport layout, and b.len contains the size of the value. Furthermore, b.ptr must have an alignment greater than or equal to that of the alignment of the value.

5.3.6.6 Bundling

A bundle is a C structure that contains one or more parameter values. Data parameters that are of a fixed size less than or equal to 16 bytes are called small parameters and may be bundled.

If there are two or more small input parameters, then all small input parameters are placed in an input bundle that is sent as the first input buffer argument. Otherwise, no input bundle is sent, and all input data parameters are sent as discrete arguments.

If there are two or more small output parameters, then all small output parameters are received in a bundle that is provided as the first output buffer argument. Otherwise, no output bundle is sent, and all output data parameters are received as discrete arguments.

The ordering of parameters within a bundle matches the ordering of the parameters in the method declaration.

5.3.7 Minkidl Compiler

The Mink IDL compiler converts the interfaces, constants, and structures defined in an IDL into source code of a desired target language. Currently, C, C++, Java, and Rust are supported target languages.

5.3.7.1 Compiler Usage

The following options are available when running the minkidl compiler:

- -o <FILE>: Specifies the name of the output file.
- –skel: Generates a skeleton header instead of a stub header.
- –c: Generates a C header. This is the default language. Need to use with '-o <FILE>'.
- –cpp: Generates a C++ header. Need to use with '-o <FILE>'.
- –java: Generates a Java class. Need to use with '-o <Directory>'.
- –rust: Generates a Rust module. Need to use with '-o <Directory>'.
- -I, –include <DIR>: Adds DIR to the include path. This option can be passed multiple times.
- –dump <DUMP>: Dumps various phases of the compiler and exits. Possible <DUMP> values are 'pst', 'ast', and 'mir'.
- –no-typed-objects: Forces C codegen to emit 'Object' as an object type instead of its own type.This option does NOT affect any other codegen backends.
- –allow—undefined-behavior: MinkIDL compiler by default is pedantic about integer widths overflowing. To allow undefined behavior to go through codegen enable this flag, outputs are not guaranteed.
- -h, –help: Prints help (see a summary with -h).
- -V, –version: Prints version information.

5.3.7.2 Restrictions

The following is the list of constraints that the compiler does not allow. If encountered, the compiler will report the error and not generate any files.

- No cyclic includes.
- Argument names within a method must be unique.
- Ensures every struct is aligned to the size of the largest member, this rule holds for recursive structs as well.
- Interface error name should not be duplicated.
- Interface function name should not be duplicated.
- Interface consts should not be duplicated, error definitions are also considered consts.
- Structs with Object in them directly or transitively cannot be used as an array in a function.
- A method cannot have both an input Object array and an input stand-alone Object as arguments. Same for Output.
- Cannot have multiple input Object arrays in a method. Same for Output.
- Object array needs to be bounded. Arguments are stack-allocated. The size must be known at compile time.
- Struct name should not be same as any other interface name.

5.3.7.3 Best Practices

When using the compiler, consider the following best practices to achieve optimal results and to enhance the code quality and functionality.

- No mixing usage of autogenerated header files of C and CPP.
- When adding new method in interface, this should be appened at the bottom. It is possible to break backwards compatibility by deleting methods or adding methods anywhere but at the end.

6 QTEE Run-time Environment

6.1 Trusted application identification

QTEE 5.0 has introduced the concept of a distinguished name and IDs to ensure unique TA identity across multiple signing authorities.

6.1.1 Distinguished name

A TA distinguished name is a dot-delimited textual identifier that is generated by combining components of the TA's certificate chain along with certain application metadata. The combination allows for the creation of a unique name where modifying any component results in a different name and thus is considered a different application from a security standpoint. The number of fields and field types of a distinguished name are dependent on the TA signing authority as well as the availability and uniqueness of certain TA metadata.

OEM-signed TAs

The distinguished name of an OEM-signed TA depends on whether the TA has defined a UUID or not as part of its metadata. If present then the UUID takes priority over the application name when generating the distinguished name and has the following format:

```
oem.<UUID>
```

For example, an OEM-signed application with a UUID of "b9a20333-2126-4f41-872bfcda3454b896" has the following distinguished name:

```
oem.b9a20333-2126-4f41-872b-fcda3454b896
```

If the OEM-signed TA does not define a UUID, then the TA name as defined in the application metadata is used instead and expected to have the following format:

```
oem.<appname>
```

An example of a distinguished name given to an OEM-signed application with the name "oemappname" is as follows:

```
oem.oemappname
```

Note: For legacy reasons, the qualified naming rules are voided and the certificate chain field can be removed for any application signed by a trusted root certificate, although it is still supported in lookup scenarios. Only OEM certificates are considered trusted at the present.

Alternate root signed TAs

In the case of TAs signed with an alternate-root, the distinguished name follows the same naming conventions of an OEM-signed TA, but adds the hash of the root certificate as an additional component to guarantee uniqueness. This is similar to the information that has been added to the oem_secapp.xml file when specifying alternate signing roots. The added component results in one of the following distinguished name formats for TAs signed with an alternate-root:

```
alt.<alt cert hash>.<UUID> alt.<alt cert hash>.<appName>
```

Examples of distinguished names given to alternate-signed TAs are as follows:

```
alt.0102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F20.
491dfeab-df8b-47f4-911f-8bf06e725e87
alt.0102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F20.
altappname
```

One is defined with a UUID of "491dfeab-df8b-47f4-911f-8bf06e725e87" and the other with an application name of "altappname".

6.1.2 Distinguished ID

In addition to the TA distinguished name a TA also has a distinguished ID. A distinguished ID is a binary representation of the distinguished name and is primarily used for key generation, access control and as a general application identifier.

6.1.3 Distinguished Name and DID Usage

Throughout the QTEE 5.0 environment, distinguished names and IDs have replaced the use of other identifiers. The following features and functionality have been improved to use the new identifiers:

- The rollback protection version table data now uses the distinguished ID in place of the certificate application ID.
- OEM configuration XML data is now required to use the distinguished name format, as defined in the preceeding sections, for identifying a TA.
- REE TA lookup is now performed using a name argument that follows the distinguished naming conventions.
- Identifying the source TA of a message retrieved from the QTEE bulletin board service is now performed using distinguished naming.
- Identifying the source TA of an inter-application message is now perfomed using distinguished naming.
- TA cryptographic keys are now derived from the application distinguished ID and as a result are no longer compatible with keys from releases before QTEE 5.0.
- Secure storage is now encrypted using the new TA cryptographic keys and as a result is incompatible with keys from releases before QTEE 5.0.

6.1.4 Trusted application SFS file persistence

Per-TA SFS file persistence is added to allow finer granularity control over whether file system data gets destroyed on a factory reset. By default, TA SFS file system data is preserved across a factory reset. This behavior can be modified on a per-TA basis by setting the new TA storageFilesNoPersist metadata property. See Section 6.2.1 for more details on this metadata property.

A side-effect of using the storage storageFilesNoPersist metadata property is that it affects the root path of the TA SFS files. The difference in the root path ultimately decides whether the content is preserved or not. Table 2-1 outlines the difference in the root path assignment depending on the storageFilesNoPerisist property setting. The base TA folder is added to the root path as described in Section 6.1.4.2.

Table 6-1 SFS file persistence path

storageFileNoPersist	Root path
True	/data/vendor/tzstorage
False (default)	/persist/data

Due to the incompatibility with this feature, the OEM control of the SFS root path via the cmnlib_oem_config.xml file has been deprecated. Specifically, the gppo_root_path and cmnlib_gppo_root_path properties are no longer supported and are ignored if used.

6.1.4.1 Increased file limit

The maximum number of SFS files supported for a given TA has increased from the QSEE 4.0 limit of 56 files up to a TA defined maximum. By using the new totalStorageFiles metadata property, a TA can increase the file limit based on its expected usage. Not setting the property results in a default maximum file limit of 60 files. See Section 6.2.1 for details on setting this metadata property. This feature has been verified with a limit of up to 2000 files.

6.1.4.2 Encoded file and directory names

SFS has been updated, and no longer uses the TA name when naming the SFS file directory and index files. Instead, the TA directory and index file naming has been improved to be more unique and is obfuscated by instead using an encoded name that looks more like the following:

```
> cd /data/vendor/tzstorage
> ls
k6SwcltMUbt7Oc0LSIGsXV6ho+56UmmvKoFpO27BfEE_
> cd k6SwcltMUbt7Oc0LSIGsXV6ho+56UmmvKoFpO27BfEE_/
> ls
eKEgHx3jo+L6bl9F4vQb
eKEgHx3jo+L6bl9F4vQb.bak
k6SwcltMUbt7Oc0LSIGsXV6ho+56UmmvKoFpO27BfEE_
k6SwcltMUbt7Oc0LSIGsXV6ho+56UmmvKoFpO27BfEE_.bak
```

6.1.5 Shared memory

Whitelist memory validation of shared memory was introduced before QTEE 5.0 to address shared memory security concerns. The RTIC privilege was added at the same time to facilitate backward compatibility of TAs affected by whitelist validations and to enable shared memory use cases where the shared memory regions are not passed by the client application.

With QTEE 5.0, the RTIC privilege has been renamed to WhitelistBypass and retains the same functionality. In addition, a new privilege, WhitelistBypassRO has also been introduced to restrict TAs to read-only access for memory not within a whitelist. See Section 7.7 for more details on both the privileges.

6.1.5.1 Shared memory interfaces

IMemSpace is the TA interface used to share memory with client applications. The interface requires shared memory regions to be described by using Mink memory objects. In the absence of support for passing shared memory objects from REE client applications using mink IPC, TAs instead create memory objects using the newly added IHlosRegionFinder interface by passing it physical memory addresses.

The IHlosRegionFinder interface performs memory validations and if successful returns a memory object that represents the capability to access a shared memory region. Access to a shared memory region is either read-only or read-write depending on validation results and TA privileges.

- Default (no whitelist privilege): Whitelist validation is performed and write access is determined by non-secure stage-2 translation table memory permissions.
- WhiteListBypassRO: Whitelist validation is performed. If successful, write access is determined by non-secure stage-2 translation table memory permissions. If unsuccessful, the returned region is read-only.
- WhitelistBypass: Whitelist validation is skipped and a read-write region is returned.

Table 6-2 TA whitelisting privilege vs. resulting permission

	Whitelisted		Unwhitelisted	
TA privilege set	**HLOS S2** RW	**HLOS S2 RO**	**HLOS S2 RW**	**HLOS S2 RO**
Default	RW	RO	None	None
WhitelistBypassRO	RW	RO	RO	RO
WhitelistBypass	RW	RW	RW	RW

Memory region objects returned by IHlosRegionFinder are generic mink objects. They may be invoked using the IMemRegion interface or passed to other domains that accept IMemRegion objects as part of a mink IPC interface.

Unlike qsee_register_shared_buffer, the IHlosRegionFinder interface provides no guarantee of exclusivity to a shared memory region. When a TA establishes a shared memory region using either qsee_register_shared_buffer or IHlosRegionFinder, other TAs may use IHlosRegionFinder to access the same shared memory region.region If a TA has gained access to a shared memory region through qsee_register_shared_buffer, other TAs will not be able to access the same shared memory region using qsee_register_shared_buffer.

The pool of available memory region mappings is shared between the IHlosRegionFinder and qsee_register_shared_buffer interfaces. As a result, the interfaces also share the limit on the number of memory regions that can be simultaneously mapped. This mapping limit is not enforced on memory region object creation but rather when attempting to map a created region into the TA's memory space.

6.1.6 Dynamically allocated TA memory

In QSEE 4.0, the TA stack and heap ranges were statically allocated as part of the TA binary layout that was generated based on TA builder parameters. Equivalently, the TA accept buffer was also declared statically based on the TA metadata. To improve upon the security of these memory buffers, these resources are now dynamically allocated during TA loading based on new TA metadata size properties rather than as previously declared via build parameters. This improvement allows for better memory containment as well as randomized placement.

Two TA metadata properties added in QTEE 5.0 are heapSize and stackSize, and their use has replaced the use of the similarly named TA builder parameters heap_size and stack_size, respectively. See Section 6.2.1 for details pertaining to the new metadata properties. Given that the TA accept buffer was already sized using the taAcceptBufSize metadata property, no new property was needed, or any configuration changes required for it.

```
md = {
   'appName':    app_name,
   'privileges': ['default'],
   'heapSize':  0x10000,
   'stackSize': 0x20000,
   'acceptBufSize': 0x4000
```

```
}
```

6.2 Metadata details

These items are trusted by the system, and will be used to grant authority to the app.

The representation of this data must not be coupled with signing formats, so that simulation environments (those that use a different binary format) can process this information in compatible way.

This data must be easily separable from the ELF file, so that privileges grants could be signed separately from the ELF (analogously to license files).

6.2.1 Metadata properties

Table 6-3 Metadata Properties

Name	Encoded Name	Type	Default Value	Description
privileges	p	IDSet	default	This list contains all of the service classes to be made available to the process. For maximum security, the application developer must ensure that this set is as small as possible. In other words, it should consist of the classes that are required for proper functioning of the application, and no others. Some of the classes are included in the default privilege set and are available to all applications. A complete list of all the classes, whether they are included in the default privilege set or not, can be found in Section 7.7 'privileges': ['default']
services	s	IDSet	None	This list contains the service classes that are implemented by the TA and exposed to the rest of the system. When the TA is loaded, its process will be registered as the "host" for services listed in its metadata. When other clients (TAs, kernel modules or processes located in other execution environments) request any of these services via their local service discovery mechanism, the request is routed to the process hosting them. When more than one loaded module claims to host the same service, it is not defined which module receives the requests. 'services': ['Adder']

Table 6-3 Metadata Properties (cont.)

Name	Encoded Name	Type	Default Value	Description
appName	n	String	None	This is a string of up to 31 bytes, typically printable ASCII or UTF-8 characters. Together with the TA signing authority, it is used by QTEE to compose the TA distinguished name, which in turn is used internally to uniquely identify the TA. The hash of the distinguished name, called distinguished ID, is used internally by QTEE to identify TAs and isolate their data and resources. It plays a role in rollback protection, key generation (including but not limited to the secure file system), and resource isolation (including but not limited to RPMB partitions). Finally the application name is added (together with the timestamp) to all log messages generated by the application. NOTE: TA is required to have either an appName or a UUID (but can have both, in which case the UUID takes precedence.) 'appName': app_name
UUID	u	Binary	None	The UUID plays the role of the application name for Global Platform TAs, and is used in its place wherever the application name would otherwise have been used. 'UUID': '12345678-1234-1234-1234-123456789abc'
objectLimit	o	Integer	40	This represent the maximum concurrent number of external Mink objects that the TA expects to have at any given time. They comprise objects from the QTEE kernel itself (service class objects), from other TAs (services implemented in other TAs) or from clients located in other execution environments. 'objectLimit' : 100

Table 6-3 Metadata Properties (cont.)

Name	Encoded Name	Type	Default Value	Description
memoryType	m	String	Protected	TAs are loaded into protected (PIMEMv1, v2, v3 and LP-DARE) or unprotected (Plain DDR) Memory. Memory defined as protected is backed by HW blocks which employ cryptography to transparently encrypt, authenticate and replay-protect read/write operations to it. Unprotected memory employs a lower level of protection, whose details depends on each chipset HW capabilities and OEM configuration. This can range from encryption and authentication (but not replay-protection), to only access control based restriction based on TrustZone-controlled Memory Protection Units. Unprotected means 'm=x', protected means 'm' does not appear in the metadata string. A TA which requires to be loaded in protected memory will never be loaded in unprotected memory as long as the platform/chipset where it is being loaded supports a form of protected memory: if there is not enough available space in the protected memory area to load it, the load request will fail. If a chipset/platform does not support protected memory, QTEE will silently fall back to loading all TAs in unprotected memory. 'memoryType': 'Unprotected'
acceptBufSize	b	Integer	4096	The acceptBufSize metadata property specifies the size of the memory region that QTEE will allocate for the TA to host its own accept buffer, i.e. the TA-owned memory where all invocations to the TA are serialized. This property has a default of 0x1000 or 4 KB. The size specified in the metadata is automatically rounded up to the nearest page-size multiple during allocation. QTEE does not support growing each TA accept buffer at runtime, so if the TA accept buffer is not large enough to contain the payload for a specific TA invocation, an error will be returned to the client originating that invocation and the call will not be allowed to reach the TA userspace code. 'acceptBufSize': 4096
multiSession	l	Boolean	True	The multiSession metadata property indicates whether a GP TA supports multiple sessions. The Global Platform implementation in QTEE uses this value internally to allow or deny requests to open new sessions with the TA. 'multiSession': 'false'

Table 6-3 Metadata Properties (cont.)

Name	Encoded Name	Type	Default Value	Description
properties	r	String	None	The custom properties metadata entry is used to declares custom, arbitrary, TA properties, for configuration purposes. From within the TA, they can be obtained via the Property API functions as defined in the Global Platform Internal API specification. The custom properties are specified in the metadata in the form of a python dictionary, the first element of each entry representing the custom property name, the second element its value. 'properties' : { 'foo': 'this is a test string', 'bar': 'true'}
version	v	String	1.0	The version metadata entry allows a developer to specify its version number in string format. This metadata entry is the source for the gpd.ta.version property, as specified in the Global Platform Internal API specification. Note that this metadata value is not used for the purpose of rollback-detection, and is totally independent from it. 'version' : '1.0'
description	d	String	None	The description metadata entry allows a developer to provide a string describing the TA. This metadata entry is the source for the gpd.ta.description property, as specified in the Global Platform Internal API specification. 'description' : 'My App'
stackSize	S	Integer	0x8000	The stackSize metadata property specifies the size of the memory region that QTEE will allocate for the TA to host its own stack. This property has a default of 0x8000 or 32 KB. The size specified in the metadata is automatically rounded up to the nearest page-size multiple during allocation. QTEE does not support growing each TA stack at runtime, so if the TA stack is exhausted at any time during the TA execution, the TA will crash. 'stackSize' : 0x10000

Table 6-3 Metadata Properties (cont.)

Name	Encoded Name	Type	Default Value	Description
heapSize	H	Integer	0xE000	The heapSize metadata property specifies the size of the memory region that QTEE will allocate for the TA to host its own heap. This property has a default value of 0xE000 or 56 KB. The size specified in the metadata is automatically rounded up to the nearest page-size multiple during allocation. QTEE does not support growing each TA heap at runtime, so when the TA reserved heap memory is exhausted, the heap will fail subsequent allocation requests. The size specified using this metadata entry refers to the overall size of the underlying memory region which the QTEE kernel will reserve for the heap of this individual TA. It does not take into account wastage due to heap internal structures, fragmentation, or usage by QTEE userspace libraries linked by the TA (e.g. for SFS metadata cache, crypto contexts, etc.). TA developers should therefore not expect to be able to allocate the full amount of heap memory as specified in this entry, but they should stress test their TA to make sure the requested heap is enough for all their use cases. 'heapSize' : 0x10000
storageFilesNo Persist	g	Boolean	False	The storageFilesNoPersist metadata property allows a TA to specify whether its SFS files should be deleted across a factory reset. The default of this property is False. When set to True, the TA SFS files will be deleted in the event of a factory reset. The setting of this property affects the TA SFS file root path as described in Section 6.1.4. NOTE: toggling this metadata value will result in corruption in previously stored SFS File. 'storageFilesNoPersist' : True
totalStorageFiles	f	Integer	60	The totalStorageFiles metadata property allows a TA to specify the maximum number of SFS files it will store. By default, the maximum file count is 60. This property has been verified with a maximum limit of up to 2000 files and this value cannot be change. If any OEM expects their TA to have more then 60 SFS file they must configure totalStorageFiles upfront to required value. NOTE: Changing this number affects the format and size of the internal per-TA SFS indexes, so a change to this property for a TA will make all of its existing files inaccessible (recognized as corrupted). 'totalStorageFiles' : 2000

Table 6-3 Metadata Properties (cont.)

Name	Encoded Name	Type	Default Value	Description
minExecTimeout	E	Integer	800	The minExecTimeout and minWaitTimeout metadata properties control execution and wait timeouts (in milliseconds) in the yield service. In services with long execution time (e.g. RSA Key Generation), the entire service can be broken into execution-wait sequences using yield service in order to prevent the service hogging the cpu for long time. minExecTimeout controls the minimum amount of time that may pass before a request is made to transfer control back to the HLOS and minWaitTimeout is the amount of time that the request waits in the HLOS. By default minExecTimeout is set to 800 msec and minWaitTimeout to 50 msec. 'minExecTimeout' : 1000
minWaitTimeout	W	Integer	50	The minExecTimeout and minWaitTimeout metadata properties control execution and wait timeouts (in milliseconds) in the yield service. In services with long execution time (e.g. RSA Key Generation), the entire service can be broken into execution-wait sequences using yield service in order to prevent the service hogging the cpu for long time. minExecTimeout controls the minimum amount of time that may pass before a request is made to transfer control back to the HLOS and minWaitTimeout is the amount of time that the request waits in the HLOS. By default minExecTimeout is set to 800 msec and minWaitTimeout to 50 msec. 'minWaitTimeout' : 10
allowUntrusted Clients	C	Boolean	False	This metadata entry can be used by the TA developer to specify if QTEE should allow or deny access to the TA to untrusted REE clients. Typically REE kernel drivers and higher privileged processes with direct access to the REE kernel are considered trusted by the REE kernel itself, but the exact determination of the criteria by which a client is considered trusted by the REE is implementation defined in the REE itself. 'allowUntrustedClients' : True

Table 6-3 Metadata Properties (cont.)

Name	Encoded Name	Type	Default Value	Description
restartTA	a	Boolean	False	If this metadata property is enabled, QTEE will take care of restarting a TA after it crashes. The call during which the TA crashed will return error, and the next call directed to the TA (using any of the supported communication stacks and interfaces by QTEE: MinkIPC service discovery, QSEECom or Global Platform Client/Internal API) will cause QTEE to restart the TA. As part of the restarting it, all it's memory will be cleared and reset to their initial value, and the TA own internal initialization functions will be called. Note that restarting a TA does not restore the validity of any of the objects, handles or sessions returned by the TA in the execution cycle during which it crashed, all of which became invalid the moment the it crashed. 'restartTA' : True
neverUnload	U	Boolean	False	A TA with the neverUnload flag set to True will never be unloaded after it has been loaded the first time. Please ensure to review use case before setting this flag as the TA will occupy secure memory which won't be freed. Multiple clients attempting to load this TA will be given references to the same TA, irrespective of the API stack they used to load it (QSEECom-based or SMCInvoke-based). If this property is enabled, it is strongly recommended to also enable the restartTA metadata property, since it is impossible to unload the TA even in the event of a TA crash. 'neverUnload' : True
noClientRestart	R	Boolean	False	This flag is specific to embedded TAs, ignored for all other TAs. If set to True, and if the TA is configured to be automatically started when one of the MinkIPC services it implements are accessed, then QTEE will take care of automatically restart it in case of TA crash when any of its hosted services are accessed again during service discovery. 'noClientRestart' : true
cryptoSelfTest	t	Boolean	False	The cryptoSelfTest metadata property enables the self-test for FIPS certifiable crypto algorithms. If enabled, self-tests are performed during TA loading and if not passed the TA load attempt will abort. This property has a default of value of False. 'cryptoSelfTest' : True

Table 6-3 Metadata Properties (cont.)

Name	Encoded Name	Type	Default Value	Description
taGpBit	B	Boolean	False	taGpBit is set to True for 64 bit TAs compiled to run on ARMv9 chipsets. This metadata entry is used by the TA loader to determine whether ARMv8.5-BTI (Branch Target Identification), where supported and enabled for the code sections of the TA. See Section 7.9 to disable this feature for 64bit TAs compiled for ARMv9 chipset.
addrSel	A	String	Low	The addrSel flag takes one of two values: 'low' or 'any'. It is used to specify where the TA should be loaded. The 'any' value simply informs QTEE that it is okay to load the TA into high memory address space (>4GB) if the target permits it. Loading into high mem is however not guaranteed. The 'low'(default) value is used to strictly load into memory space below 4GB. 'addrSel' : 'Any'
preemptible	R	Boolean	True	This flag can be set as False if the TA doesn't allow to be preempted. If it's set as True or unspecified, the TA execution can be preempted by an interrupt. When a TA is preempted, QTEE allows other requests (to QTEE kernel or another TA) to be processed. Warning : Usage on several APIs needs careful attention when TA preemption is enabled. Please refer "Module Documentation" section for `qsee_is_ns_range` and `qsee_is_s_tag_area` described earlier in this document. 'preemptible' : False
requiredTrust Level	T	Integer	1	The requiredTrustLevel flag indicates whether the TA requires a certain trust level to execute. The valid values are 0 (QTI), 1 (OEM), 2 (PLT), 3 (USR), and 4 (UNK). The default value is 1 (OEM). 'requiredTrustLevel' : 1
el2AwareKDF	B	Boolean	True	The el2AwareKDF is set to True by default. This metadata entry is used by the TA loader to determine whether the context running in EL2 should be used when deriving crypto keys thanks to KDF.

6.2.2 Metadata syntax

Metadata is provided as a contiguous sequence of bytes that encodes a sequence of name/value pairs. The syntax is formally described by the following PEG grammar and some additional restrictions.

Grammar is as follows:

- MetaData - (';' / Property)*
- Property - Name '=?' Value
- Name - (![=;] .)+

- Value - (![;] .)∗

Restrictions are as follows:

- A given name shall occur only once in the property list.
- Values shall not use ”-encoding except for "=", ";" and "%" characters.

An example is as follows:

```
p=1 ,2 ,3;s= F801 ;n= SampleApp ;o=99
```

6.2.2.1 ID set

An ID set value describes a set of 32-bit unsigned integers.

An ID set consists of a sequence of comma-delimited entries, each of which is a number (in hexadecimal) that identifies an ID in the set, followed optionally by a bitwise mask. The mask, if present, is identified by a ":" character followed by any number of hexadecimal digits, and it describes additional IDs that sequentially follow as members of the set. The first hexadecimal digit represents the next four UID values following the number, the next digit the next 4, and so on. A 1 bit indicates membership in the set. Within each digit, the least-significant bits represent the lowest ID values.

Syntax is as follows:

- IDSet: (’,’? IDEntry)∗
- IDEntry: ID (’:’ Mask)
- ID: HexDigit + an MSB-first numeric value
- Mask: HexDigit + an LSB-first bitmask (HexDigit: [0 -9a-zA -Z])

Examples are as follows:

- "1,12,123" describes a set of three IDs: 1, 18, 291.
- "1:f1" describes a set of six IDs: 1, 2, 3, 4, 5, 6.
- "1:18, a1" describes a set of five IDs: 1, 2, 9, 161.

6.2.2.2 String

A string value is a sequence of URL-encoded bytes. The metadata format does not dictate what kinds of string encoding is used - interpretation is up to the consumer of the string, but UTF-8 is recommended.

- String: (![%;] . / ” HexDigit HexDigit)∗

6.2.2.3 Binary

Binary values consist of a sequence of pairs of hexadecimal digits, each giving the value of a byte (MDB-first, within each byte).

- Bytes: (HexDigit HexDigit)∗

6.2.2.4 Integer

Integers are represented as a sequence of decimal digits optionally preceded by "-"’.

- Integer: ’-’? [0 -9]+

6.2.2.5 Boolean

Boolean value is either True or False.

- Boolean: False/True

6.3 Standard library APIs

Due to improved compatibility changes in QTEE 5.0, many C standard library functions have been removed from the default QTEE libraries linked against TAs. Standard C library calls can be resolved by either linking against the LLVM distributed version of the C standard library or by implementing the missing C standard functionality as part of the TA itself. Although it is recommended that TAs can replace C standard library interfaces with those in the LLVM supplied C standard library, it is not recommended to use any beyond this replacement.

Most of the C standard library APIs depend on system functionality that is not supported by QTEE and there is no documentation on which APIs are expected to work.

The QTEE libraries continue to support C standard library interfaces in cases where the LLVM C standard library implementation is insufficient. In QTEE 5.0, support for the C standard malloc, realloc, and free memory allocation interfaces are still available and also the file I/O interface documented in the supplied qsee_fs.h header.

7 Developing Trusted Applications

QTEE provides the execution environment for running trusted applications in the TrustZone user mode. These trusted applications are invoked by their counterparts running in the REE. The REE clients can manage and interact with their trusted applications through QTEE's public APIs.

QTEE is multithreaded in the most basic sense. Each trusted application has its own thread, but there is no interleaving of threads. When QTEE receives a command for a trusted application, QTEE signals that trusted application's thread. The thread runs until the command's completion, then it is marked as inactive and QTEE sends a response to the REE client. Trusted applications only run when prompted by a command from their REE client, and every command requires a response to conclude the interaction. During processing and execution of a command, all other commands into QTEE are blocked by the REE SCM driver.

Trusted applications for QTEE can be developed using the information in this section. We provide skeleton and sample code which can be used as a template when developing a new trusted application. These are explained in the context of the off-target environment in Section 9.3. Trusted applications can be written in C and in C++. See Section 5.3.4.3 and Section 5.3.5.2 for the necessary IDL bindings.

LLVM Version:

- 12.1.1

C++ specific settings:

- C++ TAs are compiled with -std=c++11. A newer standard can be set if supported by the used toolchain.
- C++ exceptions are disabled, i.e. C++ TAs are compiled with -fno-exceptions. A C++ developer can set a std::new_handler() to handle failed memory allocations.
- C++ RTTI is disabled, i.e. C++ TAs are compiled with -fno-RTTI.

7.1 TA entry points

Please refer TA entry point APIs.

- app_disconnectClient
- app_getAppObject
- tz_app_cmd_handler
- tz_app_init
- tz_app_shutdown

7.2 Trusted application sandboxing

A trusted application only has access to these memory regions:

- Its own load segments - When running a trusted application, QTEE maps the RO and RW segments to user mode for read and read/write access, respectively. These are unmapped when the trusted application is not running.
- Shared buffers - Each application can call the QTEE API qsee_register_shared_buffer() to register up to six shared buffers spanning a nonsecure range. Two of the six shared buffers must be optimally aligned for the specified platform while the remaining buffers might have an arbitrary alignment. For AArch64 the optimally aligned buffers must begin on a 2 MB boundary and the size must be a multiple of 2 MB, while for AArch32 the alignment must be 1 MB. This memory is mapped to the trusted application for user mode, but it is not xPU protected. It functions as shared memory between the trusted application and its REE client. Shared buffers are mutually exclusive, meaning no two trusted applications can register the same region as a shared buffer. A trusted application's shared buffers are only mapped while the trusted application is running.
- The trusted application level request/response region - A REE client communicates with its trusted application using a memory-based command interface. This request/response region is allocated by the REE client as nonsecure memory, and it is xPU protected by QTEE before handing control to the trusted application. The trusted application can write a response into this region. QTEE releases the xPU lock before handing control to the REE. The request/response region is still readable by the REE while xPU locked, only writing is protected. A REE client initiates this communication using the Client Send Data command.

7.3 Exposing a MINK service to another TA

Trusted applications can expose MINK services to other TAs.

In order to expose a MINK service, a TA must have that service listed in its metadata (see Section 6.2.1). The TA exposes services via tz_module_open(). The TA requesting the service needs the necessary privilege (refer to Section 7.7) and accesses the object via qsee_open().

An example of this, including the necessary build script modification can be found in the SMCInvoke Example App in Section 9.3.4

7.4 Expected error codes from a QTEE service

There are 3 different categories of error codes expected from a QTEE service.

- Generic error codes
  - Range: 0<= value <= 9.
  - #define Object_OK 0 // non-specific success code
  - #define Object_ERROR 1 // non-specific error
  - #define Object_ERROR_INVALID 2 // unsupported/unrecognized request
  - #define Object_ERROR_SIZE_IN 3 // supplied buffer/string too large
  - #define Object_ERROR_SIZE_OUT 4 // supplied output buffer too small
  - #define Object_ERROR_MEM 5 // out of memory
- Transport error codes

  - Range: -90 and less.
  - #define Object_ERROR_DEFUNCT -90 // object no longer exists
  - #define Object_ERROR_ABORT -91 // calling thread must exit
  - #define Object_ERROR_BADOBJ -92 // invalid object context
  - #define Object_ERROR_NOSLOTS -93 // caller's object table full
  - #define Object_ERROR_MAXARGS -94 // too many args
  - #define Object_ERROR_MAXDATA -95 // buffers too large
  - #define Object_ERROR_UNAVAIL -96 // the request could not be processed
  - #define Object_ERROR_KMEM -97 // kernel out of memory
  - #define Object_ERROR_REMOTE -98 // local method sent to remote object
  - #define Object_ERROR_BUSY -99 // cannot forward invocation, calling process is busy
  - #define Object_ERROR_AUTH -100 // cannot authenticate message
  - #define Object_ERROR_REPLAY -101 // message has been replayed
  - #define Object_ERROR_MAXREPLAY -102 // replay counter cannot be incremented
  - #define Object_ERROR_TIMEOUT -103 // target of invocation took too long to respond
- Interface error codes:
  - Range: 10 and more.
  - Defined within the interface header file. For ex: IAppLoader.h.

7.5 Legacy QTEE Service Interfaces

QTEE maintains a set of APIs to expose services to trusted applications, including heap management, logging, trusted file system access, interactions with listeners, and cryptography and hashing functions.

Trusted applications run in ARM user mode, and most of the exposed services are run in supervisor mode. Calling one of these privileged operation APIs generates a supervisor call (SVC), and the application's thread switches into supervisor mode to perform the service. The thread is then switched back into user mode.

The following sections contain the APIs exposed to trusted applications and a description for each functions are present in below link.

- Address Translation
- Bulletin Board
- Cipher
- Clock
- CMAC
- Configuration
- Core
- Crypto Hardware Lock
- Data Cache Maintenance

- Elliptic Curve Cryptography
- QTEE Env
- Embedded Secure Element
- FIPS Services
- Filesystem
- Fuses
- HASH
- HMAC
- Heap Allocation
- I2C
- Interrupts
- KDF
- Key Manager
- Logging
- Message Passing
- OEM Buffer
- OEM Utilities
- Pseudo Random Number Generator
- Public Key Algorithms
- RSA
- Secure Camera
- Secure Channel
- Secure Display
- Services
- SFS
- SPI
- Storage
- Synchronization
- String Comparison
- TA Stack Usage Profiling
- Time
- Timer
- TLMN
- Unified AES
- Unified DES
- Unified PBKDF2

- Unified SHA

7.6 Object-Based QTEE Service Interfaces

The following sections contains a list of existing MINK interfaces and a description for each method contained within the class available in below links.

- IAccessControl
- IAppClient
- IAppController
- IAppLoader
- IAppMessage
- IAttestationBuilder
- IAttestationReport
- ICipher
- ICipherOperation
- IClientEnv
- IClockConfig
- ICredentials
- ICrypto
- IDataCache
- ICertification
- IDAError
- IDeviceAttestation
- IDeviceID
- IEnv
- IESEService
- IDiagnostics
- IFeatureVersions
- IGenericService
- IGPSession
- IHash
- IHavenTokenApp
- IHdcpEncryption
- IHdcpSrm
- IHdcpTransmitter
- IHdmiStatus
- IHlosRegionFinder

- IHmac
- IHwFuse
- IHWKey
- IHWKeyFactory
- II2C
- IICE
- IIntMask
- IIO
- IIPProtector
- IKey
- IKeyManager
- IKVStore
- IKVStoreKey
- IKVStoreIterator
- IKVStoreAdmin
- ILegacyHWAttestation
- ILicenseImage
- ILicenseManager
- IListener
- IListenerCBO
- IMacchiato
- IMemManager
- IMemRegion
- IMemRegionPermEscalator
- IMemSpace
- IModule
- INotifyHdcp
- INSMem
- INSSystemReg
- IOpener
- IOPS
- IOPSSink
- IOPSSource
- IPeripheralAccessControl
- IPeripheralState
- IPeripheralStateCB

- IPFM
- IPmPon
- IPrivacyPreservingID
- IProperty
- IProvError
- IProvisioning
- IPVCLicense
- IQTEEEnvInfo
- IQWESKeyStore
- IQWESTAServices
- IRegisterListenerCBO
- IRTICDtb
- IRTICReport
- ISecureCamera
- ISecureCameraClientEvent
- ISecureCamera2
- ISecureCamera2Notify
- ISecureChannel
- ISecureChannelKeyExchange
- ISecureDisplay
- ISecureImage
- ISecureImageELFInfo
- ISecureImageParserReturnCode
- ISecureImageOemMetadataParser
- ISharedBuffer
- ISource
- ISPCOM
- ISPI
- ISwFuse
- ISync
- ITLMM
- ITLOCKey
- ITPM
- ITranslateAddr
- ITrustedReport
- IUnwrapKeys

- IUptime
- IValidate
- IVerifiedSecureImageELFInfo
- IVMDeviceUniqueKey
- IVMSessionKey
- IWait

7.7 Object-Based QTEE Service Classes to Trusted Applications

The following sections contain the MINK classes exposed to trusted applications. The classes implement the interfaces described in Section 7.6. Please refer below link for more detail.

- CAccessControl
- CAppMessage
- CCertification
- CCipher
- CClockConfig
- CCredentials
- CCrypto
- CDataCache
- CDeviceID
- CDeviceAttestation
- CHash
- CHdcpSrm
- CHdmiStatus
- CHlosRegionFinder
- CHmac
- CHwFuse
- CHWKeyFactory
- CI2C
- CICE
- CIntMask
- CIPProtector
- CKeyManager
- CKVStore
- CKVStoreAdmin
- CLicenseManager

- CListener
- CMacchiato
- CMemRegionPermEscalator
- CNSMem
- CNSSystemReg
- COEMBuf
- COPS
- COPSSink
- COPSSource
- CPeripheralAccessControl
- CCPBitstreamRWAuthority
- CCPAPPRWAuthority
- CCPCameraRWAuthority
- CCPNonPixelRWAuthority
- CPrivacyPreservingID
- CPrngSource
- CQWESTAServices
- CRTICDtb
- CRTICReport
- CSecureCamera
- CSecureCamera2
- CSecureChannel
- CSecureChannelKeyExchange
- CSecureDisplay
- CSecureGPIO
- CSecureImageParser
- CSharedBuffer
- CSPCOM
- CSPI
- CSWCrypto
- CSwFuse
- CSync
- CTLMM
- CTransNSAddr
- CUptime
- CVenusSecureChannel

- CVideoSecureChannel
- CVMSessionKey
- CWhitelistBypass
- CWhitelistBypassRO

7.8 Object-Based QTEE Service Classes to REEs

The following sections contain the MINK classes exposed to REEs. The classes implement the interfaces described in Section 7.6. Please refer below link for more detail.

- CAppClient
- CAppLoader
- CDeviceAttestation
- CFeatureVersions
- CPeripheralState
- CPmPon
- CPVCLicense
- CQTEEEnvInfo
- CRegisterListenerCBO
- CTLOCKey
- CVMDeviceUniqueKey

7.9 Creating SCons environment for TA

The following section contains the TA(sample_ta) SConscript, which is copied from qtee_tas/apps/securemsm/trustzone/qsapps/sample_ta/SConscript and serves the purpose of setting up and building the TA from sources, headers and libraries.

```
#GENERAL DESCRIPTION
#
# Steps for building the TA from sources, headers and libraries.
# 1. Importing the passed build environment from SConstruct file.
# 2. Setting up TA name based on passed TA architecture.
# 3. Setting up sources, headers and libraries required for TA compilation.
# 4. Defining the attributes of the TA using the metadata dictionary.
# 5. Build the TA using the SecureAppBuilder function.
# 6. Deploy TA artifacts using the deploy_sources and deploy_variants
#    parameters.

# Import the SCons environment passed by the SConstruct to facilitate
# TA compilation.
Import('env')
import os

# Cloning the passed SCons construction environment since changing the
# environment may interfere with subsequent TA compilations.
env = env.Clone()

# Adding entry for current TA in customer build sanity script which tests
# compilation of TAs in customer build.
try:
  # Fetch the path of current Sconscript file.
  this_sconscript = (lambda x:x).__code__.co_filename
```

```
  env.ExtSdkBldSanity(this_sconscript)
except:
  pass

# The query_supported_cmnlib_api and query_supported_krnl_svc APIs can be used
# to check for the presence of required kernel services and exposed commlonlib
# apis. The compilation of a TA can be decided based on the presence of these
# services or commonlib apis. These are some example use cases of these APIs.
if env.QuerySupportedKrnlSvc("SecureCamera2"):
  print("SecureCamera2 Kernel Service is supported.")

if env.QuerySupportedCmnlibApi("qsee_sfs_open"):
  print("qsee_sfs_open commonlib API is exposed.")

# Set this flag to true if tha current TA is chipset independent and doesn't
# require to be rebuilt for every chipset.
env['CHIPSET_INDEPENDENT'] = True

# pac-ret-bkey/bti support has been provided by default to 64-bit TA based on
# armv9 flag configured in sdk_config_<chipset>.cfg
# below reference code for forcing to downgrade security of pac-ret-bkey/bti from TA sconscript.
# env['isarmv9'] = False

# Different app names have to be provided when building TAs for 32 and 64 bits
# since they are treated as different TA compilations and different binaries
# are generated based on the app names provided. Providing the same app name
# for both instances will result in the SCons build infrastructure throwing
# errors that "multiple environments trying to build the same file" for the
# TA metadata files and binaries.
# The 'PROC' flag is set in the TA SCons environment when InitArch is called on the
# construction environment with the TA architecture.
# The 'PROC' flag can assume the values 'scorpion' and 'A53_64' for ontarget compilations
# and 'x86-32' and 'x86-64' for offtarget compilations.
if env['PROC'] == 'scorpion' or env['PROC'] == 'x86-32':
  app_name = 'sample_ta'
else:
  app_name = 'sample_ta64'

# Specifying the app_name to the construction environment so that build artifacts
# are created with the same name as the provided app_name.
env['APP_NAME'] = app_name

# The build artifacts should be compiled in chipset specific directories if the TA is
# chipset dependent otherwise its suggested to compile them in common directories so
# that they are not recompiled for each chipset recompilation.
if env.get('CHIPSET_INDEPENDENT'):
  env['OUT_DIR'] = os.path.join(env['BUILD_ROOT'], 'apps/bsp/trustzone/qsapps/${APP_NAME}/build/')

# List of sources required by TA for compilation. Using relative paths to access
# the sources and headers is the recommended and cleaner approach as compared to
# absolute path access using BUILD_ROOT and SDK_ROOT.
sources = ['./src/sample_source.c',]

# Header directory list required for TA compilation.
includes = ['${SDK_ROOT}/inc/qsee/', '../sample_shipped_lib/inc/', '../sample_compiled_lib/inc/']

# Some TAs require the use of external libraries for their compilation.
# There can be three possibilities to get external libraries for TA compilation
# based on the presence of library sources and SCons:
#
# 1. All library sources, headers and SCons files are available in both internal
#    and customer builds: In this scenario, the library SCons can directly be
#    called from the TA SConscript to generate the library files at compile time.
#
# 2. All Library sources, headers and SCons files are available in internal but
#    not in customer builds but precompiled library files are present in customer builds:
#    For these cases, presence of library SCons files can be checked and if not found,
#    the precompiled library file can be used.
#
```

```
# 3. Library sources, headers and SCons files are not present in either of the internal
#    and customer builds: In these scenarios, precompiled library files are to be directly
#    used in TA SConscript, if precompiled libraries generated libv9(with pac-ret-bkey/bti)
#    and libv8(without pac-ret-bkey/bti) version.
#    NOTE: if libv9 and libv8 versions are not present for precompiled library then disable
#    pac-ret-bkey/bti security feature using env['isarmv9'] = false and use precompiled library directly.

# pac-ret-bkey/bti support has been provided by default to 64-bit library.
# precompiled libv8(without pac-ret-bkey/bti) and libv9(with pac-ret-bkey/bti) library
# binaries are present in both internal and customer builds. For pac-ret-bkey/bti
# features TA scons will pick up precompiled libv9 library for 64-bit target
# based on below if condition checks.
# NOTE: if below condition is not handled properly then TA might crash with Exception syndrome :0x3600000x
armv9 = ''
if env.IsArmv9AppEnvStat():
  armv9 = 'armv9'

libs = []

# All sources, headers and library SCons are present in both internal and customer builds.
libs.append(env.SConscript('../sample_compiled_lib/SConscript', exports='env'))

# Sources, headers and library SCons are only present in internal builds and only precompiled
# library binaries are shipped to customer builds.
# Checking the presence of library SConscript files, calling the SConscript for library
# compilation if found, otherwise appending the precompiled library binaries.
sample_shipped_lib_scons = env.FindFile('SConscript', '../sample_shipped_lib/')
if sample_shipped_lib_scons != None:
  libs.append(env.SConscript(sample_shipped_lib_scons, exports='env'))
else:
  libs.append(os.path.join('${BUILD_ROOT}/apps/bsp/trustzone/qsapps/sample_shipped_lib', armv9, '
      ${PROC}/sample_shipped_lib.lib'))

libs.append(os.path.join('$SDK_ROOT/libs', armv9, '${APP_EXEC_MODE}/ipprot_lib.lib'))

# The metadata argument contains TA specific attributes required for TA compilation.
md = {
 'appName':    app_name,
 'privileges': ['default',],
 'heapSize' : 0x100000,
 'stackSize': 0x040000,
}

# List of files to to be deployed if TA is to be built in customer build.
sconsList = ['SConscript', 'SConstruct']

# Since SCons build environment expects only files to present inside deploy_sources, env.Glob can
# used to extract all files from a directory to be added to deploy_sources.
deploy_sources = [sconsList, env.Glob('inc/*.h'), sources]

# Build the TA from the specified sources and headers with provided attributes.
# TA SConscript must be shipped only when all sources, headers and library files required
# to build the TA are shipped. Shipping incomplete sources or headers with SConstruct or
# SConscript might suggest compilation in customer builds due to the presence of SConscript
# file which may lead to errors due to absence of dependencies.

# Different parameters passed to the builder function:
# 1. sources: Provides the list of sources to be used for compilation.
# 2. includes: Specifies the list of headers to be included.
# 3. user_libs: Add specified libs to support TA compilation.
# 4. image: Specifies the name of the TA according to which binaries will be buit.
# 5. deploy_sources: List of files to be deployed to customer builds.
# 6. deploy_variants: List of directories where the list in deploy_sources is to be shipped.
#    All files listed in deploy_sources and library binaries are shipped to the directories
#    defined in this field. Some examples of deploy_variants are HY11_1 and FEAT-BIN-PlayReady30.

# Use the deploy_sources parameter inside SecureAppBuilder to deploy all required files
# to customer builds as this is the recommended approach for file deployment.
sample_ta_units = env.SecureAppBuilder(
```

```
  sources = sources,
  includes = includes,
  user_libs = libs,
  metadata = md,
  image = app_name,
  # Define the sources, headers or SCons to be shipped to customer builds.
  deploy_sources = deploy_sources,
  # List of customer builds where the specified artifacts/sources are to be shipped.
  deploy_variants = env.GetDefaultPublicVariants()
)

# Set the built TA flag in ARGUMENTS array to True in case TA is chipset independent so
# that other chipset based construction environments can directly read the built TA objects
# and not recompile them.
built_flag = env.subst('${APP_NAME}' + '_BUILT')
if env.get('CHIPSET_INDEPENDENT'):
  ARGUMENTS[built_flag] = True

# Aliasing the TA name with the generated binaries so that any call to TA name will return
# the generated binaries.
op = env.Alias(app_name, sample_ta_units)

# Build artifacts must always be returned using string variables as SCons build environment expects
# the return in this format.
Return('sample_ta_units')
```

8 Trusted Application Clients on a REE

This chapter provides a brief overview of the various components residing on the REE used to communicate with the trusted application and describe the APIs available to a REE client.

8.1 QSEECom driver

QTEE communicator (QSEECom) is a kernel module responsible for all REE communication with QTEE. QSEECom is a platform device driver developed specifically for Linux-based REEs, such as Android. It provides IOCTLs for the user space applications to communicate with QTEE. The driver also exports APIs for kernel space applications to communicate with QTEE on the trusted end. The QSEECom driver uses the SCM interface to switch the context between the REE and QTEE. This driver also enables QTEE to use the services of the REE via a listener service interface.

8.2 QSEEComAPI library

QSEEComAPI is the library that exposes an API for the REE client/listener to load and unload the trusted application, and send and receive data to/from QTEE via the Linux QSEECom driver. The APIs allow access to the QSEECom driver to issue commands and receive responses from QTEE. All REE clients and listeners must use the API exposed in this library to communicate with TAs in QTEE. The APIs are listed and discussed in Section 8.5.1

8.3 QSEECom client

The QSEECom client is the REE application that initiates all requests to trusted applications in QTEE. Initially, it is the client that issues a request to load the trusted application by invoking QSEECom_start_app() and retrieving the handle to the QSEECom character driver. When the trusted application is loaded, the client can use the retrieved handle to send commands or requests to the trusted application.

8.4 QSEECom listener

QSEECom listener is the REE service module that serves requests originating from QTEE. These requests are mostly originated by a client or service residing in QTEE.

QTEE listeners are registered with QSEECom by invoking the QSEECom_register_listener() call. This function in turns calls QSEECOM_IOCTL_REGISTER_LISTENER_REQ. Upon successful registration, the QSEECom driver stores the listener ID in a listener service queue. The current QSEECom design handles registration of multiple listener services.

After registering with the QSEECom driver, each listener service must call QSEECom_receive_req(), which in turn calls QSEECOM_IOCTL_RECEIVE_REQ to start the listener service. One thread from each service is blocked after calling QSEECOM_IOCTL_RECEIVE_REQ. The thread is signaled when QSEECom receives a command containing a particular listener service ID.

All listener services are initiated by a daemon that is run during start-up. The majority of listeners used internally by the existing QTEE APIs are provided by and listed in a daemon called qseecomd. The purpose of this daemon is to register the available listed listeners and initiate the listener service thread that services all requests to the respective listener. Any new listener can be added to the list defined in the qseeecomd daemon and the service must be implemented accordingly.

8.5 QSEECom APIs available to trusted application REE clients/listeners

All REE clients must use the APIs listed in this section to communicate with the QSEECom driver and thereby send and receive information to and from trusted applications residing in QTEE. The API provided in this library interacts with the QSEECom driver.

Note: User space client applications or listener services may not interface directly with the QSEECom driver.

8.5.1 QSEECom APIs available to trusted application REE clients

This section describes the APIs available to REE client applications residing in user space which interact with trusted applications in QTEE.
The guidelines contained in this section refer specifically to specialized REE client APIs modelled on the QSEEComAPI which have been developed for use with the QTEE TA Software Developers Kit.

8.5.2 QSEECom_start_app

This API is called by the client application. The API does the following:

- Loads the application, fname, which resides in the location, path
- Allocates a shared memory buffer of a requested size, sb_size; this buffer is made available for the client application to send data
- Returns the client handle (tied to the application loaded)

Note: Each handle, created as a result of calling QSEECom_start_app(), is unique per process. Handles cannot be shared between processes.

```
int QSEECom_start_app(struct QSEECom_handle **clnt_handle, const char *path,
                      const char *fname, uint32_t sb_size);
```

8.5.3 QSEECom_shutdown_app

This API is called by the client application. The API does the following:

- Unloads the application to which the handle is tied
- De-allocates the shared memory buffer tied to the handle

Note: Unloading a trusted application which is awaiting a listener response is not supported.

```
int QSEECom_shutdown_app(struct QSEECom_handle *handle);
```

8.5.4 QSEECom_send_cmd

This API is called by the client application. The API sends the trusted application in QTEE a user-defined buffer (which may contain some message or command request) and receives a response from the trusted application in the response buffer, rcv_buf.

```
int QSEECom_send_cmd(struct QSEECom_handle *handle, void *send_buf, uint32_t sbuf_len,
                     void *rcv_buf, uint32_t rbuf_len);
```

QTEE is not aware of buffer pointers embedded within the command structure in the send buffer, instead treated as raw data input. This command is a blocking call. The API does the following:

- Sends the command information of sbuf_len in send_buf to the trusted application
- Receives a response to the command request in rcv_buf.

8.5.5 QSEECom_send_modified_cmd and QSEECom_send_modified_cmd_64

These APIs are called by the client application. They send the trusted application in QTEE a user-defined buffer (which may contain some message or command request) and receive a response from the trusted application in the respective buffer.
QSEECom_send_modified_cmd_64 is the 64-bit version of QSEECom_send_modified_cmd used when passing buffers from heap allocations types that are mapped above 4 GB. Heap allocations guaranteed to be mapped below 4 GB can still go through the 32-bit version even in AArch64 non-secure applications. It is unsupported to pass a buffer mapped above 4 GB to an AArch32 trusted application.

```
int QSEECom_send_modified_cmd(struct QSEECom_handle *handle,
                              void *send_buf, uint32_t sbuf_len, void *resp_buf,
                              uint32_t rbuf_len, struct QSEECom_ion_fd_info* ifd_data);

int QSEECom_send_modified_cmd_64(struct QSEECom_handle *handle,
                                 void *send_buf, uint32_t sbuf_len, void *resp_buf,
                                 uint32_t rbuf_len, struct QSEECom_ion_fd_info* ifd_data);
```

This API is the same as send_cmd, except it takes an additional parameter, ifd_data.

```
struct QSEECom_ion_fd_info {
  struct QSEECom_ion_fd_data data[4];
};

struct QSEECom_ion_fd_data {
  int32_t fd;
  uint32_t cmd_buf_offset;
};
```

Each entry in ifd_data, QSEECom_ion_fd_data, contains a representation of an ION buffer which must be allocated by the client.

For each entry in ion_fd_data, QSEECom_ion_fd_data, the QSEECom driver takes the representative ION buffer, copies to it a 4K aligned buffer useable by QTEE and populates the command at the given offset with the aligned buffer address before sending it to trusted application.

QTEE is not aware of buffer pointers embedded within the command structure in the send_buf buffer. The REE client should be aware that there is no cache management by QTEE on these embedded buffer pointers and that it must manage this accordingly.
The REE client writes to send_buf, where the trusted application writes to rcv_buf. This is a blocking call. The API does the following:

- Modifies the information at the mentioned offsets with the addresses of the memory address by the ifd_data
- Sends the modified command information of sbuf_len in the sbuf to the trusted application.
- Receives the response to the command request in the resp_buf

8.6 QSEECom APIs available to trusted application REE listeners

This section covers the APIs available to REE listener services. All listener services are initiated by the daemon. The APIs discussed in this section are available for use in the qseecomd daemon context and should not be used in the client context.

8.6.1 QSEECom_register_listener

This API is called by the REE qseecomd daemon. The API does the following:

- Allocates a shared memory buffer of a requested size, sb_length; this buffer is made available for the listener to receive a request from QTEE and send a response to QTEE
- Registers the existence of the listener service given by the identifier lstnr_id (and the shared buffer allocated to this listener service) in the driver
- Returns the listener handle (tied to the shared buffer allocated for the requested listener ID)

Note: The flags parameter in QSEECom_register_listener() is deprecated and unused.

```
int QSEECom_register_listener(struct QSEECom_handle **handle,
                              uint32_t lstnr_id, uint32_t sb_length, uint32_t flags);
```

8.6.2 QSEECom_unregister_listener

This API is called by the REE qseecomd daemon. The API de-allocates the shared memory buffer tied to the listener ID.

```
int QSEECom_unregister_listener(struct QSEECom_handle *handle);
```

8.6.3 QSEECom_receive_req

This API is called by the listener service thread spawned by the REE qseecomd daemon. The API does the following:

- Receives request information from QTEE. An IOCTL is issued to the QSEECom driver, which is blocked waiting to receive information in the shared buffer. This becomes unblocked when data is available in the shared buffer associated with this listener ID in the handle.
- Returns the request information in buf

```
int QSEECom_receive_req(struct QSEECom_handle *handle, void *buf, uint32_t len);
```

8.6.4 QSEECom_send_resp

This API is called by the service thread spawned by the REE qseecomd daemon. The API sends the data as a response to the request received when the preceding QSEECom_receive_req() was invoked.

```
int QSEECom_send_resp(struct QSEECom_handle *handle, void *send_buf, uint32_t len);
```

8.6.5 QSEECom_send_modfied_resp and QSEECom_send_modified_resp_64

These APIs are the same as QSEECom_send_resp() but contain buffers allocated by the listener service. QSEECom_send_modified_resp_64 is the 64-bit version of QSEECom_send_modified_resp used when passing buffers from heap allocations types that are mapped above 4 GB.

```
int QSEECom_send_modified_resp(struct QSEECom_handle *handle, void *send_buf,
                               uint32_t sbuf_len, struct QSEECom_ion_fd_info *ifd)
```

```
int QSEECom_send_modified_resp_64(struct QSEECom_handle *handle, void *send_buf,
                                  uint32_t sbuf_len, struct QSEECom_ion_fd_info *ifd)
```

8.7 SMCInvoke MINK APIs available to REE clients

8.7.1 SMCInvoke Overview

SMCInvoke is an object-oriented capability-based framework to enable communication between REE clients and trusted applications residing in QTEE, which utilizes MINK invoke objects. MINK object invocation is described in detail in Section 5.3 and its surrounding sections. SMCInvoke is used to expose services and interfaces implemented in QTEE to REE processes, while providing identification information about the requesting processes to QTEE.

Below is a list of types of objects involved in SMCInvoke.

8.7.2 Remote Objects

Remote objects are MINK objects owned by QTEE. REE applications obtain references to these objects via invoke operations. IAppLoader is an example of a remote object.

8.7.3 Callback Objects

Callback objects, unlike remote objects, are owned by the REE. REE client applications send references of required callback objects via invoke operations to QTEE, for example to a remote object residing in a trusted application. These callback objects can be invoked from these references for execution in the REE. IIO can be an example of a callback object.

8.7.4 Memory Objects

Memory objects are also owned by the REE and provide an efficient method of sharing large buffers between QTEE and the REE. IMemRegion is an example of a memory object.

8.8 Loading a Trusted Application via SMCInvoke

This section provides an overview of how a trusted application (TA) is loaded via SMCInvoke by a native REE client application. All invocations from an REE client into the trusted environment begin with an IClientEnv object. A native REE client can use TZCom_getClientEnvObject() to get an IClientEnv object.

The steps involved in loading a TA via SMCInvoke for native REE clients are:

- Map the TA into a buffer or provide an IMemRegion object containing the TA. N.B.: The memory region of the IMemRegion object needs to reside in the ION heap.
- Obtain an IClientEnv object
- Use the IAppLoader object to load the TA and obtain an IAppController object
- Use the IAppController object to get the TA's app object, i.e. the object returned from app_getAppObject().

8.9 APIs available to native REE clients to retrieve an IClientEnv

These APIs are available to native REE client to retrieve an IClientEnv.

- TZCom_getRootEnvObject
- TZCom_getRootEnvObjectWithCB

- TZCom_getClientEnvObject
- TZCom_getFdObject

9 Sample Applications and the QTEE Emulator (QTEEEmu)

This chapter outlines how to setup the off-target toolchain and how to build, run, and debug Trusted Applications (TA) and Client Applications (CA).

Furthermore, APIs specific for off-target testing on the QTEE Emulator (QTEEEmu) and several sample applications suitable for on- and off-target testing are introduced.

9.1 The QTEE Emulator

QTEEEmu provides a way to develop, build, run, and debug trusted applications in an off-target environment.

9.1.1 Linux Users

Ubuntu 16.04 LTS or newer is recommended for off-target QTEE development.

The following packages are required:

- LLVM / Clang toolchain: v6.0 or greater
- SCons: v4.0.1 or greater (requires Python v3.6.9 or greater)
- 64 bit and 32 bit capable libc (e.g. g++-multilib)

These can be installed on Ubuntu as follows:

```
sudo apt-get update
sudo apt-get install clang llvm scons g++-multilib
```

Note: The QTEE off-target toolchain searches the system path for the following compiler binaries: clang, clang++, llvm-ar, llvm-nm, llvm-ranlib, llvm-as, llvm-cov and llvm-profdata.

You need to ensure that the correct version of the each of these binaries is configured as the default and available via the system path. Typically this is achieved using update-alternatives, e.g.:

```
sudo update-alternatives --install /usr/bin/clang clang /usr/lib/llvm-6.0/bin/clang 100
sudo update-alternatives --install /usr/bin/llvm-ar llvm-ar /usr/lib/llvm-6.0/bin/llvm-ar 100
...
```

9.1.2 Windows Users: Windows Subsystem for Linux

Windows 10 users can use Windows Subsystem for Linux (WSL) for offtarget testing on Windows. Set up of WSL is covered comprehensively by Microsoft in the Windows 10 Installation Guide (https://docs.microsoft.com/en-us/windows/wsl/install-win10).
Note that there are alternative instructions using lxrun for those who do not have Windows build 16215 or later.

Ubuntu 16.04 LTS is recommended for off-target QTEE development using WSL.

Once WSL is running, install the required packages as per Section 9.1.1.

9.1.3 QTEEEmu Simulation

QTEEEmu aims to mimic the behavior of QTEE whenever possible. However, not all features of QTEE are supported. This section lists some of the known limitations and differences of QTEEEmu.

- The CA, the TA, and the simulated QTEE kernel share an address space. While the illusion of an address space separation exists when using the correct memory sharing APIs, a CA or TA can potentially access memory not accessible on real hardware.
- Opening unsupported services with qsee_open returns Object_ERROR. Even when a service returns an object, some functions might be stubbed. Whenever possible, an error code is returned.
- QTEEEmu creates an offtarget_fs folder for persistent storage. Should a QTEEEmu execution fail while accessing the simulated storage, the simulated storage can end up in an inconsistent state and should be deleted to avoid issues.
- An off-target TA might need additional stack space due to architectural differences. The stack space determined off-target can therefore only give an approximation of the stack space required for execution on real hardware.
- QTEEEmu does not support mixed architecture execution, e.g. a 32-bit Trusted Application running on 64-bit QTEEEmu.
- CAs and TAs must be built with the same SDK version. As the SDK contains shared libraries used in QTEEEmu, this SDK version must be present at runtime at the same absolute path used at buildtime.
- QTEEEmu registers a signal handler for SIGSEGV to gracefully exit a TA, e.g. if the TA accesses invalid memory. The handler is reserved for QTEEEmu, i.e. neither a CA nor a TA should set their own.
- The off-target C++ libraries might throw exceptions where an on-target, bare-metal C++ library wouldn't. TAs are comiled with -fno-exceptions and -fno-RTTI to align on- and off-target behavior.
- Setting a std::new_handler to handle failed memory allocations affects the whole QTEEEmu process and is therefore undefined behavior.
- QTEEEmu registers a std::terminate_handler to gracefully exit C++ TAs throwing an unhandled exception. Setting a std::terminate_handler is reserved for QTEEEmu, i.e. neither a CA nor a TA should set their own. A CA can however use exceptions if desired.
- Please refer to Services Emulated in QTEEEmu for a list of emulated services available to TA developers.

9.1.4 Off-target Application Logs

Off-target applications support different types of developer logging messages printed to stdout:

- CA logs via printf()
- TA logs via qsee_log()

TA qsee_log() messages are also written out to a unique log file with the prefix "qsee_log_". The full path to the log file is printed to stdout when the application is run, typically this is in the same directory as the executable.

Logcat-like ALOG macros are also provided as an alternative to printf() for CAs in the SDK.

In addition to TA and CA logs, QTEEEmu generates internal tzbsp_log messages, output to a unique log file with the prefix "tzbsp_log_". The logs are not human readable and are for Qualcomm use only.

9.1.5 QTEE Emulator Configuration File

Some of the QTEEEmu settings can be customized using a configuration file. QTEEEmu uses JSON format for its configuration file. Values must be set inside a "config" object, as shown below:

```
{
  "config": {
    "value1": true,
    "value2": "string"
  }
}
```

The following options are available:

- `qseeLogToStdOut` (boolean): If set to true then qsee_log() calls will be printed to stdout as well as the qsee log file. Default = true.
- `checkMemLeaks` (boolean): If set to true then the emulator will check for known memory leaks before shutting down. Default = true.
- `logfilePath` (string): Path to directory that will contain log files. Default = directory that contains the CA executable.
- `offtargetFsPath` (string): Path to directory that will contain the offtarget filesystem. Default = current working directory.

It is not required to set every available value, if an option is not present in the user configuration file the default setting will be used.

See qtee_emu_init() for how to specify the location of a config file.

9.1.6 APIs to initialize the off-target QTEE emulation environment

These APIs to initialize the off-target QTEE emulation environment.

- qtee_emu_deinit
- qtee_emu_init

9.1.7 APIs to map a trusted application into a read-only buffer

These APIs to map a trusted application into a read-only buffer.

- map_trusted_application
- unmap_trusted_application

9.1.8 APIs to create memory objects

These APIs to create memory objects.

- MemObj_getInfo
- MemObj_getPerms
- MemObj_isMapObj
- MemObj_new
- MemObj_setPerms

9.1.9 APIs to create a simulated ION buffer

These APIs to create a simulated ION buffer.

- IonSim_alloc
- IonSim_free

9.1.10 APIs to manage dependencies of GP trusted applications

These APIs to manage dependencies of GP trusted applications. gp_client_imp "APIs" to manage dependencies of GP trusted applications.

- getAppFromUUID

9.1.11 Example Config file for Debugging using Visual Studio Code

The application can be run and debugged using Visual Studio Code. GDB can be used within VSCode to debug C and C++ applications on Linux by:

- Installing Microsoft's C/C++ extension in VSCode (CTRL+Shift+X, C/C++ by Microsoft)
- Installing GDB locally, e.g. via apt-get install on Ubuntu.
- Providing the following launch file, and having GDB available in the system path.

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "(gdb) Example",
      "type": "cppdbg",
      "request": "launch",
      "program": "/path/to/application/x86-64/ca_name.elf",
      "args": [],
      "stopAtEntry": false,
      "cwd": "${workspaceFolder}",
      "environment": [],
      "externalConsole": false,
      "MIMode": "gdb",
      "setupCommands": [
      {
        "description": "Enable pretty-printing for gdb",
        "text": "-enable-pretty-printing",
        "ignoreFailures": true
      }
      ]
    }
  ]
}
```

9.2 Build and Run Sample Applications

Several sample applications can be found under SDK/samples. These samples include:

- A skeleton application for using legacy QSEECom APIs
- A skeleton application for using SMCInvoke APIs
- An SMCInvoke example app showcasing SMCInvoke's features
- C++ variants of some of these applications
- A GP skeleton application

The skeleton applications can be used as a template when developing a new trusted applications. The sample applications showcase additional features.

The basic structure of an application is as follows, taking the SMCInvoke skeleton as an example:

```
smcinvoke_skeleton/
   SConstruct
   CA/
      src/
         smci_ca_main.c
         smci_skeleton.c
         SConscript
    common/
      idl/
        CSMCIExample_open.h
        ISMCIExmaple.idl
   TA/
      src/
         SConscript
         smci_ta_main.c
         CSMCIExample.c
```

'SConstruct' is the entry point for building the CA and TA. It calls into the QTEE SDK to initialize the build environment for TAs and off-target CAs.

/CA/src/ contains the program code for the CA and the SConscript used to build it.

/TA/src/ contains the program code for the TA and the SConscript used to build it.

/common/idl/ contains the IDL file used to define a MINK interface implemented in the TA. IDL headers are autogenerated from the IDL file when included in a SConscript's include path.

Notes:

- SConstruct handles building off-target (x86-32, x86-64) as well as on-target (aarch32, aarch64). The choice is handled in InitArch. The UserModeSupportsArchitecture and UserModePreferredArchitecture functions can be used to decide if a particular target architecture is inappropriate for the specified chipset during the build, while still allowing full off-target testing. Use UserModeSupportsArchitecture to build all supported aarch binaries of TA, use UserModePreferredArchitecture to only build most appropriate.
  Example build script snippets:

```
# This example builds only the most appropriate (32bit or 64bit) version of the TA
for arch in ["aarch32", "aarch64"]:
  env = qtee_sdk_env.Clone()

  # Initialize the environment for the target architecture
  # note: for off-target builds aarchXX is mapped to x86-XX
```

```
  env.InitArch(arch)
  if not env.UserModePreferredArchitecture():
     print "Not building unneeded architecture: ", env["PROC"]
     continue

  <normal build script to build TA>


# This example builds both 32bit and 64bit versions of the TA unless one is unsupported
for arch in ["aarch32", "aarch64"]:
  env = qtee_sdk_env.Clone()

  # Initialize the environment for the target architecture
  # note: for off-target builds aarchXX is mapped to x86-XX
  env.InitArch(arch)
  if not env.UserModeSupportsArchitecture():
     print "Not building unsupported architecture: ", env["PROC"]
     continue

  <normal build script to build TA>
```

9.2.1 Building

Sample applications from the SDK/samples directory can be built with the following command from within a sample directory, changing the last argument as necessary:

```
scons CHIPSET=waipio OFF_TARGET=1 -C /path/to/TA_SConstruct BUILD_ROOT=/path/to/build_root
smcinvoke_skeleton_ca
```

where:

- CHIPSET=waipio
  When building a TA for on-target use, the CHIPSET must match the target hardware.

  Off-target TAs are not dependent on the chipset setting, however, a valid CHIPSET argument must be provided to SCons. For QTEE 5.0, 'waipio' is a suitable value.
- OFF_TARGET=1
  This specifies that the application should be built for off-target testing. Omitting this will attempt to build for on-target hardware.
- -C /path/to/TA_SConstruct
  This changes the compilation directory to the TA directory.
- BUILD_ROOT=/path/to/build/root
  This path specifies the build root and determines the top level of a build.

  Output binaries will be placed under BUILD_ROOT/build/ms
- smcinvoke_skeleton_ca
  This is the name of the target to build. Specifying 'smcinvoke_skeleton_ta' will build the TA only. Specifying 'smcinvoke_skeleton_ca' will build the QTEEEmu CA and the TA as a dependency if building with OFF_TARGET=1.

9.2.1.1 Building for Off-target Testing

When built with OFF_TARGET=1, the examples produce both 32-bit and 64-bit CA executables, which load the corresponding 32/64-bit version of their compiled TA.

The output executables are found at

```
SDK/samples/<example>/out/x86-[32|64]/<example_ca>.elf
```

Where <example> is the name of the example application you are building.

While the CA code is generic and can be easily adapted to REEs like Android, some QTEEEmu specific APIs are used, e.g. to initialize the emulation environment. Where possible and to ease porting, these QTEEEmu specific APIs have been placed in their own file.

9.2.1.2 Building for Off-target Testing Leak Detection

When built with OFFTARGET_ASAN=1 and OFFTARGET_DEBUG_LEAK=1. the examples produce 64-bit CA and TA with ASAN sanitizer enablement.

```
scons CHIPSET=waipio OFF_TARGET=1 -C /path/to/TA_SConstruct BUILD_ROOT=/path/to/build_root
smcinvoke_skeleton_ca OFFTARGET_ASAN=1 OFFTARGET_DEBUG_LEAK=1
```

where:

- OFFTARGET_ASAN=1
  This specifies that all the source will be compile with ASAN sanitizer flags.
- OFFTARGET_DEBUG_LEAK=1
  This specifies that linker will pick standard libc heap memory allocaton implementation instead of custom heap memory allocation implementaion for TA.
- Other fields are already described in Section 9.2.1
- NOTE: Currently, sanitizers are only enabled for x86-64 due to an issue with i386 libc.

The output executables can be executed to detect leaks in code and same can be found at

```
SDK/samples/<example>/out/x86-64/<example_ca>.elf
```

Where <example> is the name of the example application you are building.

9.2.1.3 Building for Devices

When a TA is built without specifying OFF_TARGET=1, both 32-bit and 64-bit versions of the TA are built for real hardware.

On-target binaries are found at:

```
SDK/build/ms/bin/WAPIONAA/<example>_ta64.mbn          [signed mbn]
SDK/build/ms/bin/WAPIONAA/signed/<example>_ta64.mbn
SDK/build/ms/bin/WAPIONAA/signed_encrypted/<example>_ta64.mbn
SDK/build/ms/bin/WAPIONAA/unsigned/<example>_ta64.mbn
```

9.2.2 Running the Application

To run the off-target CA, simply execute the CA elf binary from the relevant sample directory, e.g.

```
./out/x86-64/smcinvoke_skeleton_ca.elf
```

9.3 Sample Applications

9.3.1 QSEECom Skeleton App

The QSEECom skeleton app demonstrates the very basics of using the legacy QSEECom APIs (see Section 8.5.1) to load a TA.

When using QSEECom APIs to send a command to a TA, the TA needs to implement the tz_app_cmd_handler(). To communicate with the TA, a QSEECom CA uses QSEECom_send_cmd to send a pair of request/response structs to the TA's tz_app_cmd_handler().

In the skeleton CA, a command ID is used within the request struct. Upon receiving the request struct, the TA checks the lengths of the request and response struct and processes the command associated with the command ID.

Below is the skeleton TA's implementation of tz_app_cmd_handler(). This is an excerpt of the full code listing in later sections. The TA casts the void∗ pointers received from QSEE to the correct struct types, and then handles the cmd_id from the CA.

```
/* tz_app_cmd_handler() is the entry point for a TA to handle a command from
 a Client Application.*/
void tz_app_cmd_handler(void *cmd, uint32_t cmdlen, void *rsp, uint32_t rsplen)
{
  /* Cast the received command/response to the expected message type */
  struct cmd_req_t *cmd_ptr = (struct cmd_req_t *)cmd;
  struct cmd_rsp_t *rsp_ptr = (struct cmd_rsp_t *)rsp;

  /* Basic error check to ensure command/response are correct message types */
  if (cmdlen < sizeof(struct cmd_req_t) || rsplen < sizeof(struct cmd_rsp_t))
  {
    return;
  }

  qsee_log(QSEE_LOG_MSG_DEBUG, "skeleton TA received cmd_id %d", cmd_ptr->cmd_id);

  /* Handle the command based on cmd_id */
  switch (cmd_ptr->cmd_id) {
    case SAMPLE_CMD:
      rsp_ptr->status = SUCCESS;
      break;
    default:
      rsp_ptr->status = FAILURE;
      break;
  }
}
```

9.3.2 SMCInvoke Skeleton App

9.3.2.1 SMCInvoke Skeleton Overview

The SMCInvoke skeleton shows the basics of loading and using a TA via SMCInvoke, as described in Section 8.7.1.

When using SMCInvoke to communicate with a TA, a TA must implement app_getAppObject() returning a MINK interface. In the skeleton TA, a MINK interface 'SMCIExample' is returned.

The below code excerpt shows the SMCInvoke skeleton TA's implementation of app_getAppObject(), as well as the interface's open() function.

```
/* using app_getAppObject() we return an object for an interface
 * "ISMCIExample", which  we've defined in ISMCExample.idl. The CA gets
 * to this point using the IAppController interface; specifically
 * IAppController_getAppObject(). The CA implementation shows an example
 * of the steps involved in reaching this point. */
 int32_t app_getAppObject(Object credentials, Object *obj_ptr)
 {
    (void)credentials;
    return CSMCIExample_open(obj_ptr);
 }

/* Function declaration for the invoke function of the ISMCIExample interface. */
static ISMCIExample_DEFINE_INVOKE(CSMCIExample_invoke, CSMCIExample_, CSMCIExample *);

int32_t CSMCIExample_open(Object *objOut)
{
  /* we can use this index to identify specific instances of ISMCIExample. */
  static int global_index = 0;

  CSMCIExample *me = QSEE_ZALLOC_TYPE(CSMCIExample);
  if (!me) {
    qsee_log(QSEE_LOG_MSG_ERROR, "Memory allocation for CSMCIExample failed!");
    return Object_ERROR_MEM;
  }

  me->refs = 1;
  me->index = global_index++;

  qsee_log(QSEE_LOG_MSG_DEBUG, "ISMCIExample instance number %d!", me->index);
  *objOut = (Object){CSMCIExample_invoke, me};
  return Object_OK;
}
```

9.3.2.2 Application Credentials

Off-target, a simulated set of credentials are generated inside the TZCom APIs. The off-target environment simulates a trusted client environment, with System_UID permissions.

9.3.3 C++ SMCInvoke Skeleton App

The SDK also provides a C++ variant of the SMCInvoke skeleton, showing how to use the C++ language mappings shown in Section 5.3.5

9.3.4 SMCInvoke Example App

The SMCInvoke example app provides examples for several features of SMCInvoke-based applications.

9.3.5 Example Service TA

In the SMCInvoke example app, we host a service "IAdder" from a second 'service' TA. The service can then be used if the TA is loaded. More information on TA services can be found in Section 5.1.9.1.

- The CA loads the main TA, which implements ISMCIExampleApp.
- The CA also loads the service TA, which implements IAdder.
- The TA calls qsee_open() with IAdder's UID from CAdder.h.
- QSEE routes this request to the Service TA and calls its implementation of tz_module_open() with this UID.
- The service TA returns a reference to an object of the IAdder implementation.

9.3.5.1 TA Properties

Additional TA properties are required for TAs to be able to host or use services outside of the QTEE provided services.

A TA's properties (see Section 6.2.1) are defined in its SConscript.

9.3.5.2 Privileges

smci_example_ta SConscript:

```
md = {
'appName':      target,
'privileges': [ 'default', 'Adder'],
}
```

In the SMCInvoke example app, the main TA opens the IAdder service. Therefore, in addition to the default privilege set, the Adder privilege is required.

When adding services to the SConscript in the privileges or services section, the 'I' is omitted from the name, i.e 'IAdder' becomes 'Adder'.

9.3.5.3 Services

smci_example_svc_ta SConscript:

```
md = {
'appName':      target,
'privileges': [ 'default' ],
'services': ['Adder'],
}
```

The 'services' line is used to indicate that a TA hosts a service. The name of the service (again, without the 'I') is added to the services list in the SConscript, to add it to the TA's metadata.

9.3.6 Callback Objects

The SMCInvoke Example App also demonstrates the usage of a callback object, ICallbackObjectExample. In the example app, we show the interface ICallbackObjectExample implemented as a callback object in the CA. ICallbackObjectExample has one method, ICallbackObectExample_print, which prints a message to stdout. The ISMCIExampleApp interface implements the method ISMCIExampleApp_callbackObject Example, which takes an ICallbackObjectExample object as an argument and invokes its print method. As the TA is invoking a callback object which resides in the client side, the method is executed in the REE.

9.3.7 QSEECom Skeleton CA Source Listing

```
/*
 * Copyright (c) 2018-2020 Qualcomm Technologies, Inc.
 * All Rights Reserved.
 * Confidential and Proprietary - Qualcomm Technologies, Inc.
 */

#include "qtee_init.h"    // QTEE off-target environment initialization
#include "QSEEComAPI.h"   // QSEEComAPI Library header
#include "skeleton_app.h" // Common header between CA and TA
#include <stddef.h>
#include <stdio.h>
#include <stdint.h>

int main (void)
{
  struct QSEECom_handle *app_handle = NULL; // The application handle.
  static char const ta_name[] = SKELETON_TA_NAME; // The TA name.
```

```
  const uint32_t shared_buffer_len = 4096;
  int retval = -1;

  struct cmd_req_t req = {0};
  struct cmd_rsp_t rsp = {0};

  /*
   * Initialize QTEEEmu.
   *
   * A null pointer can be passed to use the default settings. Alternatively, a path to a JSON file
   * specifying desired settings can be provided. See 'qtee_init.h'.
   */
  retval = qtee_emu_init(NULL);
  if(retval) {
    printf("qtee_emu_init() failed\n");
    return retval;
  }

  // Load the TA. TA_PATH_DEFINE is defined in the offtarget client app builder.
  retval = QSEECom_start_app(&app_handle, TA_PATH_DEFINE, ta_name, shared_buffer_len);
  if(retval != QSEECOM_SUCCESS) {
    printf("QSEECom_start_app() failed\n");
    goto bail;
  }

  // Send a command to the TA
  req.cmd_id = SAMPLE_CMD;
  retval = QSEECom_send_cmd(app_handle, &req, sizeof(struct cmd_req_t), &rsp, sizeof(struct cmd_rsp_t));
  printf("QSEECom_send_cmd() returned %d\n", retval);

  // rsp is only valid if QSEECom_send_cmd returned successfully
  if(retval == QSEECOM_SUCCESS) {
    printf("TA returned rsp_status %d\n", rsp.status);
  }

  // Shutdown the TA, even if QSEECom_send_cmd failed
  QSEECom_shutdown_app(&app_handle);

bail:
  // Deinitialize the QTEE off-target environment
  qtee_emu_deinit();
  return retval;
}
```

9.3.8 QSEECom Skeleton TA Source Listing

```
/*
 * Copyright (c) 2018 Qualcomm Technologies, Inc.
 * All Rights Reserved.
 * Confidential and Proprietary - Qualcomm Technologies, Inc.
 */

#include "qsee_log.h"
#include "qsee_ta_entry.h"
#include "skeleton_app.h" // Common header between CA and TA
#include <stdint.h>

/* tz_app_init() is called on application start.
   Any initialization required before the TA is ready to handle commands
   should be placed here. */
void tz_app_init(void)
{
  /* Get the current log mask */
  uint8_t log_mask = qsee_log_get_mask();

  /* Enable debug level logs */
  qsee_log_set_mask(log_mask | QSEE_LOG_MSG_DEBUG);
  qsee_log(QSEE_LOG_MSG_DEBUG, "App Start");
}
```

```
/* tz_app_shutdown() is called on application shutdown.
   Any deinitialization required before the TA is unloaded should be placed here. */
void tz_app_shutdown(void)
{
  qsee_log(QSEE_LOG_MSG_DEBUG, "App shutdown");
}

/* tz_app_cmd_handler() is the entry point for a TA to handle a command from
   a Client Application.*/
void tz_app_cmd_handler(void *cmd, uint32_t cmdlen, void *rsp, uint32_t rsplen)
{
  /* Cast the received command/response to the expected message type */
  struct cmd_req_t *cmd_ptr = (struct cmd_req_t *)cmd;
  struct cmd_rsp_t *rsp_ptr = (struct cmd_rsp_t *)rsp;

  /* Basic error check to ensure command/response are correct message types */
  if (cmdlen < sizeof(struct cmd_req_t) || rsplen < sizeof(struct cmd_rsp_t))
  {
    return;
  }

  qsee_log(QSEE_LOG_MSG_DEBUG, "skeleton TA received cmd_id %d", cmd_ptr->cmd_id);

  /* Handle the command based on cmd_id */
  switch (cmd_ptr->cmd_id) {
    case SAMPLE_CMD:
      rsp_ptr->status = SUCCESS;
      break;
    default:
      rsp_ptr->status = FAILURE;
      break;
  }
}
```

9.3.9 QSEECom Skeleton C++ CA Source Listing

```
/*
 * Copyright (c) 2019-2020 Qualcomm Technologies, Inc.
 * All Rights Reserved.
 * Confidential and Proprietary - Qualcomm Technologies, Inc.
 */

#include <cstdint>
#include <cstdio>

#include "QSEEComAPI.h"   // QSEEComAPI Library header
#include "skeleton_cppapp.h" // Common header between CA and TA

extern "C" {
#include "qtee_init.h"  // QTEE off-target environment initialization
}

int main (void)
{
  struct QSEECom_handle *app_handle = NULL; // The application handle.
  static char const ta_name[] = SKELETON_TA_NAME; // The TA name.

  const uint32_t shared_buffer_len = 4096;
  int retval = -1;

  struct cmd_req_t req = {0};
  struct cmd_rsp_t rsp = {0};

  /*
   * Initialize QTEEEmu.
   *
   * A null pointer can be passed to use the default settings. Alternatively, a path to a JSON file
   * specifying desired settings can be provided. See 'qtee_init.h'.
```

```
   */
  retval = qtee_emu_init(nullptr);
  if(retval) {
    printf("qtee_emu_init() failed\n");
    return retval;
  }

  // Load the TA. TA_PATH_DEFINE is defined in the offtarget client app builder.
  retval = QSEECom_start_app(&app_handle, TA_PATH_DEFINE, ta_name, shared_buffer_len);
  if(retval != QSEECOM_SUCCESS) {
    printf("QSEECom_start_app() failed\n");
    goto bail;
  }

  // Send a command to the TA
  req.cmd_id = SAMPLE_CMD;
  retval = QSEECom_send_cmd(app_handle, &req, sizeof(struct cmd_req_t), &rsp, sizeof(struct cmd_rsp_t));
  printf("QSEECom_send_cmd() returned %d\n", retval);

  // rsp is only valid if QSEECom_send_cmd returned successfully
  if(retval == QSEECOM_SUCCESS) {
    printf("TA returned rsp_status %d\n", rsp.status);
  }

  // Shutdown the TA, even if QSEECom_send_cmd failed
  QSEECom_shutdown_app(&app_handle);

bail:
  // Deinitialize the QTEE off-target environment
  qtee_emu_deinit();
  return retval;
}
```

9.3.10 QSEECom Skeleton C++ TA Source Listing

```
/*
 * Copyright (c) 2019 Qualcomm Technologies, Inc.
 * All Rights Reserved.
 * Confidential and Proprietary - Qualcomm Technologies, Inc.
 */
extern "C"{
#include "qsee_log.h"
}
#include "skeleton_cppapp.h" // Common header between CA and TA
#include <stdint.h>

extern "C"{
  void tz_app_init(void);
  void tz_app_shutdown(void);
  void tz_app_cmd_handler(void *cmd, uint32_t cmdlen, void *rsp, uint32_t rsplen);
}

/* tz_app_init() is called on application start.
   Any initialization required before the TA is ready to handle commands
   should be placed here. */
void tz_app_init(void)
{
  /* Get the current log mask */
  uint8_t log_mask = qsee_log_get_mask();

  /* Enable debug level logs */
  qsee_log_set_mask(log_mask | QSEE_LOG_MSG_DEBUG);
  qsee_log(QSEE_LOG_MSG_DEBUG, "App Start");
}

/* tz_app_init() is called on application shutdown.
   Any deinitialization required before the TA is unloaded should be placed here. */
void tz_app_shutdown(void)
{
```

```
  /* Called on application shutdown. Can be used for any deinitialization required. */
  qsee_log(QSEE_LOG_MSG_DEBUG, "App shutdown");
}

/* tz_app_cmd_handler() is the entry point for a TA to handle a command from
   a Client Application.*/
void tz_app_cmd_handler(void *cmd, uint32_t cmdlen, void *rsp, uint32_t rsplen)
{
  /* Cast the received command/response to the expected message type */
  struct cmd_req_t *cmd_ptr = (struct cmd_req_t *)cmd;
  struct cmd_rsp_t *rsp_ptr = (struct cmd_rsp_t *)rsp;

  /* Basic error check to ensure command/response are correct message types */
  if (cmdlen < sizeof(struct cmd_req_t) || rsplen < sizeof(struct cmd_rsp_t))
  {
    return;
  }

  qsee_log(QSEE_LOG_MSG_DEBUG, "skeleton cpp TA received cmd_id %d", cmd_ptr->cmd_id);

  /* Handle the command based on cmd_id */
  switch (cmd_ptr->cmd_id) {
    case SAMPLE_CMD:
      rsp_ptr->status = SUCCESS;
      break;
    default:
      rsp_ptr->status = FAILURE;
      break;
  }
}
```

9.3.11 SMCInvoke Skeleton CA Source Listing

```
/*
 * Copyright (c) 2019, 2023 Qualcomm Technologies, Inc.
 * All Rights Reserved.
 * Confidential and Proprietary - Qualcomm Technologies, Inc.
 */

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

#include "CAppLoader.h"
#include "IAppController.h"
#include "IAppLoader.h"
#include "IClientEnv.h"
#include "ISMCIExample.h"
#include "TZCom.h"
#include "map_trusted_application.h"
#include "object.h"

#include "alog.h"
#include "smci_skeleton.h"

/* Similar to Android LOG_TAG, define a tag that appears when logging from this
 * application */
static char const LOG_TAG[] = "SMCInvoke_skeleton_CA";

/* This function demonstrates how to open a TA using SMCInvoke APIs. */
int32_t run_smcinvoke_ta_example(ta_image_data *img_data)
{
  int32_t ret = Object_OK;

  Object clientEnv = Object_NULL;      // A Client Environment that can be used to
                                       // get an IAppLoader object
  Object appLoader = Object_NULL;      // IAppLoader object that allows us to load
                                       // the TA in the trusted environment
  Object appController = Object_NULL;  // AppController contains a reference to
```

```
                                        // the app itself, after loading
  Object appObj = Object_NULL;          // An interface to our TA that allows us to send
                                        // commands to it.
  uint32_t val1 = 2;
  uint32_t val2 = 5;
  uint32_t addResult = 0;

  /* Before we can obtain an AppLoader object, a ClientEnv object is required
   * from the emulated TZ daemon. */
  ret = TZCom_getClientEnvObject(&clientEnv);

  if (Object_isERROR(ret)) {
    ALOGE("Failed to obtain clientenv from TZCom!");
    clientEnv = Object_NULL;
    goto cleanup;
  }

  /* Using the ClientEnv object we retrieved, obtain an appLoader by
   * specifying its UID */
  ret = IClientEnv_open(clientEnv, CAppLoader_UID, &appLoader);
  if (Object_isERROR(ret)) {
    ALOGE("Failed to get apploader object with %d!", ret);
    appLoader = Object_NULL;
    goto cleanup;
  }

  ALOGV("Succeeded in getting apploader object.");

  /* load the application */
  ret = IAppLoader_loadFromBuffer(appLoader, img_data->buffer, img_data->
      buffer_size, &appController);
  if (Object_isERROR(ret)) {
    ALOGE("Loading the application failed with %d!", ret);
    appController = Object_NULL;
    goto cleanup;
  }

  ALOGV("Loading the application succeeded.");

  ret = IAppController_getAppObject(appController, &appObj);
  if (Object_isERROR(ret)) {
    ALOGE("Getting the application object failed with %d!", ret);
    appObj = Object_NULL;
    goto cleanup;
  }

  ALOGV("Getting the application object succeeded.");

  /* run the ISMCIExample_add function from the ISMCIExample interface. */
  ret = ISMCIExample_add(appObj, val1, val2, &addResult);

  if (Object_isERROR(ret)) {
    ALOGE("Addition returned error %d!", ret);
  } else {
    ALOGD("Add result: %d", addResult);
  }

cleanup:
  Object_ASSIGN_NULL(appObj);
  Object_ASSIGN_NULL(appController);
  Object_ASSIGN_NULL(appLoader);
  Object_ASSIGN_NULL(clientEnv);
  return ret;
}
```

9.3.12 SMCInvoke Skeleton TA Source Listing

```
/*
 * Copyright (c) 2019, 2021 Qualcomm Technologies, Inc.
 * All Rights Reserved.
 * Confidential and Proprietary - Qualcomm Technologies, Inc.
 */
#include <stddef.h>
#include <stdint.h>

#include "CSMCIExample_open.h"
#include "ISMCIExample_invoke.h"
#include "object.h"
#include "qsee_heap.h"
#include "qsee_log.h"

typedef struct CSMCIExample {
  int refs;
  int index;
} CSMCIExample;

/* Release is called to decrement the reference counter of this object. When the
 * reference count reaches 0 (i.e. everything retaining a reference to it has
 * called release), the object is freed. */
static int32_t CSMCIExample_release(CSMCIExample *me)
{
  if (--me->refs == 0) {
    qsee_log(QSEE_LOG_MSG_DEBUG, "Freeing last instance of index: %d", me->index);
    QSEE_FREE_PTR(me);
  }
  return Object_OK;
}

/* When retain is called, this ISMCIExample object's reference count is
 * incremented. This would be called when keeping a new reference to this
 * object. */
static int32_t CSMCIExample_retain(CSMCIExample *me)
{
  me->refs++;
  return Object_OK;
}

/* A simple example showing how to add two values inside the TA and give the
 * result back to the CA. */
static int32_t CSMCIExample_add(CSMCIExample *me,
                                uint32_t val1,
                                uint32_t val2,
                                uint32_t *result_ptr)
{
  qsee_log(QSEE_LOG_MSG_DEBUG, "CSMCIExample_add called for instance index %d.", me->index);

  /* Here we use saturated math to ensure that our value doesn't overflow. */
  if (val1 >= UINT32_MAX - val2) {
    *result_ptr = UINT32_MAX;
  }

  else {
    *result_ptr = val1 + val2;
  }

  return Object_OK;
}

/* Function declaration for the invoke function of the ISMCIExample interface.
 */
static ISMCIExample_DEFINE_INVOKE(CSMCIExample_invoke, CSMCIExample_, CSMCIExample *);

int32_t CSMCIExample_open(Object *objOut)
{
  /* we can use this index to identify specific instances of ISMCIExample. */
```

```
  static int global_index = 0;

  CSMCIExample *me = QSEE_ZALLOC_TYPE(CSMCIExample);
  if (!me) {
    qsee_log(QSEE_LOG_MSG_ERROR, "Memory allocation for CSMCIExample failed!");
    return Object_ERROR_MEM;
  }

  me->refs = 1;
  me->index = global_index++;

  qsee_log(QSEE_LOG_MSG_DEBUG, "ISMCIExample instance number %d!", me->index);
  *objOut = (Object){CSMCIExample_invoke, me};
  return Object_OK;
}
```

9.3.13 SMCInvoke Skeleton C++ CA Source Listing

```
/*
 * Copyright (c) 2019 Qualcomm Technologies, Inc.
 * All Rights Reserved.
 * Confidential and Proprietary - Qualcomm Technologies, Inc.
 */

#include <cstddef>
#include <cstdint>
#include <cstdio>

#include "CAppLoader.hpp"
#include "IAppController.hpp"
#include "IAppLoader.hpp"
#include "IClientEnv.hpp"
#include "ISMCIExample.hpp"
#include "TZCom.h"
#include "object.h"

#include "alog.h"
#include "map_trusted_application.h"
#include "smci_skeleton.h"

/* Similar to Android LOG_TAG, define a tag that appears when logging from this
 * application */
static char const LOG_TAG[] = "SMCInvoke_skeleton_CA";

/* This function demonstrates how to open a TA using SMCInvoke APIs. */
int32_t run_smcinvoke_ta_example(ta_image_data *img_data)
{
  int32_t ret = Object_OK;
  IAppLoader iAppLoader;          // IAppLoader object that allows us to load
                                  // the TA in the trusted environment
  IAppController iAppController;  // AppController contains a reference to
                                  // the app itself, after loading
  ISMCIExample iSMCIExample;

  uint32_t val1 = 2;
  uint32_t val2 = 5;
  uint32_t addResult;

  do {
    /* Before we can obtain an AppLoader object, a ClientEnv object is required
     * from the emulated TZ daemon. */
    Object clientEnv = Object_NULL;
    ret = TZCom_getClientEnvObject(&clientEnv);
    if (Object_isERROR(ret)) {
      ALOGE("failed to get clientEnv object with error %d!", ret);
      break;
    }

    IClientEnv iClientEnv(clientEnv);
```

```
    clientEnv = Object_NULL;  // reference was consumed by IClientEnv instantiation

    /* Using the clientEnv object we retrieved, obtain an appLoader by
     * specifying its UID */
    ret = iClientEnv.open(CAppLoader_UID, iAppLoader);
    if (Object_isERROR(ret)) {
      ALOGE("Failed to get apploader object with %d!", ret);
      break;
    }

    ALOGV("Succeeded in getting apploader object.");

    ret = iAppLoader.loadFromBuffer(img_data->buffer, img_data->
      buffer_size, iAppController);
    if (Object_isERROR(ret)) {
      ALOGE("Loading app failed!");
      break;
    }

    ALOGV("Loading the application succeeded.");

    ret = iAppController.getAppObject(iSMCIExample);
    if (Object_isERROR(ret)) {
      ALOGE("Getting the application object failed with %d!", ret);
      break;
    }

    ALOGV("Getting the application object succeeded.");

    ret = iSMCIExample.add(val1, val2, &addResult);
    if (Object_isERROR(ret)) {
      ALOGE("Addition returned error %d!", ret);
    } else {
      ALOGD("Add result: %d", addResult);
    }
  } while (0);

  return ret;
}
```

9.3.14 SMCInvoke Skeleton C++ TA Source Listing

```
/*
 * Copyright (c) 2019, 2021 Qualcomm Technologies, Inc.
 * All Rights Reserved.
 * Confidential and Proprietary - Qualcomm Technologies, Inc.
 */

#include <cstddef>
#include <cstdint>
#include <new>

#include "CSMCIExample_open.h"
#include "ISMCIExample_invoke.hpp"
#include "object.h"

extern "C" {
#include "qsee_log.h"
}

class CSMCIExample : public ISMCIExampleImplBase
{
 public:
  CSMCIExample();
  virtual ~CSMCIExample();

  virtual int32_t add(uint32_t int1_val, uint32_t int2_val, uint32_t *result_ptr);

 private:
```

```
  /*
   * We use a simple instance counter to create an index for ISMCIExample objects.
   * We never reset its value in this example.
   */
  int index;
  static int instances;

};

int CSMCIExample::instances = 0;

CSMCIExample::CSMCIExample() : index(instances++)
{
  qsee_log(QSEE_LOG_MSG_DEBUG, "Creating new instance with index: %d", index);
}

CSMCIExample::~CSMCIExample()
{
  qsee_log(QSEE_LOG_MSG_DEBUG, "Freeing last reference to instance with index %d", index);
}

int32_t CSMCIExample::add(uint32_t val1, uint32_t val2, uint32_t *result_ptr)
{
  qsee_log(QSEE_LOG_MSG_DEBUG, "CSMCIExample::add called for instance index %d", index);

  /* We use saturated math to ensure that the addition doesn't overflow. */
  if (val1 >= UINT32_MAX - val2) {
    *result_ptr = UINT32_MAX;
  } else {
    *result_ptr = val1 + val2;
  }

  return Object_OK;
}

int32_t CSMCIExample_open(Object *objOut)
{
  CSMCIExample *me = new (std::nothrow) CSMCIExample();
  if (!me) {
    qsee_log(QSEE_LOG_MSG_ERROR, "Memory allocation for CSMCIExample failed!");
    return Object_ERROR_MEM;
  }

  *objOut = (Object){ImplBase::invoke, me};
  return Object_OK;
}
```

9.4 Services Emulated in QTEEEmu

The following MINK classes are emulated in QTEEEmu and available to TA developers. Please refer below link for more details.

- CAppMessage
- CCipher
- CClockConfig
- CCredentials
- CSWCrypto
- CDataCache
- CHash
- CHmac

- CListener
- CMemRegionPermEscalator
- COEMBuf
- CPrngSource
- CSecureChannel
- CSharedBuffer
- CSync
- CUptime

10 Deprecated List

Global qsee_is_ns_range (const void *start, uint32_t len)

This function is deprecated. QTEE kernel performs all required checks on memory ownership and accessibility before mapping it into a TA address space.

Global qsee_read_serial_num (void)

This function is deprecated. Returns only 32 bit serial_num. Use readSerialNum() exposed by CDeviceID QTEE service to get 64bit serial number.

11 Module Index

11.1 Modules

Here is a list of all modules:

12 Hierarchical Index

12.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

13 Data Structure Index

13.1 Data Structures

Here are the data structures with brief descriptions:

14 Module Documentation

14.1 TA entry points

14.1.1 Detailed Description

14.1.2 Function Documentation

14.1.2.1 int32_t app_disconnectClient (Object *credentials*)

Disconnect client from the TA while leaving the TA loaded.

Parameters

in	*credentials*	Object describing the credentials of the client requesting to disconnect from the TA.

Returns

Object_OK if successful.

14.1.2.2 int32_t app_getAppObject (Object *credentials,* Object * *obj_ptr*)

Returns an object of an interface the TA implements.

The CA gets to this point using the IAppController interface; specifically IAppController_getAppObject().

Parameters

in	*credentials*	An object describing the credentials used to request the object.
out	*obj_ptr*	An instance of the requested class.

Returns

Object_OK if successful.

14.1.2.3 void tz_app_cmd_handler (void * *req,* unsigned int *reqlen,* void * *rsp,* unsigned int *rsplen*)

The legacy entry point for a TA to handle a command from a CA.

Newer TAs wouldn't implement this API, they would implement app_getAppObject().

Parameters

in,out	*req*	Pointer to the request buffer.
in	*reqlen*	Its length.
in,out	*rsp*	Pointer to the response buffer.
in	*rsplen*	Its length.

14.1.2.4 void tz_app_init (void)

Called by QTEE on application start.

Any initialization required before the TA is ready to handle commands should be placed here.

14.1.2.5 void tz_app_shutdown (void)

Called by QTEE on application shutdown.

Any deinitialization required before the TA is unloaded should be placed here.

14.1.2.6 int32_t tz_module_open (uint32_t *uid,* Object *credentials,* Object * *obj_ptr*)

Returns an object of an interface a TA hosts.

A TA hosting services would need to implement this function.

Parameters

in	*id*	The requested class ID.
in	*credentials*	An object describing the credentials used to request the object.
out	*obj_ptr*	An instance of the requested class.

Returns

Object_OK if successful.

14.2 Legacy QTEE Service Interfaces

14.2.1 Detailed Description

Modules

- Address Translation
- Bulletin Board
- Cipher
- Clock
- CMAC
- Configuration
- Core
- Crypto Hardware Lock
- Data Cache Maintenance
- Elliptic Curve Cryptography
- QTEE Env
- Embedded Secure Element
- FIPS Services
- Filesystem
- Fuses
- HASH
- HMAC
- Heap Allocation
- I2C
- Interrupts
- KDF
- Key Manager
- Logging
- Message Passing
- OEM Buffer
- OEM Utilities
- Pseudo Random Number Generator
- Public Key Algorithms
- RSA

- Secure Camera
- Secure Channel
- Secure Display
- Services
- SFS
- SPI
- Storage
- Synchronization
- String Comparison
- TA Stack Usage Profiling
- Time
- Timer
- TLMN
- Unified AES
- Unified DES
- Unified PBKDF2
- Unified SHA

14.3 Address Translation

14.3.1 Detailed Description

14.3.2 Function Documentation

14.3.2.1 int qsee_trans_ns_addr (qsee_transm_t *tm,* uintptr_t *va,* uintptr_t * *pa*)

Translates a nonsecure virtual address to a physical address.

Parameters

in	*tm*	The processor state that should attempt translation. This affects translation outcome depending on memory access permission properties.
in	*va*	The virtual address to translate.
out	*pa*	The translation physical address result.

Note

This API truncates addresses to the page boundary. The PA result is an address on the page boundary.

Warning

Only EL1 translation methods are supported by the API.
This API only translates the given VA as NS EL1, it does _not_ confirm the resulting PA is accessible by NS EL1.

Returns

E_SUCCESS - Success
QSEE_TRANS_NS_ADDR_ERROR - Error generated from MINK IPC
QSEE_TRANS_NS_ADDR_ERROR_UNSUPPORTED_TRANSLATION_TYPE - Error when EL0 translation type is requested
QSEE_TRANS_NS_ADDR_ERROR_TRANSLATION_FAULT - Requested translation result is a translation fault

14.4 Bulletin Board

14.4.1 Detailed Description

14.4.2 Function Documentation

14.4.2.1 int qsee_bulletin_board_post (const qsee_bulletin_board_note * *note*)

Posts a note to the QTEE bulletin board, under a specific category.

Each category of the QTEE bulletin board only holds the most recent note that was posted to that category, overwriting the previous one. Notes cannot be deleted from the bulletin board and are available until they are overwritten by a newer note or the device reboots.

Note: No permission model is applied on the bulletin board. Every QTEE application can post notes under any category.

Parameters

in	*note*	Pointer to a structure that holds the note to post: • note->data shall point to a buffer that holds the data to post. • note->size of the data buffer. • note->category shall be a non-empty, NULL terminated ASCII string that holds the name of the category under which the note is posted.

Returns

SUCCESS – 0
FAILURE – Negative

14.4.2.2 int qsee_bulletin_board_read (qsee_bulletin_board_note * *note,* qsee_ta_name * *sender_name*)

Reads a note, if available, from the QTEE bulletin board from a specific category.

Note: No permission model is applied on the bulletin board. Every QTEE application can read all available notes from all categories.

Parameters

in,out	*note*	Pointer to a structure that holds the read note: • note->data shall point to a buffer that has enough space to hold the note from the desired category. • note->size is a bi-directional parameter. When used as input, it indicates buffer size. When the function returns, it holds the the bytes that were actually copied into the user-provided buffer. • note->category shall hold the category name of the category whose note is desired.
in,out	*sender_name*	Pointer to a structure that holds the TA name that posted the returned note.

Returns

SUCCESS – 0
FAILURE – Negative

14.5 Cipher

14.5.1 Detailed Description

Within Cipher API, the following return values are defined by qsee_shim.h:

- SUCCESS – 0
- FAILURE – -1

14.5.2 Enumeration Type Documentation

14.5.2.1 enum QSEE_CIPHER_ALGO_ET

QSEE Cipher supports these algorithms:

Enumerator

QSEE_CIPHER_ALGO_AES_128 AES-128 Cipher Algorithm.
QSEE_CIPHER_ALGO_AES_256 AES-256 Cipher Algorithm.
QSEE_CIPHER_ALGO_TRIPLE_DES Triple DES Cipher Algorithm.
QSEE_CIPHER_ALGO_INVALID Default or invalid algorithm value.

14.5.2.2 enum QSEE_CIPHER_BAM_PIPE_ET

QSEE Cipher Bit Accurate Model (BAM) Pipe:

Enumerator

QSEE_CIPHER_BAM_GENERIC Generic TZ use case
QSEE_CIPHER_BAM_HLOS_CPB Pattern use case and other use cases where scratch memory is used in the app Video DRM Decrypt.
QSEE_CIPHER_BAM_CPB_HLOS HDCP Encryption.
QSEE_CIPHER_BAM_HLOS_HLOS DRM Audio use case.
QSEE_CIPHER_BAM_INVALID Unknown or invalid pipe option.

14.5.2.3 enum QSEE_CIPHER_MODE_ET

Supported modes of operation:

Enumerator

QSEE_CIPHER_MODE_ECB Electronic Codebook mode.
QSEE_CIPHER_MODE_CBC Cipher block chaining mode.
QSEE_CIPHER_MODE_CTR Counter mode.
QSEE_CIPHER_MODE_XTS XEX-based tweaked codebook mode with ciphertext stealing.
QSEE_CIPHER_MODE_CCM Counter mode with CBC-MAC.
QSEE_CIPHER_MODE_CTS Cipher text stealing mode.
QSEE_CIPHER_MODE_INVALID Unknown or invalid mode.

14.5.2.4 enum QSEE_CIPHER_PAD_ET

QSEE Cipher padding algorithms:

Enumerator

QSEE_CIPHER_PAD_ISO10126 Padding mode defined by ISO 10126.
QSEE_CIPHER_PAD_PKCS7 PKCS7 padding mode as described in Internet Engineering Task Force's RFC5652.
QSEE_CIPHER_PAD_NO_PAD No padding added.
QSEE_CIPHER_PAD_INVALID Unknown or invalid padding mode.

14.5.2.5 enum QSEE_CIPHER_PARAM_ET

QSEE Cipher parameter types:

Enumerator

QSEE_CIPHER_PARAM_KEY Key value parameter.
QSEE_CIPHER_PARAM_IV Initialization Vector (IV) parameter.
QSEE_CIPHER_PARAM_MODE Cipher mode parameter. See QSEE_CIPHER_MODE_ET for valid values.
QSEE_CIPHER_PARAM_PAD Cipher padding scheme parameter. See QSEE_CIPHER_PAD_ET for valid values.
QSEE_CIPHER_PARAM_NONCE Nonce value parameter. Used in AES-CCM Mode only.
QSEE_CIPHER_PARAM_XTS_KEY AES-XTS mode secondary key value parameter.
QSEE_CIPHER_PARAM_XTS_DU_SIZE AES-XTS data unit size parameter.
QSEE_CIPHER_PARAM_CCM_PAYLOAD_LEN AES-CCM Payload length parameter.
QSEE_CIPHER_PARAM_CCM_MAC_LEN AES-CCM MAC length parameter.
QSEE_CIPHER_PARAM_CCM_HDR_LEN AES-CCM Header Data length parameter.
QSEE_CIPHER_PARAM_BAM_PIPE BAM Pipe parameter.
QSEE_CIPHER_PARAM_VA_IN Virtual address cipher input parameter.
QSEE_CIPHER_PARAM_VA_IN_LEN Virtual address cipher input parameter length.
QSEE_CIPHER_PARAM_VA_OUT Virtual address cipher output parameter.
QSEE_CIPHER_PARAM_VA_OUT_LEN Virtual address cipher output parameter length.
QSEE_CIPHER_PARAM_COPY Boolean indicating if copying of cipher is allowed.
QSEE_CIPHER_PARAM_INVALID Unknown or invalid parameter.

14.5.3 Function Documentation

14.5.3.1 int qsee_cipher_decrypt (const qsee_cipher_ctx * *cipher_ctx,* const uint8_t * *ct,* uint32_t *ct_len,* uint8_t * *pt,* uint32_t * *pt_len*)

Decrypts the passed ciphertext message using the specified algorithm.

Parameters

in	*cipher_ctx*	Pointer to cipher context.
in	*ct*	Pointer to input ciphertext buffer.
in	*ct_len*	Input ciphertext buffer length (in bytes).
out	*pt*	Pointer to output plaintext buffer.
in,out	*pt_len*	Pointer to output plaintext buffer length. **Note:** Value is modified to the actual plaintext bytes written.

Detailed description

The memory allocated for the plaintext must be large enough to hold the ciphertext equivalent. If a padding scheme is selected, the plaintext output length may be up to one block size smaller than the ciphertext length. If the output buffer is not large enough to hold the decrypted results, an error is returned.

Returns

SUCCESS – Function executes successfully.
FAILURE – Any error encountered during decryption.

14.5.3.2 int qsee_cipher_encrypt (const qsee_cipher_ctx * *cipher_ctx,* const uint8_t * *pt,* uint32_t *pt_len,* uint8_t * *ct,* uint32_t * *ct_len*)

Encrypts the passed plaintext message using the specified algorithm.

Parameters

in	*cipher_ctx*	Pointer to cipher context.
in	*pt*	Pointer to input plaintext buffer.
in	*pt_len*	Input plaintext buffer length (in bytes).
out	*ct*	Pointer to output ciphertext buffer.
in,out	*ct_len*	Pointer to output ciphertext buffer length. **Note:** Length value is modified to the actual ciphertext bytes written.

Detailed description

The memory allocated for the ciphertext must be large enough to hold the plaintext equivalent. If a padding scheme is selected, the ciphertext buffer length may need to be up to one block size larger than the plaintext length. If the output buffer is not large enough to hold the encrypted results, an error is returned.

Returns

SUCCESS – Function executes successfully.
FAILURE – Any error encountered during encryption.

14.5.3.3 int qsee_cipher_free_ctx (qsee_cipher_ctx * *cipher_ctx*)

Releases all resources with a given cipher context.

Parameters

in	*cipher_ctx*	Pointer to the cipher context to be deleted.

Returns

SUCCESS – Function executes successfully.
FAILURE – Any error encountered during context free.

14.5.3.4 int qsee_cipher_get_param (const qsee_cipher_ctx * *cipher_ctx,* QSEE_CIPHER_PARAM_ET *param_id,* void * *param,* uint32_t * *param_len*)

Retrieves the parameters for a given cipher context.

Parameters

in	*cipher_ctx*	Pointer to cipher context.
in	*param_id*	Parameter to retrieve.
out	*param*	Pointer to memory location to store the parameter.
in,out	*param_len*	Pointer to param length (in bytes). **Note:** Length value is modified to the actual length of param.

Returns

SUCCESS – Function executes successfully.
FAILURE – Any error encountered during parameter retrieval.

14.5.3.5 int qsee_cipher_init (QSEE_CIPHER_ALGO_ET *alg,* qsee_cipher_ctx ** *cipher_ctx*)

Intializes a cipher context for encrypt or decrypt operation.

Parameters

in	*alg*	Algorithm standard to use.
out	*cipher_ctx*	Double pointer to cipher context.

Returns

SUCCESS – Function executes successfully.
FAILURE – Any error encountered during cipher initialization.

14.5.3.6 int qsee_cipher_reset (qsee_cipher_ctx * *cipher_ctx*)

Resets the cipher context; does not reset the key.

Parameters

in,out	*cipher_ctx*	Pointer to cipher context.

Returns

SUCCESS – Function executes successfully.
FAILURE – Any error encountered during cipher reset.

14.5.3.7 int qsee_cipher_set_param (qsee_cipher_ctx * *cipher_ctx,* QSEE_CIPHER_PARAM_ET *param_id,* const void * *param,* uint32_t *param_len*)

Modifies the parameters for a given cipher operation.

Caution: The QSEE_CIPHER_MODE_CTS and QSEE_CIPHER_MODE_XTS cipher modes are not supported by the QTEE off-target environment.

Parameters

in,out	*cipher_ctx*	Pointer to cipher context.
in	*param_id*	Parameter to modify.
in	*param*	Pointer to parameter value to set.
in	*param_len*	Parameter length (in bytes).

Returns

SUCCESS – Function executes successfully.
FAILURE – Any error encountered during parameter setting.

14.6 Clock

14.6.1 Detailed Description

14.6.2 Function Documentation

14.6.2.1 uint32_t qsee_set_bandwidth (void * *reqClient,* uint32_t *reqClientlen,* uint32_t *res_req,* uint32_t *level,* uint32_t *flags*)

Sets the crypto/bimc/snoc bandwidth.

Parameters

in	*reqClient*	Client requesting the clock.
in	*reqClientlen*	Length of reqClient name (in bytes).
in	*res_req*	Resource to vote clocks; all associated clocks are turned on and voted for.
in	*level*	Clock level.
in	*flags*	Flags (for future use).

Returns

SUCCESS – E_SUCCESS
FAILURE – E_FAILURE

14.7 CMAC

14.7.1 Detailed Description

14.7.2 Define Documentation

14.7.2.1 #define QSEE_CMAC_DIGEST_SIZE 16

CMAC digest size.

14.7.2.2 #define QSEE_CMAC_FAILURE -1

Any error encountered.

14.7.2.3 #define QSEE_CMAC_SUCCESS 0

Function executes successfully.

14.7.3 Enumeration Type Documentation

14.7.3.1 enum QSEE_CMAC_ALGO_ET

Supported CMAC algorithms.

Enumerator

QSEE_CMAC_ALGO_AES_128 AES-128 Cipher Algorithm.
QSEE_CMAC_ALGO_AES_256 AES-256 Cipher Algorithm.

14.7.4 Function Documentation

14.7.4.1 int qsee_cmac (QSEE_CMAC_ALGO_ET *alg*, const uint8_t * *msg*, uint32_t *msg_len*, const uint8_t * *key*, uint32_t *key_len*, uint8_t * *cmac_digest*, uint32_t *cmac_len*)

Creates a cipher MAC per FIPS publication 198 using the specified hash algorithm.

Parameters

in	*alg*	CMAC algorithm to use.
in	*msg*	Pointer to message to be authenticated.
in	*msg_len*	Message length (in bytes).
in	*key*	Pointer to input key to CMAC algorithm.
in	*key_len*	Input key length (in bytes).
out	*cmac_digest*	Pointer to CMAC digest (memory provided by caller).
in	*cmac_len*	CMAC digest length (in bytes). Must be at least QSEE_CMAC_DIGEST_SIZE.

Returns

QSEE_CMAC_SUCCESS – Function executes successfully.
QSEE_CMAC_FAILURE – Any error encountered during CMAC creation.

14.8 Configuration

14.8.1 Detailed Description

14.8.2 Function Documentation

14.8.2.1 qsee_cfg_error qsee_cfg_getpropval (const char * *PropName,* uint32_t *PropNameLen,* uint32_t *PropId,* qsee_cfg_propvar_t * *pPropBuf,* uint32_t *PropBufSz,* uint32_t * *PropBufSzRet*)

Fetches a configuration value from the TZ kernel via a SYS CALL.

Parameters

in	*PropName*	Pointer to property name (a string).
in	*PropNameLen*	PropName length, including '\0' (e.g. strnlen()+1).
in	*PropId*	Property ID.
out	*pPropBuf*	Pointer to output buffer that is populated with the DAL configuration value.
in	*PropBufSz*	Output buffer size (in bytes).
out	*PropBufSzRet*	Pointer to the actual populated buffer size (in bytes). If property type is a string, output size does NOT include '\0'.

Returns

SUCCESS – 0
FAILURE – Nonzero

14.9 Core

14.9.1 Detailed Description

14.9.2 Function Documentation

14.9.2.1 int qsee_get_device_uuid (uint8_t * *uuid_ptr,* size_t * *uuid_len*)

Provides the UUID for the device.

Parameters

out	*uuid_ptr*	Pointer to a buffer filled with a struct based an IETF UUID (GP compatible).
in,out	*uuid_len*	Pointer to uuid buffer size.

Returns

CALL SUCCESS – 0

Note: The output buffer must be at least the size of an IETF UUID (32 bytes). uuid_len pointer is updated to indicate actual UUID size.

14.9.2.2 int qsee_get_fw_component_version (uint8_t * *version_ptr,* size_t * *version_len*)

The detailed firmware version number that supports the trusted OS implementation includes all privileged software involved in TEE secure booting and support, apart from the secure OS and trusted applications.

Parameters

out	*version_ptr*	Pointer to a buffer filled with the version number.
in,out	*version_len*	Pointer to version buffer size.

Returns

CALL SUCCESS – 0

Note: The version number is a printable ASCII string, but is not NULL-terminated. The maximum buffer size is 128 bytes. The version_len pointer is updated to indicate the actual version string length.

14.9.2.3 int qsee_get_secure_state (qsee_secctrl_secure_status_t * *status*)

Checks device security status.

Parameters

out	*status*	Pointer to security status (struct qsee_secctrl_secure_status_t) with the following bit field definitions: • Bit 0: secboot enabling check failed • Bit 1: Sec hardware key not programmed • Bit 2: Debug disable check failed • Bit 3: Anti-rollback check failed • Bit 4: Fuse config check failed • Bit 5: RPMB provision check failed • Bit 6: Debug check in image certificate failed Debug Bits • Bit 7: RSVD • Bit 8: TZ secure debug fuse check failed • Bit 9: MSS secure debug fuse check failed • Bit 10: CP secure debug fuse check failed • Bit 11: Nonsecure secure debug fuse check failed

Returns

CALL SUCCESS – 0

14.9.2.4 int qsee_get_trusted_os_component_version (uint8_t * *version_ptr,* size_t * *version_len*)

Details the TEE implementation version number.

Parameters

out	*version_ptr*	Pointer to a buffer filled with the QTEE version number.
in,out	*version_len*	Pointer to version buffer size.

Returns

CALL SUCCESS – 0

Note: The version number is a printable ASCII string, but is not NULL-terminated. The maximum buffer size is 128 bytes. version_len is updated to indicate the actual length of the version string.

14.9.2.5 int qsee_get_tz_app_id (uint8_t * *tz_app_id,* uint32_t *id_buf_len*)

Returns the application distinguished ID stored in the QSEE app certificate.

Parameters

out	*tz_app_id*	Pointer to buffer populated with the application distinguished ID.

in	*id_buf_len*	Output buffer length (in bytes).

Returns

CALL SUCCESS – 0

Note: The output buffer must be at least the size of distID (32 bytes).

14.9.2.6 int qsee_hdmi_status_read (uint32_t * *hdmi_enable,* uint32_t * *hdmi_sense,* uint32_t * *hdcp_auth*)

Reads HDMI link and hardware HDCP status.

Parameters

out	*hdmi_enable*	HDMI output enabled.
out	*hdmi_sense*	HDMI sense.
out	*hdcp_auth*	HDCP authentication success.

Returns

CALL SUCCESS – 0

14.9.2.7 bool qsee_is_ns_range (const void * *start,* uint32_t *len*)

Deprecated This function is deprecated. QTEE kernel performs all required checks on memory ownership and accessibility before mapping it into a TA address space.

Tests if the memory range [`start`, `start` + `len`] is in HLOS/Primary VM. `NULL` is a valid value for `start`, because physical addressing is used.

Warning : With TA preemption feature, the return value from this API isn't guaranteed to remain as valid while the caller of the API in TA code is processing the return value. It's because TZ becomes reentrantable (it can start a new task that can affect the result) when TA execution is interrupted while caller of this API is processing the result. (please refer `preemptible` Metadata property,)

To avoid the issue, we recommend to use this API over the memory region that the TA already mapped with `qsee_register_shared_buffer` or `IHlosRegionFinder`. The mapping created with the recommended APIs internally locks the characteristics of the memory itself, preventing TZ from accepting commands which would alter them. The internal locks are maintained until the memory is unmapped by the TA.

Parameters

in	*start*	Start of the memory range, physical address, included in the range.
in	*len*	Memory range length (in bytes).

Returns

true – If entire area is in HLOS/PVM memory.

false – If area contains memory outisde HLOS/PVM.

14.9.2.8 bool qsee_is_s_tag_area (uint32_t *vmid,* uint64_t *start,* uint64_t *end*)

Tests if the range [`start`, `end`] is tagged for the particular virtual machine ID (VMID), specific to the CPZ use case.

Warning : With TA preemption feature, the return value from this API isn't guaranteed to remain as valid while the caller of the API in TA code is processing the return value. It's because TZ becomes reentrantable (it can start a new task that can affect the result) when TA execution is interrupted while caller of this API is processing the result. (please refer `preemptible` Metadata property,)

To avoid the issue, we recommend to use this API over the memory region that the TA already mapped with `qsee_register_shared_buffer` or `IHlosRegionFinder`. The mapping created with the recommended APIs internally locks the characteristics of the memory itself, preventing TZ from accepting commands which would alter them. The internal locks are maintained until the memory is unmapped by the TA.

Parameters

`in`	*vmid*	VMID defined in access control layer (enum ACVirtualMachineId).
`in`	*start*	Start of the memory range, physical address, included in the range.
`in`	*end*	End of the memory range, physical address, included in the range.

Returns

true – If entire area is tagged for the specified VMID
false – If entire area is not tagged for the specified VMID

14.9.2.9 int qsee_read_jtag_id (void)

Reads the JTAG ID.

Returns

JTAG ID value.

14.9.2.10 int qsee_read_serial_num (void)

Deprecated This function is deprecated. Returns only 32 bit serial_num. Use readSerialNum() exposed by CDeviceID QTEE service to get 64bit serial number.

Reads the serial number from the PTE chain.

Returns

32bit serial number.

14.9.2.11 int qsee_tag_mem (uint32_t *vmid,* uint64_t *start,* uint64_t *end*)

Tags all memory in the range [start, end] with the specified VMID.

Parameters

in	*vmid*	VMID defined in access control layer (enum ACVirtualMachineId)
in	*start*	Start of the memory range, physical address, included in the range.
in	*end*	End of the memory range, physical address, included in the range.

Returns

CALL SUCCESS – 0

14.9.2.12 uint32_t qsee_vm_mem_count (uint32_t *vmid*)

Counts the number of 4 KB memory chunks in the specified virtual machine (VM).

Parameters

in	*vmid*	VMID defined in access control layer (enum ACVirtualMachineId).

Returns

The number of 4 KB chunks in the VM.

14.10 Crypto Hardware Lock

14.10.1 Detailed Description

14.10.2 Function Documentation

14.10.2.1 bool qsee_crypto_lock_engine (bool *lock*)

Locks or unlocks the crypto hardware by XPU protecting the resource. This function may do nothing if the chipset supports a dedicated TZ crypto engine.

Parameters

in	*lock*	Set to TRUE to lock and FALSE to unlock.

Returns

TRUE – If hardware can be locked
FALSE – If hardware cannot be locked

14.11 Data Cache Maintenance

14.11.1 Detailed Description

14.11.2 Function Documentation

14.11.2.1 void qsee_dcache_clean_region (void * addr, size_t length)

Cleans a memory region in the cache. Note that this writes back any data that is dirty, but it does not invalidate the cache region. Any further access to data in this region results in a cache-hit.

Parameters

in	*addr*	Memory region start address.
in	*length*	Memory region length.

14.11.2.2 void qsee_dcache_flush_region (void * addr, size_t length)

Cleans and invalidates a memory region in the cache. Note that the data in the cache is written back to the main memory if it is dirty, and the region is invalidated. Any further access to the data results in a cache-miss.

Parameters

in	*addr*	Memory region start address.
in	*length*	Memory region length.

14.11.2.3 void qsee_dcache_inval_region (void * addr, size_t length)

Invalidates a memory region in the cache. Note that the data in the cache is not written back to the main memory. Any further access to data in this region results in a cache-miss.

Parameters

in	*addr*	Memory region start address.
in	*length*	Memory region length.

14.12 Elliptic Curve Cryptography

14.12.1 Detailed Description

Within QTEE ECC API the following return values are defined by the Unified Crypto Environment:

- UC_E_SUCCESS = 0
- UC_E_FAILURE = 1
- UC_E_NOT_ALLOWED = 2
- UC_E_INVALID_ARG = 16
- UC_E_OUT_OF_RANGE = 17
- UC_E_DATA_INVALID = 21

14.12.2 Data Structure Documentation

14.12.2.1 struct QSEE_bigval_t

- QTEE representation of a large multiprecision integer.
  - For ECC P256, QSEE_bigval_t types hold 288-bit 2's complement numbers (9 ∗ 32-bit words).
  - For ECC P192 they hold 224-bit 2's complement numbers (7 ∗ 32-bit words).
  - Representation is little endian by word and native endian within each word.

Data fields

Type	Parameter	Description
uint32_t	data[QSEE_BIGLEN]	Array of uint32_t representing a large multi-precision integer.

14.12.2.2 struct QSEE_affine_point_t

- QTEE representation of an affine point.
- Used for ECC public keys.
- For the point at infinity, the boolean value is true and x and y values are ignored.

Data fields

Type	Parameter	Description
bool	infinity	Boolean value representing infinity point.
QSEE_bigval_t	x	Affine point X coordinate.
QSEE_bigval_t	y	Affine point Y coordinate.

14.12.2.3 struct QSEE_ECDSA_sig_t

QTEE representation for the ECDSA signature.

Data fields

Type	Parameter	Description
QSEE_bigval_t	r	ECDSA digital signature 'r' value.

Type	Parameter	Description
QSEE_bigval_t	s	ECDSA digital signature 's' value.

14.12.2.4 struct QSEE_qrlbn_modulus_data_t

QTEE representation of Qualcomm Reduced Leakage Big Number (QRLBN) modulus data.

Data fields

Type	Parameter	Description
size_t	allocated_words	Number of words allocated for QRLBN.
unsigned	flags	Flags to control reduction.
uint32_t	m_inv32	Helper value used in computing Qi for Montgomery multiplication
uint64_t	m_recip	Reciprocal value.
uint32_t *	modulus	Pointer to modulus value.
size_t	modulus_bits	Size of modulus in bits.
uint32_t *	montgomery_R	Pointer to Montgomery R value.
uint32_t *	montgomery_R2	Pointer to Montgomery R^2 value.
size_t	num_words	Number of words used by QRLBN.
int	quo_num_shift	Number of bits to shift so that the binary point is just left of bit 7.

14.12.2.5 struct QSEE_qrlbn_ecc_bigval_t

QTEE representation of QRLBN.

Data fields

Type	Parameter	Description
uint32_t	data[QSEE_ECC_BIGNUM_WORDS]	Array of uint32_t representing a large multi-precision integer.

14.12.2.6 struct QSEE_qrlbn_ecc_point_t

- QTEE representation of an ECC point in Jacobian coordinates with X, Y, and Z montgomery encoded.
- If QRLBN_IS_POINT_AT_INFINITY flag is set, the X, Y, and Z values must have values whose absolute value is less than a few times the modulus, but the actual values do not otherwise matter.

Data fields

Type	Parameter	Description
int	flags	ECC point flags.
QSEE_qrlbn_ecc_bigval_t	X	X coordinate value.
QSEE_qrlbn_ecc_bigval_t	Y	Y coordinate value.
QSEE_qrlbn_ecc_bigval_t	Z	Z coordinate value.

14.12.2.7 struct QSEE_qrlbn_ecc_modulus_data_t

QTEE representation of ECC modulus data.

Data fields

Type	Parameter	Description
QSEE_qrlbn_modulus_data_t	md	Modulus data.
QSEE_qrlbn_ecc_bigval_t	modulus_storage	QRLBN Modulus value storage.
QSEE_qrlbn_ecc_bigval_t	montgomery_R2_storage	QRLBN Montgomery R^2 value storage.
QSEE_qrlbn_ecc_bigval_t	montgomery_R_storage	QRLBN Montgomery R value storage.

14.12.2.8 struct QSEE_qrlbn_ecc_affine_point_t

- QTEE representation of an ECC affine point.
- For the point at infinity, the QRLBN_POINT_AT_INFINITY flag is set and X and Y values are ignored.

Data fields

Type	Parameter	Description
int	flags	Affine point flags.
QSEE_qrlbn_ecc_bigval_t	x	X coordinate value.
QSEE_qrlbn_ecc_bigval_t	y	Y coordinate value.

14.12.2.9 struct QSEE_qrlbn_ecc_domain_t

QTEE representation of ECC domain parameters.

Data fields

Type	Parameter	Description
QSEE_qrlbn_ecc_bigval_t	a	Elliptic curve parameter a.
QSEE_qrlbn_ecc_bigval_t	a_mont	Montgomery conversion of a.
QSEE_qrlbn_ecc_affine_point_t	affine_base_point	Base point on curve.
QSEE_qrlbn_ecc_bigval_t	b	Elliptic curve parameter b.
QSEE_qrlbn_ecc_bigval_t	b_mont	Montgomery conversion of b.
QSEE_qrlbn_ecc_point_t	base_point	Montgomery conversion of affine_base_point.
QSEE_qrlbn_ecc_modulus_data_t	base_point_order	Order of base point.
uint64_t	cofactor	Cofactor.
QSEE_qrlbn_ecc_modulus_data_t	modulus	QRLBN type modulus data.
QSEE_qrlbn_ecc_bigval_t	ts_z	Montgomery value for Tonelli Shanks square root.
int	version	Data structure version number.

14.12.2.10 struct QSEE_qrlbn_ECDSA_sig_t

QTEE representation of ECDSA signature in QRLBN values.

Data fields

Type	Parameter	Description
QSEE_qrlbn_ecc_bigval_t	r	ECDSA digital signature 'r' value.
QSEE_qrlbn_ecc_bigval_t	s	ECDSA digital signature 's' value.

14.12.2.11 struct QSEE_ecies_key_t

Structure representing an ECIES public or private key value depending on the purpose value passed to update.

Data fields

Type	Parameter	Description
union QSEE_ecies_key_t	enc_dec	Union that represents either public or private ECIES key.

14.12.2.12 union QSEE_ecies_key_t.enc_dec

Union that represents either public or private ECIES key.

Data fields

Type	Parameter	Description
QSEE_qrlbn_ecc_bigval_t *	private_key	Private key.
QSEE_qrlbn_ecc_affine_point_t *	recipient_public_key	Public key.

14.12.2.13 struct QSEE_ecies_params_t

Structure to hold elliptic curve integrated encryption scheme (ECIES) parameter values.

Data fields

Type	Parameter	Description
QSEE_ecies_balg_t	balg	Symmetric algorithm (AES) scheme.
QSEE_ecies_digest_t	digest	Digest hash function.
QSEE_ecies_curve_t	ecies_curve	ECC curve.
QSEE_ecies_kdf_t	kdf	Key derivation function.

14.12.2.14 struct QSEE_ecies_ctx_t

Structure representing an ECIES context handle.

Data fields

Type	Parameter	Description
QSEE_qrlbn_ecc_domain_t	domain	ECC domain.
QSEE_ecies_params_t	params	ECIES parameter values.

14.12.3 Define Documentation

14.12.3.1 #define QSEE_BIGLEN 9

QSEE_bigval data length for ECC P256.

14.12.4 Enumeration Type Documentation

14.12.4.1 enum QSEE_ecies_balg_t

ECIES block cipher algorithms.

Enumerator

QSEE_ECIES_AES_128 ECIES AES 128 algorithm.
QSEE_ECIES_AES_192 ECIES AES 192 algorithm.
QSEE_ECIES_AES_256 ECIES AES 256 algorithm.

14.12.4.2 enum QSEE_ecies_curve_t

Supported EC curves; used in ECDSA/ECIES.

Enumerator

QSEE_ECIES_CURVE_P_224 ECC P224 curve.
QSEE_ECIES_CURVE_P_256 ECC P256 curve.
QSEE_ECIES_CURVE_P_384 ECC P384 curve.
QSEE_ECIES_CURVE_P_521 ECC P521 curve.
QSEE_ECIES_CURVE_ED25519 ED25519 curve.

14.12.4.3 enum QSEE_ecies_digest_t

Digests provided by keymaster implementations.

Enumerator

QSEE_ECIES_DIGEST_NONE Digest optional. May not be implemented in hardware; handled in software, if needed.
QSEE_ECIES_DIGEST_MD5 MD5 Digest.
QSEE_ECIES_DIGEST_SHA1 SHA1 Digest.
QSEE_ECIES_DIGEST_SHA_2_224 SHA224 Digest.
QSEE_ECIES_DIGEST_SHA_2_256 SHA256 Digest.
QSEE_ECIES_DIGEST_SHA_2_384 SHA384 Digest.
QSEE_ECIES_DIGEST_SHA_2_512 SHA512 Digest.

14.12.4.4 enum QSEE_ecies_kdf_t

Key derivation functions; mostly used in ECIES.

Enumerator

QSEE_ECIES_KDF_RFC5869_SHA256 Do not apply a key derivation function; use the raw agreed key. HKDF defined in RFC 5869 with SHA256.
QSEE_ECIES_KDF_ISO18033_2_KDF1_SHA1 KDF1 defined in ISO 18033-2 with SHA1.
QSEE_ECIES_KDF_ISO18033_2_KDF1_SHA256 KDF1 defined in ISO 18033-2 with SHA256.
QSEE_ECIES_KDF_ISO18033_2_KDF2_SHA1 KDF2 defined in ISO 18033-2 with SHA1.
QSEE_ECIES_KDF_ISO18033_2_KDF2_SHA256 KDF2 defined in ISO 18033-2 with SHA256.

14.12.4.5 enum QSEE_ecies_purpose_t

Possible key (or pair) purposes.

Enumerator

QSEE_ECIES_PURPOSE_ENCRYPT For encrypt purposes. Usable with RSA, EC, and AES keys.
QSEE_ECIES_PURPOSE_DECRYPT For decrypt purposes. Usable with RSA, EC, and AES keys.

14.12.4.6 enum QSEE_qrlbn_field_tag_t

Tags used for specifying fields in certain operations.

Enumerator

QSEE_qrlbn_tag_m ECC domain parameter modulus.modulus_storage. Limited to 32 ∗ mod_words - 24 bits.
QSEE_qrlbn_tag_X ECC domain parameter affine_base_point.x. Limited to 32 ∗ mod_words - 24 bits.
QSEE_qrlbn_tag_Y ECC domain parameter affine_base_point.y. Limited to 32 ∗ mod_words - 24 bits.
QSEE_qrlbn_tag_a ECC domain parameter a. Limited to 32 ∗ mod_words - 24 bits.
QSEE_qrlbn_tag_b ECC domain parameter b. Limited to 32 ∗ mod_words - 24 bits.
QSEE_qrnbn_tag_n ECC domain parameter base_point_order.modulus_storage. Limited to 32 ∗ mod_words - 24 bits.
QSEE_qrlbn_tag_privkey Private key paramater. 32 ∗ base_point_order_num_words - 24 bits
QSEE_qrlbn_tag_hash Hash value. No limit
QSEE_qrlbn_tag_r ECDSA signature parameter r. Limited to 32 ∗ base_point_order_num_words - 24 bits.
QSEE_qrlbn_tag_s ECDSA signature parameter s. Limited to 32 ∗ base_point_order_num_words - 24 bits.
QSEE_qrlbn_tag_p ECC point P parameter.
QSEE_qrlbn_tag_q ECC point Q parameter.

14.12.5 Function Documentation

14.12.5.1 void qsee_ECC_hash_to_bigval (QSEE_bigval_t * *tgt,* void const * *hashp,* unsigned int *hashlen*)

Converts a hash value to bigval_t.

Parameters

in	*tgt*	Pointer to destination buffer.
in	*hashp*	Pointer to hash buffer.
in	*hashlen*	Hash buffer size.

Returns

None.

14.12.5.2 int qsee_get_random_bytes (void * *buf,* int *len*)

Software-based random bytes generator.

Parameters

in,out	*buf*	Pointer to random bytes buffer.
in	*len*	Buffer size.

Returns

SUCCESS – 0 on success
FAILURE – Negative

14.12.5.3 bool qsee_in_curveP (QSEE_affine_point_t const * *P*)

Calculate whether point P lies in the elliptic curve.

Parameters

in	*P*	Pointer to affine point type variable.

Returns

TRUE – P is in curve.
FALSE – P is not in curve.

14.12.5.4 int qsee_SW_ECC_PubPrivate_Key_generate (QSEE_affine_point_t * *pubkey,* QSEE_bigval_t * *privkey*)

- Generates public and private keys of ECC.
- Same keys are used for ECDH and ECDSA.

Parameters

out	*pubkey*	Pointer to ECC public key.
out	*privkey*	Pointer to ECC private key.

Returns

SUCCESS – 0
FAILURE – Negative

14.12.5.5 int qsee_SW_ECDH_Shared_Key_Derive (QSEE_affine_point_t * *shared_key,* QSEE_bigval_t * *privkey,* QSEE_affine_point_t * *pubkey*)

Generates a shared key from Alice's public key and Bob's private key.

Parameters

out	*shared_key*	Pointer to shared key between Alice and Bob.
in	*pubkey*	Pointer to ECC public key.
in	*privkey*	Pointer to ECC private key.

Returns

SUCCESS – 0
FAILURE – Negative

14.12.5.6 int qsee_SW_ECDSA_Sign (QSEE_bigval_t const * *msgdgst,* QSEE_bigval_t const * *privkey,* QSEE_ECDSA_sig_t * *sig*)

Signs data with the ECC private key.

Parameters

in	*msgdgst*	Pointer to message digest.
in	*privkey*	Pointer to private key for signing.
out	*sig*	Pointer to message signature.

Returns

SUCCESS – 0
FAILURE – Negative

14.12.5.7 int qsee_SW_ECDSA_Verify (QSEE_bigval_t const * *msgdgst,* QSEE_affine_point_t const * *pubkey,* QSEE_ECDSA_sig_t const * *sig*)

Verifies data with an ECC public key.

Parameters

in	*msgdgst*	Pointer to message digest.
in	*pubkey*	Pointer to public key for signing.
in	*sig*	Pointer to message signature.

Returns

SUCCESS – 0
FAILURE – Negative

14.12.5.8 int qsee_SW_ECIES_finish (QSEE_ecies_ctx_t * *ctx*)

De-initializes and resets content of the ECIES instance.

Parameters

in	*ctx*	Pointer to ECIES context.

Returns

UC_E_SUCCESS – Function executes successfully
UC_E_INVALID_ARG – Unrecognized argument

14.12.5.9 int qsee_SW_ECIES_init (QSEE_ecies_ctx_t * *ctx,* QSEE_ecies_params_t * *params*)

Initializes an ECIES instance.

Parameters

out	*ctx*	Pointer to ECIES context.
in	*params*	Pointer to parameters of ECIES context intialization.

Returns

UC_E_SUCCESS – Function executes successfully
UC_E_FAILURE – Operation failed due to unknown error
UC_E_NOT_ALLOWED – Operation currently not allowed
UC_E_INVALID_ARG – Unrecognized argument

14.12.5.10 int qsee_SW_ECIES_update (QSEE_ecies_ctx_t * *ctx,* QSEE_ecies_key_t * *key,* QSEE_ecies_purpose_t *purpose,* const uint8_t * *msg,* const uint32_t *msg_len,* uint8_t ** *ppAD,* uint32_t * *pAD_len,* uint8_t * *out,* uint32_t * *out_len*)

Updates an ECIES instance.

Parameters

in,out	*ctx*	Pointer to ECIES context.
in	*key*	Pointer of the key for encryption or decryption.
in	*purpose*	Defines update encryption or decryption.
in	*msg*	Message for encryption or decryption.
in	*msg_len*	Message length.
in,out	*ppAD*	Pointer to the AD message. In encryption, this parameter does NOT change. In decryption, the ∗ppAD changes to point to AD data.
in,out	*pAD_len*	AD message length.
out	*out*	Encryption or decryption output message.
in,out	*out_len*	Output message length. The caller needs must ensure enough memory is allocated to contain the output message.

Returns

UC_E_SUCCESS – Function executes successfully
UC_E_FAILURE – Operation failed due to unknown error
UC_E_NOT_ALLOWED – Operation currently not allowed
UC_E_INVALID_ARG – Unrecognized argument
UC_E_OUT_OF_RANGE – Value out of range
UC_E_DATA_INVALID – Data is correct but contents are invalid

14.12.5.11 bool qsee_SW_GENERIC_ECC_affine_point_on_curve (QSEE_qrlbn_ecc _affine_point_t const ∗ *Q*, QSEE_qrlbn_ecc_domain_t ∗ *dp*)

Calculates if the ECC affine point is on a curve.

Parameters

in	*Q*	Pointer to affine point type variable.
in	*dp*	Pointer to the domain curve.

Returns

TRUE – Point Q is on curve dp
FALSE – Point Q is not on curve dp

14.12.5.12 int qsee_SW_GENERIC_ECC_bigval_to_binary (uint8_t ∗ *dst*, size_t *dstlen*, const QSEE_qrlbn_ecc_bigval_t ∗ *src*, const QSEE_qrlbn_ecc_ domain_t ∗ *dp*, QSEE_qrlbn_field_tag_t *tag*)

Converts data from bigval to binary.

Parameters

out	*dst*	Pointer to destination data.
in	*dstlen*	Destination data length (in bytes).
in	*src*	Pointer to source data.
in	*dp*	Pointer to ECC domain context.
in	*tag*	Data type tag.

Returns

SUCCESS – 0
FAILURE – Negative

14.12.5.13 int qsee_SW_GENERIC_ECC_binary_to_bigval (QSEE_qrlbn_ecc_bigval_t * *dst*, const void * *src*, size_t *srclen*, const QSEE_qrlbn_ecc_domain_t * *dp*, QSEE_qrlbn_field_tag_t *tag*)

Converts data from binary to bigval.

Parameters

out	*dst*	Pointer to destination data.
in	*src*	Pointer to source data.
in	*srclen*	Source data length (in bytes).
in	*dp*	Pointer to ECC domain context.
in	*tag*	Data type tag.

Returns

SUCCESS – 0
FAILURE – Negative

14.12.5.14 int qsee_SW_GENERIC_ECC_convert_input_to_bigval (QSEE_qrlbn_ecc_bigval_t * *dst*, const void * *src*, size_t *srclen*, int *base*)

- Converts input to bigval.
- Input can be network order binary, ASCII hex, or ASCII decimal, as indicated by radix (256, 16, 10, respectively).
- When mode is 256 (binary), srclen indicates the length. Otherwise, srclen is ignored and the input must be null terminated.
- If data cannot be represented, an error occurs.

Parameters

out	*dst*	Pointer to destination.
in	*src*	Pointer to source.
in	*srclen*	Source length (in bytes).

in	*base*	Base for conversion mode.

Returns

SUCCESS – 0
FAILURE – Negative

14.12.5.15 int qsee_SW_GENERIC_ECC_deinit_ex (QSEE_qrlbn_ecc_domain_t ** *dp*)

Deinitializes the domain initialized with qsee_SW_GENERIC_ECC_init_ex.
qsee_SW_GENERIC_ECC_init_ex and qsee_SW_GENERIC_ECC_deinit_ex must be used in pair.

Parameters

out	*dp*	Pointer to ECC domain context

Returns

SUCCESS – 0
FAILURE – Negative

14.12.5.16 int qsee_SW_GENERIC_ECC_init (QSEE_qrlbn_ecc_domain_t * *dp,* char * *modulus,* char * *a,* char * *b,* char * *x,* char * *y,* char * *n,* unsigned *cofactor*)

Initializes a domain from curve hex strings and the cofactor.

Parameters

out	*dp*	Pointer to ECC domain context
in	*modulus*	Pointer to modulus.
in	*a*	Pointer to a.
in	*b*	Pointer to b.
in	*x*	Pointer to x.
in	*y*	Pointer to y.
in	*n*	Pointer to n.
in	*cofactor*	Cofactor.

Returns

SUCCESS – 0
FAILURE – Negative

14.12.5.17 int qsee_SW_GENERIC_ECC_init_ex (QSEE_qrlbn_ecc_domain_t ** *dp*, QSEE_ecies_curve_t *c*)

Initializes a domain from curve hex strings and the cofactor. FIPS certifiable ECC init API.

Parameters

out	*dp*	Pointer to ECC domain context
in	*c*	ECC curve type

Returns

SUCCESS – 0
FAILURE – Negative

14.12.5.18 int qsee_SW_GENERIC_ECC_keypair_generate (QSEE_qrlbn_ecc_bigval_t * *privkey*, QSEE_qrlbn_ecc_affine_point_t * *pubkey*, QSEE_qrlbn_ecc_domain_t * *dp*)

Generates an ECC public and private key pair.

Parameters

out	*privkey*	Pointer to private key.
out	*pubkey*	Pointer to public key.
in	*dp*	Pointer to ECC domain context.

Returns

SUCCESS – 0
FAILURE – Negative

14.12.5.19 int qsee_SW_GENERIC_ECC_pubkey_generate (QSEE_qrlbn_ecc_bigval_t const * *privkey*, QSEE_qrlbn_ecc_affine_point_t * *pubkey*, QSEE_qrlbn_ecc_domain_t const * *dp*)

Generates an ECC public key for a given private key.

Parameters

in	*privkey*	Pointer to private key.
out	*pubkey*	Pointer to public key.
in	*dp*	Pointer to ECC domain context.

Returns

SUCCESS – 0
FAILURE – Negative

14.12.5.20 int qsee_SW_GENERIC_ECDH_shared_key_derive (QSEE_qrlbn _ecc_bigval_t * *shared_key,* QSEE_qrlbn_ecc_bigval_t * *privkey,* QSEE_qrlbn_ecc_affine_point_t * *pubkey,* QSEE_qrlbn_ecc_domain_t * *dp*)

Generates a shared key from Alice's public key and Bob's private key.

Parameters

out	*shared_key*	Pointer to shared key between Alice and Bob.
in	*privkey*	Pointer to ECC private key.
in	*pubkey*	Pointer to ECC public key.
in	*dp*	Pointer to ECC domain context.

Returns

SUCCESS – 0
FAILURE – Negative

14.12.5.21 int qsee_SW_GENERIC_ECDSA_sign (uint8_t * *msgdgst,* uint32_t *msgdgst_len,* QSEE_qrlbn_ecc_bigval_t * *privkey,* QSEE_qrlbn_ECDSA_ sig_t * *sig,* QSEE_qrlbn_ecc_domain_t * *dp*)

Signs data with an ECC private key.

Parameters

in	*msgdgst*	Pointer to message that must be signed.
in	*msgdgst_len*	Message length (in bytes).
in	*privkey*	Pointer to private key for signing.
out	*sig*	Pointer to message signature.
in	*dp*	Pointer to ECC domain context.

Returns

SUCCESS – 0
FAILURE – Negative

14.12.5.22 int qsee_SW_GENERIC_ECDSA_sign_ex (QSEE_HASH_IDX *hash_alg,* uint8_t * *msg,* uint32_t *msg_len,* QSEE_qrlbn_ecc_bigval_t * *privkey,* QSEE_qrlbn_ECDSA_sig_t * *sig,* QSEE_qrlbn_ecc_domain_t * *dp*)

- Hashes input data and sign with an ECC private key.
- FIPS certifiable ECDSA signing API.

Parameters

in	*hash_alg*	Hashing algorithm to sign the message.
in	*msg*	Pointer to message that must be signed.
in	*msg_len*	Message length (in bytes).
in	*privkey*	Pointer to private key for signing.
out	*sig*	Pointer to message signature.
in	*dp*	Pointer to ECC domain context.

Returns

SUCCESS – 0
FAILURE – Negative

14.12.5.23 int qsee_SW_GENERIC_ECDSA_verify (uint8_t * *msgdgst,* uint32_t *msgdgst_len,* QSEE_qrlbn_ecc_affine_point_t * *pubkey,* QSEE_qrlbn_ECDSA_sig_t * *sig,* QSEE_qrlbn_ecc_domain_t * *dp*)

Verifies data with an ECC public key.

Parameters

in	*msgdgst*	Pointer to message that must be signed.
in	*msgdgst_len*	Message length (in bytes).
in	*pubkey*	Pointer to public key for verification.
in	*sig*	Pointer to signed message to be verfied.
in	*dp*	Pointer to ECC domain context.

Returns

SUCCESS – 0
FAILURE – Negative

14.12.5.24 int qsee_SW_GENERIC_ECDSA_verify_ex (QSEE_HASH_IDX *hash_alg,* uint8_t * *msg,* uint32_t *msg_len,* QSEE_qrlbn_ecc_affine_point_t * *pubkey,* QSEE_qrlbn_ECDSA_sig_t * *sig,* QSEE_qrlbn_ecc_domain_t * *dp*)

- Hash input data and verify with an ECC private key.
- FIPS certifiable ECDSA verification API.

Parameters

in	*hash_alg*	Hashing algorithm to sign the message.
in	*msg*	Pointer to message that must be signed.
in	*msg_len*	Message length (in bytes).
in	*pubkey*	Pointer to public key for verification.
in	*sig*	Pointer to signed message to be verfied.

in	*dp*	Pointer to ECC domain context.

Returns

SUCCESS – 0
FAILURE – Negative

14.13 QTEE Env

14.13.1 Detailed Description

14.13.2 Function Documentation

14.13.2.1 Object qsee_get_space (void)

Request the IMemSpace object associated with this TA and retains it. The caller is expected to release this object.

Returns

IMemSpace object associated with the application.

14.13.2.2 int32_t qsee_open (uint32_t *uid,* Object * *objOut*)

Request a service from the system.

On success, *objOut will hold the resulting object reference and Object_OK will be returned. In this case, the caller will hold a reference to the object and is responsible for releasing it at some point.

Otherwise, a non-zero value defined in IOpener will be returned. The value of objOut will be non-deterministic in case of an error and the caller should not access it.

Parameters

in	*uid*	The uid of the requested service.
out	*objOut*	Requested service instance.

Returns

Object_OK on success.
IOpener_ERROR_NOT_FOUND if a service matching the ID cannot be found.
IOpener_ERROR_PRIVILEGE if required privileges are not present.
IOpener_ERROR_NOT_SUPPORTED when the service is not supported (stubbed out) or when unmet dependencies are found at runtime

14.13.2.3 Object qsee_open_singleton (uint32_t *classID*)

Request a service from the system if it has not already been obtained.

If the object has already been obtained, it will be returned and the object reference will not be retained (the caller does not need to call release once for each successful call). Singleton objects are persistent for the lifetime of a TA.

Parameters

in	*classID*	The uid of the requested service.

Returns

Requested service instance.

Object_NULL on failure.

14.14 Embedded Secure Element

14.14.1 Detailed Description

14.14.2 Function Documentation

14.14.2.1 bool qsee_ese_service_block_access_basic_channel (void * *pContext*)

Checks if the TEE SE API library is configured to block access to the basic channel.

Access to the basic channel is allowed by default. It must be explicitly blocked via devcfg oem_config.xml

Parameters

in	*pContext*	ESE pContext opaque pointer.

Returns

True if blocked, otherwise allowed.

14.14.2.2 bool qsee_ese_service_block_access_default_applet (void * *pContext*)

Checks if the TEE SE API library is configured to block access to default applet selection if no AID is specified.

Access to the default applet is allowed by default. It must be explicitly blocked via devcfg oem_config.xml.

Parameters

in	*pContext*	ESE pContext opaque pointer.

Returns

True if blocked, otherwise allowed.

14.14.2.3 int qsee_ese_service_close (void * *pContext*)

Closes a logical connection using the handle.

Parameters

in,out	*pContext*	ESE pContext opaque pointer.

Returns

QSEE_ESE_SERVICE_STATUS_SUCCESS on success, otherwise failure.

See also

qsee_ese_service_open

14.14.2.4 uint32_t qsee_ese_service_feature_mask (void * *pContext*)

Gets the feature mask that enables and disables several ESE service functionalities.

Parameters

in	*pContext*	ESE pContext opaque pointer.

Returns

A value of the ESE feature mask from the OEM configuration file.

14.14.2.5 uint32_t qsee_ese_service_get_bwt (void * *pContext*)

Gets the secure element maximum block waiting (BWT).

Parameters

in	*pContext*	ESE pContext opaque pointer.

Returns

A BWT value.

14.14.2.6 uint8_t qsee_ese_service_get_cmd_nad (void * *pContext*)

Gets the T=1 NAD used for the C-APDU that is sent to the secure element.

Parameters

in	*pContext*	ESE pContext opaque pointer.

Returns

The NAD byte used in commands sent to the secure element.

14.14.2.7 uint32_t qsee_ese_service_get_poll_timeout (void * *pContext*)

Gets the SPI polling time for the secure element response message.

Parameters

in	*pContext*	ESE pContext opaque pointer.

Returns

The polling timer value.

14.14.2.8 uint8_t qsee_ese_service_get_rsp_nad (void * *pContext*)

Gets the T=1 NAD used for the R-APDU that is expected from the secure element.

Parameters

in	*pContext*	ESE pContext opaque pointer.

Returns

The NAD byte used in response from secure element.

14.14.2.9 uint32_t qsee_ese_service_get_timer (void * *pContext*)

Gets the timer mask for the ESE timers.

Parameters

in	*pContext*	ESE pContext opaque pointer.

Returns

A value of the ESE timer mask from the OEM configuration file.

14.14.2.10 uint8_t qsee_ese_service_oem_get_chip_select_id (void * *pContext*)

Retrieves the chip select identifier used by secure element from devcfg.

Parameters

in	*pContext*	ESE pContext opaque pointer.

Returns

A valid value for the chip select identifier used by secure element.

14.14.2.11 uint8_t qsee_ese_service_oem_get_spi_bits_per_word (void * *pContext*)

Retrieves the SPI number of bits per work used by secure element from devcfg.

Parameters

in	*pContext*	ESE pContext opaque pointer.

Returns

A valid number of SPI bits per work; defaults to 8 if it cannot retrieve this number from devcfg.

14.14.2.12 qsee_spi_device_id_t qsee_ese_service_oem_get_spi_id (void * *pContext*)

Retrieves the SPI device identifier used by the secure element from devcfg.

Parameters

in	*pContext*	ESE pContext opaque pointer.

Returns

A valid SPI identifier.

14.14.2.13 uint32_t qsee_ese_service_oem_get_spi_max_frequency (void * *pContext*)

Retrieves the maximum SPI frequency used by the secure element from devcfg.

Parameters

in	*pContext*	ESE pContext opaque pointer.

Returns

A valid frequency; defaults to 9600000 if it cannot be retrieved from devcfg.

14.14.2.14 int qsee_ese_service_open (void ** *ppContext*)

Opens a logical connection using the handle to communicate over SPI.

Parameters

in,out	*ppContext*	ESE pContext opaque pointer.

Returns

Object_OK on success, otherwise failure.

See also

qsee_ese_service_close

14.14.2.15 int qsee_ese_service_read (void * *pContext,* qsee_ese_service_rapdu_t * *r_apdu*)

Reads from slave data on the logical connection handle.

Parameters

in	*pContext*	ESE pContext opaque pointer.
in,out	*r_apdu*	Reads buffer information.

Returns

Object_OK on success, otherwise failure.

14.14.2.16 bool qsee_ese_service_rsp_pcb_error (uint8_t *pcb*)

Checks if T=1 Protocol Control Byte has error indications.

Parameters

in	*pcb*	Protocol control byte.

Returns

True if error bits are set, otherwise false.

14.14.2.17 int qsee_ese_service_seac (void * *pContext,* const void * *aid,* size_t *aid_size,* const void * *apdu,* size_t *apdu_size*)

Performs secure element access control (SEAC) based on the application identifier (AID) and/or application protocol data unit (APDU).

Parameters

in	*pContext*	ESE pContext opaque pointer.
in	*aid*	Applet Application Identifier NULL means AID filtering should not be applied.
in	*aid_size*	AID length (in bytes).
in	*apdu*	APDU in which SEAC filtering should be applied; NULL means no APDU filtering should be applied.
in	*apdu_size*	APDU length in bytes.

Returns

Object_OK on success, otherwise access denied.

14.14.2.18 bool qsee_ese_service_seac_apdu_filtering_enabled (void * *pContext*)

Checks if SEAC APDU filtering is enabled in QSEE.

Parameters

in	*pContext*	ESE pContext opaque pointer.

Returns

True if enabled, otherwise disabled.

14.14.2.19 bool qsee_ese_service_seac_enabled (void * *pContext*)

Checks if SEAC is enabled in QSEE.

Parameters

in	*pContext*	ESE pContext opaque pointer.

Returns

True if enabled, otherwise disabled.

14.14.2.20 bool qsee_ese_service_spi_end_apdu_session_enabled (void * *pContext*)

Checks if ESE Service can send end of APDU session message to indicate the end of an active APDU session over SPI interface. JCOP may put the chip to low power mode if no APDU session is active over all the supported interfaces

Parameters

in	*pContext*	ESE pContext opaque pointer.

Returns

true if enabled, otherwise disable

14.14.2.21 bool qsee_ese_service_spi_soft_reset_enabled (void * *pContext*)

Checks if the ESE Service can send a SPI soft reset to the ESE. This is to reset SPI T=1 protocol parameters, SPI interface, and JCOP context for the SPI interface.

Parameters

in	*pContext*	ESE pContext opaque pointer.

Returns

True if enabled, otherwise disabled.

14.14.2.22 bool qsee_ese_service_transmit_select_cmd_enabled (void * *pContext*)

Checks if the secure element GP API is allowed to send the SELECT command using the TEE_SEChannelTransmit API.

Parameters

in	*pContext*	ESE pContext opaque pointer.

Returns

True if enabled, otherwise disabled.

14.14.2.23 uint32_t qsee_ese_service_wait (qsee_ese_service_timeout_t *milliseconds*)

Waits for the specified number of milliseconds.

Parameters

in	*milliseconds*	Number of milliseconds to wait.

Returns

Object_OK on success, otherwise failure.

14.14.2.24 int qsee_ese_service_write (void * *pContext,* qsee_ese_service_capdu_t * *c_apdu*)

Writes to slave data on the logical connection handle.

Parameters

in	*pContext*	ESE pContext opaque pointer.
in	*c_apdu*	Writes buffer information.

Returns

Object_OK on success, otherwise failure.

14.15 FIPS Services

14.15.1 Detailed Description

14.15.2 Enumeration Type Documentation

14.15.2.1 enum QSEE_FIPS_APPROVAL_STATUS_TYPE

Status to be returned from qsee_get_fips_approval_status().

Enumerator

QSEE_FIPS_APPROVED Service FIPS approved

QSEE_FIPS_NOT_APPROVED Service not FIPS approved

QSEE_FIPS_SUNSET_AFTER_2023 Service not approved after 2023

QSEE_FIPS_INVALID_SERVICE_ID Service ID/name is invalid

14.15.2.2 enum QSEE_FIPS_CRYPTO_SVC_TYPE

List of Approved services to be used as parameter in qsee_get_fips_approval_status().

Enumerator

QSEE_FIPS_CRYPTO_SVC_APPROVED List of approved Hash services

QSEE_FIPS_SHA512 List of approved MAC services

QSEE_FIPS_HMAC_SHA512_KEYLEN_BETWEEN_112_AND_512_BITS List of approved symmetric cipher services

QSEE_FIPS_AES_256_XTS List of approved symmetric cipher services with text stealing

QSEE_FIPS_AES_256_CTS List of approved streaming cipher services

QSEE_FIPS_AES_256_CFB128 List of approved Public Key services

QSEE_FIPS_RSA_SIG_VER_PSS_4096_SHA512 List of approved RSA primitive services - Modexp

QSEE_FIPS_RSA_SIG_GEN_PRIMITIVE_2048 List of approved Key Derivation services

QSEE_FIPS_PBKDF2 List of approved symmetric cipher services

QSEE_FIPS_TDES_ECB_DECRYPT List of approved Public Key services

QSEE_FIPS_ECDSA_SIG_VER_COMP_P521_SHA512 List of approved Key Derivation services

QSEE_FIPS_PBKDF2_SHA512 Mark end of approved services

QSEE_FIPS_CRYPTO_SVC_APPROVED_END List of conditionally approved services

14.15.2.3 enum QSEE_FIPS_ENABLEMENT_TYPE

These values are returned when qsee_get_fips_info is called with the QSEE_FIPS_ENABLEMENT option.

This is supported to make testing FIPS features easier by allowing a tester to check whether the FIPS features are enabled.

CAUTION: The module is ONLY considered FIPS certified if the FIPS enablement fuse is blown.
To check whether the module is offically in FIPS certified mode, check the QSEE_FIPS_FUSE_STATUS option.

Enumerator

QSEE_FIPS_DISABLED FIPS is not enabled
QSEE_FIPS_ENABLED FIPS is enabled

14.15.2.4 enum QSEE_FIPS_FUSE_STATUS_TYPE

These values are returned when qsee_get_fips_info is called with QSEE_FIPS_FUSE_STATUS.

Enumerator

QSEE_FIPS_FUSE_NOT_BLOWN The FIPS enablement fuse has been not been blown.
QSEE_FIPS_FUSE_BLOWN The FIPS enablement fuse has been blown.

14.15.2.5 enum QSEE_FIPS_INFO_TYPE

These options can be used for the info_type parameter in qsee_get_fips_info().

Enumerator

QSEE_FIPS_MODULE_HMAC Pass a 32 byte buffer with this option to get the crypto module HMAC.
Returns the HMAC of the 32bit module when called by a 32bit TA.
Returns the HMAC of the 64bit module when called by a 64bit TA.

QSEE_FIPS_FUSE_STATUS Pass a 4 byte buffer with this option to check if FIPS is enabled by fuse
QSEE_FIPS_SELFTEST_STATUS Pass a 4 byte buffer with this option to check whether the selftest has run
QSEE_FIPS_ENABLEMENT Pass a 4 byte buffer with this option to check whether FIPS features are enabled
QSEE_FIPS_HW_VERSION Pass a 4 byte buffer with this option to check HW version

14.15.2.6 enum QSEE_FIPS_SELFTEST_STATUS_TYPE

These values are returned when qsee_get_fips_info is called with QSEE_FIPS_SELFTEST_STATUS.

Enumerator

QSEE_CRYPTO_SELFTEST_NOT_RUN The crypto selftest has not been run. FIPS is not enabled and no TA requiring a selftest has been loaded.

QSEE_CRYPTO_SELFTEST_PASSED The crypto selftest has passed.

14.15.3 Function Documentation

14.15.3.1 int qsee_get_fips_approval_status (QSEE_FIPS_CRYPTO_SVC_TYPE *svc*)

- Provides FIPS approval status of a crypto service or algorithm so a user seeking the approval status can get it

Parameters

in	*svc*	Crypto service enum with parameters.

Returns

QSEE_FIPS_SUCCESS or 0 if approved.
QSEE_FIPS_FAILURE or 1 if not approved.
2 if service will be not approved after 2023.

14.15.3.2 int qsee_get_fips_info (QSEE_FIPS_INFO_TYPE *info_type,* void * *buffer,* size_t *buffer_len,* size_t * *bytes_written*)

- Provides FIPS versioning information so an OEM seeking FIPS certification can prove that they are using a FIPS certified version of cmnlib with FIPS enablement.

- Requires an HMAC of the crypto module and a FIPS fuse status to determine a version.

Parameters

in	*info_type*	Type of information requested.
out	*buffer*	Buffer to write requested information.
in	*buffer_len*	Buffer length. If an info type of QSEE_FIPS_MODULE_HMAC is passed, this should be at least 32 bytes. Other information types should be 4 bytes.
out	*bytes_written*	Number of bytes written to buffer.

Returns

QSEE_FIPS_SUCCESS on success.

QSEE_FIPS_FAILURE on error.

14.16 Filesystem

14.16.1 Detailed Description

14.16.2 Function Documentation

14.16.2.1 int close (int *fd*)

Closes an open file.

Parameters

in	*fd*	File descriptor.

Note: Additional details are available at
http://man7.org/linux/man-pages/man2/close.2.html

Returns

SUCCESS – 0
FAILURE – -1

14.16.2.2 int closedir (DIR * *pdir*)

Closes a directory.

Parameters

in	*pdir*	Pointer to directory stream.

Note: Further details are available at
http://man7.org/linux/man-pages/man3/closedir.3.html

Returns

SUCCESS – 0
FAILURE – -1

14.16.2.3 int creat (const char * *path*, uint32_t *mode*)

Creates a file equivalent to open(path, O_WRONLY | O_CREAT | O_TRUNC, mode).

Parameters

in	*path*	Pointer to file path.
in	*mode*	Mode to open a new file.

Note: Values for this mode are described in
http://man7.org/linux/man-pages/man2/creat.2.html

Returns

SUCCESS – 0
FAILURE – -1

14.16.2.4 int fcntl (int *fd,* int *command,* ...)

Performs the command operation on file.

Parameters

in	*fd*	File descriptor.
in	*command*	Operation to be performed.
in	*params*	[Optional] A variable number of optional parameters that are dependent on the command.

Note: More information on optional arguments for fcntl() can be found at
http://man7.org/linux/man-pages/man2/fcntl.2.html

Returns

SUCCESS – Return value depends on the operation
FAILURE – -1

14.16.2.5 int file_dir_chown_chmod (char * *path,* uint32_t *path_len,* char * *word,* uint32_t *word_len,* char * *owner,* uint32_t *owner_len,* char * *mod,* uint32_t *mod_len,* uint32_t *level*)

chown and chmod a file or directory tree.

Parameters

in	*path*	Pointer to the parent directory to chmod and chown, e.g., "/persist/playready/drmms/".
in	*path_len*	Path string length.
in	*word*	Pointer to the start directory chown and chmod from. Must be included in path, e.g., "playready" .
in	*word_len*	Start directory string length.
in	*owner*	Pointer to owner string, e.g., "media.system".
in	*owner_len*	Owner string length.
in	*mod*	Pointer to permission mode string, e.g., "700" or "777".
in	*mod_len*	Permission mode string length.
in	*level*	Wildcarding level to apply to chown and chmod commands [0,1,2].

Note: A level value greater than 0 causes an attempt to chown and chmod with the '*' wildcard level times, e.g.:

- level = 0 :
- chown media.system /persist/playready
- chmod 700 /persist/playready

- level = 1 :
  - chown media.system /persist/playready
  - chmod 700 /persist/playready
  - chown media.system /persist/playready/ ∗
  - chmod 700 /persist/playready/ ∗

- level = 2 :
- chown media.system /persist/playready
- chmod 700 /persist/playready
- chown media.system /persist/playready/ ∗
- chmod 700 /persist/playready/ ∗
- chown media.system /persist/playready/ ∗ / ∗
- chmod 700 /persist/playready/ ∗ / ∗
  Caution: file_dir_chown_chmod() is not supported by the QTEE off-target environment.

Returns

SUCCESS – 0
FAILURE – -1

14.16.2.6 int file_get_partition_free_size (const char ∗ *path,* uint64_t ∗ *size*)

Gets the free size of a specified partition.

Parameters

in	*path*	Pointer to a file pathname.
out	*size*	Pointer to remaining bytes in partition.

Returns

SUCCESS – 0
FAILURE – -1

14.16.2.7 int frename (const char ∗ *oldfilename,* const char ∗ *newfilename*)

Renames a file.

Parameters

in	*oldfilename*	Pointer to the file name from which it is to be renamed.
in	*newfilename*	Pointer to the file name to which it is to be renamed.

Returns

SUCCESS – 0
FAILURE – -1

14.16.2.8 int fstat (int *fd,* fs_stat * *buf*)

- Gets file status.
- When file is a link, fstat() traverses the link and returns information on the linked file.
- lstat() only returns information on the link itself.

Parameters

in	*fd*	File descriptor.
out	*buf*	Pointer to file status structure.

Note: Further details are available at
http://man7.org/linux/man-pages/man2/fstat.2.html

Returns

SUCCESS – 0
FAILURE – -1

14.16.2.9 int fsync (int *fd*)

Synchronizes a file with storage.

Parameters

in	*fd*	File descriptor.

Note: Further details are available at
http://man7.org/linux/man-pages/man2/fsync.2.html

Returns

SUCCESS – 0
FAILURE – -1

14.16.2.10 int get_error_number (void)

Gets the last REE errno.

Returns

The last errno value.

14.16.2.11 int32_t lseek (int *fd*, int32_t *offset*, int *whence*)

Repositions a file offset.

Parameters

in	*fd*	File descriptor.
in	*offset*	New offset.
in	*whence*	Offset reference which can be one of: • SEEK_SET: Sets offset relative to beginning of file. • SEEK_CUR: Sets offset relative to current offset. • SEEK_END: Sets offset relative to end of file.

Note: Further details are available at
http://man7.org/linux/man-pages/man2/lseek.2.html

Returns

SUCCESS – Resulting offset as measured from the beginning of the file
FAILURE – -1

14.16.2.12 int lstat (const char * *path*, fs_stat * *buf*)

- Gets file status.
- When the file is a link, lstat() returns information on the link, unlike fstat(), which traverses the link and returns information on the linked file.

Parameters

in	*path*	Pointer to file pathname.
out	*buf*	Pointer to status buffer.

Note: Further details are available at
http://man7.org/linux/man-pages/man2/lstat.2.html

Returns

SUCCESS – 0
FAILURE – -1

14.16.2.13 int mkdir (const char * *path*, uint32_t *mode*)

Creates a directory tree.

Parameters

in	*path*	Pointer to directory pathname.
in	*mode*	Permissions mode.

Note: Further details are available at
http://man7.org/linux/man-pages/man2/mkdir.2.html

Returns

SUCCESS – 0
FAILURE – -1

14.16.2.14 int open (const char * *path*, int *flags*, ...)

Opens the specified file with the given mode.

Parameters

in	*path*	Pointer to file path.
in	*flags*	File status flags.
in	*mode*	[Optional] File access mode.

Note: Values for flags and mode are described in
http://man7.org/linux/man-pages/man2/open.2.html

Returns

SUCCESS – Valid file descriptor
FAILURE – -1

14.16.2.15 int openat (int *fd*, const char * *path*, int *flags*, ...)

Opens a relative file.

Parameters

in	*fd*	Directory file descriptor relative to which the file path is determined.
in	*path*	Pointer to file path, relative to directory described by fd.
in	*flags*	Access mode to open file.
in	*mode*	[Optional] File access mode.

Note: Values for flags and mode are described in
http://man7.org/linux/man-pages/man2/open.2.html

Returns

SUCCESS – Valid file descriptor
FAILURE – -1

14.16.2.16 DIR* opendir (const char * *path*)

Opens a directory.

Parameters

in	*path*	Pointer to the directory path to open.

Note: Further details are available at
http://man7.org/linux/man-pages/man3/opendir.3.html

Returns

SUCCESS – Pointer to directory stream
FAILURE – NULL

14.16.2.17 int32_t read (int *fd*, void * *buf*, uint32_t *count*)

Reads a number of bytes from a given file.

Parameters

in	*fd*	File descriptor.
out	*buf*	Pointer to buffer to be read into.
in	*count*	Number of bytes to read.

Note: Further details are available at
http://man7.org/linux/man-pages/man2/read.2.html

Returns

SUCCESS – Number of bytes read
FAILURE – -1

14.16.2.18 struct tzDirent* readdir (DIR * *pdir*, struct tzDirent * *entry*)

Reads a directory.

Parameters

in	*pdir*	Pointer to directory stream.
out	*entry*	Pointer to directory entry struct to be populated.

Note: Further details are available at
http://man7.org/linux/man-pages/man3/readdir.3.html

Returns

SUCCESS – Pointer to entry
FAILURE – NULL

14.16.2.19 int remove (const char * *path*)

Removes a file or directory.

Parameters

in	*path*	Pointer to pathname of file/directory.

Note: Further details are available at
http://man7.org/linux/man-pages/man3/remove.3.html

Returns

SUCCESS – 0
FAILURE – -1

14.16.2.20 int rmdir (const char * *path*)

Removes a directory.

Parameters

in	*path*	Pointer to pathname of directory.

Note: Further details are available at
http://man7.org/linux/man-pages/man2/rmdir.2.html

Returns

SUCCESS – 0
FAILURE – -1

14.16.2.21 int32_t telldir (const char * *path*)

Gets current directory location.

Parameters

in	*path*	Pointer to directory pathname.

Note: Further details are available at
http://man7.org/linux/man-pages/man3/telldir.3.html

Returns

SUCCESS – Current location of directory stream
FAILURE – -1

Caution: telldir() is not supported by the QTEE off-target environment.

14.16.2.22 int testdir (const char * *path*)

Tests if a directory exists.

Parameters

in	*path*	Pointer to directory pathname.

Returns

SUCCESS – 0 if directory is found
FAILURE – -1 if directory is not found

14.16.2.23 int unlink (const char * *path*)

Unlink a linked file.

Parameters

in	*path*	Pointer to file pathname.

Note: Further details are available at
http://man7.org/linux/man-pages/man2/unlink.2.html

Returns

SUCCESS – 0
FAILURE – -1

14.16.2.24 int unlinkat (int *dirfd,* const char * *pathname,* int *flags*)

Removes a relative directory.

Parameters

in	*dirfd*	Directory file descriptor relative to which the file path is determined.
in	*pathname*	Pointer to the file path, relative to the directory described by fd.
in	*flags*	Bit mask flag.

Note: Values for flags are described in
http://man7.org/linux/man-pages/man2/unlinkat.2.html

Returns

SUCCESS – 0
FAILURE – -1

14.16.2.25 int32_t write (int *fd,* const void * *buf,* uint32_t *count*)

Writes a number of bytes to a given file.

Parameters

in	*fd*	File descriptor.
in	*buf*	Pointer to buffer to write from.
in	*count*	Number of bytes to write.

Note: Further details are available at
http://man7.org/linux/man-pages/man2/write.2.html

Returns

SUCCESS – Number of bytes written
FAILURE – -1

14.17 Fuses

14.17.1 Detailed Description

14.17.2 Function Documentation

14.17.2.1 int qsee_blow_sw_fuse (qsee_sw_fuse_t *fuse_num*)

Changes the state of the software fuse.

Parameters

in	*fuse_num*	Software fuse to blow.

Returns

SUCCESS – 0
FAILURE – Negative

14.17.2.2 int qsee_fuse_read (uint32_t *row_address,* int32_t *addr_type,* uint32_t *row_data[2],* uint32_t * *qfprom_api_status*)

Reads the row data from the specified QFPROM row address.

Parameters

in	*row_address*	Row address in the QFPROM region from which the row data is read.
in	*addr_type*	Raw (uncorrected) or FEC-corrected data.
out	*row_data[ ]*	Array of the data (size 2x32 bits) to be read.
out	*qfprom_api_status*	Pointer to return value from the QFPROM API.

Returns

SUCCESS – 0
FAILURE – Negative

14.17.2.3 int qsee_fuse_write (uint32_t *raw_row_address,* uint32_t *row_data[2],* uint32_t *bus_clk_khz,* uint32_t * *qfprom_api_status*)

Writes the row data to the specified QFPROM raw row address.

Parameters

in	*raw_row_address*	Row address in the QFPROM region to which the row data is to be written.
in	*row_data[ ]*	Array of the data (size 2x32 bits) to write into QFPROM region.
in	*bus_clk_khz*	Bus clock frequency (in kHz) connected to QFPROM. The value is ignored.

out	*qfprom_api_status*	Pointer to return value from the QFPROM API.

Returns

SUCCESS – 0
FAILURE – Negative

14.17.2.4 int qsee_is_sw_fuse_blown (qsee_sw_fuse_t *fuse_num,* bool * *is_blown,* uint32_t *is_blown_sz*)

Queries whether the given software fuse is blown.

Parameters

in	*fuse_num*	Software fuse to query.
out	*is_blown*	Pointer to a boolean indicating whether the given software fuse is blown.
in	*is_blown_sz*	Return pointer size.

Returns

SUCCESS – 0
FAILURE – Negative

Warning

This function is not thread-safe.

14.18 HASH

14.18.1 Detailed Description

qsee_shim.h defines the following return values:

- SUCCESS – 0
- FAILURE – -1

14.18.2 Enumeration Type Documentation

14.18.2.1 enum QSEE_HASH_ALGO_ET

Hashing algorithm.

Enumerator

QSEE_HASH_NULL Do not perform any hashing.
QSEE_HASH_SHA1 SHA1 Hash algorithm.
QSEE_HASH_SHA256 SHA256 Hash algorithm.
QSEE_HASH_SHA384 SHA384 Hash algorithm.
QSEE_HASH_SHA512 SHA512 Hash algorithm.
QSEE_HASH_SHA224 SHA224 Hash algorithm.
QSEE_HASH_INVALID Unknown or invalid hash algorithm.

14.18.2.2 enum QSEE_HASH_MODE_ET

Hashing modes.

Enumerator

QSEE_HASH_MODE_HASH Plain SHA.
QSEE_HASH_MODE_HMAC HMAC SHA.
QSEE_HASH_MODE_INVALID Unknown or invalid hash mode.

14.18.2.3 enum QSEE_HASH_PARAM_ET

Parameters for hash.

Enumerator

QSEE_HASH_PARAM_MODE Hashing mode parameter.
QSEE_HASH_PARAM_HMAC_KEY HMAC key value parameter.
QSEE_HASH_PARAM_SEQ Hashing sequence parameter.
QSEE_HASH_PARAM_INVALID Unknown or invalid parameter.

14.18.2.4 enum QSEE_HASH_SEQ_ET

Hashing sequences.

Enumerator

QSEE_HASH_MODE_FIRST First block position.
QSEE_HASH_MODE_LAST Last block position.
QSEE_HASH_MODE_INVALID2 Unknown or invalid hashing sequence position.

14.18.3 Function Documentation

14.18.3.1 int qsee_hash (QSEE_HASH_ALGO_ET *alg*, const uint8_t * *msg*, uint32_t *msg_len*, uint8_t * *digest*, uint32_t *digest_len*)

Creates a message digest hash using the specified algorithm.

Parameters

in	*alg*	Hash algorithm.
in	*msg*	Pointer to message to hash.
in	*msg_len*	Message length.
out	*digest*	Pointer to digest to store.
in	*digest_len*	Length of the output digest buffer (in bytes). Must be at least equal to hash size for requested hash algorithm.

Returns

QSEE_HASH_SUCCESS – Function executes successfully
QSEE_HASH_FAILURE – Any error encountered during hash creation

14.18.3.2 int qsee_hash_final (const qsee_hash_ctx * *hash_ctx*, uint8_t * *digest*, uint32_t *digest_len*)

Computes the digest hash value.

Parameters

in	*hash_ctx*	Pointer to hash context.
out	*digest*	Pointer to output message digest hash.
in	*digest_len*	Length of the output digest buffer (in bytes). Must be at least equal to hash size for requested hash algorithm.

Returns

SUCCESS – Function executes successfully
FAILURE – Any error encountered during final hash calculation

Dependencies

qsee_hash_init

14.18.3.3 int qsee_hash_free_ctx (qsee_hash_ctx * *hash_ctx*)

Releases all resources with a given hash context.

Parameters

in	*hash_ctx*	Pointer to hash context for deletion.

Returns

QSEE_HASH_SUCCESS – Function executes successfully
QSEE_HASH_FAILURE – Any error encountered while freeing hash context

14.18.3.4 int qsee_hash_init (QSEE_HASH_ALGO_ET *alg,* qsee_hash_ctx ** *hash_ctx*)

Intializes a hash context for update and final functions.

Parameters

in	*alg*	Algorithm standard to use.
out	*hash_ctx*	Double pointer to hash context.

Returns

QSEE_HASH_SUCCESS – Function executes successfully
QSEE_HASH_FAILURE – Any error encountered during hash initialization

14.18.3.5 int qsee_hash_reset (qsee_hash_ctx * *hash_ctx*)

Resets hash context; does not reset keys.

Parameters

in,out	*hash_ctx*	Pointer to hash context.

Returns

SUCCESS – Function executes successfully
FAILURE – Any error encountered during hash reset

14.18.3.6 int qsee_hash_set_param (const qsee_hash_ctx * *hash_ctx,* QSEE_HASH_PARAM_ET *param_id,* const void * *param,* uint32_t *param_len*)

Modifies parameters for a given hash operation.

Parameters

in	*hash_ctx*	Pointer to hash context.
in	*param_id*	Parameter to modify.
in	*param*	Pointer to parameter value to set.
in	*param_len*	Param length (in bytes).

Returns

SUCCESS – Function executes successfully
FAILURE – Any error encountered during parameter setting

14.18.3.7 int qsee_hash_update (const qsee_hash_ctx * *hash_ctx*, const uint8_t * *msg*, uint32_t *msg_len*)

Hashes some data into the hash context structure (must be initialized by qsee_hash_init()).

Parameters

in,out	*hash_ctx*	Pointer to hash context.
in	*msg*	Pointer to data to hash.
in	*msg_len*	Data length to hash.

Returns

SUCCESS – Function executes successfully
FAILURE – Any error encountered during hash update

Dependencies

qsee_hash_init

14.18.3.8 int qsee_hashcipher_decrypt (const qsee_cipher_ctx * *cipher_ctx*, const qsee_hash_ctx * *hash_ctx*, uint8_t * *ct*, uint32_t *ct_len*, uint8_t * *pt*, uint32_t *pt_len*, uint8_t * *digest*, uint32_t *digest_len*)

Performs Hash and Cipher Decrypt operation simultaneously.

The memory allocated for the output plaintext buffer must be large enough to hold the ciphertext equivalent. If a padding scheme is selected, the plaintext output length may be up to one block size smaller than the ciphertext length. If the output buffer is not large enough to hold the decrypted results, an error is returned.

Parameters

in	*cipher_ctx*	Pointer to cipher context.
in	*hash_ctx*	Pointer to the hash context.
in	*ct*	Pointer to input ciphertext buffer.
in	*ct_len*	Input ciphertext buffer length.
out	*pt*	Pointer to output plaintext buffer.
in	*pt_len*	Output plaintext buffer length.
out	*digest*	Pointer to digest to store.
in	*digest_len*	Length of the output digest buffer (in bytes). Must be at least equal to hash size for requested hash algorithm.

Returns

SUCCESS – Function executes successfully
FAILURE – Any error encountered during decryption

14.18.3.9 int qsee_hashcipher_encrypt (const qsee_cipher_ctx * *cipher_ctx,* const qsee_hash_ctx * *hash_ctx,* uint8_t * *pt,* uint32_t *pt_len,* uint8_t * *ct,* uint32_t *ct_len,* uint8_t * *digest,* uint32_t *digest_len*)

Performs Hash and Cipher Encrypt operation simultaneously.

The memory allocated for the output cipher text buffer must be large enough to hold the plaintext equivalent. If a padding scheme is selected, the ciphertext buffer length may be up to one block size larger than the plaintext length. If the output buffer is not large enough to hold the encrypted results, an error is returned.

Parameters

in	*cipher_ctx*	Pointer to cipher context.
in	*hash_ctx*	Pointer to hash context.
in	*pt*	Pointer to input plaintext buffer.
in	*pt_len*	Input plaintext buffer length.
out	*ct*	Pointer to output ciphertext buffer.
in	*ct_len*	Output ciphertext buffer length.
out	*digest*	Pointer to digest to store.
in	*digest_len*	Length of the output digest buffer (in bytes). Must be at least equal to hash size for requested hash algorithm.

Returns

SUCCESS – Function executes successfully
FAILURE – Any error encountered during encryption

14.19 HMAC

14.19.1 Detailed Description

qsee_shim.h defines the following return values:

- SUCCESS – 0
- FAILURE – -1

14.19.2 Enumeration Type Documentation

14.19.2.1 enum QSEE_HMAC_ALGO_ET

Supported HMAC algorithms.

Enumerator

QSEE_HMAC_SHA1 SHA1 algorithm.
QSEE_HMAC_SHA256 SHA256 algorithm.
QSEE_HMAC_SHA384 SHA384 algorithm.
QSEE_HMAC_SHA512 SHA512 algorithm.
QSEE_HMAC_INVALID Unknown or invalid alogorithm.

14.19.2.2 enum QSEE_HMAC_PARAM_ET

HMAC parameters.

Enumerator

QSEE_HMAC_PARAM_KEY Key value parameter.
QSEE_HMAC_PARAM_INVALID Unknown or invalid parameter.

14.19.3 Function Documentation

14.19.3.1 int qsee_hmac (QSEE_HMAC_ALGO_ET *alg*, const uint8_t * *msg*, uint32_t *msg_len*, const uint8_t * *key*, uint16_t *key_len*, uint8_t * *msg_digest*)

Creates HMAC per FIPS pub 198 using the specified hash algorithm.

Parameters

in	*alg*	HMAC algorithm to use.
in	*msg*	Pointer to message to authenticate.
in	*msg_len*	Length of message in bytes.
in	*key*	Pointer to input key to HMAC algorithm.
in	*key_len*	Input key length (in bytes).
out	*msg_digest*	Pointer to message digest (memory provided by caller).

Returns

SUCCESS – Function executes successfully
FAILURE – Any error encountered during HMAC creation

14.19.3.2 int qsee_hmac_final (qsee_hmac_ctx * *ctx,* uint8_t * *msg_digest,* uint32_t *msg_digest_length*)

Final operation for HMAC per FIPS pub 198 using the specified hash algorithm.

Parameters

in	*ctx*	Pointer to HMAC context.
out	*msg_digest*	Pointer to message digest (memory provided by caller).
in	*msg_digest_length*	Message length (in bytes).

Returns

SUCCESS – Function executes successfully
FAILURE – Any error during final operation

14.19.3.3 int qsee_hmac_free_ctx (qsee_hmac_ctx * *hmac_ctx*)

Releases all resources with given HMAC context.

Parameters

in	*hmac_ctx*	Pointer to HMAC context to delete.

Returns

SUCCESS – Function executes successfully
FAILURE – Context is invalid

14.19.3.4 int qsee_hmac_init (QSEE_HMAC_ALGO_ET *alg,* qsee_hmac_ctx ** *ctx*)

Initializes HMAC per FIPS pub 198 using the specified hash algorithm.

Parameters

in	*alg*	Algorithm for HMAC.
out	*ctx*	Double pointer to HMAC context.

Returns

SUCCESS – Function executes successfully
FAILURE – Unable to initialize HMAC

14.19.3.5 int qsee_hmac_set_param (qsee_hmac_ctx * *hmac_ctx,* QSEE_HMAC_PARAM_ET *param_id,* const void * *param,* uint32_t *param_len*)

Modifies parameters for a given HMAC operation.

Parameters

in,out	*hmac_ctx*	Pointer to HMAC context.
in	*param_id*	Parameter to modify.
in	*param*	Pointer to parameter value to set.
in	*param_len*	Param length (in bytes).

Returns

SUCCESS – Function executes successfully
FAILURE – Any error during parameter setting

14.19.3.6 int qsee_hmac_update (qsee_hmac_ctx * *ctx*, const uint8_t * *msg*, uint32_t *msg_len*)

Updates HMAC per FIPS pub 198 using the specified hash algorithm.

Parameters

in,out	*ctx*	Pointer to HMAC context.
in	*msg*	Pointer to message to authenticate.
in	*msg_len*	Message length (in bytes).

Returns

SUCCESS – Function executes successfully
FAILURE – Any error during HMAC update

14.20 Heap Allocation

14.20.1 Detailed Description

14.20.2 Function Documentation

14.20.2.1 static void* qsee_calloc (size_t *num,* size_t *size*) [inline], [static]

Allocates and zero-initializes a block large enough to hold 'num' items of size 'size'.

Parameters

in	*num*	Number of items to allocate from the heap.
in	*size*	Item size in bytes.

Returns

Returns a pointer to the newly allocated block, or NULL if the block could not be allocated.

14.20.2.2 void qsee_free (void * *ptr*)

Deallocates the ptr block of memory.

Parameters

in	*ptr*	Pointer to block to be freed.

14.20.2.3 void* qsee_malloc (size_t *size*)

Allocates a block of size bytes from the heap. If size is 0, returns NULL.

Parameters

in	*size*	Number of bytes to allocate from the heap.

Returns

Returns a pointer to the newly allocated block, or NULL if the block could not be allocated.

Referenced by qsee_zalloc().

14.20.2.4 int qsee_query_heap_info (heap_info_t * *heap_info_ptr*)

Provides the heap information for a TA.

Parameters

out	*heap_info_ptr*	pointer to TA heap information.

Returns

0 for success, else -1.

14.20.2.5 void* qsee_realloc (void * *ptr,* size_t *size*)

Resizes a block of memory previously allocated using qsee_malloc() or qsee_realloc(). If size is zero, returns NULL.

Parameters

in	*ptr*	Pointer to previously allocated block. If NULL is passed, this function is akin to qsee_malloc().
in	*size*	New block size.

Returns

Returns a pointer to the newly allocated block, or NULL if the block could not be allocated.

14.20.2.6 static void* qsee_zalloc (size_t *size*) [inline], [static]

Allocates and zero-intializes a block of size bytes.

Parameters

in	*size*	Number of bytes to allocate from the heap.

Returns

Returns a pointer to the newly allocated block, or NULL if the block could not be allocated.

Referenced by qsee_calloc().

14.21 I2C

14.21.1 Detailed Description

14.21.2 Function Documentation

14.21.2.1 int qsee_i2c_close (qsee_i2cpd_device_id_t *device_id*)

Transfers access of I2C QUP back to REE.

Prerequisite: This function requires a successful call to qsee_i2c_open().

This function returns success only if the following operations are succesful:

1. Remove exclusive access lock to the I2C bus, then close device.
2. Remove QUP block protection from GSBI3.
3. Remove control block protection for GSBI3.
4. Reenable the I2C interrupt.
5. Deregister the I2C interrupt. The I2C interrupt transfers back to REE.

Parameters

in	*device_id*	I2C Device ID to close.

Returns

SUCCESS – 0
FAILURE – Negative

14.21.2.2 int qsee_i2c_open (qsee_i2cpd_device_id_t *device_id*)

Transfers access to the I2C bus to the calling application.

This function returns success only if the following operations are successful:

1. Obtain a handle to the requested device, then open device.
2. Lock I2C bus for exclusive access.
3. Register the I2C interrupt in QTEE, to transfer the interrupt from REE to TEE.
4. Disable the I2C interrupt.
5. Protect the control block for the device.
6. Protect the QUP block for the device.

Parameters

in	*device_id*	I2C device ID to open.

Returns

SUCCESS – 0
FAILURE – Negative

14.21.2.3 int qsee_i2c_read (qsee_i2cpd_device_id_t *device_id*, const qsee_i2c_config_t * *p_config*, qsee_i2c_transaction_info_t * *p_read_info*)

Reads data from a slave device.

Prerequisite: This function requires a successful call to qsee_i2c_open().

The read aborts if the slave device does not acknowledge control/address bytes written to it (before the data read starts). A read operation always terminates with a STOP condition.

An error is generated if the bus is in an inconsistent state, i.e., uninitialized or busy.

The following diagram shows the bus activity during a read. The second START shown is to be interpreted as a repeated START and may be replaced by a STOP and START by some protocols.

```
            |<------optional------>|
-------------------------------------------------------------
|S|       | |        | |S|       | |                   |N|S|
|T|Control|A|Address |A|T|Control|A|      Data         |O|T|
|A| Byte  |C|[Offset |C|A| Byte  |C|                   |A|O|
|R|       |K| or Reg]|K|R|       |K|                   |C|P|
|T|       | |        | |T|       | |                   |K| |
-------------------------------------------------------------
```

The address may be 0, 1, or 2 bytes depending on the slave. Every address byte written to the slave device must be acknowledged.

This function is blocking. It will return when either data has been read, or an error has occurred.

Parameters

in	*device_id*	I2C device ID being read from.
in	*p_config*	Pointer to bus and slave configuration for this read transaction.
in,out	*p_read_info*	Pointer to read information, i.e. buffer, length, etc.

Returns

SUCCESS – 0
FAILURE – Negative

14.21.2.4 int qsee_i2c_write (qsee_i2cpd_device_id_t *device_id*, const qsee_i2c_config_t * *p_config*, qsee_i2c_transaction_info_t * *p_write_info*)

Writes data to a slave device.

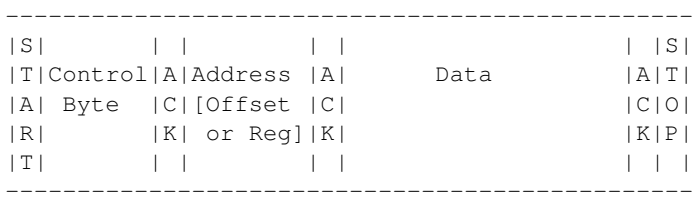
Prerequisite: This function requires a successful call to qsee_i2c_open().

The write aborts if the slave device does not acknowledge control/address/data bytes written to it. The write operation always terminates with a STOP condition.

If there is a write attempt with a count of zero, the slave device is selected and returns success if the slave device acknowledges it. Otherwise, a failure is returned. This is useful if you want to test if a slave device is addressable, as in NVRAM devices that may be busy when performing internal cache writes.

The following diagram shows the bus activity during a write operation.

```
            |<--opt.-->|
```



The address may be 0, 1, or 2 bytes depending on the slave. Every address byte written to the slave device must be acknowledged.

Parameters

in	*device_id*	I2C device ID being written to.
in	*p_config*	Pointer to bus and slave configuration for this write transaction.
in,out	*p_write_info*	Pointer to write information, i.e. buffer, length, etc.

Returns

SUCCESS – 0
FAILURE – Negative

14.22 Interrupts

14.22.1 Detailed Description

14.22.2 Function Documentation

14.22.2.1 int qsee_disable_all_interrupts (qsee_intsrc_t * *restore_mask*)

Disables all supported interrupt sources.

Parameters

out	*restore_mask*	Pointer to a bit mask of masked interrupt sources prior to calling this function. Can restore the interrupt mask state after calling this function by passing the value to qsee_set_intmask().

Warning

The interrupt state is not preserved when a syscall is made to REE.

Returns

SUCCESS – 0
FAILURE – Negative

14.22.2.2 int qsee_get_intmask (qsee_intsrc_t * *int_mask*)

Returns the masking status of all supported interrupt sources.

Parameters

out	*int_mask*	Pointer to a bit mask of currently masked interrupt sources. A value of 1 indicates the interrupt source is currently masked.

Warning

The interrupt state is not preserved when a syscall is made to REE.

Returns

SUCCESS – 0
FAILURE – Negative

14.22.2.3 int qsee_set_intmask (qsee_intsrc_t *int_mask*)

Masks and unmasks interrupt sources as directed by the input.

Parameters

in	*int_mask*	A bit mask of interrupts to enable and disable. Value of 1 indicates the interrupt source should be masked. Value of zero indicates the interrupt source should be unmasked.

Returns

SUCCESS – 0
FAILURE – Negative

14.23 KDF

14.23.1 Detailed Description

14.23.2 Function Documentation

14.23.2.1 int qsee_kdf (const void * *key*, unsigned int *key_len*, const void * *label*, unsigned int *label_len*, const void * *context*, unsigned int *context_len*, void * *output*, unsigned int *output_len*)

KDF key derivation algorithm.

Parameters

in	*key*	Pointer to key derivation key.
in	*key_len*	Key derivation key length in bytes.
in	*label*	Pointer to key derivation label.
in	*label_len*	Key derivation label length in bytes.
in	*context*	Pointer to key derivation context.
in	*context_len*	Key derivation context length in byte.
out	*output*	Pointer to derived key.
in	*output_len*	Derived key length in bytes.

Detailed description

The software is a three-level stack:

- First level: AES
- Second level: CMAC algorithm from NIST SP 800-38B
- Third level: Counter-based algorithm from NIST SP 800-108 (named KDF in implementation)

The inputs are:

- Key derivation key (key, key_len)
- Label (label, label_len)
- Context (context, context_len)

The output will be output_len bytes long stored at output. All sensitive data is set to zero before return.

When a key is set to NULL, key_len will be ignored. QTEE sets a 32 bytes key (not accessible from outside Trustzone) as the input key.

Returns

QSEE_KDF_SUCCESS – Function executes successfully.
QSEE_KDF_INVALID – Invalid parameters.
QSEE_KDF_FAILURE – All other errors during key derivation.

14.24 Key Manager

14.24.1 Detailed Description

14.24.2 Function Documentation

14.24.2.1 int qsee_km_close_cm_session (void * *hdl*)

Once the application completes using the CryptoManager (CM) core, it can relieve its usage by calling this function.

Parameters

in	*hdl*	hdl as received by qsee_km_init.

Returns

SUCCESS – 0
FAILURE – Negative

14.24.2.2 int qsee_km_cmd_service (void * *hdl*, km_cm_cmdid *cmd_hdl_id*, uint32_t *flag*, uint8_t * *buffer*, uint32_t *buff_len*)

This function passes the data blob to be unwrapped by CM hardware. The function only works if the open CM session passed.

Parameters

in	*hdl*	hdl generated by calling qsee_km_init.
in	*cmd_hdl_id*	cmd id for operation to be performed by CM, km_cm_cmdid type.
in	*flag*	Indicates suboperation, e.g., Key Controller.
in,out	*buffer*	Data command to feed to CM; also reads back information as required by command.
in	*buff_len*	Buffer length (in bytes).

Returns

SUCCESS – 0
FAILURE – Negative

14.24.2.3 int qsee_km_deinit (void * *hdl*)

Once the application no longer needs this service, it can free the context by calling this function.

Parameters

in	*hdl*	hdl as received by qsee_km_init.

Returns

SUCCESS – 0
FAILURE – -1

14.24.2.4 int qsee_km_getkey (void * *hdl*, uint32_t * *keyptr*, uint32_t *keysz*)

If the Key Controller is a sub-operation during qsee_km_cmd_service, then hardware has a key. This key can be requested by an initiating client by calling this function.

Parameters

in	*hdl*	hdl as received by qsee_km_init
out	*keyptr*	User allocated buffer, key is written to this
in	*keysz*	Length of the keyptr buffer

Returns

SUCCESS – 0
FAILURE – Negative

14.24.2.5 int qsee_km_init (void ** *hdl*)

Initialize a key manager context.

Parameters

out	*hdl*	Context initialized and returned in hdl.

Returns

SUCCESS – 0
FAILURE – Negative

14.24.2.6 int qsee_km_open_cm_session (void * *hdl*)

An open cm session returns success only if CM is not busy.

Parameters

in	*hdl*	hdl generated by qsee_km_init

Returns

SUCCESS – 0
FAILURE – Negative

14.25 Logging

14.25.1 Detailed Description

14.25.2 Function Documentation

14.25.2.1 void qsee_log (uint8_t *pri,* const char * *fmt,* ...)

Collects a set of logs in internal buffers and flushes the logs to a rolling log file at predetermined thresholds.

Parameters

in	*pri*	Priority of message to be logged.
in	*fmt*	Pointer to a string describing the message format.
in	...	Variable argument list.

Maximum length of TA message allowed for qsee logging is 120. If TA passes message of length $>$ 120, then message would be trimmed down to 120 characters and then printed in qsee_log.

Referenced by tz_app_init(), and tz_app_shutdown().

14.25.2.2 uint8_t qsee_log_get_mask (void)

Retrieves the bitmask for currently set log levels.

Returns

Bitmask value of the log levels set.

Referenced by tz_app_init().

14.25.2.3 void qsee_log_set_mask (uint8_t *pri_flags*)

Allows user to set logs using a bitmask defined by level:

- QSEE_LOG_MSG_LOW
- QSEE_LOG_MSG_MED
- QSEE_LOG_MSG_HIGH
- QSEE_LOG_MSG_ERROR
- QSEE_LOG_MSG_FATAL

This function accepts bitmask values in range QSEE_LOG_MSG_LOW to QSEE_LOG_MSG_FATAL. Any value outside this range is ignored.

Parameters

in	*pri_flags*	OR(ed) bitmask for desired log levels.

Referenced by tz_app_init().

14.25.2.4 void qsee_printf (const char * *fmt,* ...)

Collects a set of logs in internal buffers and flushes the logs to a rolling log file.

Parameters

in	*fmt*	Pointer to string describing message format.
in	...	Variable argument list.

14.26 Message Passing

14.26.1 Detailed Description

These functions are for a trusted application to prepare a message that can be sent to another trusted application.

14.26.2 Function Documentation

14.26.2.1 int qsee_decapsulate_inter_app_message (char * *source_app_name,* uint8_t * *in_buf,* uint32_t *in_len,* uint8_t * *out_buf,* uint32_t * *out_len*)

Decapsulates a message sent to this application.

This function authenticates the message, decrypts the input buffer, and writes the plaintext message into the supplied output buffer. The input buffer must be prepared by qsee_encapsulate_inter_app_message(), containing a header and MAC.

Parameters

out	*source_app_name*	Pointer to the sending application identity (maximum length is 128 bytes).
in	*in_buf*	Pointer to input buffer.
in	*in_len*	Exact size of in_buf.
out	*out_buf*	Pointer to output buffer.
in,out	*out_len*	Pointer to exact size of output buffer (must be greater than or equal to in_len).

Returns

SUCCESS – 0
All other values indicate failure and correspond to a specific error code.

14.26.2.2 int qsee_encapsulate_inter_app_message (char * *dest_app_name,* uint8_t * *in_buf,* uint32_t *in_len,* uint8_t * *out_buf,* uint32_t * *out_len*)

Prepares a message for another trusted application.

This function writes the AES128-CTR encrypted message in to the supplied output buffer, prepended with a header and appended with HMAC-SHA256. This output buffer can be given to the receiving application and then passed into qsee_decapsulate_inter_app_message() for authentication and decryption. The actual data exchange (passing the encrypted buffer) between the trusted applications can be done between their clients running in the REE.

Parameters

in	*dest_app_name*	Pointer to destination application name (maximum length 128 bytes).
in	*in_buf*	Pointer to an input buffer containing the message.
in	*in_len*	Exact input buffer length.
out	*out_buf*	Pointer to output buffer.

in,out	*out_len*	Exact output buffer length (must be greater than in_len by 144 bytes to allow room for header and MAC). out_len modified to reflect the exact length of the data written into out_buf.

Returns

SUCCESS – 0
All other values indicate failure and correspond to a specific error code.

14.27 OEM Buffer

14.27.1 Detailed Description

14.27.2 Function Documentation

14.27.2.1 size_t qsee_get_oem_buffer_length (void)

Retrieves OEM buffer length supported by QSEE.

Returns

SUCCESS – Internal OEM buffer length supported by QTEE
FAILURE – 0

14.27.2.2 int qsee_read_oem_buffer (size_t *offset*, void * *data*, size_t *data_len*)

Copies from OEM buffer at the given offset into the output buffer up to the size of the output buffer.
If the output buffer is larger than the OEM buffer at the provided offset, remaining bytes of the output buffer remain unchanged.

Parameters

in	*offset*	Offset to read from OEM buffer into.
out	*data*	Destination for copying source data into.
in	*data_len*	Amount of data to read from OEM buffer.

Returns

QSEE_OEM_BUFFER_SUCCESS on success.
QSEE_OEM_BUFFER_ERROR on error in MINK IPC.
QSEE_OEM_BUFFER_ERROR_INVALID_BUFFER_AND_OFFSET if the buffer length and offset parameters cause an integer overflow.

14.27.2.3 int qsee_write_oem_buffer (size_t *offset*, const void * *data*, size_t *data_len*)

Copies into the OEM buffer at the given offset up to the size of the parameter buffer.
If the parameter buffer length and offset combine to a greater length than the OEM buffer length, an error is returned and no data is copied.

Parameters

in	*offset*	Offset to begin writing into the destination.
in	*data*	Data source to copy to OEM buffer.
in	*data_len*	Data length to copy from source.

Returns

QSEE_OEM_BUFFER_SUCCESS on success.
QSEE_OEM_BUFFER_ERROR on error in MINK IPC.

QSEE_OEM_BUFFER_ERROR_OFFSET_OUT_OF_BOUNDS if the offset parameter falls outside of the size of the OEM buffer or if this target does not support OEM buffer.
QSEE_OEM_BUFFER_ERROR_SOURCE_BUFFER_TOO_LARGE if the provided buffer cannot fit entirely in the OEM buffer when written at the given offset.
QSEE_OEM_BUFFER_ERROR_INVALID_BUFFER_AND_OFFSET if the buffer length and offset parameters cause an integer overflow.

14.28 OEM Utilities

14.29 Pseudo Random Number Generator

14.29.1 Detailed Description

14.29.2 Function Documentation

14.29.2.1 uint32_t qsee_prng_getdata (uint8_t * *out,* uint32_t *out_len*)

Generates a random number of a specified length.

Parameters

out	*out*	Output data buffer.
in	*out_len*	Output data length. out_len must be at most QSEE_MAX_PRNG bytes.

Returns

Number of bytes read.

14.30 Public Key Algorithms

14.30.1 Detailed Description

Public key algorithms

14.30.2 Typedef Documentation

14.30.2.1 typedef void* QSEE_PKEY_HANDLE

Context handle to be used with any public-key operations.

14.30.3 Enumeration Type Documentation

14.30.3.1 enum QSEE_CE_ENGINE_TYPE

Crypto engine/implementation types

Enumerator

QSEE_CE_SW SW Crypto Engine
QSEE_CE_ARMV8 Crypto Engine using ARMv8 CE extention
QSEE_CE_HWIO HW Crypto Engine using HWIO Access
QSEE_CE_BAM HW Crypto Engine using BAM Access

14.30.3.2 enum qsee_crypto_err_enum

Error return values

Enumerator

QSEE_CRYPTO_SUCCESS Common QSEE Crypto Error Codes Operation successful
QSEE_CRYPTO_ERR_FAILURE Operation failed due to unknown err
QSEE_CRYPTO_ERR_INVALID_ARG Arg is not recognized
QSEE_CRYPTO_ERR_OUT_OF_RANGE Arg value is out of range
QSEE_CRYPTO_ERR_VERSION_MISMATCH Unexpected software or protocol ver.
QSEE_CRYPTO_ERR_NOT_SUPPORTED Operation not yet implemented
QSEE_CRYPTO_ERR_INVALID_CONTEXT Invalid context
QSEE_CRYPTO_ERR_OUT_OF_MEMORY Out of memory
QSEE_CRYPTO_ERR_INV_DGST_SIZE Invalid digest size
QSEE_CRYPTO_ERR_INV_DGST_CONTEXT Invalid digest context
QSEE_CRYPTO_ERR_ADD_OVERFLOW Addition Overflow
QSEE_CRYPTO_ERR_HEAP_LITE_FAILED Heap API failed
QSEE_CRYPTO_ERR_SELF_TEST_FAILED Self test failed
QSEE_CRYPTO_ERR_SET_CLK_BW Environment Error Codes Set Clock BW failed
QSEE_CRYPTO_ERR_VTOP_FAILED VA to PA failed
QSEE_CRYPTO_ERR_MUTEX_INIT Mutex Init API failed
QSEE_CRYPTO_ERR_MUTEX_LOCK Mutex Lock API failed
QSEE_CRYPTO_ERR_MUTEX_RELEASE Mutex Release API failed
QSEE_CRYPTO_ERR_HEAP_NOT_SUPPORTED Heap API not supported
QSEE_CRYPTO_ERR_HEAP_INIT_FAILED Heap init. failed
QSEE_CRYPTO_ERR_INV_BAM_CTX_SIZE Invalid BAM Driver CTX size
QSEE_CRYPTO_ERR_INV_BAM_CTX Invalid BAM Driver CTX
QSEE_CRYPTO_ERR_PRNG_CLK_FAILED PRNG Clock Vote API failed

QSEE_CRYPTO_ERR_INV_AUTH_XFER_SZ CE HWIO HAL Driver Error Codes Invalid Auth Transfer size

QSEE_CRYPTO_ERR_INV_CIPHER_XFER_SZ Invalid Cipher Transfer size

QSEE_CRYPTO_ERR_CE_HAL_CTX_INVALID Invalid CE HAL Context

QSEE_CRYPTO_ERR_BAM_INIT_FAILED CE BAM HAL Driver Error Codes BAM driver init. failed

QSEE_CRYPTO_ERR_BAM_INIT_TXPIPE BAM Tx pipe init. failed

QSEE_CRYPTO_ERR_BAM_INIT_RXPIPE BAM Rx pipe init. failed

QSEE_CRYPTO_ERR_BAM_TX_IRQ_MODE BAM Tx IRQ Mode set failed

QSEE_CRYPTO_ERR_BAM_RX_IRQ_MODE BAM Rx IRQ Mode set failed

QSEE_CRYPTO_ERR_BAM_XFR_CMD_DSC BAM Command Descriptor Xfer failed

QSEE_CRYPTO_ERR_BAM_XFR_DAT_DSC BAM Data Descriptor Xfer failed

QSEE_CRYPTO_ERR_BAM_XFR_RSLT_DSC BAM Result Dump Xfer failed

QSEE_CRYPTO_ERR_BAM_BUF_PA_XLATE Data buffer VTOP failed

QSEE_CRYPTO_ERR_BAM_PIPE_UNLCK BAM pipe unlock request failed

QSEE_CRYPTO_ERR_BAM_TX_PIPE_POLL Tx pipe poll API failed

QSEE_CRYPTO_ERR_BAM_RX_PIPE_POLL Rx pipe poll API failed

QSEE_CRYPTO_ERR_BAM_PIPE_BUSY Tx/Rx pipe is still busy

QSEE_CRYPTO_ERR_CIPHER_INV_KEY_PTR CE Cipher Error Codes

QSEE_CRYPTO_ERR_AES_SET_ENC_KEY CE Cipher - AES Error Codes

QSEE_CRYPTO_ERR_DES_SET_ENC_KEY CE Cipher - DES Error Codes

QSEE_CRYPTO_ERR_PRNG_ACCESS_ENABLE PRNG - Error Codes

QSEE_CRYPTO_ERR_PKEY_KEY_SIZE_TOO_SMALL CE Public-Key Error Codes Key size too small to be secure

QSEE_CRYPTO_ERR_PKEY_KEY_SIZE_TOO_LARGE Key size exceeds BigInteger size

QSEE_CRYPTO_ERR_PKEY_MSG_SIZE_TOO_LARGE Message size too large for padding scheme

QSEE_CRYPTO_ERR_DIGEST_ALG_UNSUPPORTED The digest algorithm is not supported

QSEE_CRYPTO_ERR_MSG_SIZE_DIGEST_MISMATCH Message size does not match digest size

QSEE_CRYPTO_ERR_PKEY_INVALID_SIGNATURE Invalid signature

QSEE_CRYPTO_ERR_PKEY_DECRYPT_KEY_SIZE_INVALID Invalid key size for decryption

QSEE_CRYPTO_ERR_PKEY_PADDED_PACKET_INVALID Invalid padding detected

QSEE_CRYPTO_ERR_PKEY_DECRYPT_BUF_TOO_SMALL Buffer provided for decrypted message is not large enough

QSEE_CRYPTO_ERR_PKEY_PADDING_PRNG_FAILED Failed to generate required number of bytes for padding

QSEE_CRYPTO_ERR_PKEY_POINT_AT_INFINITY Point at infinity

QSEE_CRYPTO_ERR_PKEY_CONV_TO_BIGVAL_FAILED Conversion from byte buffer to big-integer failed

QSEE_CRYPTO_ERR_PKEY_CONV_FROM_BIGVAL_FAILED Conversion from big-integer to byte-buffer failed

QSEE_CRYPTO_ERR_GENPRIME_BITS_INVALID CE BigInteger Error Codes Invalid number of bits provided for prime-generation

QSEE_CRYPTO_ERR_GENPRIME_PRNG_FAILED PRNG failed

QSEE_CRYPTO_ERR_BIGINT_OVERFLOW Operation would exceed maximum number of bits in BigInteger

QSEE_CRYPTO_ERR_BIGINT_NO_INVERSE No inverse present

QSEE_CRYPTO_ERR_BIGINT_INVALID_LENGTH Length field of BigInteger exceeds length of BigInteger array

QSEE_CRYPTO_ERR_BIGINT_IS_COMPOSITE BigInteger is trivially composite

QSEE_CRYPTO_ERR_MAC_ALG_NOT_SET MAC Error Codes Underlying Hash/Cipher algorithm was not set
QSEE_CRYPTO_ERR_MAC_KEY_NOT_SET MAC key was not set
QSEE_CRYPTO_ERR_KDF_MAC_ALG_NOT_SET KDF Error Codes pseudorandom number function not set
QSEE_CRYPTO_ERR_KDF_MAC_ALG_NOT_MATCH pseudorandom number function not match
QSEE_CRYPTO_ERR_KDF_R_LEN_NOT_SET r length not set
QSEE_CRYPTO_ERR_KDF_CTR_LOC_NOT_SET counter location not set
QSEE_CRYPTO_ERR_KDF_CIPHER_ALG_NOT_SET cipher algorithm not set
QSEE_CRYPTO_ERR_KDF_CIPHER_KEY_LEN_NOT_MATCH cipher key length not match
QSEE_CRYPTO_ERR_KDF_HASH_ALG_NOT_SET hash algorithm not match
QSEE_CRYPTO_ERR_KDF_PASSWORD_NOT_SET pbkdf password not set
QSEE_CRYPTO_ERR_KDF_SALT_NOT_SET pbkdf salt not set
QSEE_CRYPTO_ERR_KDF_ITERATION_NOT_SET pbkdf iteration not set
QSEE_CRYPTO_ERR_KDF_KO_LEN_NOT_SET pbkdf output key length not set
QSEE_CRYPTO_ERR_KDF_KO_LEN_TOO_LONG pbkdf output key length too long
QSEE_CRYPTO_ERR_KDF_KEY_NOT_SET kbkdf key not set
QSEE_CRYPTO_ERR_KDF_IV_NOT_SET kbkdf iv not set

14.30.3.3 enum QSEE_PKEY_ALG

Public-key algorithm types

Enumerator

QSEE_PKEY_SM2 SM2 digital signature algorithms as published by the Chinese Commercial Cryptography Administration Office
QSEE_PKEY_CURVE25519 Digital signature operations using Ed25519

14.30.3.4 enum QSEE_PKEY_PARAM_TYPE

Public-key control paramater types

Enumerator

QSEE_PKEY_PARAM_SET_KEYGEN_BITS Sets the key size (in bits)
QSEE_PKEY_PARAM_GET_KEYSIZE_BITS Gets the key size (in bytes)
QSEE_PKEY_PARAM_SET_MOD Accepts a byte-array and length to set the modulus
QSEE_PKEY_PARAM_SET_PUBKEY Accepts a byte-array and length to set the public key/exponent
QSEE_PKEY_PARAM_SET_PRIVKEY Accepts a byte-array and length to set the private exponent
QSEE_PKEY_PARAM_SET_PEERKEY Accepts a byte-array and length to set the peer public key
QSEE_PKEY_PARAM_GET_MOD Accepts a byte-array and length to return the modulus
QSEE_PKEY_PARAM_GET_PUBKEY Accepts a byte-array and length to return the public exponent/key
QSEE_PKEY_PARAM_GET_PRIVKEY Accepts a byte-array and length to return the private exponent/key
QSEE_PKEY_PARAM_SET_EC_CURVE Accepts a curve ID to initialize the EC parameters
QSEE_PKEY_PARAM_SET_COEFFICIENT_A Accepts a hexadecimal string and length to set the a-coefficient of the curve
QSEE_PKEY_PARAM_SET_COEFFICIENT_B Accepts a hexadecimal string and length to set the b-coefficient of the curve

QSEE_PKEY_PARAM_SET_BASE_X Accepts a hexadecimal string and length to set the X co-ordinate of the generator/base point of the curve

QSEE_PKEY_PARAM_SET_BASE_Y Accepts a hexadecimal string and length to set the Y co-ordinate of the generator/base point of the curve

QSEE_PKEY_PARAM_SET_BASE_N Accepts a hexadecimal string and length to set the order of the base point of the curve

QSEE_PKEY_PARAM_SET_COFACTOR Accepts a hexadecimal string and length to set the co-factor of the group generated by the base point

QSEE_PKEY_PARAM_SET_IDENTIFIER Set the user 'identifier' - used primarily with SM2 sign, verify and key exchange operations.

QSEE_PKEY_PARAM_SET_PEER_IDENTIFIER Set the peer 'identifier' - used primarily with SM2 sign, verify and key exchange operations.

QSEE_PKEY_PARAM_GET_SIGNATURE_SIZE Get the size of the signature in bytes

QSEE_PKEY_PARAM_GET_SHARED_KEY_PARAM Generate and retrieve the parameter 'R' (R=[r]G) for SM2 shared-key derivation

QSEE_PKEY_PARAM_SET_PEER_SHARED_KEY_PARAM Set the peer's parameter 'R', used in shared-key derivation for SM2

QSEE_PKEY_PARAM_SET_SHARED_KEY_INITIATOR Set whether the user is the shared-key derivation initiator

QSEE_PKEY_PARAM_SET_X25519_PUBKEY Accepts a byte-array and length to set the X25519 public key

QSEE_PKEY_PARAM_GET_X25519_PUBKEY Accepts a byte-array and length to return the X25519 public key

14.30.4 Function Documentation

14.30.4.1 int qsee_pkey_ctrl (QSEE_PKEY_HANDLE *h,* QSEE_PKEY_PARAM_TYPE *type,* int *val,* uint8_t * *buf,* size_t *isz,* size_t * *osz*)

Sets or gets control or context parameters, related to the operation of the chosen algorithm. When the parameter name contains _GET_, this typically signifies that the caller is requesting for the value of a parameter to be returned to them. Otherwise the parameter signifies the setting of a control variable.

Parameters

h	[in] handle to the pkey context
type	[in] pkey param type (parameter name indicates get/set)
val	[in] Integer input parameter
buf	[in, out] pointer to a buffer-parameter which can be used to pass data to or retrieve data from the context
isz	[in] Size-parameter, used to indicate the length of the buffer parameter
osz	[out] Size-parameter, used to indicate the length of the buffer being returned

Returns

0 if successful. negative value otherwise.

14.30.4.2 int qsee_pkey_decrypt (QSEE_PKEY_HANDLE *h*, const uint8_t * *in*, size_t *in_len*, uint8_t * *out*, size_t *out_buf_len*, size_t * *out_len*)

Decrypt the input cipher message by using private key. Parameters can be set up using the qsee_pkey_ctrl() API. The value in out_len indicates the length of the data filled into the out buffer.

Parameters

h	[in] handle to the pkey context
in	[in] pointer to cipher message
in_len	[in] cipher message length
out	[out] pointer to message output buffer
out_buf_len	[in] output buffer length
out_len	[out] length of data filled into output buffer

Returns

0 if successful. negative value otherwise.

14.30.4.3 int qsee_pkey_derive (QSEE_PKEY_HANDLE *h*, uint8_t * *out*, size_t *out_buf_len*, size_t * *out_len*)

Derive the shared secret/key

Parameters

h	[in] handle to the pkey context
out	[out] pointer to output buffer
out_buf_len	[in] output buffer length
out_len	[out] length of data filled into output buffer

Returns

0 if successful. negative value otherwise.

14.30.4.4 int qsee_pkey_encrypt (QSEE_PKEY_HANDLE *h*, const uint8_t * *in*, size_t *in_len*, uint8_t * *out*, size_t *out_buf_len*, size_t * *out_len*)

Encrypt the input message by using public key. Parameters can be set up using the qsee_pkey_ctrl() API. The value in out_len indicates the length of the data filled into the out buffer.

Parameters

h	[in] handle to the pkey context
in	[in] pointer to message
in_len	[in] message length
out	[out] pointer to cipher output buffer
out_buf_len	[in] output buffer length
out_len	[out] length of data filled into output buffer

Returns

0 if successful. negative value otherwise.

14.30.4.5 int qsee_pkey_free (QSEE_PKEY_HANDLE *h*)

Clears the pkey context and releases internal resources.

Parameters

h	[in] handle to the pkey context.

Returns

0 if successful, negative value otherwise.

14.30.4.6 int qsee_pkey_init (QSEE_PKEY_HANDLE *h*, QSEE_PKEY_ALG *algo*, QSEE_CE_ENGINE_TYPE *engine*)

Initializes a previously-allocated handle for pkey operations, based on the alogrithm and engine chosen.

Parameters

h	[in] handle to the pkey context
algo	[in] pkey algorithm type enumerated in QSEE_PKEY_ALG
engine	[in] engine type enumerated in QSEE_CE_ENGINE_TYPE. If the engine parameter is passed as 0, the default engine will be used.

Returns

0 if successful, negative value otherwise.

14.30.4.7 int qsee_pkey_keygen (QSEE_PKEY_HANDLE *h*)

Generates a key. Parameters can be set up using the qsee_pkey_ctrl() API.

Parameters

h	[in] handle to the pkey context

Returns

0 if successful. negative value otherwise.

14.30.4.8 QSEE_PKEY_HANDLE qsee_pkey_new (void)

Allocates memory for a context for pkey operations and returns a handle to the caller. It is the caller's responsiblity to call qsee_pkey_free() to properly clean up pkey context and release any associated resources.

Parameters

None	

Returns

Non zero value if successful.

14.30.4.9 int qsee_pkey_reset (QSEE_PKEY_HANDLE *h*)

Resets pkey context and clears internal states in order to re-use the pkey context for another pkey operation. One can use the interface to re-use the allocated pkey context for different pkey algorithm.

Parameters

h	[in] handle to the pkey context

Returns

0 if successful. negative value otherwise.

14.30.4.10 int qsee_pkey_sign (QSEE_PKEY_HANDLE *h*, const uint8_t * *md*, size_t *md_len*, uint8_t * *sig*, size_t *sig_buf_len*, size_t * *out_len*)

Signs the message digest using the private key and returns the signature. Parameters can be set up using the qsee_pkey_ctrl() API.

Parameters

h	[in] handle to the pkey context
md	[in] pointer to message digest
md_len	[in] message digest length
sig	[out] pointer to signature output buffer
sig_buf_len	[in] output buffer length
out_len	[out] length of data filled into output buffer

Returns

0 if successful. negative value otherwise.

14.30.4.11 int qsee_pkey_verify (QSEE_PKEY_HANDLE *h*, const uint8_t * *md*, size_t *md_len*, const uint8_t * *sig*, size_t *sig_len*)

Verify the message digest and signature by using public key. Parameters can be set up using the qsee_pkey_ctrl() API.

Parameters

h	[in] handle to the pkey context
md	[in] pointer to message digest
md_len	[in] message digest length
sig	[in] pointer to signature
sig_len	[in] length of signature.

Returns

0 if successful. negative value otherwise.

14.31 RSA

14.31.1 Detailed Description

14.31.2 Data Structure Documentation

14.31.2.1 struct QSEE_BigInt

QSEE_BigInt type is a QTEE representation of a large multi-precision integer.

Data fields

Type	Parameter	Description
QSEE_BLONG	a[QSEE_BLONGS_PER_KEY]	Array of QSEE_BLONG representing a large multi-precision integer.
int	n	Length of a in units of QSEE_BLONG_SIZE.

14.31.2.2 struct QSEE_S_BIGINT

QSEE_S_BIGINT type is a QTEE representation of a large multi-precision signed integer.

Data fields

Type	Parameter	Description
QSEE_BigInt	bi	QSEE_BigInt type representing a large multi-precision integer.
int	sign	0 for positive number representation, 1 for negative.

14.31.2.3 struct QSEE_RSA_KEY

QSEE_RSA_KEY is a QTEE representation of an RSA PKCS key.

Data fields

Type	Parameter	Description
int	bitLength	RSA key bit length.
QSEE_S_BIGINT *	d	Private exponent.
QSEE_S_BIGINT *	dP	d mod (p - 1) CRT param.
QSEE_S_BIGINT *	dQ	d mod (q - 1) CRT param,
QSEE_S_BIGINT *	e	Public exponent
QSEE_S_BIGINT *	N	Modulus.
QSEE_S_BIGINT *	p	p factor of N.
QSEE_S_BIGINT *	q	q factor of N.
QSEE_S_BIGINT *	qP	1/q mod p CRT param.

Type	Parameter	Description
int	type	Type of key: • QSEE_RSA_KEY_PUBLIC for encryption. • QSEE_RSA_KEY_PRIVATE for decryption. • QSEE_RSA_KEY_PRIVATE_CRT for decryption. • QSEE_RSA_KEY_PRIVATE_PUBLIC • QSEE_RSA_KEY_PRIVATE_CRT_PUBLIC for key generation.

14.31.2.4 struct QSEE_pkcs8_rsa_privkey_type

Structure representing an RSA private key (defined in RFC2313).

Data fields

Type	Parameter	Description
uint8_t *	coef_data	Chinese remainder Theorem coefficient: q**{-1} mod p
uint16_t	coef_len	coefficient length (in bytes)
uint8_t *	exp1_data	exponent1: d mod (p-1)
uint16_t	exp1_len	exponent1 length (in bytes)
uint8_t *	exp2_data	exponent2: d mod (q-1)
uint16_t	exp2_len	exponent2 length (in bytes)
uint8_t *	mod_data	modulus: n
uint16_t	mod_len	modulus length (in bytes)
uint8_t *	prime1_data	prime1 is the prime factor p of n: p
uint16_t	prime1_len	prime1 length (in bytes)
uint8_t *	prime2_data	prime2 is the prime factor q of n: q
uint16_t	prime2_len	prime2 length (in bytes)
uint8_t *	priv_exp_data	private exponent: d
uint16_t	priv_exp_len	private exponent length (in bytes)
uint8_t *	pub_exp_data	public exponent: e
uint16_t	pub_exp_len	public exponent length (in bytes)
uint8_t	version	Version number used for compatibility with future versions of RFC2313.

14.31.2.5 struct QSEE_pkcs8_ecc_privkey_type

Type representing an ECC private key (defined in RFC5915).

Data fields

Type	Parameter	Description
uint8_t	compressed_pubkey	Indicates if key is compressed or uncompressed. Only the following are supported: • 0x02: compressed • 0x03: compressed • 0x04: uncompressed
uint16_t	namedCurve	Identifies one of these well known curves: • secp192r1 • secp192r2 • secp192r3 • secp239r1 • secp239r2 • secp239r3 • secp256r1
uint8_t *	privkey_data	Pointer to private key.
uint16_t	privkey_len	Specifies private key length in bytes.
uint8_t *	pubkey_x_data	x coordinate
uint16_t	pubkey_x_len	Public key x coordinate length (in bytes)
uint8_t *	pubkey_y_data	y coordinate
uint8_t	pubkey_y_flag	RFU
uint16_t	pubkey_y_len	Public key y coordinate length (in bytes)
uint8_t	version	Specifies syntax version number of elliptic curve private key structure.

14.31.2.6 struct QSEE_pkcs8_dsa_privkey_type

DSA private key parameters.

Data fields

Type	Parameter	Description
uint8_t *	dummy	RFU
uint16_t	dummy_len	RFU

14.31.2.7 struct QSEE_pkcs8_dh_privkey_type

Type representing DH private key parameters.

Data fields

Type	Parameter	Description
uint8_t *	exp_data	Exponent
uint16_t	exp_len	Exponent length (in bytes).
uint8_t *	mod_data	Modulus
uint16_t	mod_len	Modulus length (in bytes).

14.31.2.8 struct QSEE_pkcs8_privkey_type

Type representing different private key types to enable generic APIs. Supported key types are:

- rsa
- dsa
- dh
- ecc

Data fields

Type	Parameter	Description
QSEE_pkcs8_algo_type	algo	Algo type
union QSEE_pkcs8_privkey_type	key	Union of all the supported private key types
uint8_t	ver	Key version

14.31.2.9 union QSEE_pkcs8_privkey_type.key

Union of all the supported private key types

Data fields

Type	Parameter	Description
QSEE_pkcs8_dh_privkey_type	dh	DH key type
QSEE_pkcs8_dsa_privkey_type	dsa	DSA key type
QSEE_pkcs8_ecc_privkey_type	ecc	ECC key type
QSEE_pkcs8_rsa_privkey_type	rsa	RSA key type

14.31.2.10 struct QSEE_RSA_OAEP_PAD_INFO

- RSA OAEP padding type
- PKCS #1 v2.1 RFC3447

Data fields

Type	Parameter	Description
int	hashidx	Index of Hash and Mask generation function desired.
unsigned char *	label	Label to add to message.
int	labellen	Label length.

Type	Parameter	Description
int	mgf_hashidx	Index of Hash for the MGF function when hashidx is used with QSEE_RSA_OAEP_HASH_IDX_xxx_CUSTOM_MGF, otherwise will be ignored

14.31.2.11 struct QSEE_RSA_PSS_PAD_INFO

- RSA PSS padding type
- PKCS #1 v2.1 RFC3447

Data fields

Type	Parameter	Description
int	hashidx	Index of Hash and Mask generation function desired.
int	saltlen	salt length

14.31.3 Define Documentation

14.31.3.1 #define QSEE_RSA_KEY_SIZE(*key*) ((key)->bi.n * QSEE_BLONG_SIZE)

QSEE RSA Key Size Macro.

14.31.4 Enumeration Type Documentation

14.31.4.1 enum QSEE_HASH_IDX

Defines the hash function for RSA signature generation.

Enumerator

QSEE_HASH_IDX_NULL Initial value.
QSEE_HASH_IDX_SHA1 SHA1.
QSEE_HASH_IDX_SHA256 SHA256.
QSEE_HASH_IDX_SHA224 SHA224.
QSEE_HASH_IDX_SHA384 SHA384.
QSEE_HASH_IDX_SHA512 SHA512.
QSEE_HASH_IDX_MD5 MD5.
QSEE_HASH_IDX_MAX Enum end limit.

14.31.4.2 enum QSEE_pkcs8_algo_type

Identifies the supported private key algorithms in pkcs8.

Enumerator

QSEE_PKCS8_RSA_ENCRYPTION Private key is type RSA.
QSEE_PKCS8_ECC_ENCRYPTION Private key is type ECC.
QSEE_PKCS8_ALGO_MAX Maximum supported key type value.
QSEE_PKCS8_INVALID Invalid PKCS key type.

14.31.4.3 enum QSEE_RSA_KEY_TYPE

Types used for ce_rsa_encrypt, ce_rsa_decrypt, and other related functions.

Enumerator

QSEE_RSA_KEY_PUBLIC Public key.
QSEE_RSA_KEY_PRIVATE Private key in non-CRT representation.
QSEE_RSA_KEY_PRIVATE_CRT Private key in CRT representation.
QSEE_RSA_KEY_PRIVATE_PUBLIC Used only for ce_rsa_key_gen in private/public key pair generation.
QSEE_RSA_KEY_PRIVATE_CRT_PUBLIC Private CRT/public key pair.
QSEE_RSA_KEY_INVALID Invalid RSA key type.

14.31.4.4 enum QSEE_RSA_OAEP_PAD_HASH_IDX

Hash algorithm index for RSA OAEP padding. The indices below are kept for backward compatbility only.

Note: To use SHA1 for Mask Generator Function (MGF) hash function correctly, use QSEE_RSA_OAEP_HASH_IDX_XXX_SHA1.

Enumerator

QSEE_RSA_OAEP_HASH_IDX_NULL Initial value.
QSEE_RSA_OAEP_HASH_IDX_SHA1 SHA1.
QSEE_RSA_OAEP_HASH_IDX_SHA256 SHA256.
QSEE_RSA_OAEP_HASH_IDX_SHA224 SHA224.
QSEE_RSA_OAEP_HASH_IDX_SHA384 SHA384.
QSEE_RSA_OAEP_HASH_IDX_SHA512 SHA512.
QSEE_RSA_OAEP_HASH_IDX_MD5 MD5.
QSEE_RSA_OAEP_HASH_IDX_SHA256_SHA1 SHA256_SHA1.
QSEE_RSA_OAEP_HASH_IDX_SHA224_SHA1 SHA224_SHA1.
QSEE_RSA_OAEP_HASH_IDX_SHA384_SHA1 SHA384_SHA1.
QSEE_RSA_OAEP_HASH_IDX_SHA512_SHA1 SHA512_SHA1.
QSEE_RSA_OAEP_HASH_IDX_SHA1_CUSTOM_MGF SHA1 with the user selected hash algorithm for MGF function.
QSEE_RSA_OAEP_HASH_IDX_SHA256_CUSTOM_MGF SHA256 with the user selected hash algorithm for MGF function.
QSEE_RSA_OAEP_HASH_IDX_SHA224_CUSTOM_MGF SHA224 with the user selected hash algorithm for MGF function.
QSEE_RSA_OAEP_HASH_IDX_SHA384_CUSTOM_MGF SHA384 with the user selected hash algorithm for MGF function.
QSEE_RSA_OAEP_HASH_IDX_SHA512_CUSTOM_MGF SHA512 with the user selected hash algorithm for MGF function.
QSEE_RSA_OAEP_HASH_IDX_MD5_CUSTOM_MGF MD5 with the user selected hash algorithm for MGF function.
QSEE_RSA_OAEP_HASH_IDX_MAX Enum end limit.

14.31.4.5 enum QSEE_RSA_PADDING_TYPE

- RSA padding type
- PKCS #1 v2.1 RFC3447

Enumerator

QSEE_RSA_PAD_PKCS1_V1_5_SIG PKCS1 v1.5 signature.
QSEE_RSA_PAD_PKCS1_V1_5_ENC PKCS1 v1.5 encryption.
QSEE_RSA_PAD_PKCS1_OAEP OAEP encryption.
QSEE_RSA_PAD_PKCS1_PSS PSS signature.
QSEE_RSA_NO_PAD No padding.
QSEE_RSA_PAD_PKCS1_PSS_AUTORECOVER_SALTLEN PSS with unknown saltlen.
QSEE_RSA_PAD_INVALID Invalid padding scheme.

14.31.5 Function Documentation

14.31.5.1 int qsee_BIGINT_read_radix (QSEE_BigInt * *a,* const char * *str,* uint32_t *radix*)

Reads a zero terminated string into a big integer.

Parameters

out	*a*	Pointer to QSEE_BigInt structure.
in	*str*	Pointer to zero terminated string.
in	*radix*	Radix.

Returns

E_SECMATH_SUCCESS – Function executes successfully.
-E_SECMATH_INVALID_ARG – Generic invalid argument.
-E_SECMATH_NOT_SUPPORTED – Operation not supported.

14.31.5.2 int qsee_BIGINT_read_unsigned_bin (QSEE_BigInt * *a,* const uint8_t * *buf,* uint32_t *len*)

Reads an unsigned buffer of bytes into a big integer.

Parameters

out	*a*	Pointer to big integer.
in	*buf*	Pointer to array of bytes.
in	*len*	Array length.

Returns

E_SECMATH_SUCCESS – Function executes successfully.
-E_SECMATH_INVALID_ARG – Generic invalid argument.
-E_SECMATH_BUFFER_OVERFLOW – Not enough space for output.

14.31.5.3 int qsee_rsa_decrypt (QSEE_RSA_KEY * *key,* QSEE_RSA_PADDING_TYPE *padding_type,* void * *padding_info,* unsigned char * *cipher,* int *cipherlen,* unsigned char * *msg,* int * *msglen*)

Performs PKCS #1 decryption, followed by v1.5 depad.

Parameters

in	*key*	Corresponding private RSA key.
in	*padding_type*	Padding type.
in	*padding_info*	OAEP padding parameters.
in	*cipher*	Ciphertext.
in	*cipherlen*	Ciphertext length (octets).
out	*msg*	Plaintext.
in,out	*msglen*	Max size and resulting size of plaintext.

Returns

CE_SUCCESS – Function executes successfully.
-CE_ERROR_NOT_SUPPORTED – Feature not supported.
-CE_ERROR_INVALID_PACKET – Invalid packet.
-CE_ERROR_BUFFER_OVERFLOW – Not enough space for output.
-CE_ERROR_NOP – Software crypto self test failed.

14.31.5.4 int qsee_rsa_encrypt (QSEE_RSA_KEY * *key,* QSEE_RSA_PADDING_TYPE *padding_type,* void * *padding_info,* const unsigned char * *msg,* int *msglen,* unsigned char * *cipher,* int * *cipherlen*)

Performs PKCS #1 v1.5 padding, followed by encryption.

Parameters

in	*key*	RSA key to encrypt to.
in	*padding_type*	Padding type.
in	*padding_info*	OAEP padding parameters.
in	*msg*	Plaintext.
in	*msglen*	Plaintext length (octets).
out	*cipher*	Ciphertext.
in,out	*cipherlen*	Max size and resulting size of ciphertext.

Returns

CE_SUCCESS – Function executes successfully.
-CE_ERROR_NOT_SUPPORTED – Feature not supported.
-CE_ERROR_INVALID_PACKET – Invalid packet.
-CE_ERROR_BUFFER_OVERFLOW – Not enough space for output.
-CE_ERROR_NOP – Software crypto self test failed.

14.31.5.5 int qsee_rsa_exptmod (QSEE_RSA_KEY * *key*, const unsigned char * *in*, int *inlen*, unsigned char * *out*, int * *outlen*, int *which*)

Computes an RSA modular exponentiation.

Parameters

in	*key*	RSA key to use.
in	*in*	Input data to send into RSA.
in	*inlen*	Input length (octets).
out	*out*	Destination.
in,out	*outlen*	Max size and resulting size of output.
in	*which*	Which exponent to use, e.g., PRIVATE or PUBLIC.

Returns

CE_SUCCESS – Function executes successfully.
-CE_ERROR_NOT_SUPPORTED – Feature not supported.
-CE_ERROR_INVALID_PACKET – Invalid packet.
-CE_ERROR_BUFFER_OVERFLOW – Not enough space for output.

Dependencies

None.

14.31.5.6 int qsee_rsa_key_gen (QSEE_RSA_KEY * *key*, int *keylen*, unsigned char * *pub_exp*, int *pub_exp_len*)

This function will generate an RSA private/public key pair as per FIPS 186-4

Parameters

out	*key*	The public/private RSA key
in	*keylen*	RSA key length (in Bytes)
in	*pub_exp*	Public exponent array
in	*pub_exp_len*	Public exponent array length

Returns

CE_SUCCESS – Function executes successfully.
-CE_ERROR_FAILURE – Generic Error.
-CE_ERROR_NOT_SUPPORTED – the feature is not supported.
-CE_ERROR_INVALID_ARG – Generic invalid argument.
-CE_ERROR_BUFFER_OVERFLOW – Not enough space for output.
-CE_ERROR_NO_MEMORY – Out of memory.
-CE_ERROR_INVALID_SIGNATURE – Invalid signature.

14.31.5.7 int qsee_rsa_sign_ex (QSEE_RSA_KEY * *key*, QSEE_RSA_PADDING_TYPE *padding_type*, void * *padding_info*, QSEE_HASH_IDX *hashidx*, const unsigned char * *msg*, int *msglen*, unsigned char * *signature*, int * *siglen*)

This function hashes the input plaintext, performs PKCS #1 padding then signs the signature. This is the offical FIPS certifiable RSA signing API.

Parameters

in	*key*	The private RSA key to use
in	*padding_type*	Type of padding
in	*padding_info*	PSS padding parameters
in	*hashidx*	The index of the hash desired
in	*msg*	The msg to sign (octets)
in	*msglen*	The length of the msg to sign
out	*signature*	The signature
in,out	*siglen*	The max size and resulting size of the signature

Returns

CE_SUCCESS – Function executes successfully.
-CE_ERROR_FAILURE – Generic Error.
-CE_ERROR_NOT_SUPPORTED – The feature is not supported.
-CE_ERROR_INVALID_ARG – Generic invalid argument.
-CE_ERROR_BUFFER_OVERFLOW – Not enough space for output.
-CE_ERROR_NO_MEMORY – Out of memory.
-CE_ERROR_NOP – SW crypto self test failed.

14.31.5.8 int qsee_rsa_sign_hash (QSEE_RSA_KEY * *key*, QSEE_RSA_PADDING_TYPE *padding_type*, void * *padding_info*, QSEE_HASH_IDX *hashidx*, const unsigned char * *hash*, int *hashlen*, unsigned char * *signature*, int * *siglen*)

Performs PKCS #1 padding, then signs the signature.

Parameters

in	*key*	Private RSA key to use.
in	*padding_type*	Padding type.
in	*padding_info*	PSS padding parameters.
in	*hashidx*	Hash index desired.
in	*hash*	Hash to sign (octets).
in	*hashlen*	Hash length to sign.
out	*signature*	Signature.
in,out	*siglen*	Max size and resulting size of signature.

Returns

CE_SUCCESS – Function executes successfully.
-CE_ERROR_FAILURE – Generic error.
-CE_ERROR_NOT_SUPPORTED – Feature not supported.
-CE_ERROR_INVALID_ARG – Generic invalid argument.
-CE_ERROR_BUFFER_OVERFLOW – Not enough space for output.
-CE_ERROR_NO_MEMORY – Out of memory.
-CE_ERROR_NOP – Software crypto self test failed.

14.31.5.9 int qsee_rsa_verify_signature (QSEE_RSA_KEY * *key,* QSEE_RSA_PADDING_TYPE *padding_type,* void * *padding_info,* QSEE_HASH_IDX *hashidx,* unsigned char * *hash,* int *hashlen,* unsigned char * *sig,* int *siglen*)

Performs PKCS #1 padding, then verifies the signature.

Parameters

in	*key*	Private RSA key to use.
in	*padding_type*	Padding type.
in	*padding_info*	PSS padding parameters.
in	*hashidx*	Hash index desired.
in	*hash*	Hash to sign (octets).
in	*hashlen*	Hash length to sign.
in	*sig*	Signature.
in	*siglen*	Max size and resulting size of signature.

Returns

CE_SUCCESS – Function executes successfully.
-CE_ERROR_FAILURE – Generic Error.
-CE_ERROR_NOT_SUPPORTED – Feature not supported.
-CE_ERROR_INVALID_ARG – Generic invalid argument.
-CE_ERROR_BUFFER_OVERFLOW – Not enough space for output.
-CE_ERROR_NO_MEMORY – Out of memory.
-CE_ERROR_INVALID_SIGNATURE – Invalid signature.
-CE_ERROR_NOP – Software crypto self test failed.

14.31.5.10 int qsee_rsa_verify_signature_ex (QSEE_RSA_KEY * *key,* QSEE_RSA_PADDING_TYPE *padding_type,* void * *padding_info,* QSEE_HASH_IDX *hashidx,* unsigned char * *msg,* int *msglen,* unsigned char * *sig,* int *siglen*)

This function hashes the plaintext, performs PKCS #1 padding and then verifies the signature. This is the offical FIPS certifiable RSA signing API.

Parameters

in	*key*	The private RSA key to use
in	*padding_type*	Type of padding
in	*padding_info*	PSS padding parameters
in	*hashidx*	The index of the hash desired
in	*msg*	The msg to verify (octets)
in	*msglen*	The length of the msg to verify
in	*sig*	The signature
in	*siglen*	The max size and resulting size of the signature

Returns

CE_SUCCESS – Function executes successfully.
-CE_ERROR_FAILURE – Generic Error.
-CE_ERROR_NOT_SUPPORTED – The feature is not supported.
-CE_ERROR_INVALID_ARG – Generic invalid argument.
-CE_ERROR_BUFFER_OVERFLOW – Not enough space for output.
-CE_ERROR_NO_MEMORY – Out of memory.
-CE_ERROR_INVALID_SIGNATURE – Invalid signature.
-CE_ERROR_NOP – SW crypto self test failed.

14.31.5.11 int qsee_secpkcs8_parse (uint8_t * *data_ptr,* uint16_t *data_len,* QSEE_pkcs8_privkey_type * *privkey*)

Parses a private key in PKCS#8 format. The private key contains multiple items pointing to the memory where raw PKCS#8 data is held.

Parameters

in	*data_ptr*	Pointer to raw PKCS#8 data.
in	*data_len*	Length of PKCS#8 data.
out	*privkey*	Pointer to private key extracted from raw data.

Returns

E_SUCCESS – Private key parsed properly.
-E_DATA_INVALID – Key cannot be parsed properly.
-E_NOT_SUPPORTED – An algorithm or version is not supported.
-E_INVALID_ARG – A pointer argument is NULL.
-E_FAILURE – The input data length does not match parsed length.

14.31.5.12 int qsee_util_count_bytes (QSEE_S_BIGINT * *s*)

Counts the total number of bytes in a QSEE_S_BIGINT BLONG array.

Parameters

in	*s*	QSEE_S_BIGINT structure.

Returns

Total number of bytes.

Dependencies

None.

14.31.5.13 void qsee_util_free_s_bigint (QSEE_S_BIGINT * *a*)

Frees S_BIGINT data.

Parameters

in	*a*	S_BIGINT data.

14.31.5.14 int qsee_util_init_s_bigint (QSEE_S_BIGINT ** *a*)

Allocates and initializes S_BIGINT data.

Parameters

in	*a*	S_BIGINT data.

Returns

CE_SUCCESS – Function executes successfully.
-CE_ERROR_INVALID_ARG – Generic invalid argument.
-CE_ERROR_NO_MEMORY – Out of memory.

14.32 Secure Camera

14.32.1 Detailed Description

14.32.2 Function Documentation

14.32.2.1 int qsee_sec_camera_acquire_camera (uint32_t *sessionID*)

Deprecated API.

Parameters

in	*sessionID*	Current secure camera session ID.

Returns

SUCCESS – 0
FAILURE – -1

14.32.2.2 int qsee_sec_camera_get_session (uint32_t * *sessionID*)

Deprecated API.

Parameters

out	*sessionID*	Current secure camera session ID; 0 if no session is active.

Returns

SUCCESS – 0
FAILURE – -1

14.32.2.3 int qsee_sec_camera_register_bulk_write (qsee_sec_cam_register_region_t *registerRegionId,* uint32_t *offset[],* size_t *offset_len,* uint32_t *data[],* size_t *data_len*)

Deprecated API.

Parameters

in	*registerRegionId*	Register region identifier, e.g., MMSS_A_CCI.
in	*offset*	Array of offsets at the register region.
in	*offset_len*	Number of items in the offset array.
in	*data*	Array of data values to write (corresponding to the offsets).
in	*data_len*	Number of items in data array.

Returns

SUCCESS – 0
FAILURE – -1

14.32.2.4 int qsee_sec_camera_register_callback (Object *callback_object*)

Deprecated API.

Parameters

in	*callback_object*	MINK object of type ISecureCameraClientEvent.

Returns

SUCCESS – 0
FAILURE – -1

14.32.2.5 int qsee_sec_camera_register_read (qsee_sec_cam_register_region_t *registerRegionId,* uint32_t *offset,* uint32_t * *data_ptr*)

Deprecated API.

Parameters

in	*registerRegionId*	Register region identifier, e.g., MMSS_A_CCI.
in	*offset*	Offset at the register region.
out	*data_ptr*	Returned data.

Returns

SUCCESS – 0
FAILURE – -1

14.32.2.6 int qsee_sec_camera_register_write (qsee_sec_cam_register_region_t *registerRegionId,* uint32_t *offset,* uint32_t *data*)

Deprecated API.

Parameters

in	*registerRegionId*	Register region identifier, e.g., MMSS_A_CCI.
in	*offset*	Offset at register region.
in	*data*	Data to write.

Returns

SUCCESS – 0
FAILURE – -1

14.32.2.7 int qsee_sec_camera_release_camera (uint32_t *sessionID*)

Deprecated API.

Parameters

in	*sessionID*	Current secure camera session ID.

Returns

SUCCESS – 0
FAILURE – -1

14.32.2.8 int qsee_sec_camera_set_haven_license (const uint8_t * *licenseCert_ptr,* size_t *licenseCert_len,* int32_t * *havenError_ptr*)

Deprecated API.

Parameters

in	*licenseCert_ptr*	Pointer to buffer containing the certificate in DER format.
in	*licenseCert_len*	Certificate data length (in bytes).
out	*havenError_ptr*	Haven license validation result. 0 means success. See IHavenTokenApp.h for more details on the error values.

Detailed description

Deprecated API.

Returns

SUCCESS – 0
FAILURE – -1

14.33 Secure Channel

14.33.1 Detailed Description

14.33.2 Function Documentation

14.33.2.1 int qsee_authenticate_decrypt_message (qsee_sc_ss_e_type *ssid,* qsee_sc_cid_e_type *cid,* const uint8_t * *input_msg_ptr,* uint32_t *input_msg_len,* uint8_t * *output_msg_ptr,* uint32_t * *output_msg_len_ptr*)

Authenticates and decrypts the secure blob.

The output buffer must be large enough to hold the decrypted message. The recommended output buffer size is at least input_msg_len.
Memory must be managed by the caller.

Parameters

in	*ssid*	Subsystem ID.
in	*cid*	Client ID.
in	*input_msg_ptr*	Pointer to secure blob.
in	*input_msg_len*	Length of secure blob in bytes.
out	*output_msg_ptr*	Pointer to buffer to hold decrypted data (memory provided by caller).
in,out	*output_msg_len_ptr*	Size of the above buffer (in bytes) set to decrypted data length on return.

Returns

E_SUCCESS - Successful
E_FAILURE - Operation failed

Dependencies

Secure Channel must be established successfully.

Side effects

None

14.33.2.2 int qsee_secure_message (qsee_sc_ss_e_type *ssid,* qsee_sc_cid_e_type *cid,* const uint8_t * *input_msg_ptr,* uint32_t *input_msg_len,* uint8_t * *output_msg_ptr,* uint32_t * *output_msg_len_ptr*)

Secures input message.

The output buffer must be large enough to hold the encrypted message, some internal headers, and possible padding. The recommended output buffer size is at least input_msg_len + 100 bytes.
Memory must be managed by the caller.

Parameters

in	*ssid*	Subsystem ID.
in	*cid*	Client ID.
in	*input_msg_ptr*	Pointer to plaintext data.
in	*input_msg_len*	Length of plaintext (in bytes).
out	*output_msg_ptr*	Pointer to buffer to hold secure blob (memory provided by caller).
in,out	*output_msg_len_ptr*	Size of the above buffer (in bytes), set to secure blob length by the function.

Returns

E_SUCCESS - Successful.
E_FAILURE - Operation failed.

Dependencies

Secure Channel must be established successfully.

Side effects

None.

14.34 Secure Display

14.34.1 Detailed Description

14.34.2 Function Documentation

14.34.2.1 int qsee_sd_get_session (uint32_t * *sessionID*)

Gets the current secure display session ID.

Parameters

out	*sessionID*	Current secure display session ID.

Returns

SUCCESS – 0
FAILURE – -1

14.34.2.2 int qsee_sd_set_stop_allowed (bool *allow*)

Allows or blocks REE from stopping Secure Display through the sd_ctrl syscall.

Parameters

in	*allow*	TRUE to allow, FALSE to block.

Returns

SUCCESS – 0
FAILURE – -1

14.35 Services

14.35.1 Detailed Description

14.35.2 Function Documentation

14.35.2.1 int qsee_deregister_shared_buffer (void * *address*)

Deregisters the shared buffer previously registered with QTEE.

Parameters

in	*address*	Pointer to shared buffer.

Returns

SUCCESS – 0
FAILURE – Negative

14.35.2.2 int qsee_prepare_shared_buf_for_nosecure_read (void * *address,* unsigned int *size*)

Prepares a shared buffer before sending it across to the REE.

Note: Flushes cachelines for shared memory in anticipation of non-secure access.

Parameters

in	*address*	Pointer to shared buffer.
in	*size*	Size of region. **Note:** Deprecated and unused.

Returns

SUCCESS – 0
FAILURE – Negative

14.35.2.3 int qsee_prepare_shared_buf_for_secure_read (void * *address,* unsigned int *size*)

Prepares the shared buffer sent by the REE before the trusted side reads it.

Note: Invalidates the cachelines for the shared memory, this must be done prior to first time secure access.

Parameters

in	*address*	Pointer to shared buffer.
in	*size*	Size of region. **Note:** Deprecated and unused.

Returns

SUCCESS – 0
FAILURE – Negative

14.35.2.4 int qsee_register_shared_buffer (void * *address,* unsigned int *size*)

Registers a shared buffer with QTEE.

Parameters

in	*address*	Pointer to shared buffer.
in	*size*	Shared buffer size.

Returns

SUCCESS – 0
FAILURE – Negative

14.35.2.5 int qsee_request_service (unsigned int *listener_id,* void * *req,* unsigned int *req_len,* void * *rsp,* unsigned int *rsp_len*)

Requests a service from a REE listener.

The requested REE listener must be available and running. The supplied buffer lengths must match the requested REE listener instance.

Parameters

in	*listener_id*	Requested listener ID.
in	*req*	Pointer to request buffer.
in	*req_len*	Request buffer length.
out	*rsp*	Pointer to response buffer.
in	*rsp_len*	Response buffer length.

Returns

SUCCESS – 0
FAILURE – Negative
QSEE_SERVICE_ERROR_BUFFER_TOO_LARGE – passed request/responses are too large

14.36 SFS

14.36.1 Detailed Description

14.36.2 Function Documentation

14.36.2.1 void qsee_sfs_clean_file_list (sfs_file_entry * *file_list*)

Reclaims allocated resources to create the application's file list.

Detailed description

The memory resources allocated in qsee_sfs_get_file_list() are reclaimed and can be used for future memory allocation.

Parameters

in	*file_list*	Pointer to file list returned by a successful call to qsee_sfs_get_file_list().

14.36.2.2 int qsee_sfs_close (int *fd*)

Closes an open SFS file. Releases all resources used by the file.

Parameters

in	*fd*	File descriptor.

Returns

E_SUCCESS - Closed file successfully.
E_FAILURE - Error occurred while closing file.

14.36.2.3 int qsee_sfs_error (int *fd*)

Checks error indicator.

Detailed description

Checks if the error indicator associated with fd is set; returns a value different from zero if it is.

Note: The indicator is set by a previous operation on the SFS fd that failed.

Parameters

in	*fd*	File descriptor. When qsee_sfs_open() returns zero, it will try to return the failure reason of the last open file.

Returns

Returns a non-zero value if the error indicator associated with fd is set.
Otherwise, returns zero.

14.36.2.4 int qsee_sfs_get_file_list (sfs_file_entry ** *file_list,* uint32_t * *file_list_len*)

Returns a list of files created by the calling application.

Detailed description

Returns an array of sfs_file_entry structs through the output parameter file_list, which contains the file name and file size for each file created by the calling applicaiton through qsee_sfs_open using the O_CREAT flag. The returned file_list size is returned through the output parameter file_list_len.

If file_list_len > 0, the caller must call qsee_sfs_clean_file_list to recollect resources allocated by this function.

If file_list_len == 0, there are currently no files belonging to the calling application and the caller should not call qsee_sfs_clean_file_list because the list will be set to NULL.
After editing the files, it is recommended to obtain a new file list by calling qsee_sfs_clean_file_list on the current file list and calling this function once more.

Parameters

out	*file_list*	Double pointer to sfs_file_entry list represents files created by the calling application.
out	*file_list_len*	Pointer to file list length.

Returns

0 – Created the application file list successfully.
Non-zero – An error occured while creating the application file list. Use qsee_sfs_error() to get a more detailed error code.

14.36.2.5 int qsee_sfs_getSize (int *fd,* uint32_t * *size*)

Retrieves open SFS file size.

Parameters

in	*fd*	File descriptor.
out	*size*	Pointer to open file size.

Returns

E_SUCCESS - File size stored in size parameter.
E_FAILURE - Error occurred while getting file size.

14.36.2.6 int qsee_sfs_is_anti_rollback_enabled (bool * *b*)

Reports if SFS supports hardware-backed rollback protection.

Parameters

out	*b*	Pointer to boolean rollback protection supported status.

Returns

E_SUCCESS – On success.
E_FAILURE – On failure.

14.36.2.7 int qsee_sfs_mkdir (const char * *path*)

This is a deprecated function because SFS does not support folder hierarchy. Calls to this function result in a NOP.

Parameters

in	*path*	Pointer to a fully qualified path.

Returns

Always returns E_SUCCESS.

14.36.2.8 int qsee_sfs_open (const char * *path,* int *flags*)

Opens an SFS file. The options are specified by File mode in the flag parameter.

Detailed description

In the SFS, opening a file does not actually do anything in the file system. If the file (along with the associated file segments) already exists and the file is created with O_TRUNC mode, the associated subfiles are deleted.
The first segment is created only when new bytes to be written begin arriving.

Note: The base directory must exist; otherwise, returns NULL.

Parameters

in	*path*	Pointer to a fully qualified path of filename to be opened.
in	*flags*	Bitmask field used to specify file modes: • O_RDONLY - Open for read-only access. • O_READWRITE - Open for read-write access. • O_CREAT - Creates file if it does not exist. • O_TRUNC - Truncates file to zero size after opening. • O_APPEND - Write operations occur at the end of the file.

Returns

Non-zero – A valid file descriptor.
Zero - Error occurred while opening file.

14.36.2.9 int qsee_sfs_read (int *fd,* char * *buf,* int *nbytes*)

Reads bytes from an encrypted SFS file previously opened via call to qsee_sfs_open().

Detailed description

nbytes are read from the current file position, and the file position advances by the number of read bytes. SFS performs the necessary cipher and verification operations as bytes are read from the file.

Parameters

in	*fd*	File descriptor.
in	*buf*	Pointer to buffer to hold read bytes.
in	*nbytes*	Number of bytes to read from the file.

Returns

Returns the number of bytes read from an SFS file.
Returns -1 for the error case.

14.36.2.10 int qsee_sfs_rm (const char * *path*)

Removes an SFS file that was previously created through qsee_sfs_open().

Parameters

in	*path*	Pointer to a fully qualified path of the file to be deleted.

Returns

E_SUCCESS - File removal successful.
E_FAILURE - Error occurred while removing the file.

14.36.2.11 int qsee_sfs_rmdir (const char * *path*)

This is a deprecated function because SFS does not support folder hierarchy. Calls to this function result in a NOP.

Parameters

in	*path*	Pointer to a fully qualified path

Returns

Always returns E_SUCCESS.

14.36.2.12 int qsee_sfs_seek (int *fd,* int32_t *offset,* int *whence*)

Moves the read location to point to n bytes from the start, current, or end of the file.

Detailed description

Opens, verifies, and decrypts the subfile that corresponds to the current position. The segment position advances to the current position.

Parameters

in	*fd*	File descriptor.
in	*offset*	File offset to seek in bytes.
in	*whence*	Indicates start, end, or current position. • SEEK_SET - Start of file. • SEEK_CUR - Current file position. • SEEK_END - End of file.

Returns

If successful, the current location.
If an error occurred, -1.

14.36.2.13 int qsee_sfs_write (int *fd,* const char * *buf,* int *nbytes*)

Writes bytes to a previously opened encrypted SFS file via call to qsee_sfs_open.

Detailed description

nbytes are written to the current file position.
If O_APPEND flag was set in qsee_sfs_open(), bytes are written at the end of the file unless qsee_sfs_seek() was issued before qsee_sfs_write(). In this case, data is written to the file position that qsee_sfs_seek() set. The file position advances by the number of bytes written.

Parameters

in	*fd*	File descriptor.
in	*buf*	Pointer to buffer to be written.
in	*nbytes*	Number of bytes to write to the file.

Returns

Returns the number of bytes written to an SFS file.
Returns -1 for the error case.

14.37 SPI

14.37.1 Detailed Description

14.37.2 Function Documentation

14.37.2.1 int qsee_spi_close (qsee_spi_device_id_t *device_id*)

Close the client access to the SPI device.

Parameters

in	*device_id*	The SPI device ID.

Returns

0 on success, negative on failure.

14.37.2.2 int qsee_spi_full_duplex (qsee_spi_device_id_t *device_id,* const qsee_spi_config_t * *p_config,* qsee_spi_transaction_info_t * *p_write_info,* qsee_spi_transaction_info_t * *p_read_info*)

Transfers bi-directional data on SPI bus.

Parameters

in	*device_id*	The SPI device ID to attach to.
in	*p_config*	Pointer to the desired SPI configuration.
in,out	*p_write_info*	Pointer to a buffer containing the data to be written to the SPI bus.
in,out	*p_read_info*	Pointer to a buffer populated with the data read from the SPI bus.

Returns

0 on success, negative on failure.

Note: The number of bytes written on the SPI bus is returned in p_write_info.

14.37.2.3 int qsee_spi_open (qsee_spi_device_id_t *device_id*)

Opens the SPI device and performs HW intialization.

Parameters

in	*device_id*	The SPI device ID to attach to.

Returns

0 on success, negative on failure.

14.37.2.4 int qsee_spi_read (qsee_spi_device_id_t *device_id,* const qsee_spi_config_t * *p_config,* qsee_spi_transaction_info_t * *p_read_info*)

Reads data from SPI bus.

Parameters

in	*device_id*	The SPI device ID to attach to.
in	*p_config*	Pointer to the desired SPI configuration.
in,out	*p_read_info*	Pointer to a buffer populated with the data read from the SPI bus.

Returns

0 on success, negative on failure.

14.37.2.5 int qsee_spi_write (qsee_spi_device_id_t *device_id,* const qsee_spi_config_t * *p_config,* qsee_spi_transaction_info_t * *p_write_info*)

Writes data on SPI bus.

Parameters

in	*device_id*	The SPI device ID to attach to.
in	*p_config*	Pointer to the desired SPI configuration.
in,out	*p_write_info*	Pointer to a buffer containing the data to be written to the SPI bus.

Returns

0 on success, negative on failure.

Note: The number of bytes written on the SPI bus is returned in p_write_info.

14.38 Storage

14.38.1 Detailed Description

14.38.2 Function Documentation

14.38.2.1 int qsee_query_rpmb_enablement (bool * b)

Queries whether the OEM has configured RPMB enablement on the device.

Parameters

out	*b*	Pointer to boolean query response.

Returns

QSEE_STOR_SUCCESS if no errors; otherwise, error code.

14.38.2.2 int qsee_stor_add_partition (qsee_stor_device_handle_t * *device_handle*, uint32_t *partition_id*, uint16_t *num_sectors*)

Adds a new logical partition.

Parameters

in	*device_handle*	Pointer to device handle from qsee_stor_device_init().
in	*partition_id*	Logical Partition ID.
in	*num_sectors*	Number of sectors of the new logical partition.

Returns

QSEE_STOR_SUCCESS if no errors; otherwise, error code.

14.38.2.3 int qsee_stor_client_get_info (qsee_stor_client_handle_t * *client_handle*, qsee_stor_client_info_t * *client_info*)

Returns client info.

Parameters

in	*client_handle*	Pointer to the client handle from qsee_stor_open_partition().
in	*client_info*	Pointer to a client info structure.

Returns

QSEE_STOR_SUCCESS if no errors; otherwise, error code.

14.38.2.4 int qsee_stor_device_get_info (qsee_stor_device_handle_t * *device_handle,* qsee_stor_device_info_t * *device_info*)

Returns device info.

Parameters

in	*device_handle*	Pointer to a device handle from qsee_stor_device_init().
out	*device_info*	Pointer to a device info structure.

Returns

QSEE_STOR_SUCCESS if no errors; otherwise, error code.

14.38.2.5 int qsee_stor_device_init (qsee_stor_device_id_type *device_id,* uint8_t * *partition_guid,* qsee_stor_device_handle_t * *device_handle*)

Initialize device indicated by device_id and partition_guid.

Parameters

in	*device_id*	Device partition number.
in	*partition_guid*	Partition GUID (applies only for GPT partitions).
out	*device_handle*	Pointer to a device handle.

Returns

QSEE_STOR_SUCCESS if no errors; otherwise, error code.

14.38.2.6 int qsee_stor_open_partition (qsee_stor_device_handle_t * *device_handle,* uint32_t *partition_id,* qsee_stor_client_handle_t * *client_handle*)

Open a logical partition.

Parameters

in	*device_handle*	Pointer to a device handle obtained from qsee_stor_device_init().
in	*partition_id*	Logical partition ID.
out	*client_handle*	Pointer to a client handle.

Returns

QSEE_STOR_SUCCESS if no errors; otherwise error code.

14.38.2.7 int qsee_stor_read_sectors (qsee_stor_client_handle_t * *client_handle,* uint32_t *start_sector,* uint32_t *num_sectors,* uint8_t * *data_buffer*)

Read num_sectors of data from start_sector to data_buffer.

Parameters

in	*client_handle*	Pointer to a client handle from qsee_stor_open_partition().
in	*start_sector*	Starting sector to read from.
in	*num_sectors*	Number of sectors to read.
out	*data_buffer*	Pointer to buffer containing read data.

Returns

QSEE_STOR_SUCCESS if no errors; otherwise, error code.

14.38.2.8 int qsee_stor_read_wp_config (qsee_stor_secure_wp_info_t * *wp_info*)

Read the Secure Write Protect Configuration Block.

Parameters

in,out	*wp_info*	Pointer to buffer containing information about Secure Write Protect Configuration.

Returns

QSEE_STOR_SUCCESS if no errors; otherwise, error code.

14.38.2.9 int qsee_stor_remove_client (qsee_stor_client_handle_t * *client_handle*)

Closes a logical partition within the RPMB and removes the client. Following exceptions might apply. a. Logical partition is not cleared, and may fail to add another partition of the same ID. b. If the partition is the last partition in the partition table, it will reclaim space. If the partition is not the last partition, space is not reclaimed, which may cause failure in adding more partitions.

Parameters

in	*client_handle*	Pointer to a client handle.

Returns

QSEE_STOR_SUCCESS if no errors; otherwise, error code.

14.38.2.10 int qsee_stor_write_sectors (qsee_stor_client_handle_t * *client_handle,* uint32_t *start_sector,* uint32_t *num_sectors,* uint8_t * *data_buffer*)

Write num_sectors of data from data_buffer to start_sector.

Parameters

in	*client_handle*	Pointer to a client handle from qsee_stor_open_partition().
in	*start_sector*	Starting sector to write to.
in	*num_sectors*	Number of sectors to write.

in	*data_buffer*	Pointer to buffer containing data to be written.

Returns

QSEE_STOR_SUCCESS if no errors; otherwise, error code.

14.38.2.11 int qsee_stor_write_wp_config (qsee_stor_secure_wp_info_t * *wp_info*)

Write to the Secure Write Protect Configuration Block.

Parameters

in	*wp_info*	Pointer to buffer containing information about Secure Write Protect Configuration.

Returns

QSEE_STOR_SUCCESS if no errors; otherwise, error code.

14.39 Synchronization

14.39.1 Detailed Description

14.39.2 Function Documentation

14.39.2.1 int qsee_spin (uint32_t *timeout_us*)

Performs a busy wait (spin) for the input number of microseconds.

Parameters

in	*timeout_us*	number of microseconds to wait.

Returns

O on success; otherwise, nonzero.

14.40 String Comparison

14.40.1 Detailed Description

14.40.2 Function Documentation

14.40.2.1 unsigned int qsee_strlcat (char * *pcDst,* const char * *pszSrc,* unsigned int *nDestSize*)

Concatenates a string into a destination buffer and guarantees that it is NULL-terminated.

Parameters

out	*pcDst*	Destination string to copy into.
in	*pszSrc*	Source string to copy from.
in	*nDestSize*	Size of destination buffer (in bytes).

Returns

If successful, the copied, full string length.

14.40.2.2 unsigned int qsee_strlcpy (char * *pcDst,* const char * *pszSrc,* unsigned int *nDestSize*)

Copies a string into a destination buffer and guarantees that it is NULL-terminated.

Parameters

out	*pcDst*	Destination string to copy into.
in	*pszSrc*	Source string to copy from.
in	*nDestSize*	Size of the destination buffer in bytes.

Returns

Length of the full string copied, if successful.

14.40.2.3 int qsee_strnicmp (const char * *s1,* const char * *s2,* unsigned int *n*)

Compare two substrings without case sensitivity.

Parameters

in	*s1*	The first input string
in	*s2*	The second input string
in	*n*	The length to compare

Returns

Returns 0 (zero) if s1 is identical to s2. Returns a number > 0 if s1 $>$ s2. Returns a number < 0 if s1 $<$ s2.

14.40.2.4 WCHAR* qsee_strtowstr (const char * *pszIn,* WCHAR * *pDest,* int *nSize*)

Converts a string of single-byte characters into a wide string.

Parameters

in	*pszIn*	Pointer to a NULL-terminated string comprised of single-byte characters.
out	*pDest*	Pointer to a destination buffer to receive wide string.
in	*nSize*	Size (in bytes) of pDest buffer. If nSize = 0, there is no conversion and pDest is unchanged. Negative nSize values are invalid.

Returns

Returns destination string of wide characters.

14.40.2.5 WCHAR* qsee_wstrchr (const WCHAR * *pText,* WCHAR *ch*)

Finds a specified wide character in a wide string.

Parameters

in	*pText*	NULL-terminated wide string to search.
in	*ch*	Character to locate.

Returns

Returns a pointer to the first occurrence of c in s1, or NULL if c is not found.

14.40.2.6 int qsee_wstrcmp (const WCHAR * *s1,* const WCHAR * *s2*)

Compares two wide strings, s1 and s2.
Returns an integer value that shows comparison result.

Parameters

in	*s1*	Pointer to first NULL-terminated string.
in	*s2*	Pointer to second NULL-terminated string.

Returns

Returns 0 if s1 is identical to s2.
Returns a number > 0 if s1 $>$ s2.
Returns a number < 0 if s1 $<$ s2.

14.40.2.7 int qsee_wstrlcat (WCHAR * *pcDst,* const WCHAR * *pszSrc,* long unsigned int *nDestSize*)

Concatenates a string into a destination buffer and guarantees that it is NULL-terminated.

Parameters

out	*pcDst*	Destination string to copy into.
in	*pszSrc*	Source string to copy from.
in	*nDestSize*	Size of the destination buffer in bytes.

Returns

If successful, the copied, full string length.

14.40.2.8 int qsee_wstrlcpy (WCHAR * *pwszDst,* const WCHAR * *cpwszSrc,* long unsigned int *nDestSize*)

Copies a wide string into a destination buffer and guarantees that it is NULL-terminated.

Parameters

out	*pwszDst*	Destination string to copy into.
in	*cpwszSrc*	Source string to copy from.
in	*nDestSize*	Size of the destination buffer in bytes.

Returns

If successful, the copied, full string length.

14.40.2.9 int qsee_wstrlen (const WCHAR * *s*)

Returns length of the specified wide string.

Parameters

in	*s*	Input wide string.

Returns

Length of input string without trailing nulls.

14.40.2.10 char* qsee_wstrtostr (const WCHAR * *cpwszIn,* char * *pDest,* int *nSize*)

Converts a string of wide characters into a single-byte string.

Parameters

in	*cpwszIn*	Pointer to a NULL-terminated wide string that must be converted to a single-byte character string.

out	*pDest*	Pointer to a destination buffer to receive the single-byte string.
in	*nSize*	Size (in bytes) of pDest buffer. If this is 0, there is no conversion and pDest is unchanged. Negative nSize values are invalid.

Returns

Returns destination string of single-byte characters.

14.41 TA Stack Usage Profiling

14.41.1 Detailed Description

API exposed for stack profiling.

Note: TAs should use these APIs only for development purposes. Calls to these APIs should not be made in production builds as they have a performance impact.

14.41.2 Function Documentation

14.41.2.1 uintptr_t qsee_prepare_stack_profile (void)

Prepare for stack profiling by filling the currently unused stack space of the calling TA with a well defined pattern.

Returns

Returns current stackPtr

14.41.2.2 size_t qsee_profile_stack_usage (uintptr_t *currSp*)

Look for stack usage in a function in TA

Parameters

in	*currSp*	stackPtr returned from qsee_prepare_stack_profile.

Returns

Returns the number of bytes pushed/popped on/off the stack since the calling of qsee_prepare_stack_profile()

Prerequisite: A call to qsee_prepare_stack_profile() is required before calling qsee_profile_stack_usage().

14.42 Time

14.42.1 Detailed Description

14.42.2 Function Documentation

14.42.2.1 int time_end (void)

Notifies the time service that operations can cease.

Returns

SUCCESS – 0
FAILURE – -1

14.42.2.2 int time_getmsec (void)

Gets the current system time in milliseconds.

Returns

SUCCESS – System time in milliseconds FAILURE – -1

Note: time_getmsec() is incorrect and should not be used (the 4-byte return value truncates). If the Epoch time is needed, the TZ cmnlib interface time_getutcsec() should instead be used, with explict conversion to milliseconds if desired.

14.42.2.3 int time_getsystime (tztm_t * *tzTime*)

Gets the current system time as tztm_t struct.

Parameters

out	*tzTime*	Pointer to a tztm_t struct to return system time.

Returns

SUCCESS – 0
FAILURE – -1

14.42.2.4 int time_getutcsec (tztimespec_t * *tzTimeSpec*)

Gets the current UTC time as a tztimespec_t struct.

Parameters

out	*tzTimeSpec*	Pointer to a tztimespec_t struct to return UTC time.

Returns

SUCCESS – 0
FAILURE – -1

14.42.2.5 int time_is_leap_year (unsigned int *year*)

Checks if the specified year is a leap year.

Parameters

in	*year*	Year to check.

Returns

1 if the supplied year is a leap year; otherwise, 0.

14.43 Timer

14.43.1 Detailed Description

14.43.2 Function Documentation

14.43.2.1 unsigned long long qsee_get_uptime (void)

Gets up time from bootup in ms.

Returns

Up time in ms from system bootup.

14.44 TLMN

14.44.1 Detailed Description

14.44.2 Function Documentation

14.44.2.1 int qsee_tlmm_config_gpio_id (uint32_t *gpio_id,* qsee_tlmm_config_t * *p_settings*)

Allows the GPIO ID key holder to program GPIO settings (direction, drive strength, and pull) on the pin corresponding to the GPIO ID key.

Parameters

in	*gpio_id*	- GPIO ID holder.
in	*p_settings*	- Pointer to a TLMM GPIO configuration to set.

Returns

SUCCESS – 0
FAILURE – Negative

14.44.2.2 int qsee_tlmm_get_gpio_id (const char * *psz_str,* uint32_t * *p_gpio_id*)

Retrieves a GPIO ID (key) based on a signal name string. If successful, this locks the GPIO so that only the key holder can make modifications.

Parameters

in	*psz_str*	- String signal name to use.
out	*p_gpio_id*	- GPIO ID holder.

Returns

SUCCESS – 0
FAILURE – Negative

14.44.2.3 int qsee_tlmm_gpio_id_in (uint32_t *gpio_id,* qsee_gpio_value_t * *input_val*)

Allows the GPIO ID key holder to retrieve the input value of the GPIO corresponding to the ID. The value reads high or low.

Note

This is not the same value written by the qsee_tlmm_gpio_id_out.

Parameters

in	*gpio_id*	- GPIO ID holder.
in	*input_val*	- Value to drive GPIO.

Returns

SUCCESS – 0
FAILURE – Negative

14.44.2.4 int qsee_tlmm_gpio_id_out (uint32_t *gpio_id,* qsee_gpio_value_t *output_val*)

Allows the GPIO ID key holder to drive the output value of the GPIO corresponding to the ID (high or low) when the function select is set to 0 and the direction is output.

Parameters

in	*gpio_id*	- GPIO ID holder.
in	*output_val*	- Value to drive the GPIO.

Returns

SUCCESS – 0
FAILURE – Negative

14.44.2.5 int qsee_tlmm_release_gpio_id (uint32_t *gpio_id*)

Releases a GPIO ID (key). If successful, this unlocks the GPIO and allows another entity to claim it. This function is optional and used when a GPIO must be shared between drivers that cannot share the GPIO ID.

Parameters

in	*gpio_id*	- GPIO ID holder.

Returns

SUCCESS – 0
FAILURE – Negative

14.44.2.6 int qsee_tlmm_select_gpio_id_mode (uint32_t *gpio_id,* qsee_gpio_mode_t *gpio_mode*)

Allows the GPIO ID key holder to change the function select of the GPIO corresponding to the GPIO ID to either software driven (function select 0) or its primary function select.

Parameters

in	*gpio_id*	- GPIO ID holder.
in	*gpio_mode*	- Mode to program.

Returns

SUCCESS – 0
FAILURE – Negative

14.45 Unified AES

14.45.1 Detailed Description

Within Unified AES, the following return values are defined by the Unified Crypto Environment:

- UC_E_SUCCESS = 0
- UC_E_FAILURE = 1
- UC_E_NOT_ALLOWED = 2
- UC_E_NOT_AVAILABLE = 3
- UC_E_NOT_SUPPORTED = 4
- UC_E_NO_MEMORY = 15
- UC_E_INVALID_ARG = 16
- UC_E_DATA_INVALID = 21

14.45.2 Define Documentation

14.45.2.1 #define SW_AES128_KEY_SIZE 16

Fixed Key Size for AES128.

14.45.2.2 #define SW_AES192_KEY_SIZE 24

Fixed Key Size for AES192.

14.45.2.3 #define SW_AES256_KEY_SIZE 32

Fixed Key Size for AES256.

14.45.2.4 #define SW_AES_BLOCK_BYTE_LEN 16

Fixed AES Block Byte Length

14.45.2.5 #define SW_AES_CCM_MAX_NONCE_SIZE 13

Maximum nonce length for AES CCM

14.45.2.6 #define SW_AES_CCM_MIN_NONCE_SIZE 7

Minimum nonce length for AES CCM

14.45.2.7 #define SW_AES_IV_SIZE 16

Fixed AES IV Size for cipher modes other than CCM and GCM.

14.45.2.8 #define SW_AES_MAX_IV_SIZE 60

Maximum IV Length for CCM and GCM cipher modes.

14.45.2.9 #define SW_AES_MAX_KEY_SIZE 60

Maximum buffer size for AES encryption

14.45.3 Enumeration Type Documentation

14.45.3.1 enum SW_Cipher_Alg_Type

Cipher algorithm types.

Enumerator

SW_CIPHER_ALG_AES128 AES-128 cipher algorithm.
SW_CIPHER_ALG_AES256 AES-256 cipher algorithm.
SW_CIPHER_ALG_AES192 AES-192 cipher algorithm.
SW_CIPHER_ALG_SM4 SM4 cipher algorithm.
SW_CIPHER_ALG_MAX Unknown or invalid algorithm.

14.45.3.2 enum SW_Cipher_Key_Size

Cipher key sizes.

Enumerator

SW_CIPHER_KEY_SIZE_AES128 Key size for AES-128.
SW_CIPHER_KEY_SIZE_AES256 Key size for AES-256.
SW_CIPHER_KEY_SIZE_AES192 Key size for AES-192.
SW_CIPHER_KEY_SIZE_MAX Unknown key size.

14.45.3.3 enum SW_CipherEncryptDir

Cipher direction (encrypt or decrypt).

Enumerator

SW_CIPHER_ENCRYPT Direction encrypt.
SW_CIPHER_DECRYPT Direction decrypt.
SW_CIPHER_MAX Invalid value.

14.45.3.4 enum SW_CipherModeType

Cipher modes.

Enumerator

SW_CIPHER_MODE_ECB Electronic Codebook mode.
SW_CIPHER_MODE_CBC Cipher block chaining mode.
SW_CIPHER_MODE_CTR Counter mode.
SW_CIPHER_MODE_GCM Galois Counter mode.
SW_CIPHER_MODE_CCM Counter mode with CBC-MAC.
SW_CIPHER_MODE_CTS Cipher text stealing mode.
SW_CIPHER_MODE_CFB Cipher Feedback Modde.
SW_CIPHER_MODE_OFB Output Feedback Modde. XEX-based tweaked codebook mode with ciphertext stealing.
SW_CIPHER_MODE_INVALID Invalid cipher mode.

14.45.3.5 enum SW_CipherParam

Cipher parameters.

Enumerator

SW_CIPHER_PARAM_DIRECTION Cipher direction. See SW_CipherEncryptDir.
SW_CIPHER_PARAM_KEY Key value parameter.
SW_CIPHER_PARAM_IV Initialization vector parameter.
SW_CIPHER_PARAM_MODE Cipher mode parameter.
SW_CIPHER_PARAM_BLOCKSIZE Cipher block size.
SW_CIPHER_PARAM_KEY_SIZE Key size parameter.
SW_CIPHER_PARAM_IV_SIZE Initialization vector parameter.
SW_CIPHER_PARAM_AAD Additional plaintext data for AES-GCM
SW_CIPHER_PARAM_TAG Calculated TAG by AES-GCM
SW_CIPHER_PARAM_NONCE Nonce value parameter.
SW_CIPHER_PARAM_PAYLOAD_LEN AES-CCM Payload length parameter.
SW_CIPHER_PARAM_XTS_DU_SIZE AES-XTS data unit size parameter.
SW_CIPHER_PARAM_XTS_KEY AES-XTS mode secondary key value parameter.
SW_CIPHER_PARAM_XTS_TWK_SET AES-XTS mode tweak set boolean parameter.
SW_CIPHER_PARAM_COPY Set current cipher context to be a copy of the context passed as a parameter.
SW_CIPHER_PARAM_MAX Unknown or invalid parameter.

14.45.4 Function Documentation

14.45.4.1 sw_crypto_errno_enum_type qsee_SW_AE_FinalData (CryptoCntxHandle * *handle,* lovecListType *ioVecIn,* lovecListType * *ioVecOut*)

Finalizes the Authenticated Encryption (AE) operation with the passed message using the specified algorithm, and computes the tag obtained by calling SW_Cipher_GetParam.

Parameters

in	*handle*	Pointer to AE context handle.
in	*ioVecIn*	AE input vector.
out	*ioVecOut*	Pointer to AE output vector.

Returns

UC_E_SUCCESS – Function executes successfully.
UC_E_FAILURE – Operation failed due to unknown error.
UC_E_INVALID_ARG – Unrecognized argument.
UC_E_DATA_INVALID – Tag verification failed.

14.45.4.2 sw_crypto_errno_enum_type qsee_SW_AE_UpdateAAD (CryptoCntxHandle * *handle*, void * *aad*, uint32_t *aadlen*)

Updates the Additional Authenticated Data (AAD) using the specified algorithm.

Parameters

in	*handle*	Pointer to AE context handle.
in,out	*aad*	Pointer to AAD.
in	*aadlen*	Length of AAD.

Returns

UC_E_SUCCESS – Function executes successfully.
UC_E_INVALID_ARG – Unrecognized argument.

14.45.4.3 sw_crypto_errno_enum_type qsee_SW_AE_UpdateData (CryptoCntxHandle * *handle*, IovecListType *ioVecIn*, IovecListType * *ioVecOut*)

Updates the AE operation with the passed message using the specified algorithm.

Parameters

in	*handle*	Pointer to AE context handle.
in	*ioVecIn*	AE input vector.
out	*ioVecOut*	Pointer to AE output vector.

Returns

UC_E_SUCCESS – Function executes successfully.
UC_E_INVALID_ARG – Unrecognized argument.

14.45.4.4 sw_crypto_errno_enum_type qsee_SW_Cipher_DeInit (CryptoCntxHandle ** *handle*, SW_Cipher_Alg_Type *pAlgo*)

Deintializes a cipher context.

Parameters

in	*handle*	Double pointer to cipher context handle.
in	*pAlgo*	Cipher algorithm type.

Returns

UC_E_SUCCESS – Function executes successfully.
UC_E_INVALID_ARG – Unrecognized argument.

14.45.4.5 sw_crypto_errno_enum_type qsee_SW_Cipher_GetParam (CryptoCntxHandle * *handle,* SW_CipherParam *nParamID,* void * *pParam,* uint32_t *cParam*)

Gets cipher parameters.

Parameters

in	*handle*	Pointer to cipher context handle.
in	*nParamID*	Cipher parameter ID to get.
out	*pParam*	Pointer to parameter data.
in	*cParam*	Parameter data size (in bytes).

Returns

UC_E_SUCCESS – Function executes successfully.
UC_E_INVALID_ARG – Unrecognized argument.

14.45.4.6 sw_crypto_errno_enum_type qsee_SW_Cipher_Init (CryptoCntxHandle ** *handle,* SW_Cipher_Alg_Type *pAlgo*)

Initialize a cipher context

Parameters

out	*handle*	Double pointer to cipher context handle.
in	*pAlgo*	Cipher algorithm type.

Returns

UC_E_SUCCESS – Function executes successfully.
UC_E_FAILURE – Operation failed due to unknown error.
UC_E_NOT_ALLOWED – Operation currently not allowed.
UC_E_NOT_SUPPORTED – Operation not yet implemented .
UC_E_NO_MEMORY – Allocation from a memory pool failed.

14.45.4.7 sw_crypto_errno_enum_type qsee_SW_Cipher_Reset (CryptoCntxHandle * *handle*)

Resets a cipher context; except keys.

Parameters

in,out	*handle*	Pointer to cipher context handle.

Returns

UC_E_SUCCESS – Function executes successfully.

UC_E_FAILURE – Operation failed due to unknown error.
UC_E_INVALID_ARG – Invalid context handle.

14.45.4.8 sw_crypto_errno_enum_type qsee_SW_Cipher_SetParam (CryptoCntxHandle * *handle,* SW_CipherParam *nParamID,* const void * *pParam,* uint32_t *cParam*)

Sets cipher parameters.

Parameters

in,out	*handle*	Pointer to cipher context handle.
in	*nParamID*	Cipher parameter ID to set.
in	*pParam*	Pointer to parameter data.
in	*cParam*	Parameter data size (in bytes).

Returns

UC_E_SUCCESS – Function executes successfully.
UC_E_FAILURE – Operation failed due to unknown error.
UC_E_INVALID_ARG – Unrecognized argument.

14.45.4.9 sw_crypto_errno_enum_type qsee_SW_CipherData (CryptoCntxHandle * *handle,* lovecListType *ioVecIn,* lovecListType * *ioVecOut*)

Encrypts or decrypts passed message using the specified algorithm.

Parameters

in	*handle*	Pointer to cipher context handle.
in	*ioVecIn*	Cipher input vector.
out	*ioVecOut*	Pointer to cipher output vector.

Returns

UC_E_SUCCESS – Function executes successfully.
UC_E_FAILURE – Operation failed due to unknown error.
UC_E_INVALID_ARG – Unrecognized argument.

14.46 Unified DES

14.46.1 Detailed Description

Within Unified DES, the following return values are defined by the Unified Crypto Environment:

- UC_E_SUCCESS = 0
- UC_E_FAILURE = 1
- UC_E_NOT_ALLOWED = 2
- UC_E_NO_MEMORY = 15
- UC_E_INVALID_ARG = 16

14.46.2 Enumeration Type Documentation

14.46.2.1 enum SW_Cipher_DES_Alg_Type

Cipher DES algorithm type.

Enumerator

SW_CIPHER_ALG_DES DES Algorithm.
SW_CIPHER_ALG_DES3 Triple DES algorithm.
SW_CIPHER_ALG_DES2 Triple DES algorithm.
SW_CIPHER_ALG_DES_MAX Unknown algorithm type.

14.46.2.2 enum SW_Cipher_DES_Key_Size

Cipher key size.

Enumerator

SW_CIPHER_KEY_SIZE_DES Key size for DES.
SW_CIPHER_KEY_SIZE_3DES Key size for triple DES.
SW_CIPHER_KEY_SIZE_2DES Key size for triple DES.
SW_CIPHER_KEY_SIZE_DES_MAX Unknown key size.

14.46.2.3 enum SW_CipherDESEncryptDir

Cipher DES direction.

Enumerator

SW_CIPHER_DES_ENCRYPT Cipher direction encrypt.
SW_CIPHER_DES_DECRYPT Cipher direction decrypt.
SW_CIPHER_DES_MAX Invalid direction value.

14.46.2.4 enum SW_CipherDESModeType

Cipher DES modes.

Enumerator

SW_CIPHER_DES_MODE_ECB ECB Mode.
SW_CIPHER_DES_MODE_CBC CBC Mode.

14.46.2.5 enum SW_CipherDESParam

Cipher DES parameters.

Enumerator

SW_CIPHER_DES_PARAM_DIRECTION Cipher direction (encrypt or decrypt). See SW_CipherDESEncryptDir for valid values.
SW_CIPHER_DES_PARAM_KEY Key value parameter.
SW_CIPHER_DES_PARAM_IV Initalization vector (IV).
SW_CIPHER_DES_PARAM_MODE Cipher mode. See SW_CipherDESModeType for valid values.
SW_CIPHER_DES_PARAM_BLOCKSIZE Cipher block size.
SW_CIPHER_DES_PARAM_KEY_SIZE Key size parameter. See SW_Cipher_DES_Key_Size for valid values.
SW_CIPHER_DES_PARAM_IV_SIZE IV size parameter.
SW_CIPHER_DES_PARAM_COPY Create a copy of the DES cipher handle passed as argument.
SW_CIPHER_DES_PARAM_MAX Unknown parameter.

14.46.3 Function Documentation

14.46.3.1 sw_crypto_errno_enum_type qsee_SW_Cipher_DES_DeInit (CryptoCntx Handle ** *handle*)

Deintializes cipher context.

Parameters

in	*handle*	Double pointer to cipher context handle.

Returns

UC_E_SUCCESS – Function executes successfully.
UC_E_FAILURE – Operation failed due to unknown error.

14.46.3.2 sw_crypto_errno_enum_type qsee_SW_Cipher_DES_Init (CryptoCntx Handle ** *handle,* SW_Cipher_DES_Alg_Type *pAlgo*)

Intializes a cipher context.

Parameters

out	*handle*	Double pointer to cipher context handle.
in	*pAlgo*	Cipher algorithm type.

Returns

UC_E_SUCCESS – Function executes successfully.
UC_E_NOT_ALLOWED – Operation currently not allowed.
UC_E_NO_MEMORY – Allocation from a memory pool failed.

14.46.3.3 sw_crypto_errno_enum_type qsee_SW_Cipher_DES_SetParam (CryptoCntxHandle * *handle*, SW_CipherDESParam *nParamID*, const void * *pParam*, uint32_t *cParam*)

Sets cipher parameters used by qsee_SW_CipherDESData.

Parameters

in,out	*handle*	Pointer to cipher context handle.
in	*nParamID*	Cipher parameter ID to set.
in	*pParam*	Pointer to parameter data.
in	*cParam*	Size of parameter data in bytes.

Returns

UC_E_SUCCESS – Function executes successfully.
UC_E_INVALID_ARG – Unrecognized argument.

14.46.3.4 sw_crypto_errno_enum_type qsee_SW_CipherDESData (CryptoCntxHandle * *handle*, IovecListType *ioVecIn*, IovecListType * *ioVecOut*)

Encrypts or decrypts the passed message using the specified algorithm.

Parameters

in	*handle*	Pointer to cipher context handle.
in	*ioVecIn*	Cipher input vector.
out	*ioVecOut*	Pointer to cipher output vector.

Returns

UC_E_SUCCESS – Function executes successfully.
UC_E_FAILURE – Operation failed due to unknown error.
UC_E_INVALID_ARG – Unrecognized argument.

14.47 Unified PBKDF2

14.47.1 Detailed Description

14.47.2 Function Documentation

14.47.2.1 sw_crypto_errno_enum_type qsee_SW_pbkdf2 (SW_Auth_Alg_Type *hmacAlgo,* const uint8_t * *password,* size_t *passwordLen,* const uint8_t * *salt,* size_t *saltLen,* size_t *iterations,* uint8_t * *derivedKey,* size_t *derivedKeyLen*)

This functions derives key based on password and salt.

Parameters

in	*hmacAlgo*	HMAC algorithm
in	*password*	password pointer
in	*passwordLen*	password length
in	*salt*	salt pointer
in	*saltLen*	salt length
in	*iterations*	iterations
in,out	*derivedKey*	derivedKey pointer
in	*derivedKeyLen*	derivedKey length

Returns

sw_crypto_errno_enum_type

14.48 Unified SHA

14.48.1 Detailed Description

Within Unified SHA, the following return values are defined by the Unified Crypto Environment:

- UC_E_SUCCESS = 0
- UC_E_FAILURE = 1
- UC_E_NOT_ALLOWED = 2
- UC_E_NOT_AVAILABLE = 3
- UC_E_NOT_SUPPORTED = 4
- UC_E_NO_MEMORY = 15
- UC_E_INVALID_ARG = 16

14.48.2 Enumeration Type Documentation

14.48.2.1 enum SW_Auth_Alg_Type

Software hash algorithm types.

Enumerator

SW_AUTH_ALG_NULL Default NULL algorithm.
SW_AUTH_ALG_SHA1 SHA1 hash algorithm.
SW_AUTH_ALG_SHA256 SHA256 hash algorithm.
SW_AUTH_ALG_SHA224 SHA224 hash algorithm.
SW_AUTH_ALG_SHA384 SHA384 hash algorithm.
SW_AUTH_ALG_SHA512 SHA512 hash algorithm.
SW_AUTH_ALG_MD5 MD5 hash algorithm.
SW_AUTH_ALG_SM3 SM3 hash algorithm.
SW_AUTH_ALG_RIPEMD160 ripemd160 hash algorithm.
SW_AUTH_ALG_INVALID Unknown or invalid algorithm.

14.48.2.2 enum SW_Auth_Param_Type

Software hash parameters.

Enumerator

SW_HASH_PARAM_MODE Hashing mode parameter.
SW_HASH_PARAM_IV Initialization Vector parameter.
SW_HASH_PARAM_HMAC_KEY HMAC Key value parameter.
SW_HASH_PARAM_AUTH_KEY Authentication key parameter.
SW_HASH_PARAM_COPY Make a copy of the input Hash context handle.
SW_HASH_PARAM_HMAC_COPY Make a copy of the input HMAC context handle.
SW_HASH_PARAM_MAX Unknown or invalid parameter.

14.48.3 Function Documentation

14.48.3.1 sw_crypto_errno_enum_type qsee_SW_Hash_Deinit (CryptoCntxHandle ** *cntx*)

Deinitializes hash context.

Parameters

in	*cntx*	Double pointer to hash context handle.

Returns

UC_E_SUCCESS – Function executes successfully.
UC_E_INVALID_ARG – Unrecognized argument.

14.48.3.2 sw_crypto_errno_enum_type qsee_SW_Hash_Final (CryptoCntxHandle * *handle,* IovecListType * *ioVecOut*)

Last block hash update.

Parameters

in	*handle*	Pointer to hash context handle.
out	*ioVecOut*	Pointer to hash output vector.

Returns

UC_E_SUCCESS – Function executes successfully.
UC_E_INVALID_ARG – Unrecognized argument.

14.48.3.3 sw_crypto_errno_enum_type qsee_SW_Hash_Final_Ez (CryptoCntx Handle * *handle,* uint8_t * *digest,* uint32_t *digest_len*)

Computes the digest hash value.

Parameters

in	*handle*	Pointer to hash context handle.
out	*digest*	Pointer to output message digest hash.
in	*digest_len*	Length of the output message digest hash buffer in bytes.

Returns

UC_E_SUCCESS – Function executes successfully.
UC_E_NO_MEMORY – Allocation from a memory pool failed.
UC_E_INVALID_ARG – Unrecognized argument.

Dependencies

qsee_SW_Hash_Init

14.48.3.4 sw_crypto_errno_enum_type qsee_SW_Hash_Init (CryptoCntxHandle ** *handle,* SW_Auth_Alg_Type *auth_alg*)

Initializes a hash context.

Parameters

out	*handle*	Double pointer to hash context handle.
in	*auth_alg*	Hash algorithm, see SW_Auth_Alg_Type

Returns

UC_E_SUCCESS – Function executes successfully.
UC_E_FAILURE – Operation failed due to unknown error.
UC_E_NOT_ALLOWED – Operation currently not allowed.
UC_E_NOT_SUPPORTED – Operation not yet implemented.
UC_E_NO_MEMORY – Allocation from a memory pool failed.
UC_E_INVALID_ARG – Unrecognized argument.

14.48.3.5 sw_crypto_errno_enum_type qsee_SW_Hash_Reset (CryptoCntxHandle * *handle*)

Resets a hash context.

Parameters

in,out	*handle*	Pointer to hash context handle.

Returns

UC_E_SUCCESS – Function executes successfully.
UC_E_INVALID_ARG – Unrecognized argument.

14.48.3.6 sw_crypto_errno_enum_type qsee_SW_Hash_SetParam (CryptoCntx Handle * *cntx,* SW_Auth_Param_Type *nParamID,* const void * *pParam,* uint32_t *cParam,* SW_Auth_Alg_Type *pAlgo*)

Sets hash parameters (Mode and Key for HMAC).

Parameters

in,out	*cntx*	Pointer to hash context handle.
in	*nParamID*	HMAC parameter ID to set.

in	*pParam*	Pointer to parameter data.
in	*cParam*	Size of parameter data in bytes.
in	*pAlgo*	Algorithm type.

Returns

UC_E_SUCCESS – Function executes successfully.
UC_E_INVALID_ARG – Unrecognized argument.

14.48.3.7 sw_crypto_errno_enum_type qsee_SW_Hash_Update (CryptoCntxHandle * *handle,* lovecListType *ioVecIn*)

Updates the function for intermediate data blocks hash.

Parameters

in,out	*handle*	Pointer to hash context handle.
in	*ioVecIn*	Input to hash.

Returns

UC_E_SUCCESS – Function executes successfully.
UC_E_INVALID_ARG – Unrecognized argument.

14.48.3.8 sw_crypto_errno_enum_type qsee_SW_Hash_Update_Ez (CryptoCntx Handle * *handle,* const uint8_t * *msg,* uint32_t *msg_len*)

Hashes data into the hash context structure (must be initialized by qsee_SW_Hash_Init(...)).

Parameters

in,out	*handle*	Pointer to hash context handle.
in	*msg*	Pointer to the data to hash.
in	*msg_len*	Length of the data to hash.

Returns

UC_E_SUCCESS – Function executes successfully.
UC_E_INVALID_ARG – Unrecognized argument.

Dependencies

qsee_SW_Hash_Init

14.48.3.9 sw_crypto_errno_enum_type qsee_SW_Hmac (uint8_t * *key_ptr,* uint32_t *keylen,* lovecListType *ioVecIn,* lovecListType * *ioVecOut,* SW_Auth_Alg_Type *pAlgo*)

Main function for HMAC.

Parameters

in	*key_ptr*	Pointer to Key for HMAC.
in	*keylen*	Key length.
in	*ioVecIn*	Hash input vector.
out	*ioVecOut*	Pointer to hash output vector.
in	*pAlgo*	HMAC algorithm to use (see SW_Auth_Alg_Type).

Returns

UC_E_SUCCESS – Function executes successfully.
UC_E_FAILURE – Operation failed due to unknown error.
UC_E_NOT_ALLOWED – Operation currently not allowed.
UC_E_NOT_SUPPORTED – Operation not yet implemented.
UC_E_NO_MEMORY – Allocation from a memory pool failed.
UC_E_INVALID_ARG – Unrecognized argument.

14.48.3.10 sw_crypto_errno_enum_type qsee_SW_Hmac_Deinit (CryptoCntxHandle ** *cntx*)

Deinitializes HMAC context.

Parameters

in	*cntx*	Double pointer to HMAC context handle.

Returns

UC_E_SUCCESS – Function executes successfully.
UC_E_FAILURE – Operation failed due to unknown error.

14.48.3.11 sw_crypto_errno_enum_type qsee_SW_Hmac_Final (CryptoCntxHandle * *cntx,* lovecListType * *ioVecOut*)

HMAC last block hash update.

Parameters

in	*cntx*	Pointer to HMAC context handle.
out	*ioVecOut*	Pointer to HMAC output vector.

Returns

UC_E_SUCCESS – Function executes successfully.
UC_E_FAILURE – Operation failed due to unknown error.
UC_E_INVALID_ARG – Unrecognized argument.

14.48.3.12 sw_crypto_errno_enum_type qsee_SW_Hmac_Init (CryptoCntxHandle ** *cntx,* SW_Auth_Alg_Type *pAlgo*)

Initializes hash HMAC using the specified hash algorithm.

Parameters

out	*cntx*	Double pointer to HMAC context handle.
in	*pAlgo*	Algorithm to use (see SW_Auth_Alg_Type).

Returns

UC_E_SUCCESS – Function executes successfully.
UC_E_FAILURE – Operation failed due to unknown error.
UC_E_NOT_ALLOWED – Operation currently not allowed.
UC_E_NOT_SUPPORTED – Operation not yet implemented.
UC_E_NO_MEMORY – Allocation from a memory pool failed.
UC_E_INVALID_ARG – Unrecognized argument.

14.48.3.13 sw_crypto_errno_enum_type qsee_SW_Hmac_Reset (CryptoCntxHandle * *cntx*)

Resets HMAC context.

Parameters

in	*cntx*	Pointer to HMAC context handle.

Returns

UC_E_SUCCESS – Function executes successfully.
UC_E_INVALID_ARG – Unrecognized argument.
UC_E_NO_MEMORY – Allocation from a memory pool failed.

14.48.3.14 sw_crypto_errno_enum_type qsee_SW_Hmac_Update (CryptoCntx Handle * *cntx,* IovecListType *ioVecIn*)

Updates function for intermediate data blocks hash.

Parameters

in,out	*cntx*	Pointer to HMAC context handle.
in	*ioVecIn*	Hash input vector.

Returns

UC_E_SUCCESS – Function executes successfully.
UC_E_FAILURE – Operation failed due to unknown error.
UC_E_NOT_ALLOWED – Operation currently not allowed.
UC_E_NOT_SUPPORTED – Operation not yet implemented.
UC_E_NO_MEMORY – Allocation from a memory pool failed.
UC_E_INVALID_ARG – Unrecognized argument.

14.49 Object-Based QTEE Service Interfaces

14.49.1 Detailed Description

Modules

- IAccessControl
- IAppClient
- IAppController
- IAppLoader
- IAppMessage
- IAttestationBuilder
- IAttestationReport
- ICipher
- ICipherOperation
- IClientEnv
- IClockConfig
- ICredentials
- ICrypto
- IDataCache
- ICertification
- IDAError
- IDeviceAttestation
- IDeviceID
- IEnv
- IESEService
- IDiagnostics
- IFeatureVersions
- IGenericService
- IGPSession
- IHash
- IHavenTokenApp
- IHdcpEncryption
- IHdcpSrm
- IHdcpTransmitter

- IHdmiStatus
- IHlosRegionFinder
- IHmac
- IHwFuse
- IHWKey
- IHWKeyFactory
- II2C
- IICE
- IIntMask
- IIO
- IIPProtector
- IKey
- IKeyManager
- IKVStore
- IKVStoreKey
- IKVStoreIterator
- IKVStoreAdmin
- ILegacyHWAttestation
- ILicenseImage
- ILicenseManager
- IListener
- IListenerCBO
- IMacchiato
- IMemManager
- IMemObject
- IMemRegion
- IMemRegionPermEscalator
- IMemSpace
- IModule
- INistLoggingFramework
- INotifyHdcp
- INSMem

- INSSystemReg
- IOpener
- IOPS
- IOPSSink
- IOPSSource
- IPeripheralAccessControl
- IPeripheralState
- IPeripheralStateCB
- IPFM
- IPmPon
- IPrivacyPreservingID
- IProperty
- IProvError
- IProvisioning
- IPVCLicense
- IQTEEEnvInfo
- IQWESKeyStore
- IQWESTAServices
- IRegisterListenerCBO
- IRTICDtb
- IRTICReport
- IRuntimeAttestation
- ISecureCamera
- ISecureCameraClientEvent
- ISecureCamera2
- ISecureCamera2Notify
- ISecureChannel
- ISecureChannelKeyExchange
- ISecureImageParserReturnCode
- ISecureImageOemMetadataParser
- ISecureDisplay
- ISharedBuffer

- ISource
- ISPCOM
- ISPI
- ISwFuse
- ISync
- ITLMM
- ITLOCKey
- ITPM
- ITranslateAddr
- "ITrustedReport"
- IUnwrapKeys
- IUptime
- IValidate
- IVMDeviceUniqueKey
- IVMSessionKey
- IWait

14.50 IAccessControl

14.50.1 Detailed Description

14.50.2 Function Documentation

14.50.2.1 method lockMemObject (in IAccessControl_vmidPermission[] *vmid Permissions,* in IMemRegion *memObj,* in uint32 *validationType,* out interface *lockedRegionOut*)

Parameters

in	*vmidPermissions*	List of IAccessControl_vmidPermission for validation and locking.
in	*memObj*	Memory object to lock.
in	*validationType*	One of the ValidationType_∗ values, see above for explanation.
out	*lockedRegionOut*	Handle to the locked region, can be released using Object_Release() or Object_RELEASE_IF().

Returns

Object_OK – Successful.
Object_ERROR – Any error encountered.

14.50.2.2 method lockRegion (in IAccessControl_vmidPermission[] *vmid Permissions,* in uint64 *addr,* in uint64 *size,* in uint32 *validationType,* out interface *lockedRegionOut*)

API to lock a given memory region if the client TA has privilege to use the VMs in vmidPermissions and VMs in vmidPermissions match with the existing permissions of the region as per the validationType. Once a region (addr, size) is locked for the given vmidPermissions and validationType combination, it is guaranteed to hold the permission until the region is unlocked. On success, the locked region can be unlocked by releasing the lock handle 'lockedRegionOut' using Object_Release() or Object_RELEASE_IF().

validationType - ValidationType_ALL_Exclusive

- When the validationType is 'ALL_Exclusive', the lock region is expected to be owned or shared with all the VMs in vmidPermissions list exclusively.
- Validation will fail if the given region is shared with any VMs outside of the vmidPermissions list.
- Validation will fail if the existing VM permissions on the region don't match their corresponding PERM_TYPE_∗ in vmidPermissions list.
- Once locked, permission change for any of the locked VMs is allowed only if their locked perm type allows it.
- Sharing the locked region with any other VMs outside of vmidPermissions is not allowed.

validationType - ValidationType_ANY_Exclusive

- The region is expected to be owned or shared with one or more VMs in vmidPermissions list exclusively.

- The region will be locked for exclusive ownership of all the VMs in the lock request irrespective of their current ownership or sharing status.
- Once locked, permission change for any of the locked VMs is allowed only if their locked perm type allows it.
- Any VM within locked VMs set can share or relinquish the locked region as long as the first condition is met.
- Sharing the locked region with any other VMs outside of the locked VMs set is not allowed.

validationType - ValidationType_ALL_NonExclusive

- The region is expected to be owned or shared with all the VMs in vmidPermissions list with matching permissions.
- The region can also be shared with any VMs outside of the vmidPermissions list.
- Once locked, permission change on any of the VMs within the vmidPermissions is allowed only if their locked perm type allows it.
- Permission change on any VM outside of the vmidPermissions is allowed even when the region is locked.

validationType - ValidationType_ANY_NonExclusive

- This is a most relaxed lock type of all and should be considered only when all the other validation types don't satisfy your use case.
- The region is expected to be owned or shared with one or more of the VMs in vmidPermissions list.
- The region will be locked for non-exclusive ownership of all the VMs in the lock request irrespective of their current ownership or sharing status.
- Once locked, permission change on the locked VMs is allowed only if their locked perm type allows it.
- Any VM within locked VMs set can share or relinquish the locked region as long as one or more of the locked VMs continue sharing this region.
- Permission change on any VM outside of the locked VMs set is allowed even when the region is locked.

Parameters

in	*vmidPermissions*	List of IAccessControl_vmidPermission for validation and locking.
in	*addr*	Start address of the memory region that needs to be locked.
in	*size*	Size of the memory region that needs to be locked.
in	*validationType*	One of the ValidationType_∗ values, see above for explanation.
out	*lockedRegionOut*	Handle to the locked region, can be released using Object_Release() or Object_RELEASE_IF().

Returns

Object_OK – Successful.
Object_ERROR – Any error encountered.

14.51 IAppClient

14.51.1 Detailed Description

14.51.2 Function Documentation

14.51.2.1 method getAppObject (in buffer *appDistName,* out interface *obj*)

Gets the object implementing app-provided functionalities.

Parameters

in	*appDistName*	Application distinguished name.
out	*obj*	Returned object.

Returns

Object_OK - Successful.
IAppClient_ERROR_APP_LOAD_FAILED - Failed to load application.
IAppClient_ERROR_APP_NOT_FOUND - No loaded application with distinguished name.
IAppClient_ERROR_APP_RESTART_FAILED - Failed to restart application.
IAppClient_ERROR_APP_UNTRUSTED_CLIENT - Untrusted client not allowed.
IAppClient_ERROR_CLIENT_CRED_PARSING_FAILURE - Failed to parse the client Credentials.

14.52 IAppController

14.52.1 Detailed Description

14.52.2 Function Documentation

14.52.2.1 method disconnect ( )

Disconnect client from the trusted application.

This method removes the client connection to the app while leaving the app loaded. After a successful disconnect, the only methods allowed on the IAppController object are retain and release. After a successful disconnect, a new IAppController object can be obtained for the app by calling IAppLoader_connect.

Fails if the trusted application rejects the disconnect request.

Returns

Object_OK if successful.

14.52.2.2 method getAppObject (out interface *obj*)

Gets the object implementing app-provided functionalities.

Parameters

out	*obj*	Returned object.

Returns

Object_OK on success.

14.52.2.3 method installCBO (in uint32 *uid,* in interface *obj*)

Installs a CallBack Object provided by the client.

Parameters

in	*uid*	UID of service class to register.
in	*obj*	Object implementing the interface.

Returns

Object_OK on success.

14.52.2.4 method openSession (in uint32 *cancelCode,* in uint32 *connectionMethod,* in uint32 *connectionData,* in uint32 *paramTypes,* in uint32 *exParamTypes,* in buffer *i1,* in buffer *i2,* in buffer *i3,* in buffer *i4,* out buffer *o1,* out buffer *o2,* out buffer *o3,* out buffer *o4,* in interface *imem1,* in interface *imem2,* in interface *imem3,* in interface *imem4,* out uint32 *memrefOutSz1,* out uint32 *memrefOutSz2,* out uint32 *memrefOutSz3,* out uint32 *memrefOutSz4,* out interface *session,* out uint32 *retValue,* out uint32 *retOrigin*)

Opens a session with the trusted application.

This method is part of the GP interface for the TA, and should not be directly used (internal to GP framework)

The caller/implementer (depending on parameter type) manually marshals the content of four input/output buffers.

Parameters

in	*cancelCode*	Optional code to use for cancellations.
in	*connectionMethod*	What CA identity credentials to use.
in	*connectionData*	Optional connection group identifier.
in	*paramTypes*	Parameter types, 1 byte per parameter.
in	*exParamTypes*	Extended information for parameters, 1 byte per parameter.
in	*i1*	First input buffer.
in	*i2*	Second input buffer.
in	*i3*	Third input buffer.
in	*i4*	Fourth input buffer.
out	*o1*	First output buffer.
out	*o2*	Second output buffer.
out	*o3*	Third output buffer.
out	*o4*	Fourth output buffer.
in	*imem1*	First optional memory region.
in	*imem2*	Second optional memory region.
in	*imem3*	Third optional memory region.
in	*imem4*	Fourth optional memory region.
out	*memrefOutSz1*	Desired output size for memref 1, if larger than size provided.
out	*memrefOutSz2*	Desired output size for memref 2, if larger than size provided.
out	*memrefOutSz3*	Desired output size for memref 3, if larger than size provided.
out	*memrefOutSz4*	Desired output size for memref 4, if larger than size provided.
out	*session*	Newly opened session.
out	*retValue*	GP return value.
out	*retOrigin*	Where the GP return value originated.

Returns

Object_OK if successful.

14.52.2.5 method restart ( )

Restart the TA. Only a TA that has not been unloaded can be restarted.

Returns

Object_OK if successful.

14.52.2.6 method unload ( )

Unloads the trusted application.

This function fails if the application is currently busy. The caller is expected to try again at a later time.

Returns

Object_OK if successful.

14.53 IAppLoader

14.53.1 Detailed Description

14.53.2 Function Documentation

14.53.2.1 method connect (in buffer *appName,* out IAppController *appController*)

Connect to a loaded trusted application. The trusted application with the distinguished name appName must be already loaded.

Parameters

in	*appName*	Application distinguished name string.
out	*appController*	IAppController to access the trusted application.

Returns

Object_OK if successful.

14.53.2.2 method loadFromBuffer (in buffer *appElf,* out IAppController *app Controller*)

Loads a trusted application. The application ELF binary is passed as a buffer.

Parameters

in	*appElf*	Buffer containing ELF image.
out	*appController*	IAppController to access the trusted application.

Returns

Object_OK if successful.

14.53.2.3 method loadFromRegion (in interface *appElf,* out IAppController *appController*)

Loads a trusted application. The application ELF binary is passed as an IMemRegion object.

Parameters

in	*appElf*	Region containing ELF image.
out	*appController*	IAppController to access the trusted application.

Returns

Object_OK if successful.

14.54 IAppMessage

14.54.1 Detailed Description

14.54.2 Function Documentation

14.54.2.1 method decapsulateInterAppMessage (out buffer *sourceAppName,* in buffer *encapsulatedMessage,* out buffer *originalMessage*)

API for a trusted application to decapsulate (i.e. decrypt) a message from another trusted application. This API authenticates the message, decrypts the input buffer, and writes the plaintext message into the supplied output buffer. The input buffer must have been prepared by encapsulateInterAppMessage(), containing a header and MAC.

Parameters

out	*sourceAppName*	Buffer populated with the message originator application name (maximum length is APP_NAME_MAX_LEN).
in	*encapsulated Message*	Wrapped message to be decrypted.
out	*originalMessage*	Buffer containing the unwrapped message.

Returns

0 indicates success.
All other values indicate failure and correspond to a specific error code.

14.54.2.2 method encapsulateInterAppMessage (in buffer *destAppName,* in buffer *encryptedMessage,* out buffer *encapsulatedMessage*)

API for a trusted application to prepare a message that can be sent to another trusted application.

This function writes the AES128-CTR encryped message into the supplied output buffer, prepended with a header and appended with HMAC-SHA256. This output buffer can be given to the receiving trusted application and passed to decapsulateInterAppMessage to be authenticated and decrypted.

The actual data exchange (passing the encrypted buffer) between the trusted applications can be done between clients running in the REE.

Parameters

in	*destAppName*	Destination application name in plaintext (maximum length is APP_NAME_MAX_LEN).
in	*encryptedMessage*	Buffer containing message to be encapsulated.
out	*encapsulated Message*	Buffer containing encapsulated message.

Dependencies

encapsulatedMessage buffer length should be greater than the encryptedMessage buffer length by 144 bytes, to allow room for the header and MAC.

Returns

0 indicates success.
All other values indicate failure and correspond to a specific error code.

14.55 IAttestationBuilder

14.55.1 Detailed Description

IAttestationBuilder provides an interface to build the attestation using IDeviceAttestation.

14.55.2 Function Documentation

14.55.2.1 method addBytes (in uint32 *securityLevel,* in int8[ ] *label,* in buffer *bytes*)

Adds any client/user specific data as blob.

Parameters

in	*securityLevel*	One of SECURITY_LEVEL_∗.
in	*label*	String to tag the item. The string does not have to be null-terminated.
in	*bytes*	Data bytes to be added.

Returns

Object_OK on success.

14.55.2.2 method build (in uint32 *attestationContext,* in uint64 *ADDONOptions,* in uint32 *formatType,* in uint32 *keyType,* in buffer *nonce,* out uint64 *reportStatus,* out IAttestationReport *attestationReport*)

Builds the full size of the signed and encrypted attestation.

Parameters

in	*attestationContext*	Attestation contexts.
in	*ADDONOptions*	OPT_ADDON_LOCATION, OPT_ADDON_RTIC, etc. or-ed together.
in	*formatType*	Attestation formats.
in	*keyType*	Signing key types.
in	*nonce*	Buffer containing nonce.
out	*reportStatus*	Bitmap showing which addon options were successfully fetched.
out	*attestationReport*	IAttestationReport object containing attestation.

Returns

Object_OK on success.

14.55.2.3 method clearBytes ( )

Clears the labeled client data in the attestation builder.

Returns

Object_OK on success.

14.55.2.4 method setDAParams (in buffer *DAParams*)

Set DA params (token_cert + nonce)

Parameters

in	*DAParams*	Buffer containing token_cert & nonce

Returns

Object_OK on success.

14.56 IAttestationReport

14.56.1 Detailed Description

IAttestationReport provides an interface to get the attestation bytes using IAttestationBuilder.

14.56.2 Function Documentation

14.56.2.1 method getBytes (in uint64 *offset,* out buffer *attestation*)

Gets the bytes in the attestation.

Parameters

in	*offset*	Attestation offset.
out	*attestation*	Attestation buffer.

Detailed description

Typical use is to get the size and then call this in a loop with a 4KB buffer incrementing the offset by the number of bytes returned until all the bytes have been fetched.

The caller should keep track of the bytes fetched to know when they have got them all. There is no "end of file" return code.

A buffer larger than 4KB may be used with some implementations. If the buffer is too large for the underlying IPC mechanism, then an error will be returned.

An error will be returned if offset is not in the range of 0 and the size of the attestation.

The attestation is an encrypted data blob. There is no use for part of a attestation because it cannot be decrypted; therefore, the full attestation must be fetched.

Returns

Object_OK on success.

14.56.2.2 method getSize (out uint64 *attestationReportSize*)

Gets the full size of the signed and encrypted attestation.

Parameters

out	*attestationReportSize*	Size of the attestation report.

Returns

Object_OK on success.

14.57 ICipher

14.57.1 Detailed Description

14.57.2 Function Documentation

14.57.2.1 method decrypt (in buffer *cipher,* out buffer *plain*)

Decrypts the passed ciphertext message using the specified algorithm.

Parameters

in	*cipher*	Input ciphertext buffer.
out	*plain*	Output plaintext buffer.

Detailed description

The memory allocated for plaintext must be large enough to hold the ciphertext equivalent. If a padding scheme is selected, the plaintext output length may be up to one block size smaller than the ciphertext length. If the output buffer is not large enough to hold the decrypted results, an error is returned. Note: This API does not support GCM and GCM_STRM modes which need to use update_aad/update/final APIs.

Returns

Object_OK – Successful.
Object_ERROR_INVALID – Not multiple of block length.
Object_ERROR – Any other error encountered.

14.57.2.2 method encrypt (in buffer *plain,* out buffer *cipher*)

Encrypts the passed plaintext message using the specified algorithm.

Parameters

in	*plain*	Input plaintext buffer.
out	*cipher*	Output ciphertext buffer.

Detailed description

The memory allocated for the ciphertext must be large enough to hold the plaintext equivalent. If a padding scheme is selected the ciphertext buffer length may need to be up to one block size larger than the plaintext length. If the output buffer is not large enough to hold the encrypted results, an error is returned. Note: This API does not support GCM and GCM_STRM modes which need to use update_aad/update/final APIs.

Returns

Object_OK – Successful.
Object_ERROR_INVALID – Not multiple of block length.
Object_ERROR – Any other error encountered.

14.57.2.3 method final (out buffer *obuf*)

Encrypts/Decrypts the last segment of input buffer saved in cipher context and saves the cipher/plain text to output buffer [obuf]. The API will update output size on successful return in obuf_lenout.

Parameters

out	*obuf*	Output buffer.

Returns

Object_OK – Successful.
Object_ERROR – Any error encountered.

14.57.2.4 method getParamAsData (in int32 *paramID,* out buffer *param*)

Retrieves parameters from a cipher context.

Parameters

in	*paramID*	Parameter to retrieve.
out	*param*	Memory location in which to store the parameter.

Returns

Object_OK – Successful.
Object_ERROR_INVALID – Invalid parameter encountered.
Object_ERROR – Any other error encountered.

14.57.2.5 method getParamAsU32 (in int32 *paramID,* out uint32 *param*)

Retrieves parameters from a cipher context as 32-bit unsigned int.

Parameters

in	*paramID*	Parameter to retrieve.
out	*param*	Memory location in which to store the parameter.

Returns

Object_OK – Successful.
Object_ERROR_INVALID – Invalid parameter encountered.
Object_ERROR – Any other error encountered.

14.57.2.6 method reset ( )

Resets cipher context; does not reset the key.

Returns

Object_OK – Successful.
Object_ERROR – Any error encountered.

14.57.2.7 method setParamAsData (in int32 *paramID,* in buffer *param*)

Modifies parameters for a cipher context.

Parameters

in	*paramID*	Parameter to modify.
in	*param*	Parameter value to set.

Returns

Object_OK – Successful.
Object_ERROR_INVALID – Invalid parameter encountered.
Object_ERROR – Any other error encountered.

14.57.2.8 method setParamAsObject (in int32 *paramID,* in interface *param*)

Modifies parameters for a cipher context.

Parameters

in	*paramID*	Parameter to modify.
in	*param*	Parameter value to set.

Returns

Object_OK – Successful.
Object_ERROR_INVALID – Invalid parameter encountered.
Object_ERROR – Any other error encountered.

14.57.2.9 method setParamAsU32 (in int32 *paramID,* in uint32 *param*)

Modifies the 32-bit unsigned int parameters for a cipher context.

Parameters

in	*paramID*	Parameter to modify.
in	*param*	Parameter value to set.

Returns

Object_OK – Successful.

Object_ERROR_INVALID – Invalid parameter encountered.
Object_ERROR – Any other error encountered.

14.57.2.10 method update (in buffer *ibuf,* out buffer *obuf*)

Encrypts/Decrypts input buffer [ibuf], saves the cipher/plain text to output buffer [obuf] and saves cipher context for further updates. The API will update output size on successful return in obuf_lenout.

Parameters

in	*ibuf*	Input buffer.
out	*obuf*	Output buffer.

Returns

Object_OK – Successful.
Object_ERROR – Any error encountered.

14.57.2.11 method update_aad (in buffer *aad,* out buffer *obuf*)

Updates additional authentication data when authenticated cipher mode (e.g CCM/GCM) is selected. It returns failure for other cipher modes. This API should be called before calling update() and final() APIs.

Parameters

in	*aad*	Input AAD buffer.
out	*obuf*	A dummy buffer for the output data. The Crypto HW engine requires that for all AAD data sent to the HW, the same length of data must be read out. So AAD data must be read back from the HW engine using the obuf dummy buffer. And the size of the obuf should not be less than the aad length. In the case of that the user wants to use the same buffer for both in/out buffer, the buffer must not be const.

Returns

Object_OK – Successful.
Object_ERROR – Any error encountered.

14.58 ICipherOperation

14.58.1 Detailed Description

14.58.2 Function Documentation

14.58.2.1 method finish (in buffer *input,* in buffer *inParams,* out buffer *output*)

Cipher operation - finish, for encryption or decryption only.

Parameters

in	*input*	Buffer containing input data for finish cipher operation.
in	*inParams*	Reserved for future use.
out	*output*	Buffer containing output data from finish cipher operation.

Returns

Object_OK on success.

14.58.2.2 method update (in buffer *input,* in buffer *inParams,* out buffer *output,* out uint32 *inputConsumed*)

Cipher operation - update, for encryption or decryption only.

Parameters

in	*input*	Buffer containing input data for update cipher operation.
in	*inParams*	Reserved for future use.
out	*output*	Buffer containing output data from update cipher operation.
out	*inputConsumed*	It helps in knowing whether output buffer size is enough or not.

Returns

Object_OK on success.

14.59 IClientEnv

14.59.1 Detailed Description

IClientEnv is the interface that REE clients use to obtain objects from the TEE via SMC Invoke.

14.59.2 Function Documentation

14.59.2.1 method adciAccept ( )

Registers a thread to be used for ADCI (Arbitrary Destination Callback Invocations). Will not return until successfully canceled unless an error occurs.

Returns

Object_OK on successful cancel of ADCI accept thread. Object_ERROR on failure.

14.59.2.2 method adciShutdown ( )

Cancels any previously registered ADCI accept thread.

Returns

Object_OK on success. Object_ERROR on failure.

14.59.2.3 method configTaRegion (in uint64 *appRgnAddr,* in uint32 *appRgnSize*)

Configure TA region. TA region address and length is passed as a buffer.

Parameters

in	*appRgnAddr*	TA region base address.
in	*appRgnSize*	TA region size.

Returns

Object_OK if successful.

14.59.2.4 method loadCmnlibFromBuffer (in buffer *cmnlibElf*)

Loads Cmnlib binary. The library binary is passed as a buffer.

Parameters

in	*cmnlibElf*	Buffer containing ELF image.

Returns

Object_OK if successful.

14.59.2.5 method notifyDomainChange ( )

Asserts that QTEE has no references to remote NS domain objects other than the single reference to the primordial object and that the NS domain has no references to QTEE objects other than the single reference to the root ClientEnv.

Returns

Object_OK on success, else assert on failure.

14.59.2.6 method open (in uint32 *uid,* out interface *obj*)

Error codes Gets a service object from the Client Environment.

Parameters

in	*uid*	Identifies a class of service object.
out	*obj*	Instance of the requested service.

Returns

Object_OK on success.

14.59.2.7 method registerAsClient (in IIO *credentials,* out interface *clientEnv*)

Registers a REE client with the specified credentials and returns a client environment object that the client can use to obtain service objects.

Parameters

in	*credentials*	Callback object implementing the IIO interface which can be queried to return the REE credentials of the client being registered.
out	*clientEnv*	Returned registered client environment object.

Returns

Object_OK on success.

14.59.2.8 method registerLegacy (in buffer *credentials,* out interface *clientEnv*)

Registers a REE client with the specified credentials and returns a client environment object that the client can use to obtain service objects.

Note: : This method is DEPRECATED.

Parameters

in	*credentials*	Buffer containing credentials identifying the client.
out	*clientEnv*	Returned registered client environment object.

Returns

Object_OK on success.

14.59.2.9 method registerWithCredentials (in ICredentials *credentials,* out interface *clientEnv*)

Registers a REE client with the ICredentials object and returns a client environment object that the client can use to obtain service objects.

Parameters

in	*credentials*	Callback object implementing the ICredentials interface which can be queried to return the REE credentials of the client being registered. This can be NULL only if client code is privilieged code (kernel or higher exception level.)
out	*clientEnv*	Returned registered client environment object.

Returns

Object_OK on success.

14.59.2.10 method registerWithWhitelist (in IIO *credentials,* in uint32[ ] *uids,* out interface *clientEnv*)

Registers a REE client with the specified credentials and restricts it to the specified list of UIDs. The client can use the returned client environment object to obtain service objects.

The returned client environment object can only obtain the services with UIDs in the passed UID list.

Parameters

in	*credentials*	Callback object implementing the IIO interface which can be queried to return the REE credentials of the client being registered.
in	*uids*	Class UIDs list the returned client environment object can obtain.
out	*clientEnv*	Returned registered client environment object.

Returns

Object_OK on success.

14.60 IClockConfig

14.60.1 Detailed Description

14.60.2 Function Documentation

14.60.2.1 method setBandwidth (in uint32 *resReq,* in uint32 *level,* in uint32 *flags*)

Sets the crypto/bimc/snoc bandwidth

Parameters

in	*resReq*	Resource to vote clocks; all associated clocks are turned on and voted for.
in	*level*	Clock level.
in	*flags*	Flags (for future use).

Returns

Object_OK on success.

14.61 ICredentials

14.61.1 Detailed Description

14.61.2 Function Documentation

14.61.2.1 method getPropertyByIndex (in uint32 *index,* out buffer *name,* out buffer *value*)

Gets the name of a property, given its position in the sequence of properties.

Parameters

in	*index*	A (zero-based) index into the set of properties.
out	*name*	Property name, including a terminating zero byte.
out	*value*	Property value, including a terminating zero byte.

Returns

Object_OK – Function returned successfully.
ICredentials_ERROR_NAME_SIZE – Supplied buffer not large enough to contain the name.
ICredentials_ERROR_VALUE_SIZE – Supplied buffer not large enough to contain the value.
ICredentials_ERROR_NOT_FOUND – No properties at given index.

14.61.2.2 method getValueByName (in buffer *name,* out buffer *value*)

Returns a property value, given a name.

Note that `name` and `value` are buffer types, not strings. The `name` buffer does not contain any null terminating character. `name` length should be passed as the size of the buffer if `name` is a zero-terminated string.

However, the `value` buffer will be populated with a terminating null byte and the resulting output length will be set to size, including the terminating null.

Parameters

in	*name*	Property name.
out	*value*	Property value, including a terminating zero byte.

Returns

Object_OK – Function returned successfully.
ICredentials_ERROR_VALUE_SIZE – Supplied buffer not large enough to contain value.
ICredentials_ERROR_NOT_FOUND – No properties found with given name.

14.62 ICrypto

14.62.1 Detailed Description

14.62.2 Function Documentation

14.62.2.1 method cmac (in int32 *alg,* in buffer *msg,* in buffer *key,* out buffer *digest*)

Creates a cipher MAC per FIPS PUB 198-1, using the specified hash algorithm.

Parameters

in	*alg*	128-bit or 256-bit AES algorithm.
in	*msg*	Message to be authenticated.
in	*key*	Input key to CMAC algorithm.
out	*digest*	CMAC digest.

Returns

Object_OK – Successful.
Object_ERROR_INVALID – Invalid parameter encountered.
Object_ERROR – Any other error encountered.

14.62.2.2 method device_kdf (in buffer *context,* out buffer *output*)

The KDF binding key derivation algorithm is an internally unique key-generation process derived by binding user data with the Secondary Hardware Key (SHK).

Parameters

in	*context*	Key derivation context.
out	*output*	Derived key.

Detailed description

Software is a three-level stack:

- AES (lowest level)
- CMAC algorithm from SP 800-38B
- Counter-based algorithm from SP 800-108 (named KDF in the implementation)

The input is a key derivation context. The output is the derived key. All sensitive data is zeroized before return.

Returns

Object_OK – Successful.
Object_ERROR_INVALID – Invalid parameter encountered.
Object_ERROR – Any other error encountered.

14.62.2.3 method hkdf (in int32 *hash_algo,* in buffer *ikm,* in buffer *salt,* in buffer *context,* out buffer *okm*)

HKDF key derivation algorithm.

Parameters

in	*hash_algo*	Hash Algorithm.
in	*ikm*	Input Key Material.
in	*salt*	Salt.
in	*context*	Context or Info.
out	*okm*	Output Key Material.

Detailed description

Software is a two-level stack:

- Pseudo Random Key (PRK) Derivation based on IKM and Salt
- Derive OKM using PRK and Context

The inputs are a IKM, a salt, and a context. The output is the derived key (OKM). All sensitive data is zeroized before return.

Returns

Object_OK – Successful.
Object_ERROR_INVALID – Invalid parameter encountered.
Object_ERROR – Any other error encountered.

14.62.2.4 method hwkey_cmac (in int32 *alg,* in interface *key_obj,* in buffer *msg,* out buffer *digest*)

Performs a CMAC operation using the supplied HWKey object as the key for the operation

Parameters

in	*alg*	128-bit or 256-bit AES algorithm.
in	*key_obj*	HWKey object to use for CMAC operation.
in	*msg*	Message to be authenticated.
out	*digest*	CMAC digest.

Returns

Object_OK – Successful.
Object_ERROR_INVALID – Invalid parameter encountered.
Object_ERROR – Any other error encountered.

14.62.2.5 method kdf (in buffer *key,* in buffer *label,* in buffer *context,* out buffer *output*)

KDF key derivation algorithm.

Parameters

in	*key*	Key derivation key.
in	*label*	Key derivation label.
in	*context*	Key derivation context.
out	*output*	Derived key.

Detailed description

Software is a three-level stack:

- AES (lowest level)
- CMAC algorithm from SP 800-38B
- Counter-based algorithm from SP 800-108 (named KDF in the implementation)

The inputs are a key derivation key, a label, and a context. The output is the derived key. All sensitive data is zeroized before return.

Returns

Object_OK – Successful.
Object_ERROR_INVALID – Invalid parameter encountered.
Object_ERROR – Any other error encountered.

14.63 IDataCache

14.63.1 Detailed Description

IDataCache provides virtual address-based CPU data cache management operations. These operations clean and/or invalidate to the Point of Coherency.

This interface uses buffer arguments in an exceptional manner. It is used in scenarios where the caller of this interface is executing in a process and the implementation resides in the kernel. The implementation makes use of the client-supplied buffer address, ∗not∗ the content of the buffers (as is the case for every other interface).

14.63.2 Function Documentation

14.63.2.1 method cleanAndInvalidateRegion (out buffer *region*)

Cleans and invalidates a memory region in the cache.
Data in the cache is written back to the main memory if it was dirty and the region is invalidated. Any further access to data results in a cache-miss.

Parameters

out	*region*	Memory region to clean and invalidate.

Returns

Object_OK if successful.

14.63.2.2 method cleanRegion (out buffer *region*)

Cleans a memory region in the cache.
This writes back any data that is dirty but does not invalidate the cache region. Any further access to data in this region results in a cache-hit.

Parameters

out	*region*	The memory region to clean.

Returns

Object_OK if successful.

14.63.2.3 method invalidateRegion (in buffer *region*)

Invalidates a memory region in the cache.
Data in the cache is not written back to the main memory. Any further access to data in this region results in a cache-miss.

Parameters

in	*region*	Memory region to invalidate.

Returns

Object_OK if successful.

14.64 ICertification

14.64.1 Detailed Description

14.64.2 Function Documentation

14.64.2.1 method getHybridPrngInfo (in uint32 *info_type,* out buffer *info_buf*)

Get Hybrid PRNG Module info

Parameters

in	*info_type*	Type of information requested
out	*info_buf*	Buffer to write requested information

Returns

Object_OK if successful.

14.65 IDAError

14.65.1 Detailed Description

IDAError provides a list of all the errors that might be returned by the methods of the IAttestationReport, IAttestationBuilder, and IDeviceAttestation interfaces.

14.65.2 Variable Documentation

14.65.2.1 error ADDON_CREDENTIALS_REPORT_NOT_SET

OPT_ADDON for CREDENTIALS_REPORT is not set

14.65.2.2 error ADDON_QTEE_REPORT_NOT_SET

OPT_ADDON for QTEE_REPORT is not set

14.65.2.3 error ATTESTATION_REPORT_FAILURE

Generic error that can be returned by any step while building the attestation report

14.65.2.4 error INVALID_ATTESTATION_CONTEXT

Passed attestation context is not within the expected range

14.65.2.5 error INVALID_BUFFER

Null or zero-length buffer is passed

14.65.2.6 error INVALID_CERTIFICATE

Passed certificate is invalid or it does not contain the correct feature ID

14.65.2.7 error INVALID_NONCE

Invalid nonce buffer or a nonce with an invalid length is passed

14.65.2.8 error INVALID_PARAMS_CBOR

Invalid CBOR params

14.65.2.9 error INVALID_REPORT_OFFSET

Offset exceeds the attestation report size

14.65.2.10 error INVALID_SECURITY_LEVEL

Passed security level is not within the expected range or not allowed

14.65.2.11 error INVALID_SIGNING_KEY

Passed key type is not allowed, or fetching the key to sign the attestation report based on the key type failed

14.65.2.12 error MAX_APP_DATA_LIMIT_REACHED

Maximum of 50KB cumulative app data can be added to generate a token

14.65.2.13 error NO_MEMORY

Failure during memory allocation

14.65.2.14 error NOT_ALLOWED

Generic error for a requested operation not being allowed

14.65.2.15 error WARM_UP_FAILURE

Generic error that can be returned by any step while warming up any submod and retrieving its status

14.66 IDeviceAttestation

14.66.1 Detailed Description

14.66.2 IDA Overview

Device Attestation provides APIs to create an attestation report containing a device's state information. The Device Attestation Report is signed and encrypted. Device Attestation is used in conjunction with QWES business-to-business (B2B) cloud APIs for verification and decode.

14.66.3 Function Documentation

14.66.3.1 method getWarmUpStatus (out buffer *warmUpStatus*)

Returns CBOR that shows the last time warming up completed for each individual optional data requested by the client.

Parameters

out	*warmUpStatus*	Buffer containing warmup completion timestamps in CBOR format.

Returns

Object_OK on success.

14.66.3.2 method start (in buffer *licenseCert,* out IAttestationBuilder *attestation Builder*)

Starts creating an attestation.

Parameters

in	*licenseCert*	Buffer containing license certificate.
out	*attestationBuilder*	IAttestationBuilder object to construct an attestation.

Detailed description The license certificate must chain up to a license root to encrypt the attestation.

Returns

Object_OK on success.

14.66.3.3 method warmUp (in uint64 *options,* in uint64 *timeout,* in interface *callback*)

Pokes at all necessary entities to prepare the relevant data, and initiates any other warmup activity that might be applicable to the passed options.

Parameters

in	*options*	Options for the warmup.
in	*timeout*	Time limit for the warmup to complete.
in	*callback*	IDAWarmupCallback object to call when the warmup is done.

Returns

Object_OK on success.

14.67 IDeviceID

14.67.1 Detailed Description

14.67.2 Function Documentation

14.67.2.1 method getClientDeviceID (in buffer *clientSalt,* out buffer *deviceId*)

Returns a digest of a salted SHA256 hash of the CHIP_ID concatenated with the SERIAL_NUM.

This function provides the Client with the ability to create a Client specific unique device ID.

Parameters

in	*clientSalt*	Buffer from the Client containing the hash salt. Any value is valid for the clientSalt, even empty.
out	*deviceId*	Client specific digest. The buffer needs to be at least 32 bytes in size in order to contain the digest.

Returns

- Generic errors: List of generic errors can be checked from TA development guide.
  - Object_OK if successful.
  - Object_ERROR_SIZE_OUT in case deviceId_len is exceeding TZBSP_SHA256_HASH_SZ.
  - Object_ERROR if any other failure.
- Interface errors: NA

14.67.2.2 method getComponentVersion (in uint32 *componentId,* out buffer *versionBuffer*)

Returns a buffer identifying the version of the component executing on this device.

Parameters

in	*componentId*	Component identifier. Valid value must belong to enum category: image_index_type.
out	*versionBuffer*	Componet version buffer.

Detailed description

These buffers are printable ASCII strings, but are not NULL-terminated. The maximum buffer size is 128 bytes.

Returns

- Generic errors: List of generic errors can be checked from TA development guide.
  - Object_OK if successful.
  - Object_ERROR if failure.

- Interface errors: NA

14.67.2.3 method getDeviceIdFromCM (out buffer *deviceId*)

Returns device information from CM hardware.

Parameters

out	*deviceId*	Device Unique ID from CM hardware.

Returns

- Generic errors: List of generic errors can be checked from TA development guide.
  - Object_OK if successful.
  - Object_ERROR_SIZE_OUT in case deviceId_len is exceeding sizeof(CRI_CM_DEVICE_INFO.deviceId).
  - Object_ERROR if any other failure.
- Interface errors: NA

14.67.2.4 method getDeviceUUID (out buffer *uuid*)

Returns a globally-unique ID identifying this device.

Parameters

out	*uuid*	UUID in native byte order.

Returns

- Generic errors: List of generic errors can be checked from TA development guide.
  - Object_OK if successful.
  - Object_ERROR if failure.
- Interface errors: NA

14.67.2.5 method getHWVersion (out uint32 *hwVersion*)

Returns the 32-bit hardware version from the metal layer.

Parameters

out	*hwVersion*	Hardware version integer.

Returns

- Generic errors: List of generic errors can be checked from TA development guide.

  - Object_OK if successful.
- Interface errors: NA

14.67.2.6 method getOEMID (out uint32 *oemId*)

Returns the 32-bit integer OEM ID from fuses.

Parameters

out	*oemId*	OEM ID.

Returns

- Generic errors: List of generic errors can be checked from TA development guide.
  - Object_OK if successful.
- Interface errors: NA

14.67.2.7 method getPKHash (out buffer *pkHash*)

Returns the 32-byte hash of the public key used to verify the boot image.

Parameters

out	*pkHash*	Public key hash.

Returns

- Generic errors: List of generic errors can be checked from TA development guide.
  - Object_OK if successful.
  - Object_ERROR_SIZE_OUT in case pkHash_len is exceeding SECBOOT_OTP_ROOT_OF_TRUST_BYTE_SIZE.
  - Object_ERROR if any other failure.
- Interface errors: NA

14.67.2.8 method getProductID (out uint32 *productId*)

Returns the 32-bit integer Product ID from fuses.

Parameters

out	*productId*	Product ID.

Returns

- Generic errors: List of generic errors can be checked from TA development guide.

  - Object_OK if successful.
- Interface errors: NA

14.67.2.9 method readJtagID (out uint32 *jtagId*)

Reads JTAG ID.

Parameters

out	*jtagId*	ID from the JTAG.

Returns

- Generic errors: List of generic errors can be checked from TA development guide.
  - Object_OK if successful.
- Interface errors: NA

14.67.2.10 method readSerialNum (out uint64 *serialNum*)

Reads serial number from PTE chain.

Parameters

out	*serialNum*	Device serial number.

Returns

- Generic errors: List of generic errors can be checked from TA development guide.
  - Object_OK if successful.
- Interface errors: NA

14.68 IEnv

14.68.1 Detailed Description

14.68.2 Function Documentation

14.68.2.1 method exit (in int32 *code*)

Terminates execution within the current execution environment (process or VM).

Parameters

in	*code*	Exit code. Non-zero exit conditions indicate an exception, and imply that a core dump or other diagnostic contingencies apply, depending on the environment.

Returns

Object_OK on success.

14.68.2.2 method log (in buffer *text*)

Outputs message for debugging.

Parameters

in	*text*	Message provided as a buffer (not a string).

Detailed description

The buffer must not contain a terminating NULL character. All characters in the buffer must be valid ASCII. A terminating newline character is not necessary to delimit the message from other messages.

Returns

Object_OK on success.

14.69 IESEService

14.69.1 Detailed Description

14.69.2 Function Documentation

14.69.2.1 method getAtrProperty (in uint32 *id,* out uint32 *value*)

Gets the value of the ATR property, given the identifier.

Parameters

in	*id*	Unique identifier for Secure Element ATR property.
out	*value*	Property value.

Returns

Object_OK on success.

14.69.2.2 method getOemProperty (in uint32 *id,* out uint32 *value*)

Gets the value of the OEM Configuration, given the identifier.

Parameters

in	*id*	Unique identifier for Secure Element property.
out	*value*	Property value.

Returns

Object_OK on success.

14.69.2.3 method ProvisionDeviceIds (in buffer *device,* in buffer *brand,* in buffer *model,* in buffer *product,* in buffer *manufacturer,* in buffer *serial,* in buffer *imei,* in buffer *imei_2,* in buffer *meid*)

Interface to re-provision DeviceID parameters.

Parameters

device	
brand	
model	
product	
manufacturer	
serial	
imei	
imei_2	
meid	

Returns

Object_OK on success.

14.69.2.4 method read (out buffer *rapdu*)

Reads from slave data on the logical connection handle.

Parameters

out	*rapdu*	Reads buffer information.

Returns

Object_OK on success.

14.69.2.5 method receive (out buffer *rapdu,* out int32 *result*)

Receives response from the embedded secured element.

Similar to the read() method but allows user to specify the result pointer so that the response buffer may have content even when there is no complete read yet, e.g., when result contains a WTX indication.

Parameters

out	*rapdu*	Read buffer information.
out	*result*	eseservice error/status.

Returns

Object_OK on success.

14.69.2.6 method seac (in buffer *aid,* in buffer *apdu*)

Performs Secure Element Access Control (SEAC) based on Application Identifier.

Parameters

in	*aid*	Applet Application Identifier.
in	*apdu*	APDU where SEAC filtering should be applied. NULL means no APDU filtering should be applied.

Returns

Object_OK on success; otherwise access denied.

14.69.2.7 method send (in uint8 *type,* in buffer *capdu,* out int32 *result*)

Sends a command to the Embedded Secure Element (ESE).

Similar to write function but allows user to specify type and result pointer.

Parameters

in	*type*	Identifies frame as I/R/S-Block.
in	*capdu*	Write buffer information.
out	*result*	eseservice error/status.

Returns

Object_OK on success.

See also

14.69.2.8 method setROT ( )

Request to set the RoT state.

Returns

Object_OK on success.

14.69.2.9 method updateOSUState (in uint8 *osuState*)

Update current OSU state in eseservice TA.

Parameters

in	*osuState*	Current OSU state.

Returns

Object_OK on success.

14.69.2.10 method write (in buffer *capdu*)

Writes to slave data on the logical connection handle.

Parameters

in	*capdu*	Write buffer information.

Returns

Object_OK on success.

14.70 IDiagnostics

14.70.1 Detailed Description

IDiagnostics is an interface enabling clients to obtain diagnostic information.

14.70.2 Function Documentation

14.70.2.1 method queryAppDump (in buffer *distName,* out IDiagnostics_RegisterInfo *usrRegInfo,* out buffer *usrStack,* out uint64 *usrStackAddr*)

Provides context information and stack content for specific loaded trusted application.

This API works only when debug is enabled.

Parameters

in	*distName*	distinguished trusted application name to dump information
out	*usrRegInfo*	Register values for user mode context
out	*usrStack*	User mode stack content buffer
out	*usrStackAddr*	Start address of user mode stack

Returns

Object_OK - on success.
Object_ERROR - Internal error retrieving application info.
Object_ERROR_SIZE_OUT - if provided buffer is too small for dumping stacks of given trusted application.
IDiagnostics_ERROR_ARCH_NOT_SUPPORTED - if archtecture type of given trusted application isn't supported.
IDiagnostics_ERROR_APP_NOT_FOUND - if trusted application with the distinguished name not found.
IDiagnostics_ERROR_RESTRICTED - if debug is disabled.

14.70.2.2 method queryAppInfo (in buffer *appDistName,* out IDiagnostics_AppInfo *appInfo*)

Provides the caller with information about the specified trusted application.

Parameters

in	*appDistName*	Trusted application distinguished name.
out	*appInfo*	An output structure populated with application runtime information.

Returns

Object_OK - Successful.
Object_ERROR - Internal error retrieving application info.
IDiagnostics_ERROR_APP_NOT_FOUND - No loaded application with provided distinguished name.
IDiagnostics_ERROR_RESTRICTED - if debug is disabled.

14.70.2.3 method queryAppRegion (out IDiagnostics_AppRegionInfo *pimemInfo,* out IDiagnostics_AppRegionInfo *ddrInfo,* out uint32 *scatteredAllocator Support*)

Returns the amount of available space (PIMEM and DDR) in the App Region. Also returns the size of the largest contiguous space in PIMEM and DDR and if Scattered Memory Allocation is supported.

Parameters

out	*pimemInfo*	An output structure populated with PIMEM usage information.
out	*ddrInfo*	An output structure populated with DDR usage information.
out	*scatteredAllocator Support*	Value of 1 if scattered Memory Allocation is supported, 0 if it is not.

Returns

Object_OK on success.

14.70.2.4 method queryHeapInfo (out IDiagnostics_HeapInfo *heapInfo*)

The app is blocked on a request Provides the caller with information regarding heap usage in QTEE.

Parameters

out	*heapInfo*	An output structure populated with heap usage information.

Returns

Object_OK on success.

14.70.2.5 method queryLoadedApps (out buffer *loadedApps*)

Provides the caller with a list of the currently loaded trusted applications.

Parameters

out	*loadedApps*	An output buffer populated with the distinguished name of every currently loaded app. Each entry in the list is separated by a single newline character. A null-character is appended at the end of the list. The list has no guaranteed order. The caller must provide a buffer large enough for the entire output.

Returns

Object_OK on success. Object_ERROR_SIZE_OUT if the output buffer is not sufficiently large.

14.70.3 Variable Documentation

14.70.3.1 const uint32 APP_STATUS_ABORT = 2

The app is idle

14.70.3.2 const uint32 APP_STATUS_BLOCKED = 3

The app is exited, crashed, or killed.

14.71 IFeatureVersions

14.71.1 Detailed Description

14.71.2 Function Documentation

14.71.2.1 method getVersionId (in uint32 *feature_id,* out uint32 *version*)

For a given feature, return the current version.

Parameters

in	*feature_id*	Unique identifier for a given feature.
out	*version*	Holds the version information.

Returns

Object_OK on success, ERROR_INVALID_ID on failure.

14.72 IGenericService

14.72.1 Detailed Description

14.72.2 Function Documentation

14.72.2.1 method handleCommand (in buffer *inBuf,* out buffer *outBuf*)

Generic request/response interface used to forward requests of unknown format. Do not reuse.

Parameters

in	*inBuf*	Request buffer from the client.
out	*outBuf*	Response buffer for the client.

Returns

Object_OK if successful.

14.73 IGPSession

14.73.1 Detailed Description

14.73.2 Function Documentation

14.73.2.1 method close ( )

Close the session.

Returns

Object_OK on success.

14.73.2.2 method invokeCommand (in uint32 *commandID,* in uint32 *cancelCode,* in uint32 *cancellationRequestTimeout,* in uint32 *paramTypes,* in uint32 *exParamTypes,* in buffer *i1,* in buffer *i2,* in buffer *i3,* in buffer *i4,* out buffer *o1,* out buffer *o2,* out buffer *o3,* out buffer *o4,* in interface *imem1,* in interface *imem2,* in interface *imem3,* in interface *imem4,* out uint32 *memrefOutSz1,* out uint32 *memrefOutSz2,* out uint32 *memrefOutSz3,* out uint32 *memrefOutSz4,* out uint32 *retValue,* out uint32 *retOrigin*)

Invokes a command on an open GP Session.

The content of the 4 input and output buffers is manually marshalled by the caller or implementer depending on the type of the parameter.

When a memory object is passed to represent a shared buffer, the corresponding input buffer is expected to contain a struct IGPSession_MemoryObjectParameters to detail the size and offset of the portion of the buffer being actually shared with the TA.

The memrefOutSz[1-4] parameters carry the output size of the passed memory objects, or, if a larger buffer was requested, the size of the larger buffer.

Parameters

in	*commandID*	Numeric code for the command.
in	*cancelCode*	Optional code to use for cancellations.
in	*cancellationRequest Timeout*	Timeout for automatic request cancellation.
in	*paramTypes*	Types of the 4 parameters, 1 byte per parameter.
in	*exParamTypes*	Extended information for the 4 parameters, 1 byte per parameter.
in	*i1*	First input buffer.
in	*i2*	Second input buffer.
in	*i3*	Third input buffer.
in	*i4*	Fourth input buffer.
out	*o1*	First output buffer.
out	*o2*	Second output buffer.
out	*o3*	Third output buffer.
out	*o4*	Fourth output buffer.

in	*imem1*	First optional memory object.
in	*imem2*	Second optional memory object.
in	*imem3*	Third optional memory object.
in	*imem4*	Fourth optional memory object.
out	*memrefOutSz1*	Output size for memref 1.
out	*memrefOutSz2*	Output size for memref 2.
out	*memrefOutSz3*	Output size for memref 3.
out	*memrefOutSz4*	Output size for memref 4.
out	*retValue*	GP return value.
out	*retOrigin*	Where the GP return value originated.

Returns

Object_OK if successful.

14.74 IHash

14.74.1 Detailed Description

14.74.2 Function Documentation

14.74.2.1 method decrypt (in interface *cipher_obj,* in buffer *cipher,* out buffer *plain,* out buffer *digest*)

Performs a simultaneous hash/cipher decrypt operation.

Parameters

in	*cipher_obj*	Cipher context (object).
in	*cipher*	Input ciphertext buffer.
out	*plain*	Output plaintext buffer.
out	*digest*	Digest to store.

Returns

Object_OK – Successful.
Object_ERROR – Any error encountered.

14.74.2.2 method encrypt (in interface *cipher_obj,* in buffer *plain,* out buffer *cipher,* out buffer *digest*)

Performs a simultaneous hash/cipher encrypt operation.

Parameters

in	*cipher_obj*	Cipher context (object).
in	*plain*	Input plaintext buffer.
out	*cipher*	Output ciphertext buffer.
out	*digest*	Digest to store.

Returns

Object_OK – Successful.
Object_ERROR – Any error encountered.

14.74.2.3 method final (out buffer *digest*)

Computes the digest hash value.

Parameters

out	*digest*	Message digest hash.

Returns

Object_OK – Successful.

Object_ERROR – Any error encountered.

14.74.2.4 method hash (in buffer *plain,* out buffer *digest*)

Computes the digest hash value with the input data.

Parameters

in	*plain*	Plain text message to hash.
out	*digest*	Message hash digest.

Returns

Object_OK – Successful.
Object_ERROR – Any error encountered.

14.74.2.5 method reset ( )

Resets hash context; does not reset the key.

Returns

Object_OK – Successful.
Object_ERROR – Any error encountered.

14.74.2.6 method setParamAsData (in int32 *paramID,* in buffer *param*)

Modifies parameter value for a given hash operation.

Parameters

in	*paramID*	Parameter to modify.
in	*param*	Parameter value to set.

Returns

Object_OK – Successful.
Object_ERROR – Any error encountered.

14.74.2.7 method setParamAsU32 (in int32 *paramID,* in uint32 *param*)

Modifies the 32-bit unsigned int parameter value for a given hash operation.

Parameters

in	*paramID*	Parameter to modify.
in	*param*	Parameter value to set.

Returns

Object_OK – Successful.
Object_ERROR – Any error encountered.

14.74.2.8 method squeeze (out buffer *digest*)

Computes the digest hash value.

Parameters

out	*digest*	Message digest hash.

Returns

Object_OK – Successful.
Object_ERROR – Any error encountered.

14.74.2.9 method update (in buffer *plain*)

Hashes data into the hash context.

Parameters

in	*plain*	Plain text message to hash.

Returns

Object_OK – Successful.
Object_ERROR – Any error encountered.

14.75 IHavenTokenApp

14.75.1 Detailed Description

14.75.2 Function Documentation

14.75.2.1 method addDataItem (in buffer *DataItem,* in buffer *Label,* in buffer *Data*)

Adds any data item, i.e., int, string, maps, arrays.

Parameters

in	*DataItem*	DataItem structure.
in	*Label*	Zero-terminated string to tag the item.
in	*Data*	Data to add.

Returns

Object_OK on success.

14.75.2.2 method finish ( )

Deinitialize and clean up.

Returns

Object_OK on success.

14.75.2.3 method getBytes (in uint64 *uOffset,* out buffer *pBuffer*)

Gets the bytes in the token.

Parameters

in	*uOffset*	Token offset.
out	*pBuffer*	Token buffer.

Detailed description

Typical use is to get the the size and then call this in a loop with a 4KB buffer incrementing the offset by the number of bytes returned until all the bytes have been fetched.

The caller should keep track of the bytes fetched to know when they have got them all. There is no "end of file" return code.

A buffer larger than 4KB may be used with some implementations. If the buffer is too large for the underlying IPC mechanism, then an error will be returned.

An error will be returned if uOffset is not in the range of 0 and the size of the token.

The token is an encrypted data blob. There is no use for part of a token because it cannot be decrypted; therefore, the full token must be fetched.

Returns

Object_OK on success.

14.75.2.4 method getSize (out uint64 *pnSize*)

Gets full size of the encrypted and signed token.

Parameters

out	*pnSize*	Token size.

Returns

Object_OK on success.

14.75.2.5 method start (in uint32 *uKeySelect,* in uint64 *nOpts,* in buffer *LicenseCert*)

Start creating a Haven Token.

Parameters

in	*uKeySelect*	Haven key type.
in	*nOpts*	Haven token option flags.
in	*LicenseCert*	Buffer containing license certificate.

Detailed description The License Certificate must chain up to a Haven License root to

encrypt the token in a CMS Enveloped Data format.

Returns

Object_OK if successful.

14.76 IHdcpEncryption

14.76.1 Detailed Description

Hdcp Encryption interface controls hardware blocks that implement HDCP encryption (HDMI, DP) and keeps track of the minimum encryption level per DRM policy.

14.76.2 Function Documentation

14.76.2.1 method disable (in uint32 *hdcpVersion,* in uint32 *deviceType*)

Disables HDCP encryption.

Parameters

in	*hdcpVersion*	HDCP version.
in	*deviceType*	HDCP device type.

Returns

Object_OK if successful.

14.76.2.2 method enable (in uint32 *hdcpVersion,* in uint32 *deviceType,* in buffer *key,* in buffer *randomIV*)

Enable HDCP encryption.

Parameters

in	*hdcpVersion*	HDCP version.
in	*deviceType*	HDCP device type.
in	*key*	Key to write to hardware for encryption.
in	*randomIV*	Random IV to set to hardware for encryption.

Returns

Object_OK on success.

14.76.2.3 method enforceEncryption (in uint32 *deviceType,* in int32 *enable*)

Forces encryption for Display interface.

Parameters

in	*deviceType*	HDCP device type.
in	*enable*	1 to override secure buffer to force encryption. 0 to disable override and allow secure buffer to steer encryption.

Returns

Object_OK on success.

14.77 IHdcpSrm

14.77.1 Detailed Description

Updates Output Protection Service with the revoked receiver IDs list retrieved from a signed SRM file.

14.77.2 Function Documentation

14.77.2.1 method updateRevokedIds (in uint16 *version,* in uint8[ ] *receiverIdList*)

Updates receiver IDs parsed from HDCP System Renewability Message.

Parameters

in	*version*	SRM version number.
in	*receiverIdList*	Receiver IDs to be revoked.

Returns

Object_OK on success.

14.78 IHdcpTransmitter

14.78.1 Detailed Description

HDCP Transmitter applications call IHdcpTransmitter to report HDCP encryption levels and downstream device topology. By implementing these functions, Output Protection Service (OPS) can track all HDCP Transmitters and their topologies.

14.78.2 Function Documentation

14.78.2.1 method contentProtectionLevelUpdated (in uint8 *cpl*)

HDCP Transmitter TAs call this function to update Output Protection Service with the new Content Protection Level.

Whenever content's protection level changes, OPS ensures that the HDCP Transmitter honors the new level by receiving the update. If this update is not received within a time limit, HDCP Tx receives an error through IHdcpNotify.

Parameters

in	*cpl*	New content protection level.

Returns

Object_OK on success.

14.78.2.2 method setDeviceProtectionLevel (in uint8 *dpl*)

Sets the HDCP encryption level supported by the display device. Wireless HDCP transmitters call this function.

Parameters

in	*dpl*	HDCP encryption level for the device.

Returns

Object_OK on success.

14.78.2.3 method setDeviceTopology (in uint8[] *receiverIdList,* in HdcpTopology *hdcpTopology*)

Sets HDCP Topology for a device.

Parameters

in	*receiverIdList*	Downstream device ID list.
in	*hdcpTopology*	Device topology information.

Returns

Object_OK on success.

14.79 IHdmiStatus

14.79.1 Detailed Description

14.79.2 Function Documentation

14.79.2.1 method hdmiStatusRead (out uint32 *hdmiEnable,* out uint32 *hdmiSense,* out uint32 *hdcpAuth*)

Reads status of the HDMI link and hardware HDCP.

Parameters

out	*hdmiEnable*	HDMI output enabled.
out	*hdmiSense*	HDMI sense.
out	*hdcpAuth*	HDCP authentication success.

Returns

Object_OK on success.

14.80 IHlosRegionFinder

14.80.1 Detailed Description

14.80.2 Function Documentation

14.80.2.1 method getRegion (in uint64 *physAddr,* in uint64 *size,* out IMemRegion *memRegionOut*)

Creates a memory region to access HLOS memory. Limitation: Creates region only on cached memory.

Parameters

in	*physAddr*	Physical address of HLOS memory.
in	*size*	Size of physical memory region.
out	*memRegionOut*	On success, HLOS Memory Region object.

Returns

Object_OK indicates success.
Object_ERROR indicates failure.

14.81 IHmac

14.81.1 Detailed Description

14.81.2 Function Documentation

14.81.2.1 method final (out buffer *digest*)

Final operation for HMAC per FIPS PUB 198-1 using the specified hash algorithm.

Parameters

out	*digest*	Pointer to message digest (memory provided by caller).

Returns

Object_OK – Successful.
Object_ERROR – Any error encountered.

14.81.2.2 method hmac (in buffer *msg,* out buffer *digest*)

Calculate HMAC per FIPS PUB 198-1, using the configured hash algorithm and key for the input message.

Parameters

in	*msg*	Pointer to message to be authenticated.
out	*digest*	Pointer to message digest (memory provided by caller).

Returns

Object_OK – Successful.
Object_ERROR – Any error encountered.

14.81.2.3 method setParamAsData (in int32 *paramID,* in buffer *param*)

Modifies parameters for a given HMAC operation.

Parameters

in	*paramID*	Parameter to modify.
in	*param*	Parameter value to set.

Returns

Object_OK – Successful.
Object_ERROR – Any error encountered.

14.81.2.4 method setParamAsObject (in int32 *paramID,* in interface *object*)

Modifies parameters for a given HMAC operation.

Parameters

in	*paramID*	Parameter to modify.
in	*object*	Parameter value to set.

Returns

Object_OK – Successful.
Object_ERROR – Any error encountered.

14.81.2.5 method update (in buffer *msg*)

Updates HMAC per FIPS PUB 198-1, using the configured hash algorithm.

Parameters

in	*msg*	Pointer to message to be authenticated.

Returns

Object_OK – Successful.
Object_ERROR – Any error encountered.

14.82 IHwFuse

14.82.1 Detailed Description

14.82.2 Function Documentation

14.82.2.1 method fuseRead (in uint32 *rowAddress,* in int32 *addrType,* out uint32 *rowData1,* out uint32 *rowData2,* out uint32 *qfpromApiStatus*)

Reads row data of the specified QFPROM row address.

Parameters

in	*rowAddress*	Row address in the QFPROM region from which the row data is read.
in	*addrType*	Raw (uncorrected) or FEC-corrected data.
out	*rowData1*	Lower 32 bits of data read from QFPROM region.
out	*rowData2*	Upper 32 bits of the data read from QFPROM region.
out	*qfpromApiStatus*	Return value from QFPROM function.

Returns

Object_OK – Success.
Object_ERROR – Any error occurred.

14.82.2.2 method fuseWrite (in uint32 *rawRowAddress,* in uint32 *rowData1,* in uint32 *rowData2,* out uint32 *qfpromApiStatus*)

Writes row data to the specified QFPROM raw row address.

Parameters

in	*rawRowAddress*	Row address in QFPROM region to which the row data is to be written.
in	*rowData1*	Lower 32 bits of data to write into QFPROM region.
in	*rowData2*	Upper 32 bits of data to write into QFPROM region.
out	*qfpromApiStatus*	Return value from QFPROM function.

Returns

Object_OK – success.
Object_ERROR – any error occurred.

14.83 IHWKey

14.83.1 Detailed Description

14.83.2 Function Documentation

14.83.2.1 method clearSlotId (in uint32 *slotId*)

Clear a key present in one of the persistent slots.

Parameters

in	*slotId*	Persistent slot in which the key is present.

Returns

Object_OK on success

14.83.2.2 method getBlobSize (out uint32 *size*)

Get the size of a wrapped key blob in bytes. The size and format of a wrapped key blob will vary depending on the version of the HW used in the chipset. Clients should ensure that buffers passed into wrap() are large enough to accomodate the size returned by this method.

Parameters

out	*size*	Size of a wrapped key blob in bytes

Returns

Object_OK on success

14.83.2.3 method getSlotId (out uint32 *slotId*)

If the key is present in a persistent slot, this can be used to get the slot ID provided the key object has the persistent permissions set correctly.

Parameters

out	*slotId*	Persistent slot in which the key is present.

Returns

Object_OK on success

14.83.2.4 method wrap (in interface *wrappingKey,* out buffer *blob*)

Wrap the hardware key using a hardware wrapping key. On success, the wrapped key blob will be returned to the client. This wrapped blob can be stored on the device and can be converted into a key object at any later time.

Parameters

`in`	*wrapping_key*	Key to be used for wrapping, whose type will depend on the interface invoked (i.e. for example IHWKey)
`out`	*blob*	A blob containing the wrapped key. WARNING: The size and format of a wrapped key blob may differ between chipsets. To use this API in a chipset agnostic manner, a client should ensure that the size of the output buffer is greater than or equal to the value returned by getBlobSize()

Returns

Object_OK on success

14.84 IHWKeyFactory

14.84.1 Detailed Description

14.84.2 Function Documentation

14.84.2.1 method derive (in IHWKey_policy *policy,* in IHWKey_bindings *bindings,* in IHWKey *derivationKey,* in IHWKey *mixingKey,* out interface *key*)

Derive a hardware key from a derivation key.
WARNING: Wrapping keys created through this method must not be descendants of keys created by importKey()

Parameters

in	*policy*	Key policy for the derived key
in	*bindings*	Software and device state bindings for the key
in	*derivationKey*	Key object representing the derivation key
in	*mixingKey*	Optional argument. Key object to be mixed to the context during key derivation
out	*key*	Output key object, whose type will depend on the interface invoked (i.e. for example IHWKey)

Returns

Object_OK on success
ERROR_INVALID_WRAP_KEY_SOURCE if policy's keyType is KEY_TYPE_WRAPPING and derivationKey is a descendant of a key created through importKey()

14.84.2.2 method deriveFromHWSource (in IHWKey_policy *policy,* in IHWKey_bindings *bindings,* in uint32 *source,* in IHWKey *mixingKey,* out interface *key*)

Derive a hardware key from a deterministic HW source.

Parameters

in	*policy*	Key policy for the resulting key
in	*bindings*	Software and device state bindings for the key
in	*source*	HW source to be used to derive the child key
in	*mixingKey*	Optional argument. Key object to be mixed to the context during key derivation
out	*key*	Output key object, whose type will depend on the interface invoked (i.e. for example IHWKey)

Returns

Object_OK on success

14.84.2.3 method generate (in IHWKey_policy *policy,* out interface *key*)

Generate a random hardware key.

Parameters

in	*policy*	Key policy for the generated key
out	*key*	Output key object, whose type will depend on the interface invoked (i.e. for example IHWKey)

Returns

Object_OK on success

14.84.2.4 method importKey (in IHWKey_policy *policy,* in buffer *swKey,* out interface *key*)

Import a software key to create an IHWKey object managed by HW key manager.
WARNING: Wrapping keys cannot be created through this method.

Parameters

in	*policy*	Key policy for the created key
in	*swKey*	Plaintext software key to be managed by hardware - algorithm in policy determines key size
out	*key*	Output key object, whose type will depend on the interface invoked (i.e. for example IHWKey)

Returns

Object_OK on success
ERROR_INVALID_WRAP_KEY_SOURCE if policy's keyType is KEY_TYPE_WRAPPING

14.84.2.5 method unwrap (in IHWKey *wrappingKey,* in buffer *wrappedBlob,* out interface *key*)

Create a hardware key from a wrapped key blob.

Parameters

in	*wrappingKey*	Key to be used for unwrapping.
in	*wrappedBlob*	Blob containing the wrapped key
out	*key*	Output key object, whose type will depend on the interface invoked (i.e. for example IHWKey)

Returns

Object_OK on success

14.85 II2C

14.85.1 Detailed Description

Note: The acquireExclusiveAccess() and releaseExclusiveAccess() are deprecated.

14.85.2 Function Documentation

14.85.2.1 method close (in int32 *deviceId*)

Transfers access of I2C QUP back to REE.

Parameters

in	*deviceId*	ID of I2C device to be closed.

Detailed description

This method only returns success if each operation is successful:

1. Remove the exclusive access lock to the I2C bus, then close device.
2. Remove QUP block protection for GSBI3.
3. Remove control block protection for GSBI3.
4. Reenable the I2C interrupt.
5. Deregister for the I2C interrupt to transfer the I2C interrupt back to REE.

Returns

Object_OK if successful.

Dependencies

II2C::open() method must be called successfully first.

14.85.2.2 method open (in int32 *deviceId*)

Transfers access to the I2C bus to the calling application.

Parameters

in	*deviceId*	I2C device ID to attach to.

Detailed description

This method only returns success if each operation is successful, as follows:

1. Obtain a handle to the requested device, then open device.
2. Lock I2C bus for exclusive access.
3. Register for the I2C interrupt in QTEE (transfers interrupt from REE to TEE).
4. Disable I2C interrupt (QTEE I2C is not interrupt-driven).

5. Protect the control block for the device.
6. Protect QUP block for the device.

Returns

Object_OK if successful.

14.85.2.3 method read (in int32 *deviceId,* in II2C_Config *config,* in uint32 *startAddr,* out buffer *payload*)

Reads data from a slave device.

Parameters

in	*deviceId*	I2C device ID being read from.
in	*config*	Bus and slave configuration for this read transaction.
in	*startAddr*	Address to read from.
out	*payload*	Returned data.

Detailed description

The read aborts if the slave device does not acknowledge control/address bytes written to it (before the data read starts). A read operation always terminates with a STOP condition.

An error is generated if the bus is in an inconsistent state, i.e., uninitialized or busy.

The following diagram shows the bus activity during a read. The second START is interpreted as a repeated START, but might be replaced by a STOP and START by some protocols.

```
           |<------optional------>|
-------------------------------------------------------------
|S|       | |        | |S|       | |                   |N|S|
|T|Control|A|Address |A|T|Control|A|      Data         |O|T|
|A| Byte  |C|[Offset |C|A| Byte  |C|                   |A|O|
|R|       |K| or Reg]|K|R|       |K|                   |C|P|
|T|       | |        | |T|       | |                   |K| |
-------------------------------------------------------------
```

The address can be 0, 1, or 2 bytes depending on the slave. Every address byte written to the slave device must be acknowledged.

This method is blocking. It returns when either data has been read, or an error has occurred.

Returns

Object_OK if successful.

Dependencies

The II2C::open() method must be successfully called first.

14.85.2.4 method write (in int32 *deviceId,* in II2C_Config *config,* in uint32 *startAddr,* in buffer *payload,* out uint64 *bytesWritten*)

Writes data to a slave device.

Parameters

in	*deviceId*	I2C device ID being written to.
in	*config*	Bus and slave configuration for this write transaction.
in	*startAddr*	Address to write to.
in	*payload*	Data to write.
out	*bytesWritten*	Number of bytes written.

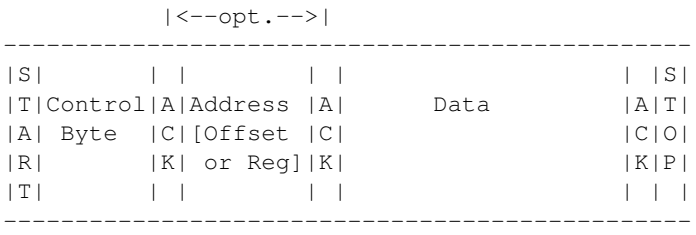
Detailed description

The write aborts if the slave device does not acknowledge control/address/data bytes written to it. The write operation always terminates with a STOP condition.

If a write is attempted with a count of zero, the slave device is selected and returns success if the slave device acknowledges. Otherwise, a failure is returned.

You can use this feature to test if a slave device is addressable, e.g., NVRAM devices performing internal cache writes.

The following diagram shows the bus activity during a write operation.



The address can be 0, 1, or 2 bytes depending on the slave. Every address byte written to the slave device must be acknowledged.

Returns

Object_OK if successful.

Dependencies

The II2C::open() method must be successfully called first.

14.86 IICE

14.86.1 Detailed Description

14.86.2 Function Documentation

14.86.2.1 method clearIceKey (in uint32 *index*)

Clear the ICE Key in the ICE hardware.

Parameters

in	*index*	Contains the key index.

Detailed description

This method is used to invalidate an already set key in the Inline Crypto Engine (ICE).

Returns

The return value is Object_OK on success and Object_ERROR otherwise.

14.86.2.2 method getWrappedKey (in buffer *key,* in uint32 *deCe,* out buffer *wrappedkey*)

Gets the wrapped key to be exported to REE.

Parameters

in	*key*	key to be wrapped. Expects an AES 256 byte key.
in	*deCe*	Device (DE) '0' or credential encrypted key (CE) '1'. DE keys are not bound to user credentials and all keys are wrapped with a key derived from a global per-boot ephemeral key. CE keys are bound to user credentials and are stored in key table to securely evict the keys when an user is locked. Once the key table is full CE keys will fallback to the same mechanism used for DE keys.
out	*wrappedkey*	wrapped key to be returned to REE. Expects a 64 byte buffer.

Detailed description

This method is used to retrieve the wrapped key based on the key type.

Returns

The return value is Object_OK on success and Object_ERROR otherwise.

14.86.2.3 method getWrappedKeyV2 (in buffer *key,* out buffer *wrappedkey*)

Gets the wrapped key (V2) to be exported to REE. The new version that was upstreamed supports variable length keyblobs, this ensures TZ no longer is required to cache any key.

Parameters

in	*key*	key to be wrapped. Expects an AES 256 byte key.
out	*wrappedkey*	Returns the REE raw secret and the per-boot ephemeral wrapped contents encryption class key in the following format. (REE raw secret (32 bytes) || IV (12 bytes) || Encrypted key (32 bytes) || Tag (16 bytes)).

Detailed description

This method is used to retrieve the wrapped key based on the key type.

Returns

The return value is Object_OK on success and Object_ERROR otherwise.

14.86.2.4 method setEphemeralContext (in buffer *context*)

Sets the ephemeral context to be used for this boot.

Parameters

in	*context*	Context to be used for generating ephemeral wrapping key.

Detailed description

This method is used to set the ephemeral context for hwkm FBE.

Returns

The return value is Object_OK on success and Object_ERROR otherwise.

14.86.2.5 method setIceKey (in uint32 *index,* in buffer *key,* in buffer *salt*)

Set the ICE Key in the ICE hardware.

Parameters

in	*index*	Contains the key index.
in	*key*	Buffer containing the key. Expected size is 32 bytes.
in	*salt*	Buffer containing the salt. Expected size is 32 bytes.

Detailed description

This method calls into the kernel API which sets the provided key in the Inline Crypto Engine (ICE) hardware. Keys set here have an index range from 0-31.

Returns

The return value is Object_OK on success and Object_ERROR otherwise.

14.86.2.6 method setSeedV2 (in buffer *seed*)

Sets the ephemeral seed to be used for this boot.

Parameters

in	*seed*	seed to be used for generating ephemeral wrapping key.

Detailed description

This method is used to set the ephemeral context for TZ FBE.

Returns

The return value is Object_OK on success and Object_ERROR otherwise.

14.87 IIntMask

14.87.1 Detailed Description

14.87.2 Function Documentation

14.87.2.1 method disableAllInterrupts (out uint32 *restoreMask*)

Modifies system behavior to prevent interrupts.

Parameters

out	*restoreMask*	Bit mask (as FIQ, IRQ, SError, and DebugException) that may be passed to setIntMask() to restore system to state prior to calling this function.

Returns

Object_OK on success.

14.87.2.2 method getIntMask (out uint32 *intMask*)

Returns the current status of interrupt masking.

Parameters

out	*intMask*	Bit mask that represents the current state of interrupt masking (as IRQ, FIQ, SError, and DebugException). A value of 1 for each interrupt source indicates masked interrupt source. A value of zero indicates interrupt source is not masked.

Returns

Object_OK on success.

14.87.2.3 method getSecureIrq (out uint32 *secIrq*)

Returns the interrupt bit used to mask secure interrupts.

Parameters

out	*secIrq*	IIntMask_IRQ or IIntMask_FIQ.

Returns

Object_OK on success.

14.87.2.4 method setIntMask (in uint32 *intMask*)

Modifies system behavior to allow interrupts.

Parameters

in	*intMask*	Bit mask of different interrupt sources (IRQ, FIQ, SError, and DebugException) to disable. A value of 1 for each interrupt source masks that interrupt source. A value of zero for each interrupt source unmasks that interrupt source.

Returns

Object_OK on success.

14.88 IIO

14.88.1 Detailed Description

14.88.2 Function Documentation

14.88.2.1 method getLength (out uint64 *len*)

Retrieves maximum value for the combination of offset and buffer length for readAtOffset() and writeAtOffset().

Parameters

out	*len*	Maximum value for the combination of offset and buffer length passed to readAtOffset() and writeAtOffset().

Returns

Object_OK on success.

14.88.2.2 method readAtOffset (in uint64 *offset,* out buffer *data*)

Copies from source at the given offset into the output buffer up to the size of the output buffer. If the output buffer is larger than the source at the provided offset, the remaining bytes of the output buffer remain unchanged.

Parameters

in	*offset*	Offset to begin reading from source buffer.
out	*data*	Destination to copy source data into.

Returns

Object_OK on success.

14.88.2.3 method writeAtOffset (in uint64 *offset,* in buffer *data*)

Copies in to the destination at the given offset up to input buffer size.
If the combined input buffer length and offset are a greater length than the destination, an error is returned and no data is copied.

Parameters

in	*offset*	Offset to begin writing into the destination.
in	*data*	Source of data to copy to destination.

Returns

Object_OK on success.

14.89 IIPProtector

14.90 IKey

14.90.1 Detailed Description

14.90.2 Function Documentation

14.90.2.1 method getParameter (in uint32 *param,* out buffer *buf*)

Gets parameter associated with the key generated by IKeyManager.

Parameters

in	*param*	Parameter ID to get.
out	*buf*	param value to set.

Detailed description

Supported params:

- PRIVATE_KEY - param value set in buf
- PUBLIC_KEY - param value set in buf
- KEY_STATUS - param value set in val

Returns

Object_OK on success. Any other error code on failure.

14.91 IKeyManager

14.91.1 Detailed Description

14.91.2 Function Documentation

14.91.2.1 method generateKey (out IKey *key*)

Generates key based on the parameters set.
On successful completion, an IKey object will be returned to the client, which can be used to retrieve the keys. All parameters set prior to generate key are flushed out on successful completion.

Parameters

out	*key*	Key Object containing generated key.

Returns

Object_OK on success; otherwise, any other error code on failure.

14.91.2.2 method setParameter (in uint32 *param,* in buffer *buf*)

Sets a parameter for key generation. All set parameters are flushed upon successful completion of generateKey.

Parameters

in	*param*	Parameter ID to set.
in	*buf*	param value to set.

Detailed description

Supported params:

- KEY_SOURCE – param value set in val
- MODULE_SEQ_1 – param value set in buf (data a signed CRI sequence)
- MODULE_SEQ_2 – param value set in buf (data a signed CRI sequence)
- KDF_SEED – param value set in buf (data is 256 byte random seed input from client)

Returns

Object_OK on success. Any other error code on failure.

14.91.2.3 method setParameterAsObject (in uint32 *param,* in interface *obj*)

Sets the HW Key object parameter for key generation. All set parameters are flushed upon successful completion of generateKey.

Parameters

in	*param*	Parameter ID to set.
in	*obj*	HWKey object to be set.

Detailed description

Supported params:

- HW_KEY – HW Key object

Returns

Object_OK on success. Any other error code on failure.

14.91.3 Variable Documentation

14.91.3.1 const uint32 KEY_SIZE_256 = 1

SET_KEY_SIZE valuesUse key size256

14.91.3.2 const uint32 KEY_SIZE_384 = 2

Use key size384

14.92 IKVStore

14.92.1 Detailed Description

14.92.2 Function Documentation

14.92.2.1 method createNewKey (in uint8[ ] *key,* in buffer *inputBuffer,* in uint32 *flags,* out interface *IKVStoreKey*)

Creates a new key value pair and returns the KeyValueStore Object.

Parameters

in	*Key*	to be created
in	*Input*	Buffer to be stored.
in	*Flags*	it's Used for inerasable/encryption keys. bit0 indicates whether to encrypt a key. bit1 is used to indicate if the key is inerasable. For example: if we set the Flags to CRYPTO_RPMB_SERVICE|INERASABLE_RPMB_SERVICE, it means we want to create an encryprted and inerasable key.
out	*Key*	Value Store interface object.

Returns

Object_OK – Success
Object_ERROR – in other error scenarios
IKVStore_ERROR_RPMB_STORAGE_FULL – No space to create the Key
IKVStore_ERROR_INVALID_PARAMETERS – Invalid parameter passed
IKVStore_ERROR_KEY_ALREADY_EXISTS – Key already created
IKVStore_ERROR_STORAGE_CORRUPTED – Storage corrupted
IKVStore_ERROR_STORAGE_NOT_AVAILABLE – Storage not available

14.92.2.2 method getIterator (out interface *IKVStoreIterator*)

Obtains the IKVStoreIterator interface object to get details of all keys created by a client

Parameters

out	*IKVStoreIterator*	object

Returns

Object_OK – Success
Object_ERROR_MEM – Mempry Allocation Failures

14.92.2.3 method getKeyHandle (in uint8[ ] *key,* out interface *IKVStoreKey*)

Obtains the IKeyValueStore interface object for the key

Parameters

in	*key*	to be queried.
out	*Key*	Value Store interface object

Returns

Object_OK – Success
Object_ERROR – Generic error
Object_ERROR_MEM – Alloc failure
IKVStore_ERROR_INVALID_PARAMETERS – Unacceptable input parameters
IKVStore_ERROR_INIT_FAILED – Failure during rpmb device init or keystore init
IKVStore_ERROR_STORAGE_NOT_AVAILABLE – Failure during or failure to interface with storage medium
IKVStore_ERROR_STORAGE_CORRUPTED – Storage data corrupted
IKVStore_ERROR_KEY_NOT_FOUND – Requested Key not found in storage

14.92.2.4 method getSpaceInfo (out IKVStore_SpaceInfo *spaceInfo*)

Obtains space info of KVStore service

Parameters

out	*IKVStore_SpaceInfo*	Space info of KVStore service

Returns

Object_OK – Success
Object_ERROR – Generic error

14.93 IKVStoreKey

14.93.1 Detailed Description

14.93.2 Function Documentation

14.93.2.1 method delete ( )

Delete the key from the storage.

Parameters

in	*Key*	to be deleted.

Returns

Object_OK – Delete successful; Object_ERROR – Any error occurred.
Object_ERROR_MEM - Memory Allocation failure
IKVStore_ERROR_STORAGE_NOT_AVAILABLE – Failure during or failure to interface with storage medium
IKVStore_ERROR_STORAGE_CORRUPTED – Storage data corrupted
IKVStore_ERROR_DELETE_KEY_FAILED – Failure other than storage medium failures.
.

14.93.2.2 method getInfo (out IKVStoreKey_Info *keyInfo*)

Retrieve metadata associated with the key 1. Size 2. Flags used for creation

Parameters

in	*key*	value for lookup.
out	*Size*	of the data associated with key.

Returns

Object_OK – Success
Object_ERROR – Any error occurred.
IKVStore_ERROR_INVALID_PARAMETERS – Invalid parameters passed.
IKVStore_ERROR_KEY_NOT_FOUND – Key doesnt not exist for the handle passed
. IKVStore_ERROR_STORAGE_CORRUPTED – Storage corrupted
IKVStore_ERROR_GET_INFO_FAILED – Failure retrieving details of the key

14.93.2.3 method getValue (in uint32 *offset,* in uint64 *bytesToRead,* out buffer *outputBuffer*)

Retrieve the buffer stored by the key.

Parameters

in	*Off*	set from which the data should be read.
in	*Bytes*	to read
out	*Buffer*	to read the requested data in.

Returns

Object_OK – Success.
Object_ERROR – Any error occurred.
Object_ERROR_MEM – Memory Allocation failures
IKVStore_ERROR_INVALID_PARAMETERS – Invalid parameters passed.
IKVStore_ERROR_STORAGE_CORRUPTED – Storage data corrupted
IKVStore_ERROR_STORAGE_NOT_AVAILABLE – When storage is full and not available for any updates. IKVStore_ERROR_GET_VALUE_FAILED When invalid parameters are passed
.

14.93.2.4 method updateValue (in buffer *inputBuffer,* in uint32 *offset*)

Modifies data for an existing key.

Parameters

in	*input*	buffer to store in storage.
in	*offset*	specifies the offset from which the data has to be updated. For the first write the offset needs to be 0.

Returns

Object_OK – Success.
Object_ERROR – Any error occurred.
Object_ERROR_MEM – Memory Allocation failures
IKVStore_ERROR_INVALID_PARAMETERS – Invalid parameters passed.
IKVStore_ERROR_STORAGE_CORRUPTED – Storage data corrupted
IKVStore_ERROR_STORAGE_NOT_AVAILABLE – When storage is full and not available for any updates. IKVStore_ERROR_UPDATE_VALUE_FAILED – Update of key data failed.

14.94 IKVStoreIterator

14.94.1 Detailed Description

14.94.2 Function Documentation

14.94.2.1 method getNextKey (out uint8[ ] *key*)

Get next key for the client.

Parameters

out	*Key*	Name

Returns

Object_OK – Retrieving the next key was successful.
Object_ERROR – All other errors.
IKVStoreIterator_ERROR_INVALID_PARAMETERS – invalid parameters passed to API
IKVStoreIterator_ERROR_NO_KEYS – if the client hasnt created any key.
IKVStoreIterator_ERROR_END_KEY_LIST – Iterated through all keys.

14.94.3 Variable Documentation

14.94.3.1 error ERROR_END_KEY_LIST

End of the iterator list

14.94.3.2 error ERROR_INVALID_PARAMETERS

Invalid parameters passed to function

14.94.3.3 error ERROR_NO_KEYS

If the requested clients have no keys created

14.95 IKVStoreAdmin

14.95.1 Detailed Description

14.95.2 Function Documentation

14.95.2.1 method deleteKeys ( )

Delete all erasable keys from all clients who have used this service.

Returns

Object_OK – Delete successful; Object_ERROR – Any error occurred.
Object_ERROR_MEM - Memory Allocation failure
IKVStoreAdmin_ERROR_STORAGE_NOT_AVAILABLE – Failure during or failure to interface with storage medium
IKVStoreAdmin_ERROR_STORAGE_CORRUPTED – Storage data corrupted
IKVStoreAdmin_ERROR_DELETE_KEY_FAILED – Failure other than storage medium failures.
.

14.95.3 Variable Documentation

14.95.3.1 error ERROR_DELETE_KEY_FAILED

Delete Key Failed

14.95.3.2 error ERROR_STORAGE_CORRUPTED

Storage is corrupted

14.95.3.3 error ERROR_STORAGE_NOT_AVAILABLE

Storage currently not available for any operations

14.96 ILegacyHWAttestation

14.96.1 Detailed Description

14.96.2 Function Documentation

14.96.2.1 method getHardwareAttestation (in uint8 *keyId,* in uint16 *context,* in uint32 *nonce,* in uint64 *regions,* out buffer *output*)

Performs HW attestation on the supplied QFPROM region(s) based on the supplied parameters.

Parameters

in	*keyId*	Key to base the derivation on.
in	*context*	Key context.
in	*nonce*	Nonce value.
in	*regions*	QFPROM region(s) of interest. If multiple QFPROM regions' HMACs are desired, OR them and send in this variable.
out	*output*	256-bit SHA HASH MAC value.

Returns

Object_OK on success. Any other error code on failure.

14.97 ILicenseImage

14.97.1 Detailed Description

14.97.2 Function Documentation

14.97.2.1 method decryptSegment (in uint32 *segmentID,* in buffer *segmentIn,* out buffer *segmentOut,* out buffer *segmentHash*)

Decrypt segment and calculate hash if required.

Parameters

in	*segmentID*	ID of the segment in the elf image.
in	*segmentIn*	Encrypted segment.
out	*segmentOut*	Decrypted segment.
out	*segmentHash*	Hash of the segment.

Detailed description

This method decrypts encrypted segment of elf file. Decrypted segment and its hash are returned in segmentOut & segmentHash respectively.

Returns

Object_OK if successful.

14.98 ILicenseManager

14.98.1 Detailed Description

14.98.2 Function Documentation

14.98.2.1 method decryptImage (in uint32 *clientID,* in buffer *params,* out ILicense Image *image*)

Decrypt an ELF Image.

Parameters

in	*clientID*	ID of the client.
in	*params*	Header segment of the elf image to be decryted.
out	*image*	ILicenseImage

Detailed description

This method initializes Decryption context with header segment data of an elf image. Same decrypt context will be used during decryption of segments of elf file.

Returns

Object_OK if successful.

14.99 IListener

14.99.1 Detailed Description

14.99.2 Function Documentation

14.99.2.1 method getSize (in int32 *listenerId,* out uint32 *size*)

Sends request to listener get size.

Parameters

in	*listenerId*	Requested listener service ID.
out	*size*	Determine the size of the request/response buffer for a given listener.

Returns

Object_OK – Success.
Object_ERROR – Failure.

14.99.2.2 method requestService (in int32 *listenerId,* in buffer *request,* out buffer *response*)

Sends request to listener service.

Parameters

in	*listenerId*	Requested listener service ID.
in	*request*	Buffer containing request to be sent.
out	*response*	Buffer to contain received response.

Returns

Object_OK on success.

14.100 IListenerCBO

14.100.1 Detailed Description

 Interface implemented by callback object listeners.

14.100.2 Function Documentation

14.100.2.1 method request (out uint32[ ] *embeddedBufOffsets,* out uint32 *is64,* out IMemRegion *smo1,* out IMemRegion *smo2,* out IMemRegion *smo3,* out IMemRegion *smo4*)

Used to invoke the callback object of a CBO-style listener, instructing it to inspect its associated shared memory object and perform the specified service.

When used with a QComCompat TA, the listener can include embedded pointers in its response, which are represented by the accompanying shared memory objects.

Parameters

out	*embeddedBufOffsets*	An array containing offsets into the request buffer at which address information for each memory object is to be written.
out	*is64*	Flag describing if the addresses are to be treated as 64-bit addressable or 32-bit addressable
out	*smo1*	Shared memory object 1
out	*smo2*	Shared memory object 2
out	*smo3*	Shared memory object 3
out	*smo4*	Shared memory object 4

Returns

Object_OK on success.

14.100.2.2 method wait ( )

When a TA/QTEE is already accessing a listener/ CBO, the current TA/QTEE will wait till the previous TA/QTEE is done accessing the listener. This function will help the current TA/QTEE to wait in the HLOS till the previous TA/QTEE is done accesing the listener/ CBO.

Parameters

in	*None*	

Returns

Object_OK on success.

14.101 IMacchiato

14.101.1 Detailed Description

14.101.2 Function Documentation

14.101.2.1 method authenticate_device (in buffer *serviceNonce,* in buffer *deviceNonce,* in buffer *opaqueData,* out buffer *signedChallengeResponse,* out uint32 *pu32SignedChallengeResponseLen,* out uint32 *retErrCode*)

Authenticates device using challenge response with Macchiato private key.

Parameters

in	*serviceNonce*	Buffer containing authentication challenge.
in	*deviceNonce*	Buffer containing device nonce.
in	*opaqueData*	Buffer containing additional opaque data to be signed (optional).
out	*signedChallenge Response*	Buffer (also allocated by calling app) to fill the challenge response output.
out	*pu32Signed ChallengeResponse Len*	Challenge response output length (in bytes).
out	*retErrCode*	Macchiato-specific return code.

Returns

Object_OK on success.

14.101.2.2 method getPublicKeyPoint (out buffer *PublicKeyData,* out uint32 *PublicKeyDataLen,* in uint32 *PubKeyType,* out uint32 *retErrCode*)

Gets the public key point.

Parameters

out	*PublicKeyData*	Buffer to contain public signing key point.
out	*PublicKeyDataLen*	Output public key data length.
in	*PubKeyType*	Public key type to be obtained.
out	*retErrCode*	Macchiato-specific return code.

Returns

Object_OK on success.

14.101.2.3 method provision_service_key (in buffer *provisionMsg,* out buffer *unwrappedKey,* out uint32 *pu32UnwarppedKeyLen,* in buffer *deviceNonce,* out uint32 *retErrCode*)

Gets unwrapped service key from the Macchiato provisioning message.

Parameters

in	*provisionMsg*	Buffer containing provisioning message.
out	*unwrappedKey*	Buffer to contain unwrapped service key (maximum size should be known to the calling application).
out	*pu32UnwarppedKeyLen*	Unwrapped service key length.
in	*deviceNonce*	Device nonce. Passed in to compare with the nonce embedded in provision message.
out	*retErrCode*	Macchiato-specific return code.

Returns

Object_OK on success.

14.101.2.4 method signServiceData (in buffer *serviceData,* out buffer *signedServiceData,* out uint32 *pu32SignedServiceDataLen,* out uint32 *retErrCode*)

Signs service data with device Macchiato private key.

Parameters

in	*serviceData*	Buffer that contains service data to be signed.
out	*signedServiceData*	Buffer (also allocated by the calling app) to fill signature output.
out	*pu32SignedServiceDataLen*	Signature output length (in bytes).
out	*retErrCode*	Macchiato specific return code.

Returns

Object_OK on success.

14.102 IMemManager

14.103 IMemObject

Describes the interface for memory objects.

This interface is identical to the generic interface (no added methods), but allows other interfaces which reference shareable memory objects to specify that the object in question is expected to be a memory object.

14.104 IMemRegion

14.104.1 Detailed Description

14.104.2 Function Documentation

14.104.2.1 method createRestrictedRegion (in uint32 *perms,* out interface *region*)

Deprecated.

Returns

Object_ERROR_INVALID deprecated API.

14.104.2.2 method getData (in uint64 *offset,* out buffer *data*)

Reads bytes present at `offset` within the region.

Parameters

`in`	*offset*	Offset location of data to be read.
`out`	*data*	Buffer to contain data to be read.

Returns

Object_OK if successful.

14.104.2.3 method setData (in uint64 *offset,* in buffer *data*)

Writes bytes at `offset` within the region.

Parameters

`in`	*offset*	Offset position to write data bytes at.
`in`	*data*	Data buffer to be written.

Returns

Object_OK on success.

14.105 IMemRegionPermEscalator

14.105.1 Detailed Description

14.105.2 Function Documentation

14.105.2.1 method escalatePerm (in IMemRegion *inRegion,* in uint32 *newPerm,* out IMemRegion *outRegion*)

Attempts to escalates access permission for input memory region to value passed in input parameter. Authority to escalate privileges for the input memory region are granted by the IMemRegionPermEscalator object creator's privileges. On success, a new memory region is returned with the requested access permission(s). The returned object will have an access control guarantee (See IAccessControl) of "ANY NonExclusive" for the set of additional access permissions granted by the privileges associated with the TA. This guarantee lasts for the lifetime of the returned memory region.

Parameters

in	*inRegion*	Memory region to attempt to escalate privileges on.
in	*newPerms*	Access permissions requested for input region. Valid values defined in IMemRegion::PERM_... constants. Executable permission is not supported.
out	*outRegion*	New region with requested permissions.

Returns

Object_OK on success

14.106 IMemSpace

14.106.1 Detailed Description

A memory space determines mapping from virtual addresses to physical addresses, and the types of access allowed at each virtual addresses.

14.106.2 Function Documentation

14.106.2.1 method map (in IMemRegion *region,* in uint32 *perms,* out uint64 *addr,* out uint64 *size,* out interface *memmap*)

Makes a memory region visible in the address space as a contiguous range of virtual addresses.

On success, virtual addresses from `addr` to `addr+size-1` are accessible with at least the specified permissions, and remain accessible until the memmap object is released.

Parameters

`in`	*region*	Identifies memory to be made visible.
`in`	*perms*	Access permissions required for mapping, as defined in IMemRegion::PERM_... constants. If these permissions cannot be satisfied, the operation fails.
`out`	*addr*	Virtual address of the mapped region.
`out`	*size*	Mapped region size.
`out`	*memmap*	An object that acts as a reference count on mapping. When deleted, the map is removed from the address space.

Returns

Object_OK on success.

14.106.2.2 method mapPartial (in IMemRegion *region,* in uint64 *offset,* in uint64 *size,* in uint32 *perms,* out uint64 *addr,* out interface *memmap*)

Makes a part of memory region visible in the address space as a contiguous range of virtual addresses.

On success, virtual addresses from `addr` to `addr+size-1` are accessible with at least the specified permissions, and remain accessible until the memmap object is released.

Parameters

`in`	*region*	Identifies memory to be made visible.
`in`	*offset*	Specifies offset (from beginning of region) to map Offset should be 4K aligned.
`in`	*size*	Specifies size to map. size should be non-zero and 4K aligned.
`in`	*perms*	Access permissions required for mapping, as defined in IMemRegion::PERM_... constants. If these permissions cannot be satisfied, the operation fails.
`out`	*addr*	Virtual address of the mapped region.

out	*memmap*	An object that acts as a reference count on mapping. When deleted, the map is removed from the address space.

Returns

Object_OK on success.

14.107 IModule

14.107.1 Detailed Description

14.107.2 Function Documentation

14.107.2.1 method open (in uint32 *id,* in interface *credentials,* out interface *obj*)

Requests an instance object.

Parameters

in	*id*	Class ID.
in	*credentials*	An object describing the credentials used to request the object.
out	*obj*	An instance of the requested class.

Returns

Object_OK if successful.

14.107.2.2 method shutdown ( )

Shuts down an instance object.

Returns

Object_OK if successful.

14.108 INistLoggingFramework

14.108.1 Detailed Description

INistLoggingFramework provides an interface to collect logs from NistLogging partition.

14.108.2 Function Documentation

14.108.2.1 method erasePartition ( )

This method will erase out the entire NIST partition.

Returns

Object_OK if successful.

14.108.2.2 method getSize (in uint32 *sizeType,* out uint32 *validRecCount,* out uint32 *size*)

This method gets the size of NIST partition corresponding to the provided argument. It will return either the entire size or the size occupied by the valid logs. Number of valid records logged in partition will be updated in validRecCount. validRecCount and size will be updated in success case only.

Parameters

in	*sizeType*	Size to read out • 1 = Size of entire NIST partition • 2 = Size of partition having valid records
out	*validRecCount*	Number of valid records in the partition
out	*size*	Size of nist log

Returns

Object_OK if successful.

Note

This API can be called first to determine how much space is require to read out the current set of records

14.108.2.3 method readRecords (out buffer *buf,* in uint32 *recordOffset,* in uint32 *maxReadCount,* out uint32 *recordRead*)

This method reads records from NIST partition into the buffer provided by the client. The records are read out in sequential order, starting with the oldest record. recordOffset can be used to start the read at a location other than the oldest record, and maxReadCount can be used to limit how many records are read. If record count provided by client is 0 this function will read up to the size of buffer provided by the client. If it's greater than zero API will read up to the specificed number of records, or until the buffer is full.

Parameters

out	*buf*	Buffer to hold the data read
in	*recordOffset*	Number of records to skip before reading out data. 0 will start from the oldest record.
in	*maxReadCount*	Indicates the maximum number of records to be read into the buffer. A value of 0 will read out as many records as will fit into the buffer
out	*recordRead*	No of valid records read out to the provided buffer

Returns

Object_OK if successful.

14.109 INotifyHdcp

14.109.1 Detailed Description

INotifyHdcp provides functions that report HDCP authentication level changes from OPS to HDCP Tx TAs.

14.109.2 Function Documentation

14.109.2.1 method HdcpNotifyError (in int32 *hdcpError*)

Notifies HDCP Authentication level errors.

Parameters

in	*hdcpError*	HDCP Error.

Returns

Object_OK on success.

14.110 INSMem

14.110.1 Detailed Description

14.110.2 Function Documentation

14.110.2.1 method countMemUsage (in uint32 *tag,* out uint32 *usageInPages*)

Counts the number of pages tagged as secure display memory.

Parameters

in	*tag*	Domain with access to protected memory region.
out	*usageInPages*	Number of pages.

Returns

Object_OK if the entire area is in secure memory; otherwise, INSMem_INVALID.

14.110.2.2 method isNSRange (in uint64 *start,* in uint64 *size*)

Checks whether the specified memory range is completely in non-secure memory.

Parameters

in	*start*	Starting address of the region. Zero is a valid value.
in	*size*	Number of bytes in region.

Returns

Object_OK if the entire area is in secure memory; otherwise, INSMem_INVALID.

14.110.2.3 method isSecureTaggedRange (in uint32 *tag,* in uint64 *start,* in uint64 *size*)

Checks whether a memory range is completely within the secure memory tagged for a use case.

Parameters

in	*tag*	Domain that has access to the protected memory region.
in	*start*	Start of memory area (included in the range).
in	*size*	Number of bytes in memory area.

Returns

Object_OK if the entire area is in secure memory; otherwise, INSMem_INVALID.

14.110.2.4 method tagMem (in uint32 *tag,* in uint64 *start,* in uint64 *end*)

Tags memory provided for a specific use-case.

Parameters

in	*tag*	Domain which should have access to the protected memory region.
in	*start*	Start of memory area (included in the range).
in	*end*	End of memory area (excluded in the range).

Returns

Object_OK if the entire area successfully tagged for the use case; otherwise, INSMem_INVALID.

14.111 INSSystemReg

14.111.1 Detailed Description

14.111.2 Function Documentation

14.111.2.1 method GetNSSystemReg (in uint32 *SysReg,* out uint64 *val*)

Retrieves value of requested NS System Register.

Parameters

in	*SysReg*	Describes the System Register that should be retrieved.
out	*val*	System Register value.

Returns

Object_OK on success.

14.112 IOpener

14.112.1 Detailed Description

IOpener requests services from the system.

14.112.2 Function Documentation

14.112.2.1 method open (in uint32 *id,* out interface *obj*)

Gets a service object from the system.

Parameters

in	*id*	Identifies a class of service object.
out	*obj*	Requested service instance.

Returns

Object_OK if successful.
ERROR_NOT_FOUND if a service matching the ID cannot be found.
ERROR_PRIVILEGE if required privileges are not present.
ERROR_NOT_SUPPORTED when the service is not supported (stubbed out) or when unmet dependencies are found at runtime.

14.113 IOPS

14.113.1 Detailed Description

14.113.2 Function Documentation

14.113.2.1 method getContentProtectionLevel (out uint8 *cpl,* out uint8 *epl,* out uint8 *type*)

Gets current DRM content protection level enforced by OPS.

Parameters

out	*cpl*	Content protection level = max(all DRM OPLs).
out	*epl*	Final protection level enforced by OPS.
out	*type*	Type 0 or 1. Type enforced by OPS.

Returns

Object_OK on success.

14.113.2.2 method getDeviceTopology (in uint32 *deviceId,* out uint8[ ] *receiverIdList,* out HdcpTopology *hdcpTopology*)

- Aggregated topology with received ID list for a particular device.
- Used by DRM TAs like WFD RX to find the downstream topology for each device type.

Parameters

in	*deviceId*	Topology requested for a particular device.
out	*receiverIdList*	Downstream device ID list.
out	*hdcpTopology*	Provides receiver ID list and information on connected downstream legacy devices.

Returns

Object_OK on success.

14.113.2.3 method getHDCPCapability (out uint8 *currHdcpLevel,* out uint8 *maxHdcp Level*)

Gets current and maximum supported HDCP levels.

Parameters

out	*currHdcpLevel*	Current HDCP version based on the device itself and the display to which it is connected.
out	*maxHdcpLevel*	Maximum supported HDCP version for device, ignoring any attached device.

Returns

Object_OK on success.

14.113.2.4 method getOPSVersion (out uint32 *version*)

Gets current OPS implementation version.

Parameters

out	*version*	Version number.

Returns

Object_OK on success.

14.114 IOPSSink

14.114.1 Detailed Description

IOPSSink provides an interface to create a HdcpTransmitter object.

14.114.2 Function Documentation

14.114.2.1 method getHdcpTransmitter (in uint32 *deviceId,* in INotifyHdcp *notify,* out IHdcpTransmitter *hdcpTx*)

Creates an object implementing IHdcpTransmitter for a specific device type.

Parameters

in	*deviceId*	Device type can be HDMI, DP, or WFD listed in IOPS.idl.
in	*notify*	Object implementing INotifyHdcp; can be invoked by OPS to notify HDCP errors.
out	*hdcpTx*	HDCP Transmitter object.

Returns

Object_OK on success.

14.115 IOPSSource

14.115.1 Detailed Description

IOPSSource Interface provides APIs to enforce output protection for DRM protected content.

14.115.2 Function Documentation

14.115.2.1 method applyOPL (in uint8 *opl,* in uint16 *srmVersion,* out uint8 *cpl,* out uint8 *dpl,* out uint8 *epl*)

- Sets content's encryption level, as required by the license.
- DRM TAs invoke this function to enforce required HDCP level.

Parameters

in	*opl*	Output Protection Level to be enforced by Source or DRM.
in	*srmVersion*	HDCP System Renewability Message version.
out	*cpl*	Content protection level = max(all source OPLs).
out	*dpl*	Display protection level = min(all sink OPLs).
out	*epl*	Final protection level enforced by OPS.

Returns

Object_OK on success.

14.116 IPeripheralAccessControl

14.117 IPeripheralState

14.118 IPeripheralStateCB

14.119 IPFM

14.119.1 Detailed Description

14.119.2 Function Documentation

14.119.2.1 method ActivateGracePeriod (in buffer *LicenseSerialNumber*)

Activate a grace period for the given license.

Note

ISV / APK clients can only activate grace periods for licenses that were installed by the client, or are bound to the client's ISV/Licensee Hash.

Parameters

in	*LicenseSerial Number*	- Serial Number of the license to activate

Returns

Object_OK on success
ERROR_FILE_NOT_FOUND No license with the given serial number was found
ERROR_NOT_GRACE_BOUND The license does not have a grace period extension
ERROR_CERT_EXPIRED No more grace periods are available
ERROR_PRIVILEGE_ERR ISV client is not the owner of the license
ERROR_NO_ACTIVATION_NEEDED if the license has not expired

14.119.2.2 method BeginUpdate (out IPFMUpdater *Updater*)

Begins an atomic transaction and copies the credentials.

Parameters

out	*Returns*	a new object of type IPFMUpdater.

Returns

Object_OK on success

14.119.2.3 method CheckFeatureIds (in buffer *RequestCBOR,* out buffer *ResponseCBOR*)

Check for licensing of multiple features.

Given a CBOR list of Feature IDs / licensee hash pairs, return a CBOR list of structures of license information. For each entry in the request list, there will be a corresponding entry at the same position in the response list.

CDDL for RequestCBOR

```
RequestCBOR = {
```

```
    blobVersion : uint32,
    FeatureIDs  : [
        *[uint32, BSTR]
    ]
}
```

RequestCBOR Example

```
RequestCBOR = {
    "blobVersion" : 2,
    "FeatureIDs" : [
        [1, 12fc331f2c],
        [2, cadfae1900],
        [3, 1502005678]
    ]
}
```

ResponseCBOR Format for version 1

```
ResponseCBOR = [
    *[uint32, BSTR, BSTR, uint32, BSTR, uint64_t]
]

ResponseCBOR = [
    *[FeatureID, LicenseeHash, SerialNumber, ResponseCode, ExtraData, Flag]
]
```

ResponseCBOR Format for version 2

```
ResponseCBOR = [
    *[uint32, BSTR, BSTR, uint32, BSTR, uint64_t, uint64_t, BSTR, BSTR]
]

ResponseCBOR = [
    *[FeatureID, LicenseeHash, SerialNumber, ResponseCode, ExtraData, Flag, IssueDate,
      SFEHandle, GPBeforeTTimeAvailable]
]
```

CBOR value descriptions

- FeatureID - Feature ID from the request
- LicenseeHash - Licensee Hash from the request
- SerialNumber - Serial Number of a license enabling the feature for the licensee. Zero-length BSTR if no matching license was found.
- ResponseCode - 0 for allowed, 1 for not allowed, 3 for processing error. Normally this value will be 0 if any licenses are listed in Result, and 1 if Result is empty. If a failure occurs while processing a potentially matching license, ResponseCode will be 3. If no licenses are in the store, ResponseCode will be 4.
- ExtraData - Version 1 onwards. Extra Data extension from the license. Zero-length BSTR if no matching license was found.
- Flag - Version 1 onwards. OPT_NO_TIME_CHECK if license validation ignored expiration. 0 otherwise.
- IssueDate - Version 2 onwards. License creation date & time details. If IssueDate extension is not present in the license then it's value will be zero.
- SFEHandle - Version 2 onwards. Valid only for SFE feature. If SFE extension is not present in the license then it's value will be Zero-length BSTR.

- GPBeforeTTimeAvailable - Version 2 onwards. It having GracePeriod and GraceCount from GPBeforeTTimeAvailable extension as a BSTR. If GPBeforeTTimeAvailable extension is not present then it's value will be Zero-length BSTR.

ResponseCBOR example for version 1

```
ResponseCBOR = [
    [800, "", 22df4e681bc159922749d591219bc3ec, 0, "", 0],
    [900, "", "", 4, "", 0]
]
```

ResponseCBOR example for version 2

```
ResponseCBOR = [
    [800, "", 22df4e681bc159922749d591219bc3ec, 0, "", 0, 51b07516, A301185F0250461FF490E8ACA4BC
    9DD745425AED470DDFA700E88B812D5FEA50A6BBCA7987AE51, A26B4772616365506572696F64183C6A47726163
    [900, "", 413235710e56a9dd91a2b98210fe0c2a, 0, "", 0, 62E1D3E2, A301185F0250413235710E56A9DD
    8C2957851F63DC8598607728035E7F5FB657B02DF8D271D62A, A26B4772616365506572696F6418786A47726163
]
```

Note

The ResponseCBOR format for version 0 is the same, except that ExtraData, Flag, Issue Date, SFEHandle and GPBeforeTTimeAvailable are not present.

Parameters

in	*RequestCBOR*	- CBOR formatted list of FeatureIDs and LicenseeHashes
out	*ResponseCBOR*	- Gives out all the serial numbers in CBOR format and the error codes correspong to the feature ID

Returns

Object_OK on success
Object_ERROR_SIZE_OUT
ERROR_CBOR_ENCODE_ERR
ERROR_CBOR_DECODE_ERR
ERROR_CBOR_DECODE_DATATYPE_ERR

14.119.2.4 method CheckFIDAndGetAllSerialNums (in buffer *RequestCBOR,* out buffer *ResponseCBOR*)

Get all license information for a list of features.

For reach FID / Licensee pair in the request, the response will contain an entry that reports information for all of the licenses that provide the given feature for the given licensee.

CDDL for RequestCBOR

```
RequestCBOR = {
    blobVersion : uint32,
    FeatureIDs  : [
        *[uint32, BSTR]
        ]
}
```

RequestCBOR Example

```
RequestCBOR = {
    "blobVersion" : 2,
    "FeatureIDs" : [
        [1,12fc331f2c],
        [2,cadfae1900],
        [3,1502005678]
    ]
}
```

CDDL for ResponseCBOR for version 1

Note

Version 0 is the same, except ExtraData and Flag are not present

```
ResponseCBOR = [
    *{
        "FID": uint32
        "LicenseeHash": BSTR
        "Result": [
          * {
                "SerialNum": BSTR
                "LicenseRestrictions": uint64
                "ExtraData": BSTR
            }
        ]
        "ResponseCode": uint32
        "Flag": uint64
    }
]
```

```
ResponseCBOR = [
    {
        "FID" : 100
        "LicenseeHash" : ""
        "Result" : [
            {
                "SerialNum" : 23456,
                "LicenseRestrictions" : 1F,
                "ExtraData" : ""
            },
            {
                "SerialNum" : 234534,
                "LicenseRestrictions" : 0C,
                "ExtraData" : "Hello"
            }
        ]
        "ResponseCode" : 0
        "Flag" : 8
    },

    {
        "FID" : 200
        "LicenseeHash" : 234568
        "Result" : [
            {
                "SerialNum" : 23454,
                "LicenseRestrictions" : 03,
                "ExtraData" : ""
```

```
            },
            {
                "SerialNum" : 234123
                "LicenseRestrictions" : 02,
                "ExtraData" : ""
            }
        ]
        "ResponseCode" : 0
        "Flag" : 8
    }
]
```

CDDL for ResponseCBOR for version 2

Note

Version 1 is the same, except IssueDate, SFEHandle and GPBeforeTTimeAvailable are not present

```
    ResponseCBOR = [
        *{
            "FID": uint32
            "LicenseeHash": BSTR
            "Result": [
              * {
                    "SerialNum": BSTR
                    "LicenseRestrictions": uint64
                    "ExtraData": BSTR
                    "IssueDate": uint64_t
                    "SFEHandle": BSTR
                    "GPBeforeTTimeAvailable": {
                                                "GracePeriod": uint64_t,
                                                "GraceCount": uint32_t
                                            }
                }
            ]
            "ResponseCode": uint32
            "Flag": uint64
        }
    ]
ResponseCBOR = [
    {
        "FID" : 100
        "LicenseeHash" : ""
        "Result" : [
            {
                "SerialNum" : 234567,
                "LicenseRestrictions" : 1F,
                "ExtraData" : "",
                "IssueDate" : 51b07516,
                "SFEHandle" : A301185F0250461FF490E8ACA4BC7132EED9439894BA0358205033B84E870622
                              9DD745425AED470DDFA700E88B812D5FEA50A6BBCA7987AE51,
                "GPBeforeTTimeAvailable": {
                                            "GracePeriod": 60,
                                            "GraceCount": 10
                                          }
            },
            {
                "SerialNum" : 234534,
                "LicenseRestrictions" : 0C,
```

```
                "ExtraData" : "Hello",
                "IssueDate" : 59b6c4c5,
                "SFEHandle" : A301185F0250461FF490E8ACA4BC7132EED9439894BA0358205033B84E870622
                              9DD745425AED470DDFA700E88B812D5FEA50A6BBCA7987AE51,
                "GPBeforeTTimeAvailable": {
                                              "GracePeriod": 120,
                                              "GraceCount": 5
                                          }
            }
        ]
        "ResponseCode" : 0
        "Flag" : 8
    }
]
```

CBOR value descriptions

- FID - Feature ID from the request
- LicenseeHash - Licensee Hash from the request
- Result - CBOR Array of license information, one entry per matching license.
- SerialNum - Serial Number of a license enabling the feature for the licensee.
- LicenseRestrictions - Bitmask of license Restrictions. Some combination of LICENSE_HAS_XXX values.
- ResponseCode - 0 for allowed, 1 for not allowed, 3 for processing error. Normally this value will be 0 if any licenses are listed in Result, and 1 if Result is empty. If a failure occurs while processing a potentially matching license, ResponseCode will be 3. If no licenses are in the store, ResponseCode will be 4.
- ExtraData - Version 1 onwards. Extra Data extension from the license. May be empty.
- Flag - OPT_NO_TIME_CHECK if license validation ignored expiration. 0 otherwise.
- IssueDate - Version 2 onwards. License creation date & time details. If IssueDate extension is not present in the license then it's value will be zero.
- SFEHandle - Version 2 onwards. Valid only for SFE feature. If SFE extension is not present in the license then it's value will be Zero-length BSTR.
- GraceCount - Version 2 onwards. GraceCount from GPBeforeTTimeAvailable extension. If GPBeforeTTimeAvailable extension is not present then GraceCount will not present in the Response.
- GracePeriod - Version 2 onwards. GracePeriod Duration from GPBeforeTTimeAvailable extension. If GPBeforeTTimeAvailable extension is not present then GracePeriod will not present in the Response.

Parameters

in	*RequestCBOR*	- CBOR list of FeatureID and LicenseeHash pairs
out	*ResponseCBOR*	- CBOR list as described above.

Returns

Object_OK on success

14.119.2.5 method CheckFIDAndGetAllWithGrace (in buffer *RequestCBOR,* out buffer *ResponseCBOR*)

Get all license information, including activatable licenses, for a list of features.

For reach FID / Licensee pair in the request, the response will contain an entry that reports information for all of the licenses that provide the given feature for the given licensee.

CDDL for RequestCBOR

```
RequestCBOR = {
    blobVersion : uint32,
    FeatureIDs  : [
        *[uint32, BSTR]
        ]
}
```

RequestCBOR Example

```
RequestCBOR = {
    "blobVersion" : 1,
    "FeatureIDs" : [
        [1,12fc331f2c],
        [2,cadfae1900],
        [3,1502005678]
    ]
}
```

CDDL for ResponseCBOR for version 1

Note

Version 0 is the same, except ExtraData and Flag are not present

```
ResponseCBOR = [
    *{
        "FID": uint32
        "LicenseeHash": BSTR
        "Result": [
          * {
                "SerialNum": BSTR
                "LicenseRestrictions": uint64
                "ResponseCode": uint32
                "ExtraData": BSTR
            }
        ]
        "Flag": uint64
    }
]
ResponseCBOR = [
    {
        "FID" : 100
        "LicenseeHash" : ""
        "Result" : [
            {
                "SerialNum" : 23456,
                "LicenseRestrictions" : 1F,
                "ResponseCode" : 0
                "ExtraData" : ""
```

```
            },
            {
                "SerialNum" : 234534,
                "LicenseRestrictions" : 0C,
                "ResponseCode" : 5
                "ExtraData" : "Hello"
            }
        ]
        "Flag" : 8
    },

    {
        "FID" : 200
        "LicenseeHash" : 234568
        "Result" : [
            {
                "SerialNum" : 23454,
                "LicenseRestrictions" : 03,
                "ResponseCode" : 0
                "ExtraData" : ""
            },
            {
                "SerialNum" : 234123
                "LicenseRestrictions" : 02,
                "ResponseCode" : 3
                "ExtraData" : ""
            }
        ]
        "Flag" : 8
    }
]
```

CDDL for ResponseCBOR for version 2

Note

Version 1 is the same, except IssueDate is not present

```
    ResponseCBOR = [
        *{
            "FID": uint32
            "LicenseeHash": BSTR
            "Result": [
              * {
                    "SerialNum": BSTR
                    "LicenseRestrictions": uint64
                    "ExtraData": BSTR
                    "IssueDate": uint64_t
                }
            ]
            "ResponseCode": uint32
            "Flag": uint64
        }
]
ResponseCBOR = [
    {
        "FID" : 100
        "LicenseeHash" : ""
        "Result" : [
```

```
            {
                "SerialNum" : 23456,
                "LicenseRestrictions" : 1F,
                "ExtraData" : "",
                "IssueDate" : 51b07516
            },
            {
                "SerialNum" : 234534,
                "LicenseRestrictions" : 0C,
                "ExtraData" : "Hello",
                "IssueDate" : 59b6c4c5
            }
        ]
        "ResponseCode" : 0
        "Flag" : 8
    }
]
```

CBOR value descriptions

- FID - Feature ID from the request
- LicenseeHash - Licensee Hash from the request
- Result - CBOR Array of license information, one entry per matching license.
- SerialNum - Serial Number of a license enabling the feature for the licensee.
- LicenseRestrictions - Bitmask of license Restrictions. Some combination of LICENSE_HAS_XXX values.
- Response - 0 for allowed, 3 for processing error, and 5 if grace activation is required. Normally this value will be 0. If the value is 5 for the license that the client wants to use, the client code must successfully call ActivateGracePeriod before enabling the feature.
- ExtraData - Version 1 only. Extra Data extension from the license. May be empty.
- Flag - OPT_NO_TIME_CHECK if license validation ignored expiration. 0 otherwise.
- IssueDate - Version 2 onwards. License creation date & time details.

Parameters

in	*RequestCBOR*	- CBOR list of FeatureID and LicenseeHash pairs
out	*ResponseCBOR*	- CBOR list as described above.

Returns

Object_OK on success

14.119.2.6 method CheckInstalledLicense (in uint32 *FeatureID,* in buffer *Licensee Hash,* out buffer *LicenseSerialNumber*)

Find a license that currently supports the given feature ID for the given licensee. If no managed licenses exist for the given licensee and feature ID, or validation of the licenses fails, ERROR_FILE_NOT_FOUND will be returned. If no managed licenses exist, ERROR_NO_LICENSES will be returned.

Parameters

in	*FeatureID*	- Feature ID to check
in	*LicenseeHash*	- ISV Hash of the client requesting the feature. This is only needed when the feature is in a license with a licensee hash extension.
out	*LicenseSerial Number*	- Serial Number of the license on success

Returns

Object_OK on success
Object_ERROR_SIZE_OUT
ERROR_FILE_NOT_FOUND
ERROR_NO_LICENSES

14.119.2.7 method CheckLicenseBuffer (in buffer *PFMLicense,* in uint32 *FeatureID,* in buffer *LicenseeHash,* out buffer *LicenseBufferCBOR*)

Check whether the given license provides the given feature to the given licensee.

CDDL Format for LicenseInfoCBOR

```
LicenseInfoCBOR = {
    "SerialNumber" : BSTR
    "Issuer" : CBOR Map
    "SubjectKeyID" : BSTR
    "SubjectDN" : CBOR Map
    "Expiration" : BSTR
    "LicenseeHashes" : BSTR
    "FIDs" : [*uint32]
    "FIDRanges" : [*[uint32, uint32]]
    "LicenseRestrictions" : uint64
    "ExtraData" : BSTR
    "Flag" : uint64_t
    "IssueDate" : uint64_t
    "SFEHandle" : BSTR
}
```

LicenseInfoCBOR Example

```
LicenseInfoCBOR = {
    "SerialNumber" : 22df4e681bc159922749d591219bc3ec
    "Issuer" : { "OU": "Issuer Org Unit", "O" : "Issuer Organization" }
    "SubjectKeyID" : 878e26b6d2b292e10ecd0cc085454118682c734f
    "SubjectDN" : { "OU": "Subject Org Unit", "CN": "Subject Common Name" }
    "Expiration" : 01000000012345678334455 66
    "LicenseeHashes" : ""
    "FIDs" : [1,22,43]
    "FIDRanges" : [[5,8], [10,15]]
    "LicenseRestrictions" : 1
    "Flag" : 8
    "IssueDate" : 51b07516
    "SFEHandle" : A301185F0250461FF490E8ACA4BC7132EED9439894BA0358205033B84E870622
                  9DD745425AED470DDFA700E88B812D5FEA50A6BBCA7987AE51
}
```

CBOR Value Descriptions

- SerialNumber - Should match LicenseSerialNumber
- Issuer - Distinguished Name of the Issuer. Refer to the description of Distinguished Names above.
- SubjectKeyID - Subject-specific Public Key Identifier. Normally 20 bytes.
- SubjectDN - Distinguished Name of the Subject. Refer to the description of Distinguished Names above.
- Expiration - Valid time range of the license. 12 bytes for time-bound licenses.
- LicenseeHashes - Accepted Client ISV Hashes for this license. 32-bit count followed by concatenated hashes.
- FIDs - FIDs provided by the license
- FIDRanges - FID Ranges provided by the license
- LicenseRestrictions - Bitmask of license Restrictions. Some combination of LICENSE_HAS_XXX values.
- ExtraData - Extra Data extension from the license, or Zero-length BSTR if the license does not have the extension.
- Flag - OPT_NO_TIME_CHECK if license validation ignored expiration. 0 otherwise.
- IssueDate - License creation date & time details. If IssueDate extension is not present in the license then it's value will be zero.
- SFEHandle - Valid only for SFE feature. If SFE extension is not present in the license then it's value will be Zero-length BSTR.

Parameters

`in`	*PFMLicense*	- Feature license in PFM or DER format
`in`	*FeatureID*	- The feature ID to check
`in`	*LicenseeHash*	- ISV Hash of the client requesting the feature. This is only needed when the license has a licensee hash extension.
`out`	*LicenseBufferCBOR*	

Returns

Object_OK on success
Object_ERROR_SIZE_OUT
Common Validation Error Codes
ERROR_LICENSE_TOO_BIG

14.119.2.8 method CheckLicenseBuffer2 (in buffer *License,* in buffer *RequestCBOR,* out buffer *ResponseCBOR*)

Fetch license information for the given License buffer. The information is encoded in a CBOR structure described below.

Only supported IPFM_VERSION is '1'.

CDDL Format for LicenseInfoCBOR

```
    RequestCBOR = {
        IPFM_VERSION : uint32
        ?IPFM_FEATUREIDS : [+uint32]
        ?IPFM_FEATURE_ENCODING_TYPE : [+int64]
        ?IPFM_NONCE : bstr
        ?IPFM_LICENSEE_HASH : bstr
    }
TimeInfoMap = {
    IPFM_QOT : QoTType
    ?IPFM_CURRENTTIME : uint64
}

QoTType = &(QOT_UNTRUSTED,
            QOT_LOW_TRUST,
            QOT_MEDIUM_TRUST,
            QOT_HIGH_TRUST)

QOT_UNTRUSTED = 0
QOT_LOW_TRUST = 1
QOT_MEDIUM_TRUST = 2
QOT_HIGH_TRUST = 3
FeatureInfoMap = {
    IPFM_FEATUREID : uint32
    IPFM_FEATURESTATUS : FeatureStatusType
    IPFM_TIMEBOUND : bool
    ?IPFM_VALIDAFTER : uint64
    ?IPFM_VALIDUNTIL : uint64
    ?IPFM_GRACEUNTIL : uint64
    ?IPFM_FEATUREVALUE : FeatureValueType
    ?IPFM_METADATA : bstr
    IPFM_BINDINGS : uint64
    IPFM_ANTIREPLAYCOUNTER : uint64
}

FeatureStatusType = &(FEATURE_STATUS_ACTIVE,
                      FEATURE_STATUS_NOTACTIVE,
                      FEATURE_STATUS_NOTPRESENT,
                      FEATURE_STATUS_DISABLED)

FEATURE_STATUS_ACTIVE = 0
FEATURE_STATUS_NOTACTIVE = 1
FEATURE_STATUS_NOTPRESENT = 2
FEATURE_STATUS_DISABLED = 3
```

ResponseCBOR

```
    ResponseCBOR = {
        IPFM_VERSION : uint32
        ?IPFM_NONCE : bstr
        ?IPFM_LICENSEEHASH : bstr
        IPFM_TIMEINFO : TimeInfoMap
        IPFM_FEATUREINFO : [+FeatureInfoMap]
        IPFM_SERIALNUMBER : bstr
    }
```

14.119.2.9 method CheckSecured (in buffer *SecuredRequest,* out buffer *Secured Response*)

Securely check feature IDs.

This API is similar to CheckFeatureIds except that it does not support the LicenseeHash option. Most client applications will not use this API. It is currently used by clients in the modem to securely check features via QMI. This requires an apps/HLOS QMI service to relay the request and response to this API.

See CheckFeatureIds for request and response details.

Parameters

in	*SecuredRequest*	- Encrypted CBOR request.
out	*SecuredResponse*	- Encrypted CBOR response

Returns

Object_OK on success
Object_ERROR_SIZE_OUT
ERROR_CBOR_ENCODE_ERR
ERROR_CBOR_DECODE_ERR
ERROR_CBOR_DECODE_DATATYPE_ERR
ERROR_BLOB_ENCAP_FAILED
ERROR_BLOB_DECAP_FAILED
ERROR_OVERFLOW
ERROR_LICENSE_TOO_BIG

14.119.2.10 method CreateProvisioning (out IProvisioning *Provisioning*)

Create an IProvisioning object.

Parameters

out	*Provisioning*	- IProvisioning object

Returns

Object_OK on success

14.119.2.11 method EnforceHWFeatures (in uint32[ ] *featureIDs,* out uint32 *HW RegisterInterfaceVersion,* out EnforceHWFeatureId_t[ ] *HWFeatureStatus*)

Enforce HW features based on license configuration.

Returns

Object_OK on success

Parameters

in	*featureIDs*	- Array of featureIDs
out	*HWRegisterInterface Version*	- HW register interface version (applicable only for TME Lite).
out	*HWFeatureStatus*	- HW feature status for provided featureIDs

14.119.2.12 method GeneratePFMReport (in buffer *license,* in uint32 *options,* out IAttestationBuilder *attestation_builder*)

Generating PFM report for attestation service. Currently including only installed license serials.

Parameters

in	*license*	- buffer containing token_cert license.
in	*options*	- options to control what info need to add in PFM report.
out	*attestationBuilder*	- IAttestationBuilder object to construct an attestation.

Returns

Object_OK on success
ERROR_CBOR_DECODE_ERR if unable to decode CBOR response of GetAllInstalledSerialNumbers
ERROR_CBOR_ENCODE_ERR if unable to encode PFM report.

14.119.2.13 method GetAllInstalledFeatureIDs (out buffer *FeatureIDsCBOR*)

Get list of installed feature IDs.

This API examines and verifies every managed license, and returns an aggregate list of currently supported featured IDs.

CDDL for FeatureIDsCBOR

```
FeatureIDsCBOR = {
    FIDs: [*uint32],
    FIDRanges: [*[uint32, uint32]]
}
```

FeatureIDsCBOR Example

```
FeatureIDsCBOR = {
    "FIDs" : [1,22,43],
    "FIDRanges" : [[10,20],[30,40]]
}
```

CBOR Value Descriptions

- FIDs - FIDs provided by currently valid licenses
- FIDRanges - FID Ranges provided by currently valid licenses

Parameters

out	*FeatureIDsCBOR*	- CBOR formatted data out for FIDs and FIDRanges

Returns

Object_OK on success
Object_ERROR_SIZE_OUT
ERROR_NO_LICENSES

14.119.2.14 method GetAllInstalledSerialNumbers (out buffer *SerialNumbersCBOR*)

Get the serial numbers for all installed licenses.

This method iterates over all the licenses in the license store and gives the caller a list of serial numbers, one for each license. The client can use this information to report the list of licenses present, to get the details of the licenses, to know which license to remove, etc...

CDDL for SerialNumbersCBOR

```
SerialNumbersCBOR = [
    *BSTR
]
```

SerialNumbersCBOR Example

```
SerialNumbersCBOR = [
    1234561,
    3af4e0d8,
    567456
]
```

Parameters

out	*SerialNumbersCBOR*	- List of installed serial numbers

Returns

Object_OK on success
ERROR_NO_LICENSES

14.119.2.15 method GetAppCapabilities (out uint64 *caps*)

Get QWES Trused appplication (TA) capabilities. It will return the capabilities of TA like whether TA supports Atomic License operations etc.

Parameters

out	*caps*	- Returns bitmaks that will correspond to each capability like Atomic license operation etc.

Returns

Object_OK on success

14.119.2.16 method GetFeatureConfig (out uint64 *fuses*)

Get the status of fuses used to request licenses.

On success, fuses is set to a bitmask indicating which sync-request fuses have been blown.

Parameters

out	*fuses*	- Bitmask indicating fuse status. Zero if no fuses blown.

Returns

Object_OK on success
ERROR_NOMEM

14.119.2.17 method GetInstalledLicenseInfo (in buffer *LicenseSerialNumber,* out buffer *LicenseInfoCBOR*)

Fetch license information for the given License Serial Number. The information is encoded in a CBOR structure described below.

CDDL Format for LicenseInfoCBOR

```
LicenseInfoCBOR = {
    "FileName" : string
    "SerialNumber" : BSTR
    "Issuer" : CBOR Map
    "SubjectKeyID" : BSTR
    "SubjectDN" : CBOR Map
    "Expiration" : BSTR
    "LicenseeHashes" : BSTR
    "FIDs" : [*uint32]
    "FIDRanges" : [*[uint32, uint32]]
    "LicenseRestrictions" : uint64
    "ExtraData" : BSTR
    "Flag" : uint64_t
    "IssueDate" : uint64_t
}
```

LicenseInfoCBOR Example

```
LicenseInfoCBOR = {
    "FileName" : ""
    "SerialNumber" : 22df4e681bc159922749d591219bc3ec
    "Issuer" : { "OU": "Issuer Org Unit", "O" : "Issuer Organization" }
    "SubjectKeyID" : 878e26b6d2b292e10ecd0cc085454118682c734f
    "SubjectDN" : { "OU": "Subject Org Unit", "CN": "Subject Common Name" }
    "Expiration" : 010000001234567833445566
    "LicenseeHashes" : ""
    "FIDs" : [1,22,43]
    "FIDRanges" : [[5,8], [10,15]]
    "LicenseRestrictions" : 1
    "ExtraData" : 81828384
    "Flag" : 8
    "IssueDate" : 51b07516
}
```

CBOR value descriptions

- FileName - Empty string, No longer used.
- SerialNumber - Should match LicenseSerialNumber
- Issuer - Distinguished Name of the Issuer. Refer to the description of Distinguished Names above.
- SubjectKeyID - Subject-specific Public Key Identifier. Normally 20 bytes.
- SubjectDN - Distinguished Name of the Subject. Refer to the description of Distinguished Names above.
- Expiration - Valid time range of the license. 12 bytes for time-bound licenses.
- LicenseeHashes - Accepted Client ISV Hashes for this license. 32-bit count followed by concatenated hashes.
- FIDs - FIDs provided by the license
- FIDRanges - FID Ranges provided by the license
- LicenseRestrictions - Bitmask of license Restrictions. Some combination of LICENSE_HAS_XXX values.
- Flag - OPT_NO_TIME_CHECK if license validation ignored expiration. 0 otherwise.
- IssueDate - License creation date & time details.

Parameters

`in`	*LicenseSerial Number*	Serial number to search for
`out`	*LicenseInfoCBOR*	CBOR structure as described above

Returns

Object_OK on success
Object_ERROR_SIZE_OUT
Common Validation Error Codes
ERROR_FILE_NOT_FOUND
ERROR_NO_LICENSES
ERROR_PFMFILER_GETFILECONTENTS_FAILED

14.119.2.18 method GetLicenseCertPFM (in buffer *LicenseSerialNumber,* out buffer *PFMLicense*)

Return an installed license in PFM format.

Parameters

`in`	*LicenseSerial Number*	- Serial Number of the license
`out`	*PFMLicense*	- License Info in PFM format

Returns

Object_OK on success
ERROR_FILE_NOT_FOUND
ERROR_NO_LICENSES
ERROR_PFMFILER_GETFILECONTENTS_FAILED

14.119.2.19 method GetNextExpiration (out uint64 *expiration*)

Get the expiration time for the next expiring license(s)

Parameters

out	*expiration*	- Epoch time when the next license(s) will expire

Returns

Object_OK on success
ERROR_INVALID_CURRENT_TIME
ERROR_NO_LICENSES

14.119.2.20 method InstallLicense (in buffer *PFMLicense,* out buffer *LicenseSerial Number,* out buffer *FeatureIDsCBOR,* out uint64 *LicenseRestrictions*)

Install a new license.

Installation will fail if device-based license restrictions do not match the device. Installation ignores licensee ID, feature ID, and expiration restrictions. Output buffers will only be set on success.

CDDL for FeatureIDsCBOR

```
FeatureIDsCBOR = {
    FIDs: [*uint32],
    FIDRanges: [*[uint32, uint32]]
}
```

FeatureIDsCBOR Format

```
FeatureIDsCBOR = {
    "FIDs" : [1,22,43],
    "FIDRanges" : [[10,20],[30,40]]
}
```

CBOR Value Descriptions

- FIDs - FIDs provided by the license
- FIDRanges - FID Ranges provided by the license

Parameters

in	*PFMLicense*	- Feature license in PFM or DER format
out	*LicenseSerial Number*	- Serial number of the license

out	*FeatureIDsCBOR*	- Features provided by the license
out	*LicenseRestrictions*	- Bitmask of license restrictions. Some combination of LICENSE_HAS_XXX values.

Returns

Object_OK on success
Common Validation Error Codes
ERROR_CBOR_ENCODE_ERR
ERROR_LICENSE_TOO_BIG
ERROR_LICENSE_STORE_FULL
ERROR_DUPLICATE
Object_ERROR_SIZE_OUT

14.119.2.21 method RegisterCallback (in IPFMCallback *Callback,* in buffer *FidsCBOR,* out buffer *ResponseCBOR,* out interface *CallbackHandle*)

Registers a callback with fid for smart notifications.

Returns

Object_OK on success

Parameters

in	*callback*	- An object of type IPFMCallback to get notified of changes to the status of FIDs.
in	*fidsCBOR*	- A list of FIDs encoded as CBOR. Callee will be notified if the status of these FIDs change.
out	*responseCBOR*	- An output CBOR buffer that contains event type and reponseCode for each affected FIDs.
out	*callbackHandle*	- An opaque object that must be held by the callee for the lifetime of the callback. When done with the callback, the callee must call Object_release on this object.

14.119.2.22 method RemoveLicense (in buffer *LicenseSerialNumber*)

Remove an installed license.

Parameters

in	*LicenseSerial Number*	- Serial number of the license to remove.

Returns

Object_OK on success
ERROR_FILE_NOT_FOUND
ERROR_NO_LICENSES

ERROR_PFMFILER_GETFILECONTENTS_FAILED
ERROR_PRIVILEGE_ERR ISV client is not the owner of the license

14.119.2.23 method RemoveLicenseExpired ( )

Remove expired licenses.

This API removes any installed licenses that have an expiration date before the current date. Licenses will only be removed if the trust level of the current date is MEDIUM.

Note

If this method is called by an ISV/APK client, it will only remove licenses that were installed by the client.

Returns

Object_OK on success
ERROR_INVALID_CURRENT_TIME

14.119.2.24 method SetOptions (in uint64 *uOpts*)

Disable license validation checks.

Note

This API is for internal testing only

Parameters

in	*uOpts*	Option bitmask

Returns

Object_OK on success
ERROR_OPTS_NOT_SUPPORTED

14.119.2.25 method SetTrustedTime (in uint64 *time,* in uint32 *trust_level*)

Update internal time with the latest trusted time value.

Note

This method can only be called by trusted clients.

Returns

Object_OK on success
ERROR_OPTS_NOT_SUPPORTED - Client is not allowed to set trusted time.

14.119.3 Variable Documentation

14.119.3.1 const uint64 CAPS_ATOMIC_LICENSE_OP = 0x00000001

TA supports Atomic license operations

14.119.3.2 error ERROR_ACTIVATION_NOT_NEEDED

License is currently active so grace period activation is not needed

14.119.3.3 error ERROR_BLOB_DECAP_FAILED

Decryption of encrypted request failed.

14.119.3.4 error ERROR_BLOB_ENCAP_FAILED

Generation of encrypted response failed.

14.119.3.5 error ERROR_CALLBACK_STORE_FULL

No space is available to store a new callback

14.119.3.6 error ERROR_CBOR_DECODE_DATATYPE_ERR

Data type is incorrect in client CBOR input parameter.

14.119.3.7 error ERROR_CBOR_DECODE_ERR

Client CBOR input parameter is invalid.

14.119.3.8 error ERROR_CBOR_ENCODE_ERR

Internal error.

14.119.3.9 error ERROR_CERT_DEVICEID

The device's Device ID does not match the license.

14.119.3.10 error ERROR_CERT_EXPIRED

The license certificate is expired.

14.119.3.11 error ERROR_CERT_FEATUREID

The requested Feature ID does not match the license.

14.119.3.12 error ERROR_CERT_GENERAL_ERR

Error while verifying the cert chain.

14.119.3.13 error ERROR_CERT_HWVERSION

The device's CHIP ID (chip family) does not match the license.

14.119.3.14 error ERROR_CERT_LEAF_IS_CA

Leaf certificate in the license is marked as a CA.

14.119.3.15 error ERROR_CERT_LICENSEE_HASH

The requested Licensee Hash does not match the license.

14.119.3.16 error ERROR_CERT_NOT_TRUSTED

The license is not signed with a recognized root CA.

14.119.3.17 error ERROR_CERT_NOTYETVALID

The license certificate is not yet valid.

14.119.3.18 error ERROR_CERT_OEM

The device's OEM ID does not match the license.

14.119.3.19 error ERROR_CERT_PKHASH

The device's PKHash does not match the license.

14.119.3.20 error ERROR_CERT_PRODUCTID

The device's Product/Model ID does not match the license.

14.119.3.21 error ERROR_DUPLICATE

The license can not be installed because it has the same serial number as a managed license.

14.119.3.22 error ERROR_FILE_NOT_FOUND

No managed license matches the client request.

14.119.3.23 error ERROR_FLUSH

Error writing data to RPMB

14.119.3.24 error ERROR_GPBTTA_GRACE_EXPIRED

Error due to Allowed Grace Period have been expired

14.119.3.25 error ERROR_GRACE_COUNT_EXHAUSTED

Error due to Allowed Boot Count for Grace have been Exhauted

14.119.3.26 error ERROR_HASH_GENERATION

The signing hash used by the license has the wrong length or the hash operation failed.

14.119.3.27 error ERROR_INCORRECT_LICENSEE_HASH

Licensee field did not match client, and client is an APK

14.119.3.28 error ERROR_INVALID_CERT

License cannot be parsed or is corrupt.

14.119.3.29 error ERROR_INVALID_CURRENT_TIME

Time-based operation can not be completed because current time is unknown, or trust level is too low

14.119.3.30 error ERROR_INVALID_PARAM

Error due to passed incorrect input/output params.

Errors

14.119.3.31 error ERROR_INVALID_TRANSACTION_COUNT

Revocation License has invalid TransactionID count to perform license revocation

14.119.3.32 error ERROR_LICENSE_ALREADY_REVOKED

License which is requested to be revoked is already revoked

14.119.3.33 error ERROR_LICENSE_STORE_FULL

No space is available to install a license.

14.119.3.34 error ERROR_LICENSE_TOO_BIG

License data is too large.

14.119.3.35 error ERROR_NO_FEATURE_CONFIG

Feature Configs are not defined for this chipset

14.119.3.36 error ERROR_NO_LICENSES

Get or Check failed because no licenses are available.

14.119.3.37 error ERROR_NOMEM

Memory allocation failed. Retry may succeed.

14.119.3.38 error ERROR_NOT_GRACE_BOUND

License does not have a grace period extension.

14.119.3.39 error ERROR_OPTS_NOT_SUPPORTED

SetOptions is not allowed for the current client.

14.119.3.40 error ERROR_OVERFLOW

Request would cause an overflow in an internal calculation.

14.119.3.41 error ERROR_PFM_REPORT_REVOCATION_LICENSE

Error in adding revocation/revoked licenses in PFM Submod report

14.119.3.42 error ERROR_PFM_SUBMOD_REVOCATION_SERIAL

Error in adding revocation and/or revoked license to PFM Submod report

14.119.3.43 error ERROR_PFMFILER_FAILED

License data store is unavailable. Retry later.

14.119.3.44 error ERROR_PFMFILER_GETFILECONTENTS_FAILED

Managed license exists but could not be read.

14.119.3.45 error ERROR_PRIVILEGE_ERR

ISV client is not the owner of the license.

14.119.3.46 error ERROR_QWESSTORE_MEMORY_FULL

Not able to store the key because of QWESStore memory is full.

14.119.3.47 error ERROR_REV_LIC_ALREADY_PROCESSED

License with the revocation details is already processed

14.119.3.48 error ERROR_REVOCATION_FID_UNAVAILABLE

License with Revocation extension requires revocaiton FID

14.119.3.49 error ERROR_REVOCATION_LIST

Error while accessing revocation list

14.119.3.50 error ERROR_REVOCATION_LIST_FULL

Revocation List data is full

14.119.3.51 error ERROR_REVOCATION_UNSUPPORTED

Device does not either support RPMB or it isn't properly provisioned/configured

14.119.3.52 error ERROR_REVOKE_LICENSE

Failed to update revocation data

14.119.3.53 error ERROR_UNSUPPORTED_REVOCATION_FID

This happens if Revocation License file contains FeatureIDs that are not related to revocation feature

14.119.3.54 const uint64 EVENT_TYPE_STATE_CHANGE = 0x00000000

Event change such as install, transaction, expiry, and remove.

14.119.3.55 const uint64 FEATURE_STATUS_ACTIVE = 0x00

Feature Status

14.119.3.56 const uint64 FEATURE_STATUS_ALLOWED = 0x00000000

The feature is allowed.

14.119.3.57 const uint64 FEATURE_STATUS_GRACE_ACTIVATION_NEEDED = 0x00000005

Grace activation is required.

14.119.3.58 const uint64 FEATURE_STATUS_NO_LICENSES = 0x00000004

No licenses are in the store.

14.119.3.59 const uint64 FEATURE_STATUS_NOT_ALLOWED = 0x00000001

The feature is not allowed.

14.119.3.60 const uint64 FEATURE_STATUS_PROCESSING_ERROR = 0x00000003

Failure occurs while processing a potentially matching license.

14.119.3.61 const uint64 LICENSE_HAS_DEVICEID = 0x00000004

Device ID must match license

14.119.3.62 const uint64 LICENSE_HAS_GRACE_PERIOD = 0x00000080

VALIDTIME may be affected by grace period

14.119.3.63 const uint64 LICENSE_HAS_HWVERSION = 0x00000002

Hardware Version must match license

14.119.3.64 const uint64 LICENSE_HAS_LICENSEEHASH = 0x00000010

client ISV Hash must match license

14.119.3.65 const uint64 LICENSE_HAS_OEMID = 0x00000001

OEM ID must match license

14.119.3.66 const uint64 LICENSE_HAS_PKHASH = 0x00000008

Device PKHash must match license

14.119.3.67 const uint64 LICENSE_HAS_PRODUCTID = 0x00000040

Product ID must match license

14.119.3.68 const uint64 LICENSE_HAS_VALIDTIME = 0x00000020

Current time must match license

14.119.3.69 const uint64 QOT_UNTRUSTED = 0x00

QoT

14.119.3.70 const uint64 TRANSACTION_STATUS_FAIL = 0x00000002

Transaction fail.

14.119.3.71 const uint64 TRANSACTION_STATUS_INCOMPLETE = 0x00000001

Transaction is incomplete.

14.119.3.72 const uint64 TRANSACTION_STATUS_SUCCESS = 0x00000000

Transaction success.

14.119.3.73 const uint64 VERSION = 1

CheckLicenseBuffer2 labels

14.120 IPmPon

14.120.1 Detailed Description

14.120.2 Function Documentation

14.120.2.1 method ponResetCfg (in uint32 *ponResetSource,* in uint32 *ponResetTypeCfg,* in uint32 *s1TimerMs,* in uint32 *s2TimerMs*)

Configures the PowerON (PON) reset source, reset time, s1 timer, and s2 timer for all applicable PMICs.

Parameters

in	*ponResetSource*	Reset source type
in	*ponResetTypeCfg*	PS_HOLD reset configure type
in	*s1TimerMs*	S1 timer in ms for PMIC bark
in	*s2TimerMs*	S2 timer in ms for PMIC bite

Returns

Object_OK – Success.
ERROR_RESET_CFG_INVALID_PARAM – Invalid parameter(s) encountered.
ERROR_RESET_CFG_ILLEGAL_WARM_RESET – Rejected warm reset cfg because debug policy disallows dumps.
Object_ERROR – Any other error encountered.

14.121 IPrivacyPreservingID

14.121.1 Detailed Description

14.121.2 Function Documentation

14.121.2.1 method getClientDeviceID (in interface *credentials,* in buffer *salt,* out buffer *id*)

Get a 32-byte privacy preserving chipset ID
This method can only be invoked by QTI signed TAs The ID will be computed as:
SHA256(ChipID || SerialNumber || salt || Credentials)

- ChipID is common to all devices in the chipset family
- Salt is supplied by the client
- SerialNumber is unique to the device
- Credentials will be parsed from the caller and computed as follows:
  - If the client is a Trusted App
    - Credentials = DistID
  - If the client is an Android vendor daemon
    - Credentials = SHA256(vmid || uid)
  - If the client is an APK
    - Credentials = SHA256(vmid || PackageCert)

Parameters

in	*credentials*	Credentials for the client using this API
in	*salt*	Salt provided by client
out	*id*	32-byte client specific device ID

Returns

Object_OK on success

14.121.2.2 method getDeviceID (in buffer *salt,* out buffer *id*)

Get a 32-byte privacy preserving chipset ID
The ID will be computed as:
SHA256(ChipID || SerialNumber || salt || Credentials)

- ChipID is common to all devices in the chipset family
- Salt is supplied by the client
- SerialNumber is unique to the device
- Credentials will be parsed from the caller and computed as follows:
  - If the client is a Trusted App
    - Credentials = DistID
  - If the client is an Android vendor daemon
    - Credentials = SHA256(vmid || uid)
  - If the client is an APK
    - Credentials = SHA256(vmid || PackageCert)

Parameters

`in`	*salt*	Salt provided by client
`out`	*id*	32-byte client specific device ID

Returns

Object_OK on success
ERROR_FEATURE_NOT_LICENSED if this feature is not enabled in PFM license
ERROR_LICENSE_UNAVAILABLE if PFM license cannot be accessed

14.122 IProperty

14.122.1 Detailed Description

14.122.2 Function Documentation

14.122.2.1 method getProperty (in buffer *name,* out buffer *dest*)

Gets the value of the named property.

Parameters

in	*name*	Property name.
out	*dest*	Requested property value.

Returns

Object_OK on success.

14.122.2.2 method setProperty (in buffer *name,* in buffer *src*)

Sets the value of the named property.

Parameters

in	*name*	Property name.
in	*src*	Value to be assigned to the named property.

Returns

Object_OK on success.

14.123 IProvError

14.123.1 Detailed Description

14.123.2 Variable Documentation

14.123.2.1 error INVALID_ARGUMENT

Invalid arguments.

14.123.2.2 error INVALID_CLOUD_CERTIFICATE

Invalid cloud certificate.

14.123.2.3 error INVALID_DA_CERTIFICATE

Invalid attestation certificate.

14.123.2.4 error INVALID_PARAMS_CBOR

Invalid CBOR parameters.

14.123.2.5 error KEY_LIST_FULL

Key list is full.

14.123.2.6 error NO_MEMORY

Failure during memory allocation.

14.123.2.7 error NOT_SUPPORTED

Not supported.

14.124 IProvisioning

14.124.1 Detailed Description

14.124.2 Function Documentation

14.124.2.1 method beginCipher (in buffer *keyBlob,* in buffer *inParams,* out buffer *outParams,* out ICipherOperation *cipherOperation*)

Creates a cipher operation for encryption or decryption only.

Parameters

in	*keyBlob*	Buffer containing the key material.
in	*inParams*	Reserved for future use.
out	*outParams*	Reserved for future use.
out	*cipherOperation*	ICipherOperation object to encrypt or decrypt data.

Returns

Object_OK on success.

14.124.2.2 method createKeyList (out IProvKeyList *keyList*)

Creates a key list and returns a handle to it to add or clear key blobs.

Parameters

out	*keyList*	IProvKeyList object to add the keys into the key list.

Returns

Object_OK on success.

14.124.2.3 method deriveCEK (in uint32 *cekType,* in buffer *keyBlob,* in buffer *inParams,* in buffer *cloudPublicKey,* out buffer *outParams,* out buffer *cekBlob*)

Derives a CEK using the keyBlob (Device's key) and cloudPublicKey (Cloud's key).

Parameters

in	*cekType*	cek algorithm type.
in	*keyBlob*	Buffer containing the input key material (Device's key).
in	*inParams*	CBOR encoded parameters such as AES mode & key size.
in	*cloudPublicKey*	Buffer containing the cloud key material (Cloud's key).
out	*outParams*	Reserved for future use.
out	*cekBlob*	Output buffer containing derived cek.

Returns

Object_OK on success.

14.124.2.4 method exportKey (in buffer *keyBlob,* in int32 *keyFormat,* out buffer *keyData*)

Exports public key

Parameters

in	*keyBlob*	Buffer that contains the key material.
in	*keyFormat*	The key format in which the public key should be exported.
out	*keyData*	Buffer containing the bytes of the exported public key.

Returns

Object_OK on success.

14.124.2.5 method generateAttestation (in buffer *licenseCert,* in buffer *keyBlob,* in buffer *serviceCtx,* out IAttestationBuilder *builder*)

Generates an attestation report for provisioning purpose.

Parameters

in	*licenseCert*	Buffer containing license to use attestation feature.
in	*keyBlob*	Buffer containing key material.
in	*serviceCtx*	Bytes that make sense to the 3rd party cloud and is passed along by the QWES cloud back to the 3rd party cloud.
out	*builder*	IAttestationBuilder object to add user data.

Returns

Object_OK on success.

14.124.2.6 method generateAttestationWithKeyList (in buffer *licenseCert,* in IProvKeyList *keyList,* out IAttestationBuilder *builder*)

Generates a report attesting to multiple keys for provisioning purpose.

Parameters

in	*licenseCert*	Buffer containing license to use attestation feature.
in	*keyList*	IProvKeyList object that contains the keys to be attested to.
out	*builder*	IAttestationBuilder object to add user data.

Returns

Object_OK on success.

14.124.2.7 method generateKey (in uint32 *keyType,* in buffer *keyParams,* out buffer *keyBlob*)

Generates a key pair based on input algorithm and key parameters.

Parameters

in	*keyType*	Either RSA or ECC.
in	*keyParams*	CBOR encoded parameters such as keyType, purpose, keysize, padding, etc.
out	*keyBlob*	Buffer containing key material which is bound to caller and device.

Returns

Object_OK on success.

14.124.2.8 method importKey (in uint32 *keyType,* in buffer *keyParams,* in buffer *keyData,* out buffer *keyBlob*)

Imports an existing key for IProvisioning use

Parameters

in	*keyType*	Type of key to import.
in	*keyParams*	CBOR-encoded parameters such as key purpose, key size, block mode, etc.
in	*keyData*	Buffer containing the bytes of the key to import.
out	*keyBlob*	Buffer containing key material which is bound to caller and device.

Returns

Object_OK on success.

14.124.2.9 method signAsym (in buffer *keyBlob,* in buffer *tbs,* out buffer *signature*)

Signs the given input using the key identified by keyBlob.

Parameters

in	*keyBlob*	Buffer containing the key material.
in	*tbs*	To be signed data.
out	*signature*	Signature data.

Returns

Object_OK on success.

14.124.2.10 method verifySignature (in buffer *keyBlob,* in buffer *tbs,* in buffer *signature*)

Verifies the given signature using the key identified by keyBlob.

Parameters

in	*keyBlob*	Buffer containing the key material.
in	*tbs*	Signed data.
in	*signature*	Signature to verify.

Returns

Object_OK on success.

14.124.3 Variable Documentation

14.124.3.1 const uint32 AES128_KEY_SIZE = 128

Key size 128.

14.124.3.2 const uint32 AES192_KEY_SIZE = 192

Key size 192.

14.124.3.3 const uint32 AES256_KEY_SIZE = 256

Key size 256.

14.124.3.4 const uint32 BLOCK_MODE_GCM = 32

GCM mode.

14.124.3.5 const uint32 CEK_TYPE_AES = 0

AES key.

14.124.3.6 const uint32 DIGEST_SHA_2_256 = 4

SHA256.

14.124.3.7 const uint32 DIGEST_SHA_2_384 = 5

SHA384.

14.124.3.8 const uint32 DIGEST_SHA_2_512 = 6

SHA512.

14.124.3.9 const uint32 EC_CURVE_P_256 = 1

p256.

14.124.3.10 const uint32 EC_CURVE_P_384 = 2

p384.

14.124.3.11 const uint32 EC_CURVE_P_521 = 3

p521.

14.124.3.12 const uint32 KEY_PURPOSE_DECRYPT = 1

Decrypt purpose.

14.124.3.13 const uint32 KEY_PURPOSE_DERIVE_KEY = 4

Derive key purpose.

14.124.3.14 const uint32 KEY_PURPOSE_ENCRYPT = 0

Encrypt purpose.

14.124.3.15 const uint32 KEY_PURPOSE_SIGN = 2

Sign purpose.

14.124.3.16 const uint32 KEY_PURPOSE_VERIFY = 3

Verify purpose.

14.124.3.17 const uint32 KEY_TYPE_AES = 2

AES key.

14.124.3.18 const uint32 KEY_TYPE_ECC = 1

ECC key.

14.124.3.19 const uint32 KEY_TYPE_RSA = 0

RSA key.

14.124.3.20 const uint32 Label_Caller_Data = 7

CBOR label, Caller Data.

14.124.3.21 const uint32 Label_Caller_Params = 6

CBOR label, Caller Parameters.

14.124.3.22 const uint32 Label_Hash_Algorithm = 31

CBOR label, Hash Algorithm.

14.124.3.23 const uint32 Label_MGF_Hash_Algorithm = 32

CBOR label, MFG Hash Algorithm.

14.124.3.24 const uint32 Label_PublicKey_Algorithm = 1

CBOR label, PublicKey Algorithm.

14.124.3.25 const uint32 Label_PublicKey_Content = 2

CBOR label, PublicKey Content.

14.124.3.26 const uint32 Label_RSA_Modulus = 21

CBOR label, RSA Modulus.

14.124.3.27 const uint32 Label_RSA_PublicExponent = 22

CBOR label, RSA Exponent.

14.124.3.28 const uint32 Label_RSAES_OAEP_Params = 3

CBOR label, RSA-AES OAEP Params.

14.124.3.29 const uint32 PAD_RSA_OAEP = 1

OAEP padding.

14.124.3.30 const uint32 PAD_RSA_PSS = 2

PSS padding.

14.124.3.31 const uint32 RSA2048_KEY_SIZE = 2048

Key size 2048.

14.124.3.32 const uint32 RSA4096_KEY_SIZE = 4096

Key size 4096.

14.124.3.33 const uint32 RSA65537_PUB_EXP = 65537

Public exponent 65537.

14.124.3.34 const uint32 TAG_ASSOCIATED_DATA = 13

CBOR tag, aad.

14.124.3.35 const uint32 TAG_BLOCK_MODE = 7

CBOR tag, block mode.

14.124.3.36 const uint32 TAG_CALLER_NONCE = 11

CBOR tag, caller nonce.

14.124.3.37 const uint32 TAG_CONTEXT = 15

CBOR tag, context.

14.124.3.38 const uint32 TAG_DIGEST = 4

CBOR tag, digest.

14.124.3.39 const uint32 TAG_EC_CURVE = 8

CBOR tag, ecc curve.

14.124.3.40 const uint32 TAG_KEY_PURPOSE = 1

CBOR tag, key purpose.

14.124.3.41 const uint32 TAG_KEY_SIZE = 2

CBOR tag, key size.

14.124.3.42 const uint32 TAG_MAC = 14

CBOR tag, authentication tag.

14.124.3.43 const uint32 TAG_NONCE = 9

CBOR tag, nonce.

14.124.3.44 const uint32 TAG_PADDING = 3

CBOR tag, padding.

14.124.3.45 const uint32 TAG_RSA_MGF1_DIGEST = 12

CBOR tag, digest.

14.124.3.46 const uint32 TAG_RSA_PUBLIC_EXPONENT = 5

CBOR tag, public exponent.

14.124.3.47 const uint32 TAG_SALT = 16

CBOR tag, salt.

14.124.3.48 const uint32 Value_Algorithm_RSAES_OAEP = 1

CBOR Value, Algorithm RSAES_OAEP.

14.124.3.49 const uint32 Value_Digest_SHA1 = 1

CBOR Value, Digest SHA1.

14.124.3.50 const uint32 Value_Digest_SHA224 = 2

CBOR Value, Digest SHA224.

14.124.3.51 const uint32 Value_Digest_SHA256 = 3

CBOR Value, Digest SHA256.

14.124.3.52 const uint32 Value_Digest_SHA384 = 4

CBOR Value, Digest SHA384.

14.124.3.53 const uint32 Value_Digest_SHA512 = 5

CBOR Value, Digest SHA512.

14.125 IPVCLicense

14.125.1 Detailed Description

14.125.2 Function Documentation

14.125.2.1 method setHavenLicense (in buffer *LicenseCert,* out int32 *havenError*)

Set and validate Haven license certificate for the PVC feature.

Parameters

in	*LicenseCert*	Buffer containing the certificate in DER format.
out	*havenError*	Result of license validation.

Detailed description

This method needs to be called at least once after the device was rebooted to enable the PVC feature. Access to the PVC feature without calling this method first, or if the provided certificate is not valid, will fail.

In order for PVC to be enabled, the function needs to return Object_OK, and havenError needs to be Object_OK. This function will return Object_OK if it succeeded in contacting the Haven License service, in which case the result of the certificate validation is returned in havenError.

Returns

Object_OK if the license was evaluated by the Haven License service.

14.126 IQTEEEnvInfo

14.126.1 Detailed Description

14.126.2 Function Documentation

14.126.2.1 method getSupportedArchitectures (out uint8 *architecturesSupported*)

Get the architectures supported in usermode.

Parameters

out	*architectures Supported*	8-bit integer value showing the supported architectures.

Returns

Object_OK if successful.

14.127 IQWESKeyStore

14.127.1 Detailed Description

14.127.2 Function Documentation

14.127.2.1 method clearValue (in buffer *keyAlias*)

Clears the data or cert that is saved and associated with the key identified by keyAlias.

Parameters

in	*keyAlias*	Alias name for the input key.

Returns

Object_OK on success.

14.127.2.2 method loadKey (in buffer *keyAlias,* out buffer *keyValue*)

Loads key blob identified by keyAlias from secure storage.

Parameters

in	*keyAlias*	Alias name for the input key.
out	*keyValue*	Buffer containing the key material.

Returns

Object_OK on success.

14.127.2.3 method loadValue (in buffer *keyAlias,* out buffer *valueBytes*)

Loads type and value of the data or certificate saved and associated with key identified by keyAlias.

Parameters

in	*keyAlias*	Alias name for the input key.
out	*valueBytes*	Buffer containing either key blob or x509 certificate.

Returns

Object_OK on success.

14.127.2.4 method removeKey (in buffer *keyAlias*)

Removes the key and any cert associated with the key identified by keyAlias.

Parameters

in	*keyAlias*	Alias name for the input key.

Returns

Object_OK on success.

14.127.2.5 method saveKey (in buffer *keyBlob,* in buffer *keyAlias*)

Saves a key in secure storage (SFS).

Parameters

in	*keyBlob*	Buffer containing the key material.
in	*keyAlias*	Alias name for the input key.

Returns

Object_OK on success.

14.127.2.6 method saveValue (in buffer *keyAlias,* in buffer *valueBytes*)

Save either a key blob or x509 certificate given by valueBytes and associates them with the key identified by keyAlias.

Parameters

in	*keyAlias*	Alias name for the input key.
in	*valueBytes*	Buffer containing either key blob or x509 certificate.

Returns

Object_OK on success.

14.127.3 Variable Documentation

14.127.3.1 error ERROR_DUPLICATE_ALIAS

Alias passed to saveKey is already used.

14.127.3.2 error ERROR_INVALID_ARGUMENT

Invalid arguments.

14.127.3.3 error ERROR_INVALID_KEY_BLOB

Invalid key blob.

14.127.3.4 error ERROR_KEY_STORE_FULL

Available/allowed key storage capacity is reached.

14.127.3.5 error ERROR_NO_MEMORY

Failure during memory allocation.

14.128 IQWESTAServices

14.128.1 Detailed Description

14.128.2 Function Documentation

14.128.2.1 method createServiceById (in buffer *license,* in uint32 *serviceId,* out interface *service*)

Returns service interface

Parameters

in	*license*	Buffer containing license to check feature IDs.
in	*serviceId*	QWES TA service ID
out	*service*	Interface to get service from QWES TA.

Returns

Object_OK on success.

14.129 IRegisterListenerCBO

14.129.1 Detailed Description

 Interface used by REE clients to register callback object listeners into QTEE.

14.129.2 Function Documentation

14.129.2.1 method register (in uint32 *listenerId,* in IListenerCBO *cbo,* in interface *memRegion*)

Register a CBO-style listener with QTEE.

Each CBO listener can be associated with a shared memory object. Each CBO implements IListenerCBO .

Parameters

in	*listenerId*	The listener id being registered.
in	*cbo*	The callback object associated with this listener.
in	*memRegion*	The shared memory object associated with this listener. Can be Object_NULL.

Returns

Object_OK on success.

14.130 IRTICDtb

14.130.1 Detailed Description

14.130.2 Function Documentation

14.130.2.1 method getDtb (out uint64 *KernelPhysBaseAddr,* out uint64 *DTBAddress,* out buffer *rtic_dtb*)

Retrieve the contents of the DTB that is sent by APPS BL.

Parameters

out	*KernelPhysBaseAddr*	Kernel physical base address.
out	*DTBAddress*	Dtb struct address.
out	*rtic_dtb*	Buffer containing DTB sent by APPS BL.

Returns

Object_OK on success.

14.131 IRTICReport

14.131.1 Detailed Description

14.131.2 Function Documentation

14.131.2.1 method getMPHash (out uint8[ ] *MPHash*)

Retrieves SHA256 hash of the MP Catalog.

Parameters

out	*MPHash*	SHA256 hash of MP Data.

Returns

Object_OK on success.

14.131.2.2 method getReport (out IRTICReport_EI2ActorReportIDL *actor_report,* out buffer *asset_name,* out IRTICReport_EI2ActorDataIDL[ ] *actor_data,* out buffer *actor_names*)

Retrieves contents of Report generated by EL2.

Parameters

out	*actor_report*	Instance of Actor Report generated by EL2.
out	*asset_name*	Asset name affected by EL2 violation.
out	*actor_data*	Array of Actor Data generated by EL2.
out	*actor_names*	Array of Actor names generated by EL2.

Returns

Object_OK on success.

14.131.2.3 method getReport2 (out buffer *el2_report_buff*)

Retrieves contents of Report generated by EL2.

Parameters

out	*el2_report_buff*	EL2 report buffer.

Returns

Object_OK on success.

14.131.2.4 method getReportHeader (out IRTICReport_reportHeader *reportheader*)

Retrieves contents of Report Header generated by EL2.

Parameters

out	*reportheader*	Output Report Header Structure.

Returns

Object_OK on success.

14.131.2.5 method getReportPFCounters (out uint32[ ] *PFCounters*)

Retrieve Page Fault Counters from the report.

Parameters

out	*PFCounters*	Array of Page Fault Counters indexed by PF_COUNTER_∗.

Returns

Object_OK on success.

14.131.2.6 method SetGetAssetHash (in uint32 *AssetID,* in buffer *InHash,* out buffer *hash*)

Retrieves SHA256 hash of the MP Assets.

Parameters

in	*AssetID*	ID of asset for which Hash to be fetched.
in	*start_addr*	Start Address of Asset.
in	*len*	size/length of Asset.
out	*Hash*	SHA256 Hash of Asset calculated on Asset with addr and len.
out	*out_len*	length of output buffer.

Returns

Object_OK on success.

14.131.3 Variable Documentation

14.131.3.1 error ERROR_INVALID_PARAM

Specifying Errors returned from this interface.

14.131.3.2 const uint32 TEXTASSETID = 0

Specifies Asset ID to calculate hash.

14.132 IRuntimeAttestation

14.132.1 Detailed Description

IRuntimeAttestation provides an interface to generate attestation report which can be initiated by OEM using an HLOS app. This report should contain data which indicates whether any corruption was found in critical data partitions. This report will contain label, GUID and hashes of all critical data partitions(primary and back up) in SPINOR. Also the report will contain boot time corruption log from NIST Log framework. This hash can be used by the OEM to compare with an original hash they possess. If the hashes do not match, it indicates that the partition is corrupted.

14.132.2 Function Documentation

14.132.2.1 method getPartitionHashData (out buffer *data*)

This method is designed to create a buffer from an array of hashes representing all critical data partitions, and it appends boot time corruption log from NIST Log framework at the end of the buffer. The purpose of this buffer is to facilitate the generation of reports by the OEMs. By default, the buffer is formatted in CBor. However, the method also allows the customer to modify the format according to their specific needs, providing flexibility in how the data and logs are presented for report generation.

Parameters

out	*data*	The data of the report

Returns

Object_OK if successful.
Object_ERROR_SIZE_OUT if output length is less than required length.
Object_ERROR or other error codes on failure.

14.132.2.2 method getPartitionHashSize (out uint32 *size*)

This method is responsible for calculating the size of the report that is created from an array of hashes representing all critical data partitions, with an attached NIST log at the end. The function returns the size of the report, which is crucial for memory allocation. OEMs should call this method prior to generating the buffer, ensuring that they allocate sufficient memory to accommodate the entire report.

Parameters

out	*size*	The total size of the report

Returns

Object_OK if successful.
Object_ERROR or other error codes on failure.

14.132.3 Variable Documentation

14.132.3.1 error ERROR_CLOSE_PARTITION_FAILURE

Returned when unable to close partition.

14.132.3.2 error ERROR_END_STORAGE_ITERATOR_FAILURE

Returned when failed to close partition iterator.

14.132.3.3 error ERROR_ERASE_NISTLOG_FAILURE

Returned when unable to erase nist log from nistlog partition.

14.132.3.4 error ERROR_FAILURE

Deprecated and should no longer be used.

14.132.3.5 error ERROR_GPT_ENTRY_UNAVAILABLE

Deprecated and should no longer be used.

14.132.3.6 error ERROR_INSUFICIENT_MEMORY

Deprecated and should no longer be used.

14.132.3.7 error ERROR_INVALID_PARAM

Deprecated and should no longer be used.

14.132.3.8 error ERROR_OPEN_PARTITION_FAILURE

Returned when unable to open partition.

14.132.3.9 error ERROR_PARTITION_HASH_FAILURE

Returned when failed to calculate partition hash.

14.132.3.10 error ERROR_PARTITION_NOT_FOUND

Returned when no primary partition found.

14.132.3.11 error ERROR_QCBOR_ENCODE_FAILURE

Returned when failed to encode partition hash info in qcbor format.

14.132.3.12 error ERROR_READ_NISTLOG_FAILURE

Returned when unable to read nist log from nistlog framework.

14.132.3.13 error ERROR_READ_PARTITION_FAILURE

Returned when unable to read partition.

14.132.3.14 error ERROR_RELEASE_STORAGE_OBJECT_FAILURE

Deprecated and should no longer be used.

14.132.3.15 error ERROR_STORAGE_ITERATOR_UNAVAILABLE

Deprecated and should no longer be used.

14.133 ISecureCamera

14.133.1 Detailed Description

14.133.2 Function Documentation

14.133.2.1 method acquireCamera (in uint32 *sessionID*)

- Blocks camera from being released out of secure mode.
- Must be called by the client to ensure the secure session cannot be terminated by REE or other TAs.

Parameters

in	*sessionID*	Current secure camera session ID.

Returns

Object_OK – Success.
ISecureCamera_ERROR_LICENSE_IS_INVALID – License not set or invalid.
Object_ERROR – Generic error.

14.133.2.2 method getSession (out uint32 *sessionID*)

Gets the current secure camera session ID.

Parameters

out	*sessionID*	Current secure camera session ID; 0 if no session is active.

Returns

Object_OK – Success.
ISecureCamera_ERROR_LICENSE_IS_INVALID – License not set or invalid.
Object_ERROR – Generic error.

14.133.2.3 method registerBulkWrite (in uint32 *registerRegionId,* in uint32[ ] *offset,* in uint32[ ] *data*)

Bulk writes 32-bit data to the specified camera register region address space.

Parameters

in	*registerRegionId*	Register region identifier, e.g., MMSS_A_CCI.
in	*offset*	Array of offsets at the register region.
in	*data*	Array of data values to write (corresponding to the offsets).

Returns

Object_OK on success.

14.133.2.4 method registerEventsCallback (in ISecureCameraClientEvent *cb_obj*)

- Registers notification callback for secure camera framework.
- Callback functions are invoked once an event triggers them (see ISecureCameraClientEvent).

Parameters

in	*cb_obj*	Mink object of ISecureCameraClientEvent type.

Returns

Object_OK – Success.
Object_ERROR – Generic error.

14.133.2.5 method registerRead (in uint32 *registerRegionId,* in uint32 *offset,* out uint32 *data*)

Reads 32-bit data to the specified register address space.

Parameters

in	*registerRegionId*	Register region identifier, e.g., MMSS_A_CCI.
in	*offset*	Offset at the register region.
out	*data*	Returned data.

Returns

Object_OK on success.

14.133.2.6 method registerWrite (in uint32 *registerRegionId,* in uint32 *offset,* in uint32 *data*)

Writes 32-bit data to the specified camera register region address space.

Parameters

in	*registerRegionId*	Register region identifier, e.g., MMSS_A_CCI.
in	*offset*	Offset at register region.
in	*data*	Data to write.

Returns

Object_OK – Success.
ISecureCamera_ERROR_LICENSE_IS_INVALID – License not set or invalid.
Object_ERROR – Generic error.

14.133.2.7 method releaseCamera (in uint32 *sessionID*)

- Allows camera to exit secure mode.
- Must be called when the client allows REE to switch camera to non-secure mode.

Parameters

in	*sessionID*	Current secure camera session ID.

Returns

Object_OK – Success.
ISecureCamera_ERROR_LICENSE_IS_INVALID – License not set or invalid.
Object_ERROR – Generic error.

14.133.2.8 method setOEMHavenLicense (in buffer *LicenseCert,* out int32 *haven Error*)

Sets and validates Haven license certificate for Secure Camera feature.

Parameters

in	*LicenseCert*	Buffer containing certificate in DER format.
out	*havenError*	Internal error code returned by IHavenTokenApp service. Returned code can be used for diagnostics and is only valid when the the function completes with Object_OK , e.g., license bypass.

Detailed description

This method must be called at least once after the device reboots to enable the secure camera feature. Invoking secure camera without calling this method first, or if the provided certificate is not valid, results in a disabled secure camera feature.

Note

Non-production devices bypass the license check to support debug.

Returns

Object_OK – Success.
ISecureCamera_ERROR_LICENSE_IS_INVALID – Failed license validation.
Object_ERROR – Generic error.

14.134 ISecureCameraClientEvent

14.134.1 Detailed Description

14.134.2 Function Documentation

14.134.2.1 method phyProtectEvent (in uint32[] *state*)

- Called by QTEE kernel when camera PHYs are protected.
- Implementation must be present in TAs that use secure camera functionality.

Parameters

in	*state*	Reserved for future use.

Returns

Object_OK on success.

14.134.2.2 method phyUnprotectEvent (in uint32[] *state*)

- Called by QTEE kernel when secure camera PHYs are unprotected.
- Implementation must be present in TAs that use secure camera functionality.

Parameters

in	*state*	Reserved for future use.

Returns

Object_OK on success.

14.135 ISecureCamera2

14.135.1 Detailed Description

14.135.2 Function Documentation

14.135.2.1 method getCSFVersion (out uint32 *archVer,* out uint32 *maxVer,* out uint32 *minVer*)

This method gets the current Secure Camera version info.

Parameters

out	*archVer*	architecture version info
out	*maxVer*	max version info
out	*minVer*	min version info

Returns

Object_OK on success. Any other error code on failure.

14.135.2.2 method registerNotifyCB (in ISecureCamera2Notify *cb*)

Registers a ISecureCamera2Notify callback object.

Parameters

in	*cb*	ISecureCamera2Notify callback object

Detailed description

This method is optional. Only if TA wants to receive a notification when all sensors are protected/unprotected, should it register the callback via this method.

Returns

Object_OK on success. Any other error code on failure.

14.135.2.3 method resetCamera ( )

Resets camera to clear secure memory.

Detailed description

This method must be called to clear secure memory between two authenticated users. Before calling, the secure camera session should already be started.

Returns

Object_OK on success. Any other error code on failure.

14.135.2.4 method setParam (in uint32 *paramId,* in buffer *paramBuf*)

Number of Protected Sensors required for this session. Range (1 up to total # sensors on device) Sets and Validates supported Secure Camera parameters.

Parameters

in	*paramId*	Parameter that needs to be set (see PARAM_$*$)
in	*paramBuf*	Buffer/BufferSize used for the specified paramId

Detailed description

This method must be called at least once after the device reboots to enable the secure camera feature by setting the license certificate. Invoking secure camera without calling this method first, or if the provided certificate is not valid, results in a disabled secure camera feature.

Returns

Object_OK on success. Any other error code on failure.

14.135.3 Variable Documentation

14.135.3.1 error ERROR_UNSUPPORTED

ISecureCamera2 is in the wrong state for the requested operation.

14.135.3.2 error ERROR_WRONG_STATE

Haven license validation failed.

14.135.3.3 const uint32 PARAM_LICENSE = 1

ISecureCamera2 does not support this operation.

14.135.3.4 const uint32 PARAM_NUM_SENSORS = 2

PARAM_LICENE MUST be set using a HAVEN license provisioned with the correct OEM_ID / FEAT_ID for this device

14.136 ISecureCamera2Notify

14.136.1 Detailed Description

14.136.2 Function Documentation

14.136.2.1 method event (in uint32 *state*)

Sensor(s) will transition into non-secure streaming mode after this notification This method is called by the secure camera driver after an event has occured. A TrustedApp should implement this if it wants to receive notifications when sensors are protected/unprotected.

Parameters

in	*state*	Indicates the camera streaming state

Returns

Error code

14.136.3 Variable Documentation

14.136.3.1 const uint32 EVENT_UNPROTECTED = 2

Sensor(s) now operating in secure streaming mode

14.137 ISecureChannel

14.137.1 Detailed Description

14.137.2 Function Documentation

14.137.2.1 method authenticateDecryptMessage (in uint32 *subsystemId,* in uint32 *clientId,* in buffer *input,* out buffer *output*)

Authenticates and decrypts secure blob.

Parameters

in	*subsystemId*	Subsystem ID.
in	*clientId*	Client ID.
in	*input*	Buffer containing secure blob.
out	*output*	Buffer to hold decrypted data.

Detailed description

- Output buffer must be large enough to hold the decrypted message.
- Recommended output buffer size is at least input size.
- Caller must manage memory.

Returns

Object_OK on success.

Dependencies

Secure Channel must be established successfully.

14.137.2.2 method secureMessage (in uint32 *subsystemId,* in uint32 *clientId,* in buffer *input,* out buffer *output*)

Secures the input message.

Parameters

in	*subsystemId*	Subsystem ID.
in	*clientId*	Client ID.
in	*input*	Buffer containing plaintext data.
out	*output*	Buffer to hold the output secure blob.

Detailed description

- Output buffer must be large enough to hold encrypted message, some internal headers, and possible padding.
- Recommended output buffer size is at least input message size + 100 bytes.

- Caller must manage memory.

Returns

Object_OK on success.

Dependencies

Secure Channel must be established successfully.

14.138 ISecureChannelKeyExchange

14.138.1 Detailed Description

14.138.2 Function Documentation

14.138.2.1 method commitObjectToRPMB (in uint32 *key_type,* in interface *param_obj*)

Commits the key object to persistent RPMB storage.

Parameters

in	*param_buf*	The key object to be committed.
in	*key_type*	The type of key to be committed. Supported params: • PARAM_LOCAL_CONTEXT_KEY – key derived from base key.

Returns

Object_OK on success. Any other error code on failure.

14.138.2.2 method commitToRPMB (in uint32 *key_type,* in buffer *param_buf*)

Commits the key buffer to persistent RPMB storage.

Parameters

in	*param_buf*	The value of the key to be committed.
in	*key_type*	The type of key to be committed. Supported params: • PARAM_BASE_KEY – locally generated base key. • PARAM_REMOTE_KEY – key received from a remote source.

Returns

Object_OK on success. Any other error code on failure.

14.138.2.3 method deriveContextKey (in buffer *nonce,* out interface *key_obj*)

Derives a key using the base key and nonce. Base key value is taken from RPMB.

Parameters

in	*nonce*	Input nonce value.
out	*key_obj*	Output derived key object.

Returns

Object_OK on success. Any other error code on failure.

14.138.2.4 method generateBaseKey (out buffer *base_key*)

Generates a base key using PRNG and returns it in plaintext.

Parameters

out	*base_key*	Output base key value.

Returns

Object_OK on success. Any other error code on failure.

14.138.2.5 method generateNonce (out buffer *nonce_buf*)

Generates a nonce for use in key derivation.

Parameters

out	*nonce_buf*	Output base key value.

Returns

Object_OK on success

14.138.2.6 method getFromRPMB (in uint32 *param,* out buffer *buf*)

Reads the parameter value from RPMB and returns the buffer.

Parameters

in	*param*	ID of the parameter to fetch.
out	*buf*	Output param value.

Supported params:

- PARAM_REMOTE_KEY – key received from a remote source.

Returns

Object_OK on success. Any other error code on failure.

14.138.2.7 method getObjectFromRPMB (in uint32 *param,* out interface *key_obj*)

Reads the parameter value from RPMB and returns the object.

Parameters

in	*param*	ID of the parameter to fetch.
out	*key_obj*	Output param value.

Supported params:

- PARAM_LOCAL_CONTEXT_KEY – key derived from base key

Returns

Object_OK on success. Any other error code on failure.

14.138.2.8 method isProvisioningDone ( )

Returns whether or not the keys have been provisioned.

Returns

Object_OK on success. Any other error code on failure.

14.138.2.9 method lockProvisioning ( )

Locks the provisioning. New keys can no longer be committed after this step.

Returns

Object_OK on success. Any other error code on failure.

14.138.2.10 method setUsecase (in uint32 *param*)

Sets the usecase ID in the context. This ID is used to manage RPMB partitions.

Parameters

in	*param*	ID of the USECASE. Supported params: • USECASE_IRIS

Returns

Object_OK on success. Any other error code on failure.

14.138.2.11 method unlockProvisioning ( )

Unlocks the provisioning. New keys can be committed again after this step.

Returns

Object_OK on success. Any other error code on failure.

14.139 ISecureImageParserReturnCode

The parent/common interface defining the error codes, shared by the Secure Image Parser interfaces.

14.140 ISecureImageOemMetadataParser

14.140.1 Detailed Description

Generic interface for extracting hash table segment OEM metadata information from Secboot images. Multiple image formats are accepted(i.e., MBNv6, MBNv7, etc.).

14.140.2 Function Documentation

14.140.2.1 method getAntiRollbackVersion (in buffer *imageFileContent,* out uint32 *antiRollbackVersion*)

Returns an Anti Rollback Version for the provided image file.

Parameters

in	*imageFileContent*	Image file content to parse. Uses buffer and size and accepts smaller files.
out	*antiRollbackVersion*	The Anti Rollback Version version value.

Returns

ISecureImageParserReturnCode return code.

14.140.2.2 method getAntiRollbackVersionFromMemObj (in interface *imageFileContent,* out uint32 *antiRollbackVersion*)

Returns an Anti Rollback Version for the provided image file.

Parameters

in	*imageFileContent*	Image file content to parse. Uses a memory object interface and accepts any file size.
out	*antiRollbackVersion*	The Anti Rollback Version version value.

Returns

ISecureImageParserReturnCode return code.

14.140.2.3 method getInterfaceVersion (out uint32 *interfaceVersion*)

Returns the interface version implemented by the ISecureImageOemMetadataParser. That is just a convenience method. Interface version mismatch is not a problem with Mink IPC. If a caller asks for a method that is not implemented, Mink provides a "not implemented" error. However, in some cases, that could be desired to know the root cause of the mismatch. For example, to report mismatched interface versions in the log.

Parameters

out	*interfaceVersion*	The INTERFACE_VERSION (see above) used to implement the TZ service.

Returns

ISecureImageParserReturnCode return code.

14.140.2.4 method getJtagIdBinding (in buffer *imageFileContent,* out uint64 *jtagId*)

Returns an JTAG ID binding for the provided image file if bound to it. If image is unbound, corresponding error code returned (in that case the value of the jtagId is undefined).

Parameters

in	*imageFileContent*	Image file content to parse. Uses buffer and size and accepts smaller files.
out	*jtagId*	The JTAG ID binding value if the image bound to it.

Returns

ISecureImageParserReturnCode return code.

14.140.2.5 method getJtagIdBindingFromMemObj (in interface *imageFileContent,* out uint64 *jtagId*)

Returns an JTAG ID binding for the provided image file if bound to it. If image is unbound, corresponding error code returned (in that case the value of the jtagId is undefined).

Parameters

in	*imageFileContent*	Image file content to parse. Uses a memory object interface and accepts any file size.
out	*jtagId*	The JTAG ID binding value if the image bound to it.

Returns

ISecureImageParserReturnCode return code.

14.140.2.6 method getOemIdBinding (in buffer *imageFileContent,* out uint64 *oemId*)

Returns an OEM ID binding for the provided image file if bound to it. If image is unbound, corresponding error code returned.

Parameters

in	*imageFileContent*	Image file content to parse. Uses buffer and size and accepts smaller files.
out	*oemId*	The OEM ID binding value if the image bound to it.

Returns

ISecureImageParserReturnCode return code.

14.140.2.7 method getOemIdBindingFromMemObj (in interface *imageFileContent,* out uint64 *oemId*)

Returns an OEM ID binding for the provided image file if bound to it. If image is unbound, corresponding error code returned.

Parameters

in	*imageFileContent*	Image file content to parse. Uses a memory object interface and accepts any file size.
out	*oemId*	The OEM ID binding value if the image bound to it.

Returns

ISecureImageParserReturnCode return code.

14.140.2.8 method getOemLifecycleStateBinding (in buffer *imageFileContent,* out uint64 *oemLifecycleState*)

Returns an OEM LIFECYCLE STATE binding for the provided image file if bound to it. Applies to MBNv7 and newer images only. If image is unbound, corresponding error code returned.

Parameters

in	*imageFileContent*	Image file content to parse. Uses buffer and size and accepts smaller files.
out	*oemLifecycleState*	The OEM LIFECYCLE STATE binding value if the image bound to it.

Returns

ISecureImageParserReturnCode return code.

14.140.2.9 method getOemLifecycleStateBindingFromMemObj (in interface *image FileContent,* out uint64 *oemLifecycleState*)

Returns an OEM LIFECYCLE STATE binding for the provided image file if bound to it. Applies to MBNv7 and newer images only. If image is unbound, corresponding error code returned.

Parameters

in	*imageFileContent*	Image file content to parse. Uses a memory object interface and accepts any file size.
out	*oemLifecycleState*	The OEM LIFECYCLE STATE binding value if the image bound to it.

Returns

ISecureImageParserReturnCode return code.

14.140.2.10 method getOemProductIdBinding (in buffer *imageFileContent,* out uint64 *oemProductId*)

Returns an OEM PRODUCT ID binding for the provided image file if bound to it. If image is unbound, corresponding error code returned.

Parameters

in	*imageFileContent*	Image file content to parse. Uses buffer and size and accepts smaller files.
out	*oemProductId*	The OEM PRODUCT ID binding value if the image bound to it.

Returns

ISecureImageParserReturnCode return code.

14.140.2.11 method getOemProductIdBindingFromMemObj (in interface *imageFileContent,* out uint64 *oemProductId*)

Returns an OEM PRODUCT ID binding for the provided image file if bound to it. If image is unbound, corresponding error code returned.

Parameters

in	*imageFileContent*	Image file content to parse. Uses a memory object interface and accepts any file size.
out	*oemProductId*	The OEM PRODUCT ID binding value if the image bound to it.

Returns

ISecureImageParserReturnCode return code.

14.140.2.12 method getSerialNumberBindings (in buffer *imageFileContent,* out uint64[ ] *serialNumbers*)

Returns SERIAL NUMBER bindings for the provided image file if bound to it. If image is unbound, corresponding error code returned. On success, the API will return an array of uint64 values. The caller must allocate sufficient buffer for that array. Generically, images bind to 8 or fewer serial numbers.

Parameters

in	*imageFileContent*	Image file content to parse. Uses buffer and size and accepts smaller files.
out	*serialNumbers*	The array of SERIAL NUMBER bindings if the image bound to it.

Returns

ISecureImageParserReturnCode return code.

14.140.2.13 method getSerialNumberBindingsFromMemObj (in interface *imageFileContent,* out uint64[ ] *serialNumbers*)

Returns SERIAL NUMBER bindings for the provided image file if bound to it. If image is unbound, corresponding error code returned. On success, the API will return an array of uint64 values. The caller must allocate sufficient buffer for that array. Generically, images bind to 8 or fewer serial numbers.

Parameters

in	*imageFileContent*	Image file content to parse. Uses a memory object interface and accepts any file size.
out	*serialNumbers*	The array of SERIAL NUMBER bindings if the image bound to it.

Returns

ISecureImageParserReturnCode return code.

14.140.3 Variable Documentation

14.140.3.1 const uint32 INTERFACE_VERSION = 1

The version of the interface. Allows a user of the interface to verify if implementation (in TZ) corresponds to the interface file version (i.e., this IDL).

14.141 ISecureDisplay

14.141.1 Detailed Description

14.141.2 Function Documentation

14.141.2.1 method getLMRegCount (out uint32 *lm_count*)

Gets the Layer Mixer Register count for CRC calculation.

Parameters

out	*lm_count*	Count of Layer Mixer registers used by display 0 to disable frame authentication based on MISR.

Returns

Object_OK if successful.

14.141.2.2 method getSession (out uint32 *sessionID*)

Gets current secure display session ID.

Parameters

out	*sessionID*	Current secure display session ID. 0 if no active session.

Returns

Object_OK on success.

14.141.2.3 method readMISR (out uint32[ ] *lm_misr*)

Reads the Layer Mixer MISR registers.

Parameters

out	*lm_misr*	Pointer to output array for MISR vaues. Array size must be at least equal to 2∗lm_count.

Returns

Object_OK if successful.

14.141.2.4 method setDisplayId (in uint32 *disp_id*)

set display id for topology lockdown.

Parameters

in	*disp_id*	identifier for primary or secondary display

Returns

Object_OK if successful.

14.141.2.5 method setStopAllowed (in int32 *isAllowed*)

Controls whether REE can stop the secure display using Android's sd_ctrl syscall.

Parameters

in	*isAllowed*	1 allows REE to stop secure display; otherwise, 0.

Returns

Object_OK on success.

14.142 ISharedBuffer

14.142.1 Detailed Description

14.142.2 Function Documentation

14.142.2.1 method deregisterSharedBuffer (in uint64 *start,* in uint64 *size*)

Deregisters the shared buffer previously registered with QTEE for trusted applications.

Parameters

in	*start*	Buffer reference, must match registered buffer.
in	*size*	Buffer size (deprecated).

Returns

Object_OK – Success.
Object_ERROR – Failure.

14.142.2.2 method flushCache (in uint64 *start*)

- Flushes cachelines for the shared memory buffer.
- Must be called prior to passing buffer contents to consumer.

Parameters

in	*start*	Buffer reference, must match registered buffer.

Returns

Object_OK – Success.
Object_ERROR – Failure.

14.142.2.3 method invalidateCache (in uint64 *start*)

- Invalidates cachelines for the shared memory buffer.
- Must be called prior to accessing memory region.

Parameters

in	*start*	Buffer reference, must match registered buffer.

Returns

Object_OK – Success.
Object_ERROR – Failure.

14.142.2.4 method registerSharedBuffer (in uint64 *start,* in uint64 *size*)

Deregisters the shared buffer with QTEE for trusted applications.

Parameters

in	*start*	Buffer reference, must match registered buffer.
in	*size*	Buffer size.

Returns

Object_OK – Success.
Object_ERROR – Failure.

14.143 ISource

14.143.1 Detailed Description

14.143.2 Function Documentation

14.143.2.1 method read (out buffer *data*)

Returns a random number.

Parameters

out	*data*	Random number.

Returns

Object_OK on success.

14.144 ISPCOM

14.144.1 Detailed Description

14.144.2 Function Documentation

14.144.2.1 method client_is_server_connected (in uint64 *handle,* out uint32 *is_connected*)

- Checks if remote server is connected.
- Checks that logical channel is fully connected between the TZ client and SP server.

Parameters

in	*handle*	Client handle.
out	*is_connected*	Boolean value indicates whether channel is connected.

Returns

Object_OK on success.
Object_ERROR or negative value on failure.

14.144.2.2 method client_send_message_sync (in uint64 *handle,* in buffer *req_ptr,* in uint32 *req_size,* in buffer *resp_ptr,* in uint32 *resp_size,* in uint32 *timeout_msec,* out uint32 *retRespSize*)

Sends a request and receives a response.

Parameters

in	*handle*	Client handle.
in	*req_ptr*	Request buffer.
in	*req_size*	Request buffer size.
in	*resp_ptr*	Response buffer.
in	*resp_size*	Response buffer size (max response size).
in	*timeout_msec*	Timeout in msec between command and response. 0=no timeout.
out	*retRespSize*	Actual response size.

Returns

Object_OK on success.
Object_ERROR or negative value on failure.

14.144.2.3 method get_shared_memory (in uint32 *size,* out interface *mem,* out uint64 *phy_addr*)

Returns a shared memory region to be used with TZSPCOM.

Spcom Service returns returns a region that can be read or written by the TA and the memory range is guaranteed to only be read-write accessible to the SPSS and TZ domains for the duration of the lifetime of the memory object in the TA.

Note: The memory range can be accessed by other TAs at the same time and the SPU has no way to tell which TA wrote what memory to the shared memory.

Parameters

in	*size*	size of the buffer
out	*mem*	shared memory region object
out	*phy_addr*	shared memory region physical address

Returns

Object_OK on success.
Object_ERROR or negative value on failure.

14.144.2.4 method is_sp_subsystem_link_up (out uint32 *is_link_up*)

Check if SPSS link is up.

Parameters

out	*is_link_up*	Boolean value indicates if link is up.

Returns

Object_OK on success.
Object_ERROR or negative value on failure.

14.144.2.5 method register_client (in buffer *chName,* out uint64 *retHandle*)

Registers client for communication channel.

Parameters

in	*chName*	Channel name string.
out	*retHandle*	Client handle.

Returns

Object_OK on success.
Object_ERROR or negative value on failure.

14.144.2.6 method register_shared_memory (in uint64 *phy_addr,* in uint32 *size,* out interface *mem*)

Returns a shared memory region object.

The memory passed for registration belongs to an area of memory reserved for SP and TZ domains read-write access.

Spcom Service returns a memory region object that can be read or written by the TA and the memory range is guaranteed to only be read-write accessible to the SPSS and TZ domains for the duration of the lifetime of the memory object in the TA.

Note: The memory range can be accessed by other TAs at the same time and the SPU has no way to tell which TA wrote what memory to the shared memory.

Parameters

in	*phy_addr*	physical address of the buffer.
in	*size*	size of the buffer
out	*mem*	shared memory region object

Returns

Object_OK on success.
Object_ERROR or negative value on failure.

14.144.2.7 method register_shared_memory_from_obj (in interface *memObjIn,* out interface *memObjOut*)

Returns a shared memory region object.

The memory object passed for registration belongs to an area of memory reserved for SP and TZ domains read-write access.

Spcom Service returns a memory region object that can be read or written by the TA and the memory range is guaranteed to only be read-write accessible to the SPSS and TZ domains for the duration of the lifetime of the memory object in the TA.

Note: The memory range can be accessed by other TAs at the same time and the SPU has no way to tell which TA wrote what memory to the shared memory.

Parameters

in	*memObjIn*	shared memory object of the buffer
out	*memObjOut*	shared memory region object

Returns

Object_OK on success.
Object_ERROR or negative value on failure.

14.144.2.8 method reset_sp_subsystem ( )

- Sends reset command to secure processor.
- Ask remote SP to reset itself. SP initiates a Watch-Dog-Bite.

Returns

Object_OK on success.
Object_ERROR or negative value on failure.

14.144.2.9 method unregister_client (in uint64 *handle*)

Unregisters client for communication channel.

Parameters

in	*handle*	Client handle.

Returns

Object_OK on success.
Object_ERROR or negative value on failure.

14.144.3 Variable Documentation

14.144.3.1 error ERROR_INVALID_SIZE

Error Codes

14.145 ISPI

14.145.1 Detailed Description

14.145.2 Function Documentation

14.145.2.1 method close (in int32 *deviceId*)

Closes client access to the SPI device.

Parameters

in	*deviceId*	SPI device ID.

Returns

Object_OK on success.

14.145.2.2 method open (in int32 *deviceId*)

Opens device and performs hardware intialization.

Parameters

in	*deviceId*	SPI device ID to attach to.

Returns

Object_OK on success.

14.145.2.3 method read (in int32 *deviceId,* in ISPI_Config *config,* out buffer *data*)

Reads data from SPI bus.

Parameters

in	*deviceId*	SPI device ID.
in	*config*	Desired SPI configuration.
out	*data*	Reads buffer information.

Returns

Object_OK on success.

14.145.2.4 method write (in int32 *deviceId,* in ISPI_Config *config,* in buffer *data,* out uint64 *bytesWritten*)

Writes data on SPI bus.

Parameters

in	*deviceId*	SPI device ID to attach to.
in	*config*	Desired SPI configuration.
in	*data*	Write buffer.
out	*bytesWritten*	Number of bytes actually written.

Returns

Object_OK on success.

14.145.2.5 method writeFullDuplex (in int32 *deviceId,* in ISPI_Config *config,* in buffer *data,* out uint64 *bytesWritten,* out buffer *returnedData*)

Transfers bidirectional data on SPI bus.

Parameters

in	*deviceId*	SPI device ID to attach to.
in	*config*	Desired SPI configuration.
in	*data*	Write buffer.
out	*bytesWritten*	Number of bytes actually written.
out	*returnedData*	Data buffer for data read back.

Returns

Object_OK on success.

14.145.2.6 method writeFullDuplexExt (in int32 *deviceId,* in ISPI_Config_Ext *config,* in buffer *data,* out uint64 *bytesWritten,* out buffer *returnedData*)

Transfers bidirectional data on SPI bus.

Parameters

in	*deviceId*	SPI device ID to attach to.
in	*config*	Desired SPI configuration specified by ISPI_Config_Ext.
in	*data*	Write buffer.
out	*bytesWritten*	Number of bytes actually written.
out	*returnedData*	Data buffer for data read back.

Returns

Object_OK on success.

14.146 ISwFuse

14.146.1 Detailed Description

14.146.2 Function Documentation

14.146.2.1 method blowSwFuse (in int32 *fuse*)

Blow the SW fuse.

Note: Function is not thread-safe.

Parameters

in	*fuse*	The SW fuse to blow.

Returns

Object_OK – success.
ERROR_INVALID_CLIENT – invalid client requesting the service
Object_ERROR – any error occurred except invalid client.

14.146.2.2 method getSecureState (out uint32 *status1,* out uint32 *status2*)

Check security status on the device.

Parameters

out	*status1*	The security status with the following bit field definitions: • Bit 0: secboot enabling check failed • Bit 1: Sec HW key is not programmed • Bit 2: debug disable check failed • Bit 3: Anti-rollback check failed • Bit 4: fuse config check failed • Bit 5: rpmb provisioned check failed • Bit 6: debug check in image certificate failed (debug ou field missing from image cert) • Bit 7: RSVD • Bit 8: tz secure debug fuse check failed • Bit 9: mss secure debug fuse check failed • Bit 10: cp secure debug fuse check failed • Bit 11: non-secure secure debug fuse check failed
out	*status2*	The security status with the following bit field definitions: • Bits 0-31: reserved

Returns

Object_OK – success.
Object_ERROR – any error occurred.

14.146.2.3 method isSwFuseBlown (in int32 *fuse_num,* out uint32 *is_blown*)

Test whether the specified SW fuse is blown.

Parameters

in	*fuse_num*	The SW fuse to query.
out	*is_blown*	Whether the given SW fuse is blown.

Returns

Object_OK – success.
Object_ERROR – any error occurred.

14.147 ISync

14.147.1 Detailed Description

14.147.2 Function Documentation

14.147.2.1 method spin (in uint32 *timeoutUs*)

Performs a busy wait (spin) for the input number of microseconds.

Parameters

in	*timeoutUs*	Number of microseconds to spin.

Returns

Always returns Object_OK.

14.148 ITLMM

14.148.1 Detailed Description

14.148.2 Function Documentation

14.148.2.1 method ConfigGpioId (in uint32 *gpioId,* in ITLMM_Config *settings*)

Configures a GPIO based on the GPIO ID.

Parameters

in	*gpioId*	GPIO key that allocates a GPIO to a user.
in	*settings*	Configurable settings (pull, drive, direction).

Returns

Object_OK on success.

14.148.2.2 method GetGpioId (in buffer *gpioStr,* out uint32 *gpioId*)

- Retrieves the GPIO ID for a specific GPIO pin.
- This function is required to perform operations on a GPIO.

Parameters

in	*gpioStr*	String name of GPIO signal.
out	*gpioId*	GPIO key that allocates GPIO to a user.

Returns

Object_OK on success. ITLMM_ERROR_ACCESS_DENIED when the client cannot use the GPIO.

14.148.2.3 method GpioIdIn (in uint32 *gpioId,* out uint32 *value*)

Reads the input value of a GPIO based on the GPIO ID.

Parameters

in	*gpioId*	GPIO key that allocates GPIO to a user.
out	*value*	Value (0 = LOW, 1 = HIGH) of the input signal.

Returns

Object_OK on success.

14.148.2.4 method GpioIdOut (in uint32 *gpioId,* in uint32 *value*)

Drives output value of a GPIO based on the GPIO ID.

Parameters

in	*gpioId*	GPIO key that allocates a GPIO to a user.
in	*value*	Value (0 = LOW, 1 = HIGH) to drive an output.

Returns

Object_OK on success.

14.148.2.5 method ReleaseGpioId (in uint32 *gpioId*)

- Releases GPIO and destroys GPIO ID key.
- This function is optional in case a GPIO is no longer needed.

Parameters

in	*gpioId*	GPIO key that allocates a GPIO to a user.

Returns

Object_OK on success.

14.148.2.6 method SelectGpioIdMode (in uint32 *gpioId,* in uint32 *mode*)

- Selects the function select (either 0 or primary) of a GPIO based on the GPIO ID.
- Optionally sets configuration parameters.

Parameters

in	*gpioId*	GPIO key that allocates GPIO to a user.
in	*mode*	Mode IO (0) or PRIMARY (1) to set GPIO mux.

Returns

Object_OK on success.

14.148.3 Variable Documentation

14.148.3.1 error ERROR_ACCESS_DENIED

Failure on Access Control privileges (see CSecureGPIO).

14.148.3.2 const uint32 GPIO_2MA = 0

In idle state whether the Chip Select is high or low.

14.148.3.3 const uint32 GPIO_INPUT = 0

Specifies if the GPIO should be an input or an output.

14.148.3.4 const uint32 GPIO_LOW = 0

The drive value for an output GPIO.

14.148.3.5 const uint32 GPIO_MODE_IO = 0

Specifier to switch between primary and IO (bit-bang) mode.

14.148.3.6 const uint32 GPIO_NO_PULL = 0

Specifies the weak pull of the GPIO when configured as input GPIO or in some cases of alternate functions.

14.149 ITLOCKey

14.149.1 Detailed Description

14.149.2 Function Documentation

14.149.2.1 method getKeyData (out buffer *keydata*)

The function gets key data using secure channel API. The Trusted Location (TLOC) system running in HLOS shall pass the key data to MDM. In the case of fusion target, existing secure channel shared memory based key exchange between TZ-Modem is not possible, as there is no shared memory between APQ to MDM. The communication between these processors is established by data transfer over PCIe through TLOC in HLOS. As per the design, the expected size of the buffer for the key from the caller is 256 bytes.

Parameters

out	*keydata*	An output buffer populated with key data

Returns

Object_OK indicates success Object_ERROR indicates failure

14.149.2.2 method getKeyNonce (out buffer *noncedata*)

The function gets nonce data using secure channel API. The Trusted Location (TLOC) system running in HLOS shall pass the nonce data to MDM. To mitigate the anti-reply, the nonce will be shared between APQ to MDM for every boot. As per the design, the expected size of the buffer for the nonce from the caller is 128 bytes.

Parameters

out	*noncedata*	An output buffer populated with nonce data

Returns

Object_OK indicates success Object_ERROR indicates failure

14.149.3 Variable Documentation

14.149.3.1 error ERROR_KEY_ALREADY_GENERATED

Error code returned by getKeyData method when the key is already generated.

14.150 ITPM

14.150.1 Detailed Description

14.150.2 Function Documentation

14.150.2.1 method CheckNVFlushState (out uint8 *flushPending*)

Returns status of NV Flush to storage.

Parameters

out	*flushPending*	1 if NV flush is pending. 0 if NV has been flushed.

Returns

Object_OK on success.

14.150.2.2 method GetControlAreadAddr (out uint64 *addr*)

get tpm control area address

Parameters

out	*addr*	pointer to tpm control area address

Returns

Object_OK on success.

14.150.2.3 method SendTPMCommand (in uint8 *locality,* in buffer *reqBuffer,* out buffer *rspBuffer*)

Sends TPM command to TpmApp.

Parameters

in	*locality*	Locality for the command ($4 <$ locality < 32).
in	*reqBuffer*	Request buffer containing command.
out	*rspBuffer*	Response buffer.

Returns

Object_OK on success.

14.150.2.4 method TpmCheckLocality ( )

Sends TPM check locality command.

Returns

Object_OK on success which means no active locality of 0,1,2,3

14.150.2.5 method TpmHashData (in buffer *data*)

Sends TPM _TPM_HASH_DATA command.

Parameters

in	*data*	data to be hashed.

Returns

Object_OK on success.

14.150.2.6 method TpmHashEnd ( )

Sends TPM _TPM_HASH_END command.

Returns

Object_OK on success.

14.150.2.7 method TpmHashStart ( )

Sends TPM _TPM_HASH_START command.

Returns

Object_OK on success.

14.151 ITranslateAddr

14.151.1 Detailed Description

14.151.2 Function Documentation

14.151.2.1 method getPA (in uint32 *translationMethod,* in uint64 *VA,* out uint64 *PA*)

Translates a virtual address to physical address.

Parameters

in	*translationMethod*	Describes what processor state should attempt the translation. Affects translation outcome depending on memory access permission properties.
in	*VA*	Virtual address to translate.
out	*PA*	Physical address result of translation.

Note: This function truncates addresses to the page boundary. The PA result is also an address on the page boundary.

Returns

Object_OK on success.

14.152 "ITrustedReport"

14.152.1 Detailed Description

ITrustedReport provides an interface to communicate between TZ and Modem to receive secure information, e.g. trusted location, trusted time, etc.

14.152.2 Function Documentation

14.152.2.1 method attestToCustomKey (in buffer *licenseCert,* in buffer *keyBlob,* in buffer *serviceCtx,* out IAttestationBuilder *builder*)

Generates an attestation report that attests to the given custom key.

Parameters

in	*licenseCert*	Buffer containing license to use attestation feature.
in	*keyBlob*	Buffer containing the key material.
in	*serviceCtx*	Bytes that make sense to the 3rd party cloud and is passed along by the QWES cloud back to the 3rd party cloud.
out	*builder*	IAttestationBuilder object to add user data.

Returns

Object_OK on success.

14.152.2.2 method generateCustomKey (in uint32 *keyType,* in buffer *inParams,* out buffer *keyBlob*)

Generates a key pair based on the custom keyType.

Parameters

in	*keyType*	Type of the custom key.
in	*inParams*	Reserved for future use.
out	*keyBlob*	Buffer containing the key material which is bound to caller and device.

Returns

Object_OK on success.

14.152.2.3 method getReport (in uint64 *reportType,* out buffer *report*)

Get CDDL Format for trustedReport.

Parameters

in	*reportType*	for secure contents
out	*report*	in CBOR

Detailed description

Generate CBOR report with cache data in QWES TA

Returns

Object_OK on success.

14.152.2.4 method getStatus (in uint64 *reportType,* out buffer *statusReport*)

Get CDDL Fromat for statusReport.

Parameters

in	*reportType*	for secure contents
out	*status*	in CBOR, age of the data in cache

Detailed description

Typical use is to get the freshness of when the data in cache was last obtained from modem.

Returns

Object_OK on success.

14.152.2.5 method refresh (in uint64 *reportType,* in uint64 *timeout,* in IRefresh Callback *callback*)

Get up-to-date data from modem.

Parameters

in	*reportType*	for secure contents
in	*timeout*	limit for the refresh to complete, currently unused.
in	*callback*	object to call when refresh is done.

Detailed description

Typical use is to call refresh before getReport, this is to update cache by fetch latest data from modem

Returns

Object_OK on success.

14.152.2.6 method setCustomKey (in uint32 *keyType,* in buffer *keyBlob,* in buffer *inParams*)

Unwraps the key identified by keyBlob and sets the internal key of the trusted report that is identified by keyType.

Parameters

in	*keyType*	Type of the custom key.
in	*keyBlob*	Buffer containing the key material.
in	*inParams*	Reserved for future use.

Returns

Object_OK on success.

14.152.3 Variable Documentation

14.152.3.1 const uint32 CUSTOM_KEY_TYPE_TMEHW_ATTEST_SIGN = 0

Custom TME HW attestation signing key.

14.152.3.2 error INVALID_ARGUMENT

Invalid arguments.

14.152.3.3 error INVALID_LICENSE

invalid license or does not contain feature ID

14.152.3.4 error INVALID_REPORT_TYPE

Passed report type is invalid

14.152.3.5 error NO_MEMORY

Failure during memory allocation

14.152.3.6 error NOT_ALLOWED

Not allowed.

14.152.3.7 error NOT_SUPPORTED

Not supported.

14.152.3.8 error REPORT_FAILURE

Failure during report generation

14.152.3.9 error TLOC_PERMISSION_NOT_SET

location permission is not enabled

14.153 IUnwrapKeys

14.153.1 Detailed Description

14.153.2 Function Documentation

14.153.2.1 method unwrap (in buffer *wrapped,* out buffer *unwrapped*)

Unwraps keys in an OEM-specific manner.

Parameters

in	*wrapped*	Wrapped key data.
out	*unwrapped*	Unwrapped key data.

Returns

Object_OK on success.
Object_ERROR on failure to unwrap the data.

14.154 IUptime

14.154.1 Detailed Description

14.154.2 Function Documentation

14.154.2.1 method getBootCycleId (out uint64 *bootCycleId*)

Gets a unique boot-cycle identifier.

Parameters

out	*bootCycleId*	boot-cycle identifier.

Returns

Object_OK – Success.
Object_ERROR – Any error occurred.

14.154.2.2 method getUptime (out uint64 *upTimeMS*)

Gets the up time from bootup in milliseconds.

Parameters

out	*upTimeMS*	Time from bootup in milliseconds.

Returns

Object_OK – Success.
Object_ERROR – Any error occurred.

14.154.2.3 method getUptimeUS (out uint64 *upTimeUS*)

Gets up time from bootup in microseconds.

Parameters

out	*upTimeUS*	Time from bootup in microseconds.

Returns

Object_OK – Success.
Object_ERROR – Any error occurred.

14.155 IValidate

14.155.1 Detailed Description

14.155.2 Function Documentation

14.155.2.1 method validate (in buffer *data*)

Performs data buffer validation.

Parameters

in	*data*	Data buffer to be validated.

Returns

Object_OK on success.

14.156 IVMDeviceUniqueKey

14.156.1 Detailed Description

14.156.2 Function Documentation

14.156.2.1 method derive (out buffer *outKey*)

Derive and return a VM device unique key.

Parameters

out	*outKey*	holds the generated key.

Detailed description

The generated key will have the length of the output buffer.
This key length is bounded such that $0 <$ outKeyLen $\leq (2^{32}\text{-}1)/8$.
An error will be returned if these bounds are violated.

Returns

Object_OK on success.
Object_ERROR on failure.

14.157 IVMSessionKey

14.157.1 Detailed Description

14.157.2 Function Documentation

14.157.2.1 method derive (in uint32 *sessionID,* in buffer *VMUniqueId,* out buffer *outKey*)

Derive a key based on TA distinguished ID, VM unique ID and Session ID.
TA uses this interface for deriving key with VM unique ID, session ID inputs.

Parameters

in	*sessionID*	Used as factor for key derivation and has to be the same as passed by VM to derive the same key.
in	*VMUniqueId*	Holds the VM unique identifier and used as factor for Key derivation.
out	*outKey*	Output buffer to hold the key of desired size.

Returns

Object_OK - success.
Object_ERROR_SIZE_IN - Invalid input arguments
Object_ERROR_MEM - Memory allocation failure
IVMSessionKey_ERROR_PRNG_FAILED - Session Key generation failed due to RNG failure
IVMSessionKey_ERROR_CRYPTO_FAILED - Key Derivation failed due to crypto operation
IVMSessionKey_ERROR_MAX_SESSIONS - Key Derivation failed due to maximum active sessions
IVMSessionKey_ERROR_IMPROPER_CTXT - Key request failed as other endpoint has not taken this session key

14.158 IWait

14.158.1 Detailed Description

14.158.2 Function Documentation

14.158.2.1 method signal (in uint32 *code,* in uint32 *events*)

Signal the object with the specified event(s).

Parameters

in	*code*	Optional code to use for signaling: if non-0, the object will only process a signal with matching code.
in	*events*	See list of supported events above.

Returns

Object_OK if the events were delivered.

14.158.2.2 method wait (in uint32 *msec,* in uint32 *code,* in uint32 *events,* out uint32 *result*)

Wait for the specified amount of milliseconds for an event to occur.

The object will only respond to events passed in the events parameter, to be interpreted as a mask, one event per bit.

Parameters

in	*msec*	How long to wait, in milliseconds, or WAIT_INFINITE.
in	*code*	Optional code to use when processing an incoming signal: if non-0, the object will only accept a signal with matching code.
in	*events*	See list of supported events above.
out	*result*	Mask of the received events.

Returns

Object_OK on success.

14.159 Object-Based QTEE Service Classes to Trusted Applications

14.159.1 Detailed Description

Modules

- CAccessControl
- CAppMessage
- CCertification
- CCipher
- CClockConfig
- CCredentials
- CCrypto
- CDataCache
- CDeviceID
- CDeviceAttestation
- CHash
- CHdcpSrm
- CHdmiStatus
- CHlosRegionFinder
- CHmac
- CHwFuse
- CHWKeyFactory
- CI2C
- CICE
- CIntMask
- CIPProtector
- CKeyManager
- CKVStore
- CKVStoreAdmin
- CLicenseManager
- CListener
- CMacchiato
- CMemRegionPermEscalator

- CNistLoggingFramework
- CNSMem
- CNSSystemReg
- COEMBuf
- COPS
- COPSSink
- COPSSource
- CPeripheralAccessControl
- CCPBitstreamRWAuthority
- CCPAPPRWAuthority
- CCPCameraRWAuthority
- CCPNonPixelRWAuthority
- CPrivacyPreservingID
- CPrngSource
- CQWESTAServices
- CRTICDtb
- CRTICReport
- CRuntimeAttestation
- CSecureCamera
- CSecureCamera2
- CSecureChannel
- CSecureChannelKeyExchange
- CSecureDisplay
- CSecureGPIO
- CSecureImageParser
- CSharedBuffer
- CSPCOM
- CSPI
- CSWCrypto
- CSwFuse
- CSync
- CTLMM

- CTransNSAddr
- CUptime
- CVenusSecureChannel
- CVideoSecureChannel
- CVMSessionKey
- CWhitelistBypass
- CWhitelistBypassRO

14.160 CAccessControl

14.160.1 Detailed Description

Class CAccessControl implements IAccessControl interface.

The class ID ‘AccessControl‘ is not included in the default privilege set. It needs to be added to the privileges section in the metadata to allow the TA to use the CAccessControl class.

14.161 CAppMessage

14.161.1 Detailed Description

Class CAppMessage implements IAppMessage interface.

The class ID ‘AppMessage‘ is included in the default privilege set, therefore the CAppMessage class is available to every TA.

14.162 CCertification

14.162.1 Detailed Description

Class CCertification implements ICertification interface.

14.163 CCipher

14.163.1 Detailed Description

Class CCipher implements ICipher interface.

The class IDs ‘CipherAES128‘, ‘CipherAES256‘, ‘CipherTripleDES‘ and ‘CipherSM4‘ are included in the default privilege set, therefore the CCipher class is available to every TA.

14.164 CClockConfig

14.164.1 Detailed Description

Class CClockConfig implements IClockConfig interface.

The class ID ‘ClockConfig‘ is included in the default privilege set.

14.165 CCredentials

14.165.1 Detailed Description

Class CCredentials implements ICredentials interface.

When a TA opens CCredentials, it gets its own credentials object.

The class ID ‘CCredentials‘ is included in the default privilege set, therefore the CCredentials class is available to every TA.

14.166 CCrypto

14.166.1 Detailed Description

Class CCrypto implements ICrypto interface in kernel space.

The class ID ‘CCrypto‘ is included in the default privilege set, therefore the CCrypto class is available to every TA.

14.167 CDataCache

14.167.1 Detailed Description

Class CDataCache implements IDataCache interface.

The class ID ‘DataCache‘ is included in the default privilege set, therefore the CDataCache class is available to every TA.

14.168 CDeviceID

14.168.1 Detailed Description

Class CDeviceID implements IDeviceID interface.

The class ID ‘DeviceID‘ is included in the default privilege set, therefore the CDeviceID class is available to every TA.

This class provides interfaces to TA to request various sort of device ID such as jtag id, OEM id, device id, HW Version etc.

This service is part of default privilege set for TAs.

14.169 CDeviceAttestation

14.169.1 Detailed Description

Class CDeviceAttestation implements IDeviceAttestation interface.

The class ID ‘CDeviceAttestation‘ is not included in the default privilege set. It needs to be added to the privileges section in the metadata to allow the TA to use the CDeviceAttestation class.

14.170 CHash

14.170.1 Detailed Description

Class CHash implements IHash interface.

The class IDs ‘HashNone‘, ‘HashSHA1‘, ‘CHashSHA256‘, ‘HashSHA384‘, ‘HashSHA512‘, ‘HashSM3‘, ‘HashSHA3_224‘, ‘HashSHA3_256‘, ‘HashSHA3_384‘ , ‘HashSHA3_512‘ , ‘HashSHAKE_128‘ , ‘HashSHAKE_256‘ , ‘HashCSHAKE_128‘ and ‘HashCSHAKE_256‘ are included in the default privilege set, therefore the CHash class is available to every TA.

14.171 CHdcpSrm

14.171.1 Detailed Description

Class CHdcpSrm implements IHdcpSrm interface.

The class ID ‘HdcpSrm‘ is not included in the default privilege set. It needs to be added to the privileges section in the metadata to allow the TA to use the CHdcpSrm class.

14.172 CHdmiStatus

14.172.1 Detailed Description

Class CHdmiStatus implements IHdmiStatus interface.

The class ID ‘HdmiStatus‘ is not included in the default privilege set. It needs to be added to the privileges section in the metadata to allow the TA to use the CHdmiStatus class.

14.173 CHlosRegionFinder

14.173.1 Detailed Description

Class CHlosRegionFinder implements IHlosRegionFinder interface.

The class ID ‘HlosRegionFinder‘ is included in the default privilege set granted to all TAs that allows them to use the IHlosRegionFinder interfaces for acquiring REE memory regions.

14.174 CHmac

14.174.1 Detailed Description

Class CHmac implements IHmac interface.

The class IDs ‘HmacSHA1‘, ‘HmacSHA256‘, ‘HmacSHA384‘, ‘HmacSHA512‘, ‘HmacSHA3_224‘, ‘HmacSHA3_256‘, ‘HmacSHA3_384‘ and ‘HmacSHA3_512‘ are included in the default privilege set, therefore the CHmac class is available to every TA.

14.175 CHwFuse

14.175.1 Detailed Description

Class CHwFuse implements IHwFuse interface.

The class ID ‘HwFuse‘ is not included in the default privilege set. It needs to be added to the privileges section in the metadata to allow the TA to use the CHwFuse class.

14.176 CHWKeyFactory

14.176.1 Detailed Description

Class CHWKeyFactory implements IHWKeyFactory interface.

The class ID ‘HWKeyFactory‘ is not included in the default privilege set. It needs to be added to the privileges section in the metadata to allow the TA to use the CHWKeyFactory class.

14.177 CI2C

14.177.1 Detailed Description

Class CI2C implements II2C interface.

The class ID ‘I2C‘ is not included in the default privilege set. It needs to be added to the privileges section in the metadata to allow the TA to use the CI2C class.

14.178 CICE

14.178.1 Detailed Description

Class CICE implements IICE interface.

The class ID ‘ICE‘ is not included in the default privilege set. It needs to be added to the privileges section in the metadata to allow the TA to use the CICE class.

14.179 CIntMask

14.179.1 Detailed Description

Class CIntMask implements IIntMask interface.

The class ID ‘IntMask‘ is not included in the default privilege set. It needs to be added to the privileges section in the metadata to allow the TA to use the CIntMask class.

14.180 CIPProtector

14.180.1 Detailed Description

Class CIPProtector implements IIPProtector interface.

The class ID ‘IPProtector‘ is not included in the default privilege set. It needs to be added to the privileges section in the metadata to allow the TA to use the CIPProtector class.

14.181 CKeyManager

14.181.1 Detailed Description

Class CKeyManager implements IKeyManager interface.

The class ID ‘KeyManager‘ is not included in the default privilege set. It needs to be added to the privileges section in the metadata to allow the TA to use the CKeyManager class.

14.182 CKVStore

14.182.1 Detailed Description

Class CKVStore implements IKVStore interface.

14.183 CKVStoreAdmin

14.183.1 Detailed Description

Class CKVStoreAdmin implements IKVStoreAdmin interface.

14.184 CLicenseManager

14.184.1 Detailed Description

Class CLicenseManager implements ILicenseManager interface.

The class ID ‘LicenseManager‘ is not included in the default privilege set. It needs to be added to the privileges section in the metadata to allow the TA to use the CLicenseManager class.

14.185 CListener

14.185.1 Detailed Description

Class CListener implements IListener interface.

The class ID ‘Listener‘ is included in the default privilege set, therefore the CListener class is available to every TA.

14.186 CMacchiato

14.186.1 Detailed Description

Class CMacchiato implements IMacchiato interface.

The class ID ‘Macchiato‘ is not included in the default privilege set. It needs to be added to the privileges section in the metadata to allow the TA to use the CMacchiato class.

14.187 CMemRegionPermEscalator

14.187.1 Detailed Description

Class CMemRegionPermEscalator implements IMemRegionPermEscalator interface.

The class ID ‘MemRegionPermEscalator‘ is included in the default privilege set, therefore the CMemRegionPermEscalator class is available to every TA.

TAs must have additional privileges associated with memory permission escalation, such as CCPBitstreamRWAuthority in order to open this class. A TA with no such privileges will fail to open an object with this class ID.

14.188 CNistLoggingFramework

14.188.1 Detailed Description

Class CNistLoggingFramework implements INistLoggingFramework interface which can be use to read logs from NistLogging partition.

The class ID ‘NistLoggingFramework‘ is not included in the default privilege set. It needs to be added to the privileges section in the metadata to allow the TA to use the CNistLoggingFramework class.

14.189 CNSMem

14.189.1 Detailed Description

Class CNSMem implements INSMem interface.

The class ID ‘NSMem‘ is included in the default privilege set, therefore the CNSMem class is available to every TA.

14.190 CNSSystemReg

14.190.1 Detailed Description

Class CNSSystemReg implements INSSystemReg interface.

The class ID ‘NSSystemReg‘ is not included in the default privilege set. It needs to be added to the privileges section in the metadata to allow the TA to use the CNSSystemReg class.

14.191 COEMBuf

14.191.1 Detailed Description

Class COEMBuf implements IIO interface.

The class ID ‘OEMBuf‘ is not included in the default privilege set. It needs to be added to the privileges section in the metadata to allow the TA to use the COEMBuf class.

14.192 COPS

14.192.1 Detailed Description

Class COPS implements IOPS interface.

The class ID ‘OPS‘ is not included in the default privilege set. It needs to be added to the privileges section in the metadata to allow the TA to use the COPS class.

14.193 COPSSink

14.193.1 Detailed Description

Class COPSSink implements IOPSSink interface.

14.194 COPSSource

14.194.1 Detailed Description

Class COPSSource implements IOPSSource interface.

The class ID ‘OPSSource‘ is not included in the default privilege set. It needs to be added to the privileges section in the metadata to allow the TA to use the COPSSource class.

14.195 CPeripheralAccessControl

14.196 CCPBitstreamRWAuthority

14.196.1 Detailed Description

The class ID CCPBitstreamRWAuthority has no class. TAs with this privilege grant the CMemRegionPermEscalator class the authority to escalate access permissions to memory inaccessible to the calling client, but accessible to the CP Bitstream VMID. TAs may escalate to either RO or RW access permissions with this privilege.

The class ID ‘CCPBitstreamRWAuthority‘ is not included in the default privilege set. It must be added to the privileges section in the TA’s metadata in order to gain the above mentioned authority.

14.197 CCPAPPRWAuthority

14.197.1 Detailed Description

The class ID CCPAPPRWAuthority has no class. TAs with this privilege grant the CMemRegionPermEscalator class the authority to escalate access permissions to memory inaccessible to the calling client, but accessible to the CP APP VMID. TAs may escalate to either RO or RW access permissions with this privilege.

The class ID ‘CCPAPPRWAuthority‘ is not included in the default privilege set. It must be added to the privileges section in the TA’s metadata in order to gain the above mentioned authority.

14.198 CCPCameraRWAuthority

14.198.1 Detailed Description

The class ID CCPCameraRWAuthority has no class. TAs with this privilege grant the CMemRegionPermEscalator class the authority to escalate access permissions to memory inaccessible to the calling client, but accessible to the CP Camera VMID. TAs may escalate to either RO or RW access permissions with this privilege.

The class ID ‘CPCameraRWAuthority‘ is not included in the default privilege set. It must be added to the privileges section in the TA’s metadata in order to gain the above mentioned authority.

14.199 CCPNonPixelRWAuthority

14.199.1 Detailed Description

The class ID CCPNonPixelRWAuthority has no class. TAs with this privilege grant the CMemRegionPermEscalator class the authority to escalate access permissions to memory inaccessible to the calling client, but accessible to the CP Non-Pixel VMID. TAs may escalate to either RO or RW access permissions with this privilege.

The class ID ‘CPNonPixelRWAuthority‘ is not included in the default privilege set. It must be added to the privileges section in the TA’s metadata in order to gain the above mentioned authority.

14.200 CPrivacyPreservingID

14.200.1 Detailed Description

Class CPrivacyPreservingID implements IPrivacyPreservingID interface.

The class ID ‘PrivacyPreservingID‘ is not included in the default privilege set. It needs to be added to the privileges section in the metadata to allow the TA to use the CPrivacyPreservingID class.

14.201 CPrngSource

14.201.1 Detailed Description

Class CPrngSource implements ISource interface.

The class ID ‘PrngSource‘ is included in the default privilege set, therefore the CPrngSource class is available to every TA.

14.202 CQWESTAServices

14.202.1 Detailed Description

Class CQWESTAServices implements IQWESTAServices interface.

The class ID ‘QWESTAServices‘ is not included in the default privilege set. It needs to be added to the privileges section in the metadata to allow the TA to use the CQWESTAServices class.

14.203 CRTICDtb

14.203.1 Detailed Description

Class CRTICDtb implements IRTICDtb interface.

The class ID ‘RTICDtb‘ is not included in the default privilege set. It needs to be added to the privileges section in the metadata to allow the TA to use the CRTICDtb class.

14.204 CRTICReport

14.204.1 Detailed Description

Class CRTICReport implements IRTICReport interface.

The class ID ‘RTICReport‘ is not included in the default privilege set. It needs to be added to the privileges section in the metadata to allow the TA to use the CRTICReport class.

14.205 CRuntimeAttestation

14.205.1 Detailed Description

Class CRuntimeAttestation implements IRuntimeAttestation interface which can be use to generate report to check if a partition is corrupted or not.

The class ID ‘RuntimeAttestation‘ is not included in the default privilege set. It needs to be added to the privileges section in the metadata to allow the TA to use the CRuntimeAttestation class.

14.206 CSecureCamera

14.206.1 Detailed Description

Class CSecureCamera implements ISecureCamera interface.

The class ID ‘SecureCamera‘ is not included in the default privilege set. It needs to be added to the privileges section in the metadata to allow the TA to use the CSecureCamera class.

14.207 CSecureCamera2

14.207.1 Detailed Description

Class CSecureCamera2 implements ISecureCamera2 interface.

The class ID ‘SecureCamera2‘ needs to be added to the TA privileges set to allow the TA to use the CSecureCamera2 class.

With this class ID, there is a possibility that the protect/unprotect event callback of ISecureCamera2Notify may never reach TA if the TA is busy processing a request or is out on a listener.

14.208 CSecureChannel

14.208.1 Detailed Description

Class CSecureChannel implements ISecureChannel interface.

The class ID ‘SecureChannel‘ is included in the default privilege set, therefore the CSecureChannel class is available to every TA.

14.209 CSecureChannelKeyExchange

14.209.1 Detailed Description

Class CSecureChannelKeyExchange implements ISecureChannelKeyExchange interface.

The class ID ‘SecureChannelKeyExchange‘ is not included in the default privilege set. It needs to be added to the privileges section in the metadata to allow the TA to use the CHWKeyFactory class.

14.210 CSecureDisplay

14.210.1 Detailed Description

Class CSecureDisplay implements ISecureDisplay interface.

The class ID ‘SecureDisplay‘ is not included in the default privilege set. It needs to be added to the privileges section in the metadata to allow the TA to use the CSecureDisplay class.

14.211 CSecureGPIO

14.211.1 Detailed Description

The class IDs 'SecureGPIO' are not included in the default privilege set. The IDs need to be added to the privileges section in the metadata to allow the client (e.g. a TA) to use the CSecureGPIO class.

This class is used to enforce an Access Control (AC) policy on GPIOs. This class needs to be used in conjunction with TLMM (see ITLMM). Therefore, the client should have the CTLMM (CTLMM) privilege in conjunction with at least one of the IDs defined by this class.

How it works: Each time a GPIO is requested through TLMM, QTEE will check if the GPIO is subject to the AC policy. If AC needs to be verified and the client has the required UID in its metadata, then this client can open and use the GPIO.

How to protect a GPIO: the AC policy is defined through a mapping between GPIOs and UIDs at compilation-time. The mapping should be declared through devcfg (e.g. 'oem_config.xml'). Inside the devcfg file, the device 'secure_gpio' needs to be added and each protected GPIO has to be declared as a 'props' field, where the 'name' is the string identifying the GPIO, the type is 'DALPROP_ATTR_TYPE_UINT32' and the content of this field is one of the SecureGPIOIds defined by this class.

For example, the following snippet showcases how the GPIO 'tlmm_gpio_test_pin' would be protected by 'CSecureGPIOId_0_UID'.

```
<device id="secure_gpio_dev">
 <props name="tlmm_gpio_test_pin" type=DALPROP_ATTR_TYPE_UINT32>
   650 <!-- CSecureGPIOId_0_UID -->
 </props>
</device>
```

14.212 CSecureImageParser

14.212.1 Detailed Description

The CSecureImageParser implements ICSecureImageParser interface.

Generic interface for extracting metadata information from Secboot images. The class ID ‘CSecureImageParser‘ is included in the default privilege set, therefore the CSecureImageParser class is available to every TA.

14.213 CSharedBuffer

14.213.1 Detailed Description

Class CSharedBuffer implements ISharedBuffer interface.

The class ID ‘SharedBuffer‘ is included in the default privilege set, therefore the CSharedBuffer class is available to every TA.

14.214 CSPCOM

14.214.1 Detailed Description

Class CSPCOM implements ISPCOM interface.

The class ID ‘SPCOM‘ is not included in the default privilege set. It needs to be added to the privileges section in the metadata to allow the TA to use the CSPCOM class.

14.215 CSPI

14.215.1 Detailed Description

Class CSPI implements ISPI interface.

The class ID ‘SPI‘ is not included in the default privilege set. It needs to be added to the privileges section in the metadata to allow the TA to use the CSPI class.

14.216 CSWCrypto

14.216.1 Detailed Description

Class CSWCrypto is software implementation of ICrypto interface in user space.

The class ID ‘CSWCrypto‘ is included in the default privilege set, therefore the CSWCrypto class is available to every TA.

14.217 CSwFuse

14.217.1 Detailed Description

Class CSwFuse implements ISwFuse interface.

The class ID ‘CSwFuse‘ is included in the default privilege set, therefore the CSwFuse class is available to every TA.

14.218 CSync

14.218.1 Detailed Description

Class CSync implements ISync interface.

The class ID ‘Sync‘ is included in the default privilege set, therefore the CSync class is available to every TA.

14.219 CTLMM

14.219.1 Detailed Description

Class CTLMM implements ITLMM interface.

The class ID ‘TLMM‘ is not included in the default privilege set. It needs to be added to the privileges section in the metadata to allow the TA to use the CTLMM class.

14.220 CTransNSAddr

14.220.1 Detailed Description

Class CTransNSAddr implements ITranslateAddr interface.

The class ID ‘TransNSAddr‘ is not included in the default privilege set. It needs to be added to the privileges section in the metadata to allow the TA to use the CTransNSAddr class.

14.221 CUptime

14.221.1 Detailed Description

Class CUptime implements IUptime interface.

The class ID ‘Uptime‘ is included in the default privilege set, therefore the CUptime class is available to every TA.

14.222 CVenusSecureChannel

14.222.1 Detailed Description

This service is not currently available in this product.

14.223 CVideoSecureChannel

14.223.1 Detailed Description

This service is not currently available in this product.

14.224 CVMSessionKey

14.224.1 Detailed Description

Class CVMSessionKey implements IVMSessionKey interface.

The class ID ‘VMSessionKey‘ is not included in the default privilege set.

14.225 CWhitelistBypass

14.225.1 Detailed Description

The class ID ‘WhitelistBypass‘ privilege is used to grant a TA permission to bypass whitelist memory validation checks. This validation is bypassed during shared memory registration when using the legacy qsee_register_shared_buffer interface or when acquiring memory regions using the new IHlosRegionFinder interface. Due to the potential risk of the privilege, this privilege is not part of the default privilege set granted to all TAs. Note: This privilege should only be granted where necessary as it creates additional risk by circumventing validation checks.

14.226 CWhitelistBypassRO

14.226.1 Detailed Description

The class ID ‘WhitelistBypassRO‘ is not included in the default privilege. The WhitelistBypassRO privilege is used to grant a TA reduced permission to access a memory region acquired using the IHlosRegionFinder interface despite failing whitelist validation on the region. This privilege has no effect on whitelist address validation on calls to the legacy qsee_register_shared_buffer interface. Due to the potential risk of the privilege. Note: This privilege should only be granted where necessary as it creates additional risk by circumventing validation checks.

14.227 Object-Based QTEE Service Classes to REEs

14.227.1 Detailed Description

Modules

- CAppClient
- CAppLoader
- CDeviceAttestation
- CFeatureVersions
- CPeripheralState
- CPmPon
- CPVCLicense
- CQTEEEnvInfo
- CRegisterListenerCBO
- CTLOCKey
- CVMDeviceUniqueKey

14.228 CAppClient

14.228.1 Detailed Description

Class CAppClient implements IAppClient interface. This class provides an interface to obtain app-provided functionalities.

The class ID ‘AppClient‘ is not included in the default privilege set.

14.229 CAppLoader

14.229.1 Detailed Description

Class CAppLoader implements IAppLoader interface.

The class ID ‘AppLoader‘ is not included in the default privilege set. It needs to be added to the privileges section in the metadata to allow the TA to use the CAppLoader class.

14.230 CFeatureVersions

14.230.1 Detailed Description

Class CFeatureVersions implements IFeatureVersions interface.

The class ID ‘FeatureVersions‘ is not included in the default privilege set. It needs to be added to the privileges section in the metadata to allow the TA to use the CFeatureVersions class.

14.231 CPeripheralState

14.232 CPmPon

14.232.1 Detailed Description

Class CPmPon implements IPmPon interface.

The class ID ‘PmPon‘ stands for Power Management PowerON.

14.233 CPVCLicense

14.233.1 Detailed Description

Class CPVCLicense implements IPVCLicense interface.

The class ID ‘PVCLicense‘ is not included in the default privilege set. It needs to be added to the privileges section in the metadata to allow the TA to use the CPVCLicense class.

14.234 CQTEEEnvInfo

14.234.1 Detailed Description

Class CQTEEEnvInfo implements IQTEEEnvInfo interface.

14.235 CRegisterListenerCBO

14.235.1 Detailed Description

Class CRegisterListenerCBO implements IRegisterListenerCBO interface.

The class ID ‘RegisterListenerCBO‘ is not included in the default privilege set. It needs to be added to the privileges section in the metadata to allow the TA to use the CRegisterListenerCBO class.

14.236 CTLOCKey

14.236.1 Detailed Description

Class CTLOCKey implements ITLOCKey interface.

The class ID ‘CTLOCKey‘ is not included in the default privilege set. It needs to be added to the privileges section in the metadata to allow the TA to use the CTLOCKey class.

14.237 CVMDeviceUniqueKey

14.237.1 Detailed Description

Class CVMDeviceUniqueKey implements IVMDeviceUniqueKey interface.

The class ID ‘VMDeviceUniqueKey‘ is not included in the default privilege set.

14.238 APIs available to native REE clients to retrieve an IClientEnv

14.238.1 Detailed Description

14.238.2 Function Documentation

14.238.2.1 int TZCom_getClientEnvObject (Object * *obj*)

Client get a new IClientEnv obj

This interface retrieves the client's credentials and registers the client with the TEE. The

Parameters

out	*obj*	client's IClientEnv object

Returns

0 on success,
negative otherwise

14.238.2.2 int TZCom_getFdObject (int *fd,* Object * *obj*)

Wrap a file descriptor into an object

Parameters

in	*fd*	the fd to be wrapped into an obj
out	*obj*	fd object that takes ownership of fd i.e.release of obj would close fd.

Returns

0 on success,
negative otherwise

14.238.2.3 int TZCom_getRootEnvObject (Object * *obj*)

Get root object

This is used to create a root IClientEnv object. It supports default 16 callback threads and 4K callback request buffer.

Parameters

out	*rootobj*	root IClientEnv Obj

Returns

Object_OK on success,
Object_ERROR otherwise.

14.238.2.4 int TZCom_getRootEnvObjectWithCB (size_t *cbthread_cnt,* size_t *cbbuf_len,* Object * *obj*)

Get root object with configurable callback threads and/or callback buffer size

This is used by client to create a root IClientEnv Obj when number of callback threads and/or callback buffer size are different from default values

Parameters

in	*cbthread_cnt*	cb thread count
in	*cbbuf_len*	cb request buffer length
out	*rootobj*	root IClientEnv Obj

Returns

Object_OK on success,
Object_ERROR otherwise.

14.239 APIs to initialize the off-target QTEE emulation environment

14.239.1 Detailed Description

14.239.2 Function Documentation

14.239.2.1 int32_t qtee_emu_deinit (void)

Deinitialize the QTEE Emulation environment.

Returns

Object_OK on success
Object_ERROR otherwise.

14.239.2.2 int32_t qtee_emu_init (const char * *config*)

Initialize the QTEE Emulation environment.

Optionally, a path to a user config file can be passed and the settings in this file will overwrite the defaults. If the default settings are desired, a null pointer can be passed instead.

Parameters

in	*config*	Path to user config file. May be NULL to use default config.

Returns

Object_OK on success
Object_ERROR otherwise.

14.240 APIs to map a trusted application into a read-only buffer

14.240.1 Detailed Description

14.240.2 Data Structure Documentation

14.240.2.1 struct ta_image_data

Data fields

Type	Parameter	Description
void *	buffer	Read-only buffer containing the trusted application.
size_t	buffer_size	Size of the buffer.

14.240.3 Function Documentation

14.240.3.1 int map_trusted_application (char const * *appname,* ta_image_data * *img_data*)

Map a trusted application into a read-only buffer.

Parameters

in	*appname*	The full path to the file.
out	*img_data*	Pointer to a ta_image_data struct containing the TA buffer and its size.

Returns

0 on success.
Non-zero if mapping fails.

14.240.3.2 void unmap_trusted_application (ta_image_data * *img_data*)

Unmap a trusted application referenced by a ta_image_data struct.

Safe to call on a zeroed struct (e.g. if initialized with {0} or memset to 0).

Parameters

in	*img_data*	Pointer to a ta_image_data struct containing the TA buffer and its size.

14.241 APIs to create memory objects

14.241.1 Detailed Description

14.241.2 Function Documentation

14.241.2.1 int32_t MemObj_getInfo (Object *mo,* void ** *addr,* size_t * *len,* uint32_t * *perm*)

Returns the address space information for a memory object represented by the passed object, this needs to be a map object associated with the memobj to be able to provide mapping information.

Parameters

in	*memObj*	Memory object.
out	*addr*	Address of memory range
out	*len*	Length of memory range
out	*perm*	Permissions for the range

Returns

Object_OK on success

14.241.2.2 uint32_t MemObj_getPerms (Object *memObj*)

Get the permissions of a memory object.

Parameters

in	*memObj*	Memory object.

Returns

A uint32_t representing the IMemRegion permissions.

14.241.2.3 bool MemObj_isMapObj (Object *mo*)

Returns true if the object passed is a map object representing a locked memory object, i.e. mapped/fixed in memory.

Parameters

in	*memObj*	Memory object.

Returns

A bool indicating if this is a map object, i.e. locked memory object.

14.241.2.4 Object MemObj_new (void * *addr,* size_t *size,* uint32_t *perms,* bool *isCached*)

Create a new memory object.

This API creates a memory region object of a given source buffer.

The memory object works directly on the source buffer. The supplied permissions have to be a subset of the access rights of the source buffer. The supplied permissions are only respected by APIs using the memory object. Access to the source buffer via its address is not prevented.

Parameters

in	*addr*	Pointer to a buffer representing the memory region.
in	*sizes*	Size of the buffer.
in	*perms*	IMemRegion permissions of the memory object.
in	*isCached*	Whether MO memory allocated from cached pool or not.

Returns

Object representing the memory region on success,
Object_NULL otherwise.

14.241.2.5 void MemObj_setPerms (Object *memObj,* uint32_t *perms*)

Set the permissions of a memory object.

Parameters

out	*memObj*	Memory object.
in	*perms*	IMemRegion permissions of the memory object.

14.242 APIs to create a simulated ION buffer

14.242.1 Detailed Description

14.242.2 Data Structure Documentation

14.242.2.1 class BufferAllocatorSimHandle

Public member functions

- int AllocSim (size_t size, size_t alignment)
- int32_t **getFd** ()
- void * **getBuffer** ()
- size_t **getBufferSize** ()

Static Public Member Functions

- static void freeSim (int32_t fd)
- static void * **getAddrFromFd** (int32_t fd)

14.242.3 Function Documentation

14.242.3.1 int BufferAllocatorSimHandle::AllocSim (size_t *size*, size_t *alignment*)

Allocate a simulated ion buffer represented by a handle.

Parameters

in	*handle*	Pointer to the handle representing a simulated ion buffer.
in	*size*	Size of the allocated buffer.
in	*alignment*	Alignment of the allocated buffer.

Returns

0 on success,
negative otherwise.

14.242.3.2 static void BufferAllocatorSimHandle::freeSim (int32_t *fd*) [static]

Frees a simulated ion buffer represented by a handle.

Parameters

in	*handle*	Pointer to the handle representing a simulated ion buffer.

14.243 APIs to manage dependencies of GP trusted applications

14.243.1 Detailed Description

14.243.2 Data Structure Documentation

14.243.2.1 struct TEEC_uuid_dep_names

Data fields

Type	Parameter	Description
char	app_name[MAX_UUID_DEP_NAME_COUNT][GP_APP_NAME_BUF_SIZE]	Array containing the names of the dependencies of a GP TA.
size_t	num	Number of TA dependencies.

14.243.3 Define Documentation

14.243.3.1 #define GP_APP_NAME_BUF_SIZE 64

Maximum buffer size for a TAs name.

14.243.3.2 #define GP_APP_PATH_BUF_SIZE 4096

Maximum buffer size for a TA's path.

14.243.3.3 #define MAX_UUID_DEP_NAME_COUNT 8

Maximum number of TA dependencies.

14.243.4 Function Documentation

14.243.4.1 TEEC_Result getAppFromUUID (const TEEC_UUID * *uuid,* char * *app_name,* char * *ta_path,* TEEC_uuid_dep_names * *deps*)

Get the application names of GP trusted applications identified by its UUID.

A GP client application identifies GP trusted applications by its UUID, whereby QTEE identifies and loads them by its name. This API is called from the GP client layer to determine the name of the GP trusted application as well as the name of its dependencies.

Note: An off-target GP client application must implement this function.

Parameters

in	*uuid*	Pointer to the UUID of the called GP trusted application.
out	*app_name*	Pointer to the name of the called GP trusted application.
out	*ta_path*	Pointer to the path where the called GP trusted application is installed. All TAs used by an application must be installed to the same path.
out	*deps*	Pointer to a struct containing the names of the dependencies of the GP trusted application.

Returns

TEEC_SUCCESS on success,
TEEC_GENERIC_ERROR otherwise.

14.244 Services Emulated in QTEEEmu

CAppMessage

CCipher

CClockConfig

CCmac

CCredentials

CCrypto CSWCrypto

CDataCache

CFileTable

CHash

CHmac

CListener

CMemRegionPermEscalator

COEMBuf

CPrngSource

CSecureChannel

CSecureImageParser

CSharedBuffer

CSync

CUptime

14.245 CCmac

14.245.1 Detailed Description

Class CCmac implements ICmac interface.

14.246 CFileTable

14.246.1 Detailed Description

Class CFileTable implements IFileTable interface in user space.

The class ID ‘CFileTable‘ is included in the default privilege set, therefore the CFileTable class is available to every TA.

This class provides an interface to the SFS to map file descriptors to persistent object handles

14.247 CSecureImage

14.247.1 Detailed Description

Class CSecureImage implements ISecureImage interface.

The class ID ‘SecureImage‘ is not included in the default privilege set. It needs to be added to the privileges section in the metadata to allow the TA to use the CSecureImage class.

14.248 CSecureImagePrivate

14.248.1 Detailed Description

Class CSecureImagePrivate implements ISecureImagePrivate interface.

The class ID ‘SecureImagePrivate‘ is not included in the default privilege set. It needs to be added to the privileges section in the metadata to allow the TA to use the CSecureImagePrivate class.

14.249 CStorageAccess

14.249.1 Detailed Description

Class CStorageAccess implements IStorageAccess interface.

The class ID ‘StorageAccess‘ is not included in the default privilege set. It needs to be added to the privileges section in the metadata to allow the TA to use the CStorageAccess class.

14.250 CStorRd

14.250.1 Detailed Description

Class CStorRd implements IStorRd interface. It is used for accessing resilient partitions in storage.

The class ID ‘StorRd‘ is not included in the default privilege set. It needs to be added to the privileges section in the metadata to allow the TA to use the CStorRd class.

14.251 CSWHash

14.251.1 Detailed Description

Class CSWHash is software implementation of ISWHash interface in user space.

The class ID ‘CSWHash‘ is included in the default privilege set, therefore the CSWHash class is available to every TA.

14.252 CSystemManager

14.252.1 Detailed Description

Class CSystemManager implements ISystemManager interface.

The class ID ‘SystemManager‘ is included in the default privilege set, therefore the CSystemManager class is available to every TA. It provides interface for system management operations such as fetching credentials of systems.

14.253 CTALog

14.253.1 Detailed Description

Class CTALog implements ITALog interface.

The class ID ‘TALog‘ is included in the default privilege set, therefore the CTALog class is available to every TA.

The class ID ‘LogSink‘ is for internal use for QTEE only, i.e. trying to open an object with this UID will always fail.

14.254 CTrustedCameraDriver

14.254.1 Detailed Description

Class CTrustedCameraDriver implements ITrustedCameraDriver interface.

This class provides an interface to protect camera PHY lanes in QSEE

14.255 IPFMUpdater

14.255.1 Detailed Description

14.255.2 Function Documentation

14.255.2.1 method Finalize (out buffer *TransactionInfo*)

Finalizes the transaction.

Parameters

out	*TransactionInfo.*	If the licenses don't contain Transaction Extension, the response CBOR array will be empty

CDDL for TransactionInfo

```
TransactionInfo CBOR Response:
[
    +{
        ?"TransactionID":tstr,
        ?"InstalledLicenses":[+bstr],
        ?"RevokedLicenses":[+bstr],
        ?"Status":uint64_t
    }
]
```

TransactionInfo Example

```
TransactionInfo:
[
    {
        "TransactionID": "b02f096b-ecd4-47af-b5bd-38ce798846a3",
        "InstalledLicenses": [h'4535DFC452A36B57794EC60EE15661F0'],
        "RevokedLicenses": [h'688FF7B4A0FB0DAA1D8C7D798E1F665F'],
        "Status": 0
    },
    {
        "TransactionID": "f1c5111f-4d46-4d7e-a6ab-a113af77ca70",
        "InstalledLicenses": [h'4FBB16D936B909F1A2E525DA1B5AA540'],
        "RevokedLicenses": [h'191068B7BBD82C83C5ACCB9FC71D9FA9'],
        "Status": 0
    }
]
```

TransactionInfo CBOR value descriptions

- TransactionID - Unique identifier for a given transaction for a bundle of licenses.
- InstalledLicenses - Serial numbers of licenses installed for this transaction.
- RevokedLicenses - Serial numbers of licenses revoked for this transaction.
- Status - 0 for transaction success, 1 for transaction incomplete, 3 for transaction failure.

Returns

Object_OK on success

14.255.2.2 method Update (in buffer *PFMLicense*)

Collects all the licenses for a transaction.

Parameters

in	*PFMLicense*	- Feature license in PFM or DER format.

Returns

Object_OK on success

14.256 IProvKeyList

14.256.1 Detailed Description

14.256.2 Function Documentation

14.256.2.1 method appendKey (in buffer *keyBlob,* in buffer *serviceCtx*)

Adds the given key blob to the internal key list.

Parameters

in	*keyBlob*	Buffer containing key material.
in	*serviceCtx*	Bytes that make sense to the 3rd party cloud and is passed along by the QWES cloud back to the 3rd party cloud.

Returns

Object_OK on success.

14.256.2.2 method clear ( )

Clears the internal key list.

Returns

Object_OK on success.

14.257 ISecureImageELFInfoSnapshot

14.257.1 Detailed Description

Inner interface for retrieving information from a parsed ELF image. The methods defined here can also be used with the IVerifiedSecureImageELFInfo interface.

14.257.2 Function Documentation

14.257.2.1 method getELFHeader (out ISecureImage_elfHeader *elfHeader*)

Returns an ISecureImage_elfHeader with information from the parsed image's ELF Header.

Parameters

out	*elfHeader*	ISecureImage_elfHeader with relevant ELF Header fields.

Returns

Object_OK on success. Any other ISecureImage error code on failure.

14.257.2.2 method getProgramHeader (in uint32 *phdrIndex,* out ISecureImage_programHeader *programHeader*)

Returns an ISecureImage_programHeader with all the information from the Program Header at phdrIndex.

Parameters

in	*phdrIndex*	Index of the requested Program Header.
out	*programHeader*	ISecureImage_programHeader with all Program Header fields.

Returns

Object_OK on success. Any other ISecureImage error code on failure.

14.258 ISecureImageELFInfo

14.258.1 Detailed Description

An extension to ISecureImageELFInfoSnapshot adding getProgramHeaders.

14.258.2 Function Documentation

14.258.2.1 method getProgramHeaders (out ISecureImage_programHeader[ ] *programHeaders*)

Returns an ISecureImage_programHeader array of desired type with all the information from all the Program Headers.

Parameters

out	*programHeaders*	ISecureImage_programHeader with all Program Header fields.

Returns

Object_OK on success. Any other ISecureImage error code on failure.

14.259 IVerifiedSecureImageELFInfoSnapshot

14.259.1 Detailed Description

Inner interface for retrieving information from a verified ELF image. The methods defined in ISecureImageELFInfoSnapshot can also be used by this interface.

14.259.2 Function Documentation

14.259.2.1 method getBufParameter (in uint32 *param,* out buffer *value*)

Returns the parameter indicated by param in the output buffer.

Parameters

in	*param*	One of the PARAM_OUT IDs corresponding to a buffer parameter defined in the ISecureImage interface.
out	*value*	The requested parameter.

Returns

Object_OK on success. Any other ISecureImage error code on failure.

14.259.2.2 method getIntParameter (in uint32 *param,* out uint32 *value*)

Returns the parameter indicated by param.

Parameters

in	*param*	One of the PARAM_OUT IDs corresponding to an int parameter defined in the ISecureImage interface.
out	*value*	The requested parameter.

Returns

Object_OK on success. Any other ISecureImage error code on failure.

14.260 IVerifiedSecureImageELFInfo

14.260.1 Detailed Description

An extension to IVerifiedSecureImageELFInfoSnapshot, adding getObjParameter() and getProgramHeaders().

14.260.2 Function Documentation

14.260.2.1 method getObjParameter (in uint32 *param,* out interface *value*)

Returns the parameter indicated by param in the output object.

Parameters

in	*param*	One of the PARAM_OUT IDs corresponding to an object parameter defined in the ISecureImage interface.
out	*value*	The requested parameter.

Returns

Object_OK on success. Any other ISecureImage error code on failure.

14.261 IVerifiedSecureImageMDTInfoSnapshot

14.261.1 Detailed Description

Outer interface for MDT use cases.

14.261.2 Function Documentation

14.261.2.1 method verifySegmentFromBuf (in uint32 *phdrIndex,* in buffer *segment,* out buffer *decryptedSegment*)

Verifies the provided segment. Decrypts the segment if encrypted.

Parameters

in	*phdrIndex*	The segment's Program Header index.
in	*segment*	Segment data to be verified and decrypted if encrypted.
out	*decryptedSegment*	Decrypted segment data, if segment is encrypted. If NULL or equal to segment, decryption is performed in-place.

Returns

Object_OK on success. Any other ISecureImage error code on failure.

14.261.2.2 method verifySegmentFromMemObj (in uint32 *phdrIndex,* in IMemRegion *segment*)

Verifies the provided segment. Decrypts the segment if encrypted.

Parameters

in	*phdrIndex*	The segment's Program Header index.
in	*segment*	Segment data to be verified and decrypted if encrypted.

Returns

Object_OK on success. Any other ISecureImage error code on failure.

14.261.2.3 method verifySegmentsFromBuf (in buffer *segments,* out buffer *decryptedSegments*)

Verifies the provided segments. Decrypts the segments if encrypted.

Parameters

in	*segments*	Segment data to be verified and decrypted if encrypted. Should contain the data for all the image's segments.
out	*decryptedSegments*	Decrypted segment data, if segments are encrypted. If NULL or equal to segments, decryption is performed in-place.

Returns

Object_OK on success. Any other ISecureImage error code on failure.

14.261.2.4 method verifySegmentsFromMemObj (in IMemRegion *segments*)

Verifies the provided segments. Decrypts the segments if encrypted.

Parameters

in	*segments*	Segment data to be verified and decrypted if encrypted. Should contain the data for all the image's segments.

Returns

Object_OK on success. Any other ISecureImage error code on failure.

14.262 IVerifiedSecureImageMDTInfo

An extension to IVerifiedSecureImageMDTInfoSnapshot, adding getObjParameter() and getProgramHeaders().

14.263 ISecureImageSnapshot

14.263.1 Detailed Description

WARNING!!! Do not add any new methods to this interface. You may add but not modify error codes and constants.

14.263.2 Function Documentation

14.263.2.1 method getEncryptionPublicKey (in uint32 *encryptionScheme,* in buffer *decryptionSoftwareContext,* in uint32 *featureId,* out buffer *key*)

Returns the encryption public key for the Encryption Scheme, Software Context, and Feature ID provided.

Parameters

in	*encryptionScheme*	One of the encryption schemes enumerated elsewhere in this documentation.
in	*decryptionSoftware Context*	The Software Context for the desired public key. Ignored by some Encryption Schemes.
in	*featureId*	The Feature ID for the desired public key.
out	*key*	The encryption public key for the Encryption Scheme, Software Context, and Feature ID provided.

Returns

Object_OK on success. Any other ISecureImage error code on failure.

14.263.2.2 method getSecureImageELFInfoFromMemObj (in IMemRegion *elf,* out ISecureImageELFInfo *secureImageElfInfo*)

Returns an ISecureImageELFInfo Object which allows the caller to get ELF Header and Program Header info without verifying the ELF.

Parameters

in	*elf*	ELF image.
out	*secureImageElfInfo*	Object from which to retrieve image information, such as the ELF Header and Program Headers.

Returns

Object_OK on success. Any other ISecureImage error code on failure.

14.263.2.3 method getServiceVersion (out uint32 *majorVersion,* out uint32 *minor Version*)

Returns the version of the ISecureImage service. This is just a convenience method. Version mismatch is not a problem with Mink IPC. If a caller asks for a method that is not implemented, Mink provides a "not implemented" error. However, in some cases, this could be used to determine the root cause of the mismatch. For example, to report mismatched interface versions in the log.

Parameters

out	*majorVersion*	The SERVICE_MAJOR_VERSION (see above) of the service.
out	*minorVersion*	The SERVICE_MINOR_VERSION (see above) of the service.

Returns

Object_OK on success. Any other ISecureImage error code on failure.

14.263.2.4 method verifyELFFromMemObj (in IMemRegion *elf,* in buffer *decryptionSoftwareContext,* in ISecureImage_verifyInputs *verifyInputs,* out IVerifiedSecureImageELFInfo *verifiedSecureImageElfInfo*)

Verifies the provided ELF and decrypts it if encrypted. Returns an IVerifiedSecureImageELFInfo Object which allows the caller to get information about the verified ELF.

Parameters

in	*elf*	The ELF image to verify and decrypt, if encrypted.
in	*decryptionSoftwareContext*	Software Context to use for decryption of the ELF, if encrypted. Ignored by some Encryption Schemes.
in	*verifyInputs*	ISecureImage_verifyInputs struct containing the enforcement policy and other verification constraints.
out	*verifiedSecureImageElfInfo*	Object which allows the caller to get information about the verified ELF.

Returns

Object_OK on success. Any other ISecureImage error code on failure.

14.263.3 Variable Documentation

14.263.3.1 error ERROR_CREDENTIALS

Failed to decrypt segment data.

14.263.3.2 error ERROR_CREDENTIALS_DID

User provided credentials object is invalid.

14.263.3.3 error ERROR_CTX_DTOR

Attempted to access verification information about an unverified image.

14.263.3.4 error ERROR_DECRYPT

The image's Feature ID does not match that provided by the user.

14.263.3.5 error ERROR_DECRYPT_UNSUPPORTED

Failed to set QBEC key context.

14.263.3.6 error ERROR_ELF_HDR_PARSE

Failed to retrieve or parse data. Possible causes include a buffer overflow, or less data was provided in a buffer than expected.

14.263.3.7 error ERROR_EXTRACT_DATA

Input parameter is invalid.

14.263.3.8 error ERROR_EXTRACT_FEATURE_ID

Failed to extract the Secondary Software ID from the provided image.

14.263.3.9 error ERROR_EXTRACT_SECONDARY_SW_ID

Obtained key is not of the expected size.

14.263.3.10 error ERROR_FEATURE_ID_MISMATCH

Failed to extract the Feature ID from the provided image.

14.263.3.11 error ERROR_FEATURE_UNSUPPORTED

Segment decryption not supported for image format.

14.263.3.12 error ERROR_GET_KEY

Failed to obtain a key because it is not available in software.

14.263.3.13 error ERROR_HASH

Multiple Program Header Table segments were detected in the provided image.

14.263.3.14 error ERROR_HASH_SEGMENT_CONTEXT

Another segment overlaps with the hash table segment.

14.263.3.15 error ERROR_HASH_SEGMENT_MISSING

Failed to parse a Program Header Table segment from a provided image buffer.

14.263.3.16 error ERROR_HASH_SEGMENT_OVERLAP

Failed to find a hash table segment in the provided image.

14.263.3.17 error ERROR_HASH_TABLE_SEGMENT_VERSION

Failed to clean up a context.

14.263.3.18 error ERROR_HASH_TABLE_SIZE

Multiple hash table segments were detected in the provided image.

14.263.3.19 error ERROR_HW_KEY_FACTORY_DERIVE

Failed to open the Hardware Key Factory service.

14.263.3.20 error ERROR_HW_KEY_FACTORY_OPEN

Image is not OEM encrypted, but user included ENFORCE_OEM_ENCRYPTED in the enforcement policy.

14.263.3.21 error ERROR_IMAGE_UNVERIFIED

Failed to map a memory object.

14.263.3.22 error ERROR_INVALID_ENCRYPTION_SCHEME

Invalid Enforcement Policy is set.

14.263.3.23 error ERROR_INVALID_ENFORCEMENT_POLICY

Error returned for QBEC FEATURE_ID request.

14.263.3.24 error ERROR_INVALID_PARAMETER

Failed to allocate heap memory.

14.263.3.25 error ERROR_INVALID_QBEC_FEATURE_ID

Failed to extract QBEC public key from device.

14.263.3.26 error ERROR_KEY_ACCESS

Failed to get some parameter from the Key Manager service.

14.263.3.27 error ERROR_KEY_MANAGER_GENERATE_KEY

Failed to set a parameter in the Key Manager service.

14.263.3.28 error ERROR_KEY_MANAGER_GET_PARAM

Failed to pass some parameter to the Key Manager service.

14.263.3.29 error ERROR_KEY_MANAGER_OPEN

Failed to derive a key from hardware source.

14.263.3.30 error ERROR_KEY_MANAGER_SET_PARAM

Failed to open the Key Manager service.

14.263.3.31 error ERROR_KEY_SIZE

Failed to obtain a key.

14.263.3.32 error ERROR_MEM_REGION_OPEN

Failed to copy memory.

14.263.3.33 error ERROR_MEMORY_COPY

Failed to initialize a UIE context.

14.263.3.34 error ERROR_MEMORY_MAP

Failed to bind to a memory object.

14.263.3.35 error ERROR_MULTIPLE_HASH_SEGMENTS

Failed to initialize the secboot context.

14.263.3.36 error ERROR_MULTIPLE_PHDR_SEGMENTS

Another segment overlaps with the Program Header Table segment.

14.263.3.37 error ERROR_OEM_UNENCRYPTED

Image is not QTI signed, but user included ENFORCE_QTI_SIGNED in the enforcement policy.

14.263.3.38 error ERROR_OEM_UNSIGNED

Failed to verify the hash table segment of the provided image.

14.263.3.39 error ERROR_PHDR_PARSE

Failed to parse the ELF Header from a provided image buffer.

14.263.3.40 error ERROR_PHDR_SEGMENT_MISSING

Parsed hash table length was found to be 0.

14.263.3.41 error ERROR_PHDR_SEGMENT_OVERLAP

Failed to find a Program Header Table Segment in the image.

14.263.3.42 error ERROR_QBEC_CXT

Image has an unsupported Hash Table Segment version.

14.263.3.43 error ERROR_QBEC_PARAMETER

Failed to initialize a QBEC context.

14.263.3.44 error ERROR_QBEC_PUBLIC_KEY

Failed to get QBEC parameters from Hash Table Segment.

14.263.3.45 error ERROR_QBEC_SET_KEY_CONTEXT

Invalid Encryption Scheme.

14.263.3.46 error ERROR_QTI_UNSIGNED

Image is not OEM signed, but user included ENFORCE_OEM_SIGNED in the enforcement policy.

14.263.3.47 error ERROR_UIE_CXT

Failed to obtain a Distinguished ID from the provided credentials object.

14.263.3.48 error ERROR_VERIFY_SEGMENT

Failed to hash provided segment data.

14.263.3.49 error ERROR_VERIFY_SIGNATURE

Provided segment data failed verification against the hash table segment.

14.263.3.50 const uint32 PARAM_OUT_AUTHORITIES = 20

Should be used with getIntParameter(). IDs for signing authority. Specified which authorities signed and/or encrypted the image.

14.263.3.51 const uint32 PARAM_OUT_OEM_AR_VERSION = 0

Requested feature is not supported. Should be used with getIntParameter(). IDs for OU/Metadata OEM-specific outputs.

14.263.3.52 const uint32 PARAM_OUT_OEM_ROT = 30

Should be used with getBufParameter(). IDs for RoT outputs.

14.263.3.53 const uint32 PARAM_OUT_QBEC_DE_SCHEME_ID = 43

Should be used with getIntParameter(). The Data Encryption Scheme ID from QBEC. Use this to determine the data encryption parameter format.

14.263.3.54 const uint32 PARAM_OUT_QBEC_ENC_SEG_BITMASK = 60

Should be called with getBufParameter(). This returns the bitmask of the encrypted segments, and should be called after getting the number of segments in the image. The number of segments can be determined by calling getElfHeader() and reading the e_phnum member of the ISecureImage_elfHeader output structure. This bitmask is in little endian order and always a multiple of 4 bytes (uint32). The output buffer needs at most 4∗(ceil((e_phnum-1)/32)+1) bytes. Count the number of set bits (popcount) to determine the number of encrypted segments. This is only available for certain QBEC_DE_SCHEME_IDs.

14.263.3.55 const uint32 PARAM_OUT_QBEC_FEATURE_ID = 40

Should be used with getIntParameter(). IDs for QBEC FEATURE_ID output.

14.263.3.56 const uint32 PARAM_OUT_QBEC_GCM_IVS_AUTH_TAGS = 61

Should be called with getBufParameter(). This returns the initialization vectors and authentication tags for each encrypted segment as described by PARAM_OUT_QBEC_ENC_SEG_BITMASK. The size of this parameter is 28∗N bytes where N is the number of set bits in the output of PARAM_OUT_QBEC_ENC_SEG_BITMASK. These are output as [IV_0, TAG_0, IV_1, TAG_1, ..., IV_N-1, TAG_N-1] where each IV is 12 bytes and each authentication tag is 16 bytes. This is only available for certain QBEC_DE_SCHEME_IDs.

14.263.3.57 const uint32 PARAM_OUT_QBEC_KM_SCHEME_ID = 42

Should be used with getIntParameter(). The Key Management Scheme ID from QBEC. Use this to determine the key format.

14.263.3.58 const uint32 PARAM_OUT_QBEC_UNWRAPPED_KEY_OBJ = 51

Should be used with getObjParameter(). This returns a hardware key derived from the QBEC struct used for decryption. This is only available for certain QBEC_KM_SCHEME_IDs.

14.263.3.59 const uint32 PARAM_OUT_QBEC_UNWRAPPED_SW_KEY = 50

Should be called with getBufParameter(). This returns an unwrapped software key from the QBEC struct used for decryption. Note that a software key is inherently less secure than a hardware key! This is only available for certain QBEC_KM_SCHEME_IDs.

14.263.3.60 const uint32 PARAM_OUT_QBEC_WRAPPED_ENC_KEY = 41

Should be used with getBufParameter(). This returns a buffer containing QBEC Data Encryption parameters along with the wrapped encryption key. The wrapped encryption key is of length 68 bytes and the Data Encryption parameter is of size (16 + N∗(12+16)) bytes where N is the number of encrypted segments. This is only available for certain QBEC_KM_SCHEME_IDs.

14.263.3.61 const uint32 PARAM_OUT_QTI_AR_VERSION = 10

Should be used with getIntParameter(). IDs for OU/Metadata QTI-specific outputs.

14.264 ISecureImage

14.264.1 Detailed Description

Outer-level interface for parsing and optionally validating ELF and MDT images.

14.264.2 Function Documentation

14.264.2.1 method verifyMDTFromBuf (in buffer *mdt,* in buffer *decryptionSoftwareContext,* in ISecureImage_verifyInputs *verifyInputs,* out IVerifiedSecureImageMDTInfo *verifiedSecureImageMdtInfo*)

Verifies the provided MDT. Returns an IVerifiedSecureImageMDTInfo Object which allows the caller to verify segments and get information about the verified MDT.

Parameters

in	*mdt*	The MDT to verify.
in	*decryptionSoftwareContext*	Software Context to use for decryption of the segments via the segment verification APIs, if encrypted. Ignored by some Encryption Schemes.
in	*verifyInputs*	ISecureImage_verifyInputs struct containing the enforcement policy and other verification constraints.
out	*verifiedSecureImageMdtInfo*	Object which allows the caller to verify segments and get information about the verified MDT.

Returns

Object_OK on success. Any other ISecureImage error code on failure.

14.265 IStorageAccess

14.265.1 Detailed Description

14.265.2 Function Documentation

14.265.2.1 method EraseSectors (in uint32 *stor_type,* in uint32 *slot,* in uint32 *partition,* in StorAccess_Guid *GUID,* in uint64 *start_sector,* in uint64 *num_sectors*)

Parameters

in	*stor_type*	- Type of Storage device
in	*slot*	- Slot number of the storage device
in	*partition*	- Partition number of the storage device
in	*GUID*	- GUID ID
in	*start_sector*	- Address of start sector
in	*num_sectors*	- Number of Sectors to erase

Returns

Object_OK on success

14.265.2.2 method ReadBytes (in uint32 *stor_type,* in uint32 *slot,* in uint32 *partition,* in StorAccess_Guid *GUID,* out buffer *data_buffer,* in uint64 *start_byte,* in uint64 *num_bytes*)

Parameters

in	*stor_type*	- Type of Storage device
in	*slot*	- Slot number of the storage device
in	*partition*	- Partition number of the storage device
in	*GUID*	- GUID ID
out	*data_buffer*	- Data buffer to read the data
in	*start_byte*	- Address of start byte
in	*num_bytes*	- Number of bytes to be read

Returns

Object_OK on success

14.265.2.3 method WriteBytes (in uint32 *stor_type,* in uint32 *slot,* in uint32 *partition,* in StorAccess_Guid *GUID,* in buffer *data_buffer,* in uint64 *start_byte,* in uint64 *num_bytes*)

Parameters

in	*stor_type*	- Type of Storage device
in	*slot*	- Slot number of the storage device
in	*partition*	- Partition number of the storage device

in	*GUID*	- GUID ID
in	*data_buffer*	- Data buffer to write the data
in	*start_byte*	- Address of start byte
in	*num_bytes*	- Number of bytes to be written

Returns

Object_OK on success

14.266 IStorRd

14.266.1 Detailed Description

14.266.2 Function Documentation

14.266.2.1 method EnableBackupAndWriteProtect ( )

Enables resiliency backup and applies write protect for all the partitions that require it based on the devcfg / device tree settings.

Returns

Object_OK on success.

14.266.2.2 method OpenPartition (in IStorRd_Guid *guid,* out interface *IStorRd Partition*)

Opens a partition

Parameters

in	*guid*	GUID of the partition to open
out	*IStorRdPartition*	IStorRdPartition object

Returns

Object_OK on success.

14.266.2.3 method SetManufacturingMode (in uint8 *mode_val*)

Enables/disables manufacturing mode for resiliency driver

Parameters

in	*mode_val*	1 to enable, 0 to disable manufacturing mode

Returns

Object_OK on success.

14.266.2.4 method StartCodePartitionIterator (in uint32 *dev_type,* in uint32 *slot,* in uint32 *lun,* out uint32 *expected_num_entries,* out interface *IStorRdIterator*)

Starts a partition iterator session to iterate all the code partitions

Parameters

in	*dev_type*	Device type
in	*slot*	Device slot
in	*lun*	Device logical unit number
out	*expected_num_ entries*	Holds value of the expected number of partitions
out	*IStorRdIterator*	IStorRdIterator object

Returns

Object_OK on success.

14.267 IStorRdPartition

14.267.1 Detailed Description

14.267.2 Function Documentation

14.267.2.1 method GetPartitionInfo (out IStorRd_PartitionInfo *partition_info*)

Returned when a hash mismatch is seen Gets partition information

Parameters

out	*partition_info*	Pointer to information about the partition

Returns

Object_OK on success.

14.267.2.2 method ReadLba (in uint64 *lba,* in uint64 *lba_count,* out buffer *data_buffer*)

Reads data from a partition

Parameters

in	*lba*	Starting sector to read
in	*lba_count*	Number of sectors to read
out	*data_buffer*	Pointer to a buffer for the data

Returns

Object_OK on success.

14.267.2.3 method WriteLba (in uint64 *lba,* in uint64 *lba_count,* in buffer *data_buffer*)

Writes data to a partition

Parameters

in	*lba*	Starting sector to write
in	*lba_count*	Number of sectors to write
out	*data_buffer*	Pointer to a buffer for the data

Returns

Object_OK on success.

14.267.3 Variable Documentation

14.267.3.1 error ERROR_ALREADY_RUNNING

Returned when resources are not available

14.267.3.2 error ERROR_DEVICE_ERROR

Returned when an operation is not supported

14.267.3.3 error ERROR_HASH_GENERATION

Returned when no partition is found

14.267.3.4 error ERROR_HASH_MISMATCH

Returned when hash calculation fails

14.267.3.5 error ERROR_IN_MFG_MODE

Returned for APIs that can only be called in manufacturing mode

14.267.3.6 error ERROR_INVALID_PARAM

Errors

14.267.3.7 error ERROR_KEY_CLEAR

Returned when the KDF fails

14.267.3.8 error ERROR_KEY_GENERATION

Returned when HMAC calculation fails

14.267.3.9 error ERROR_KEY_WRAP

Returned when clearing the key fails

14.267.3.10 error ERROR_MAC_GENERATION

Returned when a MAC mismatch is seen

14.267.3.11 error ERROR_MAC_MISMATCH

Returned for any device errors

14.267.3.12 error ERROR_NOT_IN_MFG_MODE

Returned when an operation is already in progress

14.267.3.13 error ERROR_NOT_SUPPORTED

Returned when an invalid parameter is encountered

14.267.3.14 error ERROR_OUT_OF_RESOURCES

Returned when wrapping the key fails

Returned for any device errors

14.267.3.15 error ERROR_PARTITION_NOT_FOUND

Returned when writing to a write-protected partition

14.267.3.16 error ERROR_READONLY

Returned for APIs that can only be called out of manufacturing mode

14.268 IStorRdIterator

14.268.1 Detailed Description

14.268.2 Function Documentation

14.268.2.1 method GetNextCodePartitionEntry (out IStorRd_GptEntry *gpt_entry,* out uint32 *result*)

Returned when an operation is already in progress Gets the next code partition GPT entry

Parameters

out	*entry*	Pointer to the next GPT partition entry
out	*result*	Indication of the job status (running or not, complete)

Returns

Object_OK on success.

14.268.3 Variable Documentation

14.268.3.1 error ERROR_NOT_SUPPORTED

Returned when an invalid parameter is encountered

14.268.3.2 const uint32 STATUS_NOT_STARTED = 1

Partition iterator status values

14.269 ISWHash

14.269.1 Detailed Description

14.269.2 Function Documentation

14.269.2.1 method final (out buffer *digest*)

Computes the digest hash value.

Parameters

out	*digest*	Message digest hash.

Returns

Object_OK – Successful.
Object_ERROR – Any error encountered.

14.269.2.2 method hash (in buffer *plain,* out buffer *digest*)

Computes the digest hash value with the input data.

Parameters

in	*plain*	Plain text message to hash.
out	*digest*	Message hash digest.

Returns

Object_OK – Successful.
Object_ERROR – Any error encountered.

14.269.2.3 method init (in int32 *algo*)

Initializes hash context based on requested hash algorithm and acceleration engine. If engine parameter is passed as zero, the default engine will be used. The default engine is defined in UCLIB configuration file.

Parameters

in	*hash*	type enumerated in UCLIB_HASH_ALG

Returns

Object_OK – Successful.
Object_ERROR – Any error encountered.

14.269.2.4 method reset ( )

Resets hash context; does not reset the key.

Returns

Object_OK – Successful.
Object_ERROR – Any error encountered.

14.269.2.5 method squeeze (out buffer *digest*)

Squeeze step of XOF Hash. Squeezes arbitraily many bytes The max digest length returned is equal to the squeeze rate.

Parameters

out	*digest*	Message digest hash.

Returns

Object_OK – Successful.
Object_ERROR – Any error encountered.

14.269.2.6 method squeezeblocks (out buffer *digest*)

Squeeze step of XOF Hash. Squeezes full blocks of rate bytes each. Can be called multiple times to keep squeezing

Parameters

out	*digest*	Message digest hash.

Returns

Object_OK – Successful.
Object_ERROR – Any error encountered.

14.269.2.7 method update (in buffer *plain*)

Hashes data into the hash context. Also used for absorb operation for XOF functions

Parameters

in	*plain*	Plain text message to hash.

Returns

Object_OK – Successful.
Object_ERROR – Any error encountered.

14.270 ISystemManager

14.270.1 Detailed Description

14.270.2 Function Documentation

14.270.2.1 method getCredentials (in uint32 *systemId,* out ICredentials *credentials*)

Number of Systems managed by this System Manager. Returns the ICredentials object associated with the System.

Parameters

in	*systemId*	system ID of the system against which the credentials are requested.
out	*credentials*	The Credentials Object against the system ID.

Returns

Object_OK on Success. ERROR_INVALID_SYSTEM_ID when the provided systemId is invalud.

14.270.2.2 method registerHandler (in ICredentials *credentials,* in INotification Handler *handler*)

Register a TA handler that will be executed when a new EL2 entity will be set.

Parameters

in	*credentials*	The SystemCredentials we want to be notified on.
in	*handler*	The INotificationHandler that the client will register.

Returns

Object_OK on success. Object_ERROR when there is an error upon registration.

14.270.3 Variable Documentation

14.270.3.1 const uint32 EL2SYSTEM_ID = 1

Constant representing QTEE

14.270.3.2 error ERROR_INVALID_STATE

The given SYSTEM_ID does not correspond to a valid System.

14.270.3.3 const uint32 MODEM_ID = 2

Constant representing the system currently operating in EL2. (UEFI, DLME, HYP)

14.270.3.4 const uint32 NUM_IDs = 3

Constant representing Modem

14.270.3.5 const uint32 QTEE_ID = 0

The given system is not ready for update. Constants for different systems

14.271 ITALog

14.271.1 Detailed Description

14.271.2 Function Documentation

14.271.2.1 method registerSink (in interface *ILogSink*)

Register a log sink to the TA. If a log sink has been already registered, the new one will replace it. Only a single log sink can be registered by a TA.

Parameters

`in`	*ILogSink*	A log sink object that will implement the custom logging function. When passing an Object_NULL sink, the current log sink will be deregistered.

14.272 ILogSink

14.272.1 Detailed Description

14.272.2 Function Documentation

14.272.2.1 method onLog (in uint8 *pri,* in buffer *msg*)

Method signature for log sinks to implement. Developers should avoid manipulating the logging mask (qsee_log_set_mask()) to avoid infinite recursion when qsee_log() is called.

Parameters

in	*pri*	Priority of the message.
in	*msg*	Message to be handled by the logger.

14.273 ITrustedCameraDriver

14.273.1 Detailed Description

14.273.2 Function Documentation

14.273.2.1 method dynamicConfigureFDPort (in uint32 *protect*)

Description: Dynamic configuration to allow secure/non-secure FD port on all the CSIDs

Parameters

in	*protect*	To secure or non-secure the FD port.

Returns

Object_OK on success.

14.273.2.2 method dynamicConfigurePortsV2 (in PortInfo[] *port_info*)

Description: This method allows dynamically configuring ports of camera hw_type.

Parameters

in	*port_info*	Array indicating which ports to be configured for the hw_type specified. They can be protected or not-protected depending on the request.

Returns

Object_OK on success.

14.273.2.3 method dynamicProtectSensor (in ITCDriverSensorInfo *phy_info*)

Description: This method allows protecting a camera sensor based on the sensor information provided.

Parameters

in	*phy_info*	Camera HW settings required for securing the usecase

Returns

Object_OK on success.

14.273.2.4 method getVersion (out uint32 *arch_ver,* out uint32 *max_ver,* out uint32 *min_ver*)

Description: This method gets the current version info.

Parameters

out	*arch_ver*	architecture version info
out	*max_ver*	max version info
out	*min_ver*	min version info

Returns

Object_OK on success.

14.273.2.5 method registerNotifyCB (in ITrustedCameraDriverNotify *cb*)

Description: This method registers a callback to the trusted camera driver.

Parameters

in	*cb*	the callback called by the trusted camera driver

Returns

Object_OK on success.

14.273.3 Variable Documentation

14.273.3.1 error ERROR_NOT_ALLOWED

Error codeInvalid request to configure FD port

14.273.3.2 const uint32 IFE = 0

Camera HW types

15 Data Structure Documentation

15.1 EnforceHWFeatureId_t Struct Reference

Data Fields

- uint32 **featureID**
- uint32 **HWFeatureStatus**

15.1.1 Detailed Description

15.1.2 IPFM Overview

IPFM is the interface to the Platform Feature Manager (PFM), which provides access to a managed collection of feature licenses. PFM also refers to an ASCII-encoded license format based on PEM. A complete description of feature licenses is beyond the scope of this document. From a high level, a feature license is an X.509 certificate with extensions that define what features are enabled and under what conditions. The extensions provide a blacklist. If a value specified by any extension does not match either the query or the state of the device, license validation will fail, with an error code that indicates the first extension where a mismatch was detected. Technically, a Feature ID or FID is an integer that identifies a feature. However, the terms feature, Feature ID, and FID are used somewhat interchangeably in the rest of this document.

15.1.2.1 CBOR

Many of the IPFM methods use CBOR to encode inputs or outputs. A full description of CBOR is beyond the scope of this document. One reason it is used in the API is so responses can include additional data in the future without requiring changes to client code or the .idl. With that in mind, client code should not depend on the exact order of items in a CBOR map in a response, or the exact number of items in a CBOR structure/array. Clients should skip unexpected trailing items in arrays and unknown keys in maps, and should only fail if any expected fields are NOT present.

CBOR inputs and outputs are described using both CDDL and a CBOR diagnostic format, which a JSO N-like text version of the CBOR data. Square braces indicate a CBOR array, which may be heterogeneous. Curly braces indicate a CBOR map. CBOR maps used by IPFM always use text strings as map key values.

Note

CBOR permits both definite- and indefinite-length arrays and maps, but IPFM methods only accept and produce definite-length arrays and maps. Depending on the CBOR library used by the client, client code may have to provide additional options to generate definite-length arrays and maps.

15.1.2.2 Distinguished Names

The normal purpose of an X.509 certificate is for a certificate Issuer to assert that a public key is owned by a Subject. The Issuer and Subject are identified by Distinguished Names. A Distingushed Name (DN) is a series of well-known attribute types (keys) and values. Most key / value pairs are optional. Several IPFM methods return the Issuer and Subject DNs in a CBOR map. The values in the CBOR map have been converted from the raw DER encoding into a CBOR map. Both the keys and values in the map are TSTR values. The following key values are supported at a minimum.

- `"O"` - Organization
- `"OU"` - Organizational Unit
- `"CN"` - Common Name

15.1.2.3 Common Validation Error Codes

A number of IPFM methods return errors that indicate which restriction failed to match, as described above, or a more general problem with the X.509 certificate structure. These error codes all include _CERT in their names and are listed here.

Restriction Mismatch Errors

- ERROR_CERT_PKHASH
- ERROR_CERT_FEATUREID
- ERROR_CERT_EXPIRED
- ERROR_CERT_OEM
- ERROR_CERT_HWVERSION
- ERROR_CERT_LICENSEE_HASH
- ERROR_CERT_DEVICEID
- ERROR_CERT_NOTYETVALID
- ERROR_CERT_PRODUCTID

Errors for corrupted or incorrectly generated licenses

- ERROR_INVALID_CERT
- ERROR_CERT_NOT_TRUSTED
- ERROR_CERT_GENERAL_ERR
- ERROR_CERT_LEAF_IS_CA

15.1.2.4 Other Common Error Codes

Object_ERROR_SIZE_OUT

Many IPFM methods return variable-length data in CBOR format. If the output buffer provided by the client is not large enough to hold the data, Object_ERROR_SIZE_OUT will be returned, and the output buffer length will be set to a value large enough to contain the data. Note that this is an upper bound on the data length, and not necessarily the exact data length. If the client allocates a new buffer of this length and calls the method again with the new buffer, it will succeed, and the buffer length will

be set to the exact data size.

Object_ERROR

Object_ERROR can be returned from most methods. This indicates an internal error of some kind. It may indicate a system misconfiguration. For example, it can be returned if PFM can not find the persistent storage for licenses. The circumstances under which the error occurred and the qsee_log should be reported to the maintainers of PFM.

Other error codes

Note that while every effort is made to correctly list the errors that each method can return, the list should not be considered exhaustive. A method may return error codes in addition to those listed, either now or in the future.

15.1.3 Field Documentation

15.1.3.1 uint32 EnforceHWFeatureId_t::featureID

15.1.3.2 uint32 EnforceHWFeatureId_t::HWFeatureStatus

15.2 IAppController Class Reference

Public member functions

- **IAppController** (Object impl)
- virtual int32_t **openSession** (uint32_t cancelCode_val, uint32_t connectionMethod_val, uint32_t connectionData_val, uint32_t paramTypes_val, uint32_t exParamTypes_val, const void ∗i1_ptr, size_t i1_len, const void ∗i2_ptr, size_t i2_len, const void ∗i3_ptr, size_t i3_len, const void ∗i4_ptr, size_t i4_len, void ∗o1_ptr, size_t o1_len, size_t ∗o1_lenout, void ∗o2_ptr, size_t o2_len, size_t ∗o2_lenout, void ∗o3_ptr, size_t o3_len, size_t ∗o3_lenout, void ∗o4_ptr, size_t o4_len, size_t ∗o4_lenout, const ProxyBase &imem1, const ProxyBase &imem2, const ProxyBase &imem3, const ProxyBase &imem4, uint32_t ∗memrefOutSz1_ptr, uint32_t ∗memrefOutSz2_ptr, uint32_t ∗memrefOutSz3_ptr, uint32_t ∗memrefOutSz4_ptr, ProxyBase &session, uint32_t ∗retValue_ptr, uint32_t ∗retOrigin_ptr)
- virtual int32_t **unload** ()
- virtual int32_t **getAppObject** (ProxyBase &obj)
- virtual int32_t **installCBO** (uint32_t uid_val, const ProxyBase &obj)
- virtual int32_t **disconnect** ()
- virtual int32_t **restart** ()

Additional Inherited Members

15.3 IAppControllerImplBase Class Reference

Protected Member Functions

- virtual int32_t **invoke** (ObjectOp op, ObjectArg ∗a, ObjectCounts k)

Additional Inherited Members

15.4 IAppLoader Class Reference

Public member functions

- **IAppLoader** (Object impl)
- virtual int32_t **loadFromBuffer** (const void ∗appElf_ptr, size_t appElf_len, IAppController &appController)
- virtual int32_t **loadFromRegion** (const ProxyBase &appElf, IAppController &appController)
- virtual int32_t **loadEmbedded** (const void ∗appName_ptr, size_t appName_len, IAppController &appController)
- virtual int32_t **connect** (const void ∗appName_ptr, size_t appName_len, IAppController &appController)

Additional Inherited Members

15.5 IAppLoaderImplBase Class Reference

Protected Member Functions

- virtual int32_t **invoke** (ObjectOp op, ObjectArg ∗a, ObjectCounts k)

Additional Inherited Members

15.6 IClientEnv Class Reference

Public member functions

- **IClientEnv** (Object impl)
- virtual int32_t **open** (uint32_t uid_val, ProxyBase &obj)
- virtual int32_t **registerLegacy** (const void ∗credentials_ptr, size_t credentials_len, ProxyBase &clientEnv)
- virtual int32_t **registerAsClient** (const IIO &credentials, ProxyBase &clientEnv)
- virtual int32_t **registerWithWhitelist** (const IIO &credentials, const uint32_t ∗uids_ptr, size_t uids_len, ProxyBase &clientEnv)
- virtual int32_t **notifyDomainChange** ()
- virtual int32_t **registerWithCredentials** (const ICredentials &credentials, ProxyBase &clientEnv)
- virtual int32_t **loadCmnlibFromBuffer** (const void ∗cmnlibElf_ptr, size_t cmnlibElf_len)
- virtual int32_t **configTaRegion** (uint64_t appRgnAddr_val, uint32_t appRgnSize_val)
- virtual int32_t **adciAccept** ()
- virtual int32_t **adciShutdown** ()

Additional Inherited Members

15.7 IClientEnvImplBase Class Reference

Protected Member Functions

- virtual int32_t **invoke** (ObjectOp op, ObjectArg ∗a, ObjectCounts k)

Additional Inherited Members

15.8 ICredentials Class Reference

Public member functions

- **ICredentials** (Object impl)
- virtual int32_t **getPropertyByIndex** (uint32_t index_val, void ∗name_ptr, size_t name_len, size_t ∗name_lenout, void ∗value_ptr, size_t value_len, size_t ∗value_lenout)
- virtual int32_t **getValueByName** (const void ∗name_ptr, size_t name_len, void ∗value_ptr, size_t value_len, size_t ∗value_lenout)

Additional Inherited Members

15.9 ICredentialsImplBase Class Reference

Protected Member Functions

- virtual int32_t **invoke** (ObjectOp op, ObjectArg ∗a, ObjectCounts k)

Additional Inherited Members

15.10 IIAppController Class Reference

Public member functions

- virtual int32_t **openSession** (uint32_t cancelCode_val, uint32_t connectionMethod_val, uint32_t connectionData_val, uint32_t paramTypes_val, uint32_t exParamTypes_val, const void ∗i1_ptr, size_t i1_len, const void ∗i2_ptr, size_t i2_len, const void ∗i3_ptr, size_t i3_len, const void ∗i4_ptr, size_t i4_len, void ∗o1_ptr, size_t o1_len, size_t ∗o1_lenout, void ∗o2_ptr, size_t o2_len, size_t ∗o2_lenout, void ∗o3_ptr, size_t o3_len, size_t ∗o3_lenout, void ∗o4_ptr, size_t o4_len, size_t ∗o4_lenout, const ProxyBase &imem1, const ProxyBase &imem2, const ProxyBase &imem3, const ProxyBase &imem4, uint32_t ∗memrefOutSz1_ptr, uint32_t ∗memrefOutSz2_ptr, uint32_t ∗memrefOutSz3_ptr, uint32_t ∗memrefOutSz4_ptr, ProxyBase &session, uint32_t ∗retValue_ptr, uint32_t ∗retOrigin_ptr)=0
- virtual int32_t **unload** ()=0
- virtual int32_t **getAppObject** (ProxyBase &obj)=0
- virtual int32_t **installCBO** (uint32_t uid_val, const ProxyBase &obj)=0
- virtual int32_t **disconnect** ()=0
- virtual int32_t **restart** ()=0

Static Public Attributes

- static const uint32_t **CBO_INTERFACE_WAIT** = UINT32_C(1)
- static const int32_t **ERROR_APP_SUSPENDED** = INT32_C(10)
- static const int32_t **ERROR_APP_BLOCKED_ON_LISTENER** = INT32_C(11)
- static const int32_t **ERROR_APP_UNLOADED** = INT32_C(12)

- static const int32_t **ERROR_APP_IN_USE** = INT32_C(13)
- static const int32_t **ERROR_NOT_SUPPORTED** = INT32_C(14)
- static const int32_t **ERROR_CBO_UNKNOWN** = INT32_C(15)
- static const int32_t **ERROR_APP_UNLOAD_NOT_ALLOWED** = INT32_C(16)
- static const int32_t **ERROR_APP_DISCONNECTED** = INT32_C(17)
- static const int32_t **ERROR_USER_DISCONNECT_REJECTED** = INT32_C(18)
- static const int32_t **ERROR_STILL_RUNNING** = INT32_C(19)
- static const int32_t **ERROR_APP_RESTART_FAILED** = INT32_C(20)

Static Protected Attributes

- static const ObjectOp **OP_openSession** = 0
- static const ObjectOp **OP_unload** = 1
- static const ObjectOp **OP_getAppObject** = 2
- static const ObjectOp **OP_installCBO** = 3
- static const ObjectOp **OP_disconnect** = 4
- static const ObjectOp **OP_restart** = 5

15.11 IIAppLoader Class Reference

Public member functions

- virtual int32_t **loadFromBuffer** (const void ∗appElf_ptr, size_t appElf_len, IAppController &appController)=0
- virtual int32_t **loadFromRegion** (const ProxyBase &appElf, IAppController &appController)=0
- virtual int32_t **loadEmbedded** (const void ∗appName_ptr, size_t appName_len, IAppController &appController)=0
- virtual int32_t **connect** (const void ∗appName_ptr, size_t appName_len, IAppController &appController)=0

Static Public Attributes

- static const int32_t **ERROR_INVALID_BUFFER** = INT32_C(10)
- static const int32_t **ERROR_PIL_ROLLBACK_FAILURE** = INT32_C(11)
- static const int32_t **ERROR_ELF_SIGNATURE_ERROR** = INT32_C(12)
- static const int32_t **ERROR_METADATA_INVALID** = INT32_C(13)
- static const int32_t **ERROR_MAX_NUM_APPS** = INT32_C(14)
- static const int32_t **ERROR_NO_NAME_IN_METADATA** = INT32_C(15)
- static const int32_t **ERROR_ALREADY_LOADED** = INT32_C(16)
- static const int32_t **ERROR_EMBEDDED_IMAGE_NOT_FOUND** = INT32_C(17)

- static const int32_t **ERROR_TZ_HEAP_MALLOC_FAILURE** = INT32_C(18)
- static const int32_t **ERROR_TA_APP_REGION_MALLOC_FAILURE** = INT32_C(19)
- static const int32_t **ERROR_CLIENT_CRED_PARSING_FAILURE** = INT32_C(20)
- static const int32_t **ERROR_APP_UNTRUSTED_CLIENT** = INT32_C(21)
- static const int32_t **ERROR_APP_NOT_LOADED** = INT32_C(22)
- static const int32_t **ERROR_APP_MAX_CLIENT_CONNECTIONS** = INT32_C(23)
- static const int32_t **ERROR_APP_BLACKLISTED** = INT32_C(24)
- static const int32_t **ERROR_APP_ARCH_NOT_SUPPORTED** = INT32_C(25)
- static const int32_t **ERROR_VM_NOT_WHITELISTED_TO_CONNECT_TA** = INT32_C(26)
- static const int32_t **ERROR_ELF_LOADING** = INT32_C(27)

Static Protected Attributes

- static const ObjectOp **OP_loadFromBuffer** = 0
- static const ObjectOp **OP_loadFromRegion** = 1
- static const ObjectOp **OP_loadEmbedded** = 2
- static const ObjectOp **OP_connect** = 3

15.12 IIClientEnv Class Reference

Public member functions

- virtual int32_t **open** (uint32_t uid_val, ProxyBase &obj)=0
- virtual int32_t **registerLegacy** (const void ∗credentials_ptr, size_t credentials_len, ProxyBase &clientEnv)=0
- virtual int32_t **registerAsClient** (const IIO &credentials, ProxyBase &clientEnv)=0
- virtual int32_t **registerWithWhitelist** (const IIO &credentials, const uint32_t ∗uids_ptr, size_t uids_len, ProxyBase &clientEnv)=0
- virtual int32_t **notifyDomainChange** ()=0
- virtual int32_t **registerWithCredentials** (const ICredentials &credentials, ProxyBase &clientEnv)=0
- virtual int32_t **loadCmnlibFromBuffer** (const void ∗cmnlibElf_ptr, size_t cmnlibElf_len)=0
- virtual int32_t **configTaRegion** (uint64_t appRgnAddr_val, uint32_t appRgnSize_val)=0
- virtual int32_t **adciAccept** ()=0
- virtual int32_t **adciShutdown** ()=0

Static Public Attributes

- static const int32_t **ERROR_COMMONLIBS_ALREADY_LOADED** = INT32_C(10)
- static const int32_t **ERROR_APP_REGION_ALREADY_SET** = INT32_C(11)

Static Protected Attributes

- static const ObjectOp **OP_open** = 0
- static const ObjectOp **OP_registerLegacy** = 1
- static const ObjectOp **OP_registerAsClient** = 2
- static const ObjectOp **OP_registerWithWhitelist** = 3
- static const ObjectOp **OP_notifyDomainChange** = 4
- static const ObjectOp **OP_registerWithCredentials** = 5
- static const ObjectOp **OP_loadCmnlibFromBuffer** = 6
- static const ObjectOp **OP_configTaRegion** = 7
- static const ObjectOp **OP_adciAccept** = 8
- static const ObjectOp **OP_adciShutdown** = 9

15.13 IICredentials Class Reference

Public member functions

- virtual int32_t **getPropertyByIndex** (uint32_t index_val, void ∗name_ptr, size_t name_len, size_t ∗name_lenout, void ∗value_ptr, size_t value_len, size_t ∗value_lenout)=0
- virtual int32_t **getValueByName** (const void ∗name_ptr, size_t name_len, void ∗value_ptr, size_t value_len, size_t ∗value_lenout)=0

Static Public Attributes

- static const int32_t **ERROR_NOT_FOUND** = INT32_C(10)
- static const int32_t **ERROR_NAME_SIZE** = INT32_C(11)
- static const int32_t **ERROR_VALUE_SIZE** = INT32_C(12)

Static Protected Attributes

- static const ObjectOp **OP_getPropertyByIndex** = 0
- static const ObjectOp **OP_getValueByName** = 1

15.14 IIIO Class Reference

Public member functions

- virtual int32_t **getLength** (uint64_t ∗len_ptr)=0
- virtual int32_t **readAtOffset** (uint64_t offset_val, void ∗data_ptr, size_t data_len, size_t ∗data_lenout)=0
- virtual int32_t **writeAtOffset** (uint64_t offset_val, const void ∗data_ptr, size_t data_len)=0

Static Public Attributes

- static const int32_t **ERROR_OFFSET_OUT_OF_BOUNDS** = INT32_C(10)
- static const int32_t **ERROR_SOURCE_BUFFER_TOO_LARGE** = INT32_C(11)
- static const int32_t **ERROR_INVALID_BUFFER_AND_OFFSET** = INT32_C(12)

Static Protected Attributes

- static const ObjectOp **OP_getLength** = 0
- static const ObjectOp **OP_readAtOffset** = 1
- static const ObjectOp **OP_writeAtOffset** = 2

15.15 IIO Class Reference

Public member functions

- **IIO** (Object impl)
- virtual int32_t **getLength** (uint64_t ∗len_ptr)
- virtual int32_t **readAtOffset** (uint64_t offset_val, void ∗data_ptr, size_t data_len, size_t ∗data_lenout)
- virtual int32_t **writeAtOffset** (uint64_t offset_val, const void ∗data_ptr, size_t data_len)

Additional Inherited Members

15.16 IIOImplBase Class Reference

Protected Member Functions

- virtual int32_t **invoke** (ObjectOp op, ObjectArg ∗a, ObjectCounts k)

Additional Inherited Members

15.17 ImplBase Class Reference

15.18 ISecureImage_elfHeader Struct Reference

Data Fields

- uint64 **e_ident_class**
- uint64 **e_type**
- uint64 **e_machine**
- uint64 **e_entry**
- uint64 **e_phoff**
- uint64 **e_flags**
- uint64 **e_ehsize**
- uint64 **e_phentsize**
- uint64 **e_phnum**

15.18.1 Field Documentation

15.18.1.1 uint64 ISecureImage_elfHeader::e_ehsize

15.18.1.2 uint64 ISecureImage_elfHeader::e_entry

15.18.1.3 uint64 ISecureImage_elfHeader::e_flags

15.18.1.4 uint64 ISecureImage_elfHeader::e_ident_class

15.18.1.5 uint64 ISecureImage_elfHeader::e_machine

15.18.1.6 uint64 ISecureImage_elfHeader::e_phentsize

15.18.1.7 uint64 ISecureImage_elfHeader::e_phnum

15.18.1.8 uint64 ISecureImage_elfHeader::e_phoff

15.18.1.9 uint64 ISecureImage_elfHeader::e_type

15.19 ISecureImage_programHeader Struct Reference

Data Fields

- uint64 **p_type**
- uint64 **p_offset**
- uint64 **p_vaddr**
- uint64 **p_paddr**
- uint64 **p_filesz**
- uint64 **p_memsz**
- uint64 **p_flags**
- uint64 **p_align**

15.19.1 Field Documentation

15.19.1.1 uint64 ISecureImage_programHeader::p_align

15.19.1.2 uint64 ISecureImage_programHeader::p_filesz

15.19.1.3 uint64 ISecureImage_programHeader::p_flags

15.19.1.4 uint64 ISecureImage_programHeader::p_memsz

15.19.1.5 uint64 ISecureImage_programHeader::p_offset

15.19.1.6 uint64 ISecureImage_programHeader::p_paddr

15.19.1.7 uint64 ISecureImage_programHeader::p_type

15.19.1.8 uint64 ISecureImage_programHeader::p_vaddr

15.20 ISecureImage_verifyInputs Struct Reference

Data Fields

- uint64 **enforcement_policy**
- uint32 **encryption_scheme**
- uint32 **feature_id**
- uint32 **sw_id**
- uint32 **secondary_sw_id**
- uint32 **oem_min_ar_version**
- uint32 **qti_min_ar_version**

15.20.1 Field Documentation

15.20.1.1 uint32 ISecureImage_verifyInputs::encryption_scheme

15.20.1.2 uint64 ISecureImage_verifyInputs::enforcement_policy

15.20.1.3 uint32 ISecureImage_verifyInputs::feature_id

15.20.1.4 uint32 ISecureImage_verifyInputs::oem_min_ar_version

15.20.1.5 uint32 ISecureImage_verifyInputs::qti_min_ar_version

15.20.1.6 uint32 ISecureImage_verifyInputs::secondary_sw_id

15.20.1.7 uint32 ISecureImage_verifyInputs::sw_id

15.21 ProxyBase Class Reference