Qualcomm Technologies, Inc.

Qualcomm Linux Security Guide - Addendum

80-70018-11A AA

April 4, 2025

Contents

1 Security overview

This addendum serves as a supplementary guide, providing additional details about the security features. It's intended for users with full access to the Qualcomm proprietary software shipped with Qualcomm® Linux®.

Read this addendum in conjunction with the Qualcomm Linux Security Guide, which outlines the framework and tools available to protect your device's hardware and software from potential threats.

Note: See Hardware SoCs that are supported on Qualcomm Linux.

2 Security APIs

The security APIs offer the ability to interface with the Qualcomm Linux Kernel and the device's hardware. They also facilitate various software services that can be run in a trusted execution environment.

2.1 User space APIs

The user space APIs are functions that the Linux operating system uses to communicate with the kernel.

For more information about other APIs, see Qualcomm Linux Security Guide → $SecurityAPIs$.

Global platform TEE client APIs

The global platform for trusted execution environment (TEE) specification client APIs are designed for writing client applications in the Linux-embedded user space.

For more information, see GlobalPlatform Technology TEE Internal Core API Specification Version 1.1.2.50 (Target v1.2).

For the global platform client API header, see:

`LE.QCLINUX.1.0.r1/build-qcom-wayland/tmp-glibc/sysroots-components/qcm6490/securemsm-headers/usr/include/securemsm-headers/TEE_client_api.h`.

SMC invoke MINK APIs

The SMC invoke APIs are accessible to native Linux clients. These APIs use MINK to invoke objects.

int TZCom_getClientEnvObject (Object _ obj)

Description

This function allows the client to obtain a new `IClientEnv` object. This interface retrieves the client credentials and registers the client with the Qualcomm TEE.

Parameters

Parameters		Description
`out`	`obj`	`client IClientEnv object`

Returns

The function returns 0 on success. Else, it returns a negative value.

int TZCom_getFdObject (int fd, Object _ obj)

Description

This function wraps the direct memory access (DMA) allocated file descriptor into an object. This object is referred to as a memory object in the SMC invoke transport protocol to Qualcomm TEE.

Parameters

Parameters		Description
`in`	`fd`	File descriptor that is to be wrapped into an object.
`out`	`obj`	The file descriptor object that takes ownership of the file descriptor, that is, release of the object would close the file descriptor

Returns

The function returns 0 on success. Else, it returns a negative value.

int TZCom_getRootEnvObject (Object _ obj)

Description

This function retrieves the root object. It's used to create a root `IClientEnv` object. It supports the default four callback threads and a 4K callback request buffer.

Parameters

Parameters		Description
out	rootobj	root IClientEnv Obj

Returns

The function returns `Object_OK` upon success. Else, it returns `Object_ERROR`.

int TZCom_getRootEnvObjectWithCB (size_t cbthread_cnt, size_t cbbuf_len, Object _ obj)

Description

This function retrieves the root object with the configurable callback threads and callback buffer size. The client uses this API to create an `IClientEnv` object root when the number of callback threads and callback buffer size differs from the default values.

Parameters

Parameters		Description
in	Cbthread_cnt	Thread count.
out	cbbuf_len	Configurable callback request buffer length.
out	rootobj	root IClientEnv Obj

Returns

The function returns `Object_OK` on success. Else, it returns `Object_ERROR`.

int32_t IClientEnv_open (Object self, uint32 uid_val, Object * obj_ptr)

Description

This function uses the `IClientEnv` interface to allow Linux user space clients to retrieve objects from Qualcomm TEE using SMC invoke. It fetches a service object from the client environment.

Parameters

Parameters		Description
in	uid_val	Identifies a class of service objects.
out	obj_ptr	Instance of the requested service.

Returns

The function returns `Object_OK` on success. Else, it returns `Object_ERROR`.

int32_t IAppLoader_loadFromBuffer (Object self, const void *appElf_ptr, size_t appElf_len, Object *appController_ptr)

Description

This function loads a trusted application. The application executable and linking format (ELF) binary is supplied as a buffer.

Parameters

Parameters		Description
in	appElf_ptr	Buffer containing ELF image.
in	appElf_len	ELF image length.
out	appController_ptr	`IAppController` to access the trusted application.

Returns

The function returns `Object_OK` on success. Else, it returns `Object_ERROR`.

int32_t IAppLoader_loadFromRegion (Object self, Object appElf_val, Object *appController_ptr)

Description

This function loads a trusted application. The application ELF binary is supplied as an `IMemRegion` object.

Parameters

Parameters		Description
in	appElf_val	Region containing ELF image.
out	appController_ptr	IAppController to access the TA.

Returns

The function returns `Object_OK` on success. Else, it returns `Object_ERROR`.

int32_t IAppController_getAppObject (Object self, Object *obj_ptr)

Description

This function retrieves the object that implements the functionalities of the application.

Parameters

Parameters		Description
out	obj_ptr	Returned object.

Returns

The function returns `Object_OK` on success. Else, it returns `Object_ERROR`.

int32_t IAppClient_getAppObject (Object self, const void *appDistName_ptr, size_t appDistName_len, Object *obj_ptr)

Description

This function retrieves the object that implements application-provided functionalities.

Parameters

Parameters		Description
in	appDistName_ptr	Application distinguished name.
in	appDistName_len	Application length.
out	obj_ptr	Returned object.

Returns

Return message	Description
Object_OK	Successful
IAppClient_ERROR_APP_LOAD_FAILED	Failure to load the application.
IAppClient_ERROR_APP_NOT_FOUND	No loaded application with a distinguished name.
IAppClient_ERROR_APP_RESTART_FAILED	Failure to restart the application.
IAppClient_ERROR_APP_UNTRUSTED_CLIENT	Untrusted clients aren't allowed.
IAppClient_ERROR_CLIENT_CRED_PARSING_FAILURE	Failure to parse the client credentials.

int32_t IAppController_unload (Object self)

Description

This function unloads the trusted application. It fails if the application is busy and the caller is expected to try again.

Returns

The function returns Object_OK on success. Else, it returns Object_ERROR.

2.2 Qualcomm TEE APIs

Qualcomm TEE provides a collection of APIs that offer services to secure applications. These services include heap management, logging, access to the secure file system (SFS), interactions with listeners, and functions for cryptography and hashing.

The secure applications run in the Arm® User mode, while these services run in the Supervisor mode.

1. Qualcomm TEE maintains a software interface (SWI) layer along with an internal syscall handler.
2. When one of these APIs is called, a SWI instruction is generated, causing the application thread to switch into the Supervisor mode to execute the service.

3. Subsequently, the thread is switched back to the User mode.

The APIs that are exposed to the secure applications, along with a brief description for each function is outlined in this addendum. For a more comprehensive description and details about parameters, see the header files at `ssg/api/securemsm/trustzone/qsee`.

For more information, see Qualcomm Linux Security Guide → $SecurityAPIs$.

Cipher APIs

A cipher facilitates the encryption and decryption of data. The cipher APIs enable various operations to encrypt and decrypt the data within a cipher context.

qsee_cipher_encrypt()

Description

This function encrypts the provided plain text message using the specified algorithm.

```
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 )
```

Parameters

Parameters		Description
in	cipher_ctx	Pointer to the cipher context.
in	pt	Pointer to the input plain text buffer.
in	pt_len	Length of the input plain text buffer (in bytes).
out	ct	Pointer to the output cipher text buffer.
In,out	ct_len	Pointer to the output buffer length of the cipher text. **Note**: The length value is modified to the actual number of cipher text bytes written.

The memory allocated for the cipher text must be enough to hold the equivalent plain text. If a padding scheme is selected, the cipher text buffer length should be up to one BLOCKSize larger than the plain text length. If the output buffer isn't large enough to hold the encrypted results, an error is returned.

Returns

- SUCCESS – The function executes successfully.
- FAILURE – Any error encountered during encryption.

qsee_cipher_free_ctx()

Description

This function releases all resources associated with a given cipher context.

```
int qsee_cipher_free_ctx ( qsee_cipher_ctx  cipher_ctx )
```

Parameters

Parameters		Description
in	cipher_ctx	Pointer to the cipher context to be deleted

Returns

- SUCCESS - The function executes successfully.
- FAILURE - Any error encountered while freeing the context.

qsee_cipher_get_param()

Description

This function retrieves the parameters associated with a given cipher context.

```
int qsee_cipher_get_param ( const qsee_cipher_ctx  cipher_ctx, QSEE_
CIPHER_PARAM_ET param_id, void  param, uint32_t  param_len )
```

Parameters

Parameters		Description
in	cipher_ctx	Pointer to the cipher context.
in	param_id	Parameter (param) to be retrieved.
out	param	Pointer to a memory location where the parameter will be stored.

Parameters		Description
In, out	param_len	Pointer to the length of the parameter (in bytes). **Note**: The length value is modified to the actual length of the parameter.

Returns

- SUCCESS - The function executes successfully.
- FAILURE - Any error encountered during the retrieval of the parameter.

qsee_cipher_init()

Description

This function initializes a cipher context for either encryption or decryption operations.

```
int qsee_cipher_init ( QSEE_CIPHER_ALGO_ET alg, qsee_cipher_ctx
cipher_ctx )
```

Parameters

Parameters		Description
in	alg	A standard algorithm to be used.
out	cipher_ctx	Double pointer to the cipher context.

Returns

- SUCCESS - The function executes successfully.
- FAILURE - Any error encountered during the initialization of the cipher.

qsee_cipher_reset()

Description

This function resets the cipher context without resetting the key.

```
int qsee_cipher_reset ( qsee_cipher_ctx  cipher_ctx )
```

Parameters

Parameters		Description
in,out	cipher_ctx	Pointer to the cipher context.

Returns

- SUCCESS - The function executes successfully.
- FAILURE - Any error encountered during the reset of the cipher.

qsee_cipher_set_param()

Description

This function modifies the parameters for a specified cipher operation.

```
int qsee_cipher_set_param ( qsee_cipher_ctx  cipher_ctx, QSEE_CI
PHER_PARAM_ET param_id, const void  param, uint32_t param_len)
```

Caution: The Qualcomm TEE off-target environment doesn't support the QSEE_CIPHER_MODE_CTS and QSEE_CIPHER_MODE_XTS cipher modes.

Parameters

Parameters		Description
in,out	cipher_ctx	Pointer to the cipher context.
in	param_id	Parameter to be modified.
in	param	Pointer to the value of the parameter to be set.
in	param_len	Length of the parameter (in bytes).

Returns

- SUCCESS - The function executes successfully.
- FAILURE - Any error encountered while resetting the cipher.

For more cipher APIs, see <TZ.APPS>>qtee_tas/sdk/latest/external/inc/qsee/qsee_cipher.h.

Clock APIs

A clock handles the secure time keeping and cryptographic operations.

qsee_set_bandwidth()

Description

This function sets the bandwidth for the crypto, bus-integrated memory controller (BIMC), and system network on chip (SNoC) clock.

```
uint32_t qsee_set_bandwidth ( void  reqClient, uint32_t reqClientlen,
uint32_t res_req, uint32_t level, uint32_t flags )
```

Parameters

Parameters		Description
in	reqClient	Client that requests the clock.
in	reqClientlen	Length of reqClient name (in bytes).
in	res_req	The resource to vote clocks; all associated clocks are turned on and voted.
in	level	Clock level
in	flags	Flags (currently set to 0)

Returns

- SUCCESS - The function executes successfully.
- FAILURE - Any error encountered during the reset.

For more information about clock APIs, see <TZ.APPS>/qtee_tas/sdk/latest/external/inc/qsee/qsee_clk.h.

CMAC API

A type of message authentication code (MAC) constructed using a block cipher.

qsee_cmac() API

Description

The function uses a specified hash algorithm to create a cipher-based MAC (CMAC) in accordance with the keyed hash MAC (HMAC) (FIPS PUB 198-1) standard.

The following are the supported CMAC algorithms:

- QSEE_CMAC_ALGO_AES_128, which uses the AES-128 cipher algorithm.
- QSEE_CMAC_ALGO_AES_256, which uses the AES-256 cipher algorithm.

```
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)
```

Parameters

Parameters		Description
in	alg	CMAC algorithm to be used.
in	msg	Pointer to the message to be authenticated.
in	msg_len	Length of the message (in bytes).
in	key	Pointer to the input key for the CMAC algorithm.
in	key_len	Length of the key (in bytes).
out	cmac_digest	Pointer to the CMAC digest (memory provided by the caller).
in	cmac_len	Length of the CMAC digest (in bytes). It must be at least QSEE_CMAC_DIGEST_SIZE.

Returns

- QSEE_CMAC_SUCCESS - The function executes successfully.
- QSEE_CMAC_FAILURE - Any error encountered during the creation of CMAC.

For more information about CMAC APIs, see: <TZ.APPS>/qtee_tas/sdk/latest/external/inc/qsee/qsee_cmac.h

Configuration APIs

A property configuration value from the TrustZone kernel.

qsee_cfg_getpropval()

Description

This function retrieves the property configuration value from the TrustZone kernel using a system call.

```
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)
```

Parameters

Parameters		Description
in	PropName	Pointer to a property name (a string).
in	PropNameLen	Length of the property name, including the null character '0'. For example, strnlen() +1.
in	PropId	Property ID.
out	PropBuf	Pointer to an output buffer that is populated with the DAL configuration value.
in	PropBufSz	Size of the output buffer (in bytes).
out	PropBufSzRet	Pointer to the actual size of the populated buffer (in bytes). If the property type is a string, the output size does NOT include the null character '0'.

Returns

- SUCCESS - 0

- FAILURE - Any nonzero value.

For more information about configuration APIs, see <TZ.APPS>/qtee_tas/sdk/latest/external/inc/qsee/qsee_cfg_prop.h.

Core APIs

The core APIs are designed to read, verify, and return the status of various core values for applications operating in a TEE.

qsee_get_device_uuid()

Description

The function provides the universal unique identifier (UUID) of the device.

```
int qsee_get_device_uuid ( uint8_t  uuid_ptr, size_t  uuid_len )
```

Parameters

Parameters		Description
out	uuid_ptr	Pointer to a buffer that is filled with a struct-based Internet engineering task force (IETF) UUID (GP compatible)
in.out	uuid_len	Pointer to the size of the UUID buffer.

Returns

- CALL SUCCESS - 0
- The output buffer must be at least the size of an IETF UUID (32 bytes). The uuid_len pointer is updated to indicate the actual size of the UUID.

qsee_get_fw_component_version()

Description

This function provides the detailed firmware version number that supports the trusted Linux implementation. It includes all the privileged software involved in TEE secure boot and support, excluding the secure OS and TA.

```
int qsee_get_fw_component_version ( uint8_t  version_ptr, size_t
version_len )
```

Parameters

Parameters		Description
out	version_ptr	Pointer to a buffer that is filled with the version number.
in.out	version_len	Pointer to the size of the version buffer.

Returns

- CALL SUCCESS - 0
- The version number is a printable ASCII string, but isn't NULL-terminated. The maximum buffer size is 128 bytes. The version_len pointer is updated to indicate the actual string length of the version.

qsee_get_secure_state()

Description

This function checks the security status of the device.

```
int qsee_get_secure_state ( qsee_secctrl_secure_status_t  status )
```

Parameters

Parameters		Description
out	status	Pointer to the 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 configuration check failed • Bit 5: RPMB provision check failed • Bit 6: Debug check in image certificate failed debug bits • Bit 7: Reserved (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

qsee_get_tz_app_id()

Description

This function returns the application distinguished ID that's stored in the Qualcomm TEE application certificate.

```
int qsee_get_tz_app_id ( uint8_t  tz_app_id, uint32_t id_buf_len )
```

Parameters

Parameters		Description
out	tz_app_id	Pointer to the buffer that will be populated with the application distinguished ID.
in	id_buf_len	Length of the output buffer (in bytes).

Returns

- CALL SUCCESS - 0
- The output buffer must be at least the size of distID (32 bytes).

qsee_hdmi_status_read()

Description

This function reads the status of the HDMI link and the hardware's high-bandwidth digital content protection (HDCP).

```
int qsee_hdmi_status_read ( uint32_t  hdmi_enable, uint32_t  hdmi_
sense, uint32_t  hdcp_auth)
```

Parameters

Parameters		Description
out	hdmi_enable	Indicates whether the HDMI output is enabled.
out	hdmi_sense	Provides the HDMI sense status.
out	hdcp_auth	Indicates the success of the HDCP authentication

Returns

CALL SUCCESS - 0

qsee_is_ns_range()

Description

This function tests if the memory range [start, start + length] falls within the nonsecure memory and is a convenience function that accesses tzbsp_is_ns_area. NULL is a valid value for the start parameter as physical addressing is used.

```
bool qsee_is_ns_range ( const void  start, uint32_t len )
```

Parameters

Parameters		Description
in	start	The starting point of the memory range that is a physical address and is included in the range.
in	len	Length of the memory range (in bytes).

Returns

- TRUE - If the entire area is in nonsecure memory.
- FALSE - If the area contains secure memory.

qsee_is_s_tag_area()

Description

This function tests if the memory range [start, end] is tagged for the virtual machine ID (VMID) and is relevant for the content-protected zone (CPZ) use cases.

```
bool qsee_is_s_tag_area ( uint32_t vmid, uint64_t start, uint64_t end
)
```

Parameters

Parameters		Description
in	vmid	VMID defined in the access control layer (enum ACVirtualMachineId).
in	start	The starting point of the memory range that is a physical address and is included in the range.
in	end	The endpoint of the memory range that is a physical address and is included in the range.

Returns

- TRUE - If the entire area is tagged for the specified VMID.
- FALSE - If the entire area isn't tagged for the specified VMID.

qsee_read_jtag_id()

Description

This function reads the joint test action group (JTAG) ID and returns the JTAG ID value.

```
int qsee_read_jtag_id ( void )
```

qsee_read_serial_num()

Description

This function reads the serial number from the product test engineering (PTE) chain and returns the serial number.

```
int qsee_read_serial_num ( void )
```

qsee_tag_mem()

Description

This function tags all the memory range [start, end] with the specified VMID.

```
int qsee_tag_mem ( uint32_t vmid, uint64_t start, uint64_t end )
```

The table lists the API parameters:

Parameters		Description
in	vmid	VMID defined in the access control layer (enum ACVirtualMachineId)
in	start	The starting point of the memory range that is a physical address and is included in the range.
in	end	The ending point of the memory range that is a physical address and is included in the range.

Returns

CALL SUCCESS - 0

qsee_vm_mem_count()

Description

This function counts the number of 4 kB memory chunks in a specified virtual machine.

```
uint32_t qsee_vm_mem_count ( uint32_t vmid )
```

Parameters

Parameters		Description
in	vmid	VMID defined in the access control layer (enum ACVirtualMachineId).

Returns

- If successful, returns the number of 4 kB chunks in the virtual machine.
- Returns 0 if there is an error.

For more information about core APIs, see TZ.APPS.1.0-01587-KODIAKAAAAANAZT-1/qtee_tas/sdk/latest/external/inc/qsee/qsee_core.h.

Data cache maintenance APIs

These APIs enable clean-up of the data in the cache memory.

qsee_dcache_clean_region()

Description

This function cleans a memory region in the cache. This API writes back any data that's dirty; however, it doesn't invalidate the cache region. Any further access to data in this region results in a cache-hit.

```
void qsee_dcache_clean_region ( void  addr, size_t length )
```

The following table lists the API parameters:

Parameters		Description
in	addr	Starting address of the memory region.
in	length	Length of the memory region.

qsee_dcache_flush_region()

Description

This function cleans and invalidates a memory region in the cache. The data in the cache is written back to the main memory when it's dirty, and the region becomes invalidated. Any further access to the data results in a cache-miss.

```
void qsee_dcache_flush_region ( void  addr, size_t length )
```

Parameters

Parameters		Description
in	addr	Starting address of the memory region.
in	length	Length of the memory region.

qsee_dcache_inval_region()

Description

Invalidates a memory region in the cache. The data in the cache isn't written back to the main memory. Any further access to data in this region results in a cache-miss.

```
void qsee_dcache_inval_region ( void  addr, size_t length )
```

Parameters

Parameters		Description
in	addr	Memory region start address.
in	length	Memory region length.

For more information about data cache maintenance APIs, see <TZ.APPS>/qtee_tas/sdk/latest/external/inc/qsee/qsee_dcache.h.

ECC APIs

An elliptic curve cryptography (ECC) is a key-based data encryption technique. These APIs allow various encryption and decryption operations on the public and private key pairs.

qsee_ECC_hash_to_bigval()

Description

This function converts a hash value to bigval_t.

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

Parameters

Parameters		Description
in	tgt	Pointer to the destination buffer.
in	hashp	Pointer to the hash buffer.
in	hashlen	Buffer size.

Returns

None

qsee_get_random_bytes()

Description

This function generates a software-based random byte.

```
int qsee_get_random_bytes ( void  buf, int len )
```

Parameters

Parameters		Description
in,out	buf	Pointer to a random byte buffer.
in	len	Buffer size.

Returns

- SUCCESS - 0 on success
- FAILURE - Negative

qsee_in_curveP()

Description

This function calculates if the point P lies within an elliptic curve.

```
bool qsee_in_curveP ( QSEE_affine_point_t const  P )
```

Parameters

Parameters		Description
in	P	Pointer to the variable that is an affine point type.

Returns

- TRUE - P is in a curve.
- FALSE - P isn't in the curve.

qsee_SW_ECC_PubPrivate_Key_generate()

Description

This function generates the public and private keys of an ECC. The same keys are used for an elliptic curve diffie-hellman (ECDH) and an elliptic curve digital signature algorithm (ECDSA).

```
int qsee_SW_ECC_PubPrivate_Key_generate ( QSEE_affine_point_t
pubkey, QSEE_bigval_t  privkey )
```

Parameters

Parameters		Description
out	pubkey	Pointer to an ECC public key.
out	privkey	Pointer to an ECC private key.

Returns

- SUCCESS - 0
- FAILURE - Negative

qsee_SW_ECDH_Shared_Key_Derive()

Description

This function generates a shared key from Alice's public key and Bob's private key.

```
int qsee_SW_ECDH_Shared_Key_Derive ( QSEE_affine_point_t  shared_key,
QSEE_bigval_t  privkey, QSEE_affine_point_t  pubkey )
```

Parameters

Parameters		Description
out	shared_key	Pointer to the shared key between Alice and Bob.
in	pubkey	Pointer to an ECC public key.
in	privkey	Pointer to an ECC private key.

Returns

- SUCCESS - 0
- FAILURE - Negative

qsee_SW_ECDSA_Sign()

Description

This function signs the data with an ECC private key.

```
int qsee_SW_ECDSA_Sign ( QSEE_bigval_t const  msgdgst, QSEE_ bigval_t
const  privkey, QSEE_ECDSA_sig_t  sig )
```

Parameters

Parameters		Description
in	msgdgst	Pointer to the message digest.
in	privkey	Pointer to the private key for signing.
out	sig	Pointer to the message signature.

Returns

- SUCCESS - 0
- FAILURE - Negative

qsee_SW_ECDSA_Verify()

Description

This function verifies the data with an ECC public key.

```
int qsee_SW_ECDSA_Verify ( QSEE_bigval_t const  msgdgst, QSEE_affine_
point_t const  pubkey, QSEE_ECDSA_sig_t const  sig )
```

Parameters

Parameters		Description
in	msgdgst	Pointer to the message digest.
in	privkey	Pointer to the private key for signing.
in	sig	Pointer to the message signature.

Returns

- SUCCESS - 0
- FAILURE - Negative

qsee_SW_ECIES_finish()

Description

This function de-initializes and resets the content of an elliptic curve-integrated encryption scheme (ECIES) instance.

```
int qsee_SW_ECIES_finish ( QSEE_ecies_ctx_t  ctx )
```

Parameters

Parameters		Description
int	ctx	Pointer to an ECIES context.

Returns

- UC_E_SUCCESS - The function executes successfully.
- UC_E_INVALID_ARG - An unrecognized argument.

qsee_SW_ECIES_init()

Description

This function initializes an ECIES instance.

```
int qsee_SW_ECIES_init ( QSEE_ecies_ctx_t  ctx, QSEE_ecies_params_t
params )
```

Parameters

Parameters		Description
out	ctx	Pointer to an ECIES context.
in	params	Pointer to the parameters of an ECIES context initialization.

Returns

- UC_E_SUCCESS - The function executes successfully.
- UC_E_FAILURE - The operation failed due to an unknown error.
- UC_E_NOT_ALLOWED - The operation isn't allowed.
- UC_E_INVALID_ARG - An unrecognized argument.

qsee_SW_ECIES_update()

Description

This function updates an ECIES instance.

```
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 )
```

Parameters

Parameters		Description
in,out	ctx	Pointer to an ECIES context.
in	key	Pointer of the key for an encryption or a decryption.
in	purpose	Defines an updated encryption or decryption.
in	msg	Message for an encryption or a 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 the AD data.
in,out	pAD_len	The length of an AD message.
out	Out	An output message of an encryption or a decryption.
in,out	out_len	The length of an output message. The caller needs must ensure that enough memory is allocated to contain the message of the output.

Returns

- UC_E_SUCCESS - The function executes successfully.
- UC_E_FAILURE - The operation failed due to an unknown error.
- UC_E_NOT_ALLOWED - The operation isn't allowed.
- UC_E_INVALID_ARG - An unrecognized argument.

- UC_E_OUT_OF_RANGE - A value out of range.
- UC_E_DATA_INVALID - The data is correct. However, the contents are invalid.

qsee_SW_GENERIC_ECC_affine_point_on_curve()

Description

This function calculates if an ECC affine point is on a curve.

```
bool qsee_SW_GENERIC_ECC_affine_point_on_curve ( QSEE_qrlbn _ecc_
affine_point_t const  Q, QSEE_qrlbn_ecc_domain_t  dp )
```

Parameters

Parameters		Description
in	Q	Pointer to a variable that is an affine point type.
in	dp	Pointer to the domain curve.

Returns

- TRUE - The point Q is on the curve dp.
- FALSE - The point Q isn't on the curve dp.

qsee_SW_GENERIC_ECC_deinit_ex()

Description

This function de-initializes the domain that was initialized with qsee_SW_GENERIC_ECC_init_ex,

qsee_SW_GENERIC_ECC_init_ex, and qsee_SW_GENERIC_ECC_deinit_ex. This function must be used in a pair.

```
int qsee_SW_GENERIC_ECC_deinit_ex ( QSEE_qrlbn_ecc_domain_t  dp )
```

Parameters

Parameters		Description
out	dp	Pointer to an ECC domain context

Returns

- SUCCESS - 0

- FAILURE - Negative

qsee_SW_GENERIC_ECC_init()

Description

This function initializes a domain from the curve hexadecimal strings and the cofactor.

```
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 )
```

Parameters

Parameters		Description
out	dp	Pointer to an ECC domain context
in	modulus	Pointer to the modulus.
in	a	Pointer to the a.
in	b	Pointer to the b.
in	x	Pointer to the x.
in	y	Pointer to the y.
in	n	Pointer to the n.
in	cofactor	Indicates a cofactor.

Returns

- SUCCESS - 0
- FAILURE - Negative

qsee_SW_GENERIC_ECC_keypair_generate()

Description

This function generates an ECC public and private key pair.

```
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 )
```

Parameters

Parameters		Description
out	privkey	Pointer to the private key.
out	pubkey	Pointer to the public key.
in	dp	Pointer to an ECC domain context.

Returns

- SUCCESS - 0
- FAILURE - Negative

qsee_SW_GENERIC_ECC_pubkey_generate()

Description

This function generates an ECC public key for a given private key.

```
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 )
```

Parameters

Parameters		Description
in	privkey	Pointer to the private key.
out	pubkey	Pointer to the public key.
in	dp	Pointer to an ECC domain context.

Returns

- SUCCESS - 0
- FAILURE - Negative

qsee_SW_GENERIC_ECDH_shared_key_derive()

Description

This function generates a shared key from Alice's public key and Bob's private key.

```
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 )
```

Parameters

Parameters		Description
out	shared_key	Pointer to the shared key between Alice and Bob.
in	privkey	Pointer to the private key.
in	pubkey	Pointer to the public key.
in	dp	Pointer to an ECC domain context.

Returns

- SUCCESS - 0
- FAILURE - Negative

qsee_SW_GENERIC_ECDSA_sign()

Description

This function signs the data with an ECC private key.

```
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 )
```

Parameters

Parameters		Description
in	msgdgst	Pointer to a message to be signed.
in	msgdgst_len	Length of the message (in bytes).
in	privkey	Pointer to the private key for signing.
out	sig	Pointer to the signature of the message.
in	dp	Pointer to an ECC domain context.

Returns

- SUCCESS - 0
- FAILURE - Negative

qsee_SW_GENERIC_ECDSA_sign_ex()

Description

This function hashes the input data and signs with an ECC private key. It's a FIPS certifiable ECDSA signing API.

```
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 )
```

Parameters

Parameters		Description
in	hash_alg	The algorithm that is hashed for signing the message.
in	msg	Pointer to a message to be signed.
in	msg_len	Length of the message (in bytes).
in	privkey	Pointer to the private key for signing.
out	sig	Pointer to the signature of the message.
in	dp	Pointer to an ECC domain context.

Returns

- SUCCESS - 0
- FAILURE - Negative

qsee_SW_GENERIC_ECDSA_verify()

Description

This function verifies the data with an ECC public key.

```
int qsee_SW_GENERIC_ECDSA_verify ( uint8_t  msgdgst, uint32_t
msgdgst_len, QSEE_qrlbn_ecc_affine_point_t  pubkey, QSE E_qrlbn_
ECDSA_sig_t  sig, QSEE_qrlbn_ecc_domain_t  dp
```

Parameters

Parameters		Description
in	msgdgst	Pointer to a message that must be signed.
in	msgdgst_len	Length of the message (in bytes).
in	pubkey	Pointer to the public key for verification.
in	sig	Pointer to the signature of the message.
in	dp	Pointer to an ECC domain context.

Returns

- SUCCESS - 0
- FAILURE - Negative

qsee_SW_GENERIC_ECDSA_verify_ex()

Description

This function hashes the input data and verifies with an ECC private key. This API is a FIPS certifiable ECDSA verification API.

```
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 )
```

Parameters

Parameters		Description
in	hash_alg	The algorithm that is hashed for signing the message.
in	msg	Pointer to a message to be signed.
in	msg_len	Length of the message (in bytes).
in	pubkey	Pointer to the public key for verification.
in	sig	Pointer to the signature of the message.

Parameters		Description
in	dp	Pointer to an ECC domain context.

Returns

- SUCCESS - 0
- FAILURE - Negative

For more information about ECC APIs, see: TZ.APPS.1.0/qtee_tas/sdk/latest/external/inc/qsee/qsee_ecc.h.

Hash APIs

The hash transforms a key or a string of characters into another value called a message digest. These hash APIs perform various hash operations on data.

qsee_hash()

Description

This function creates a message digest hash using the specified algorithm.

```
int qsee_hash ( QSEE_HASH_ALGO_ET alg, const uint8_t  msg, uint32_t
msg_len, uint8_t  digest, uint32_t digest_len )
```

Parameters

Parameters		Description
in	alg	Indicates the hash algorithm.
in	msg	Pointer to the message to hash.
in	msg_len	Length of the message.
out	digest	Pointer to the digest to store.
in	digest_len	The length of the output digest buffer (in bytes). This length must be at least equal to the size of the hash that has requested the hash algorithm.

Returns

- QSEE_HASH_SUCCESS - The function executes successfully.
- QSEE_HASH_FAILURE - Any error encountered during the creation of the hash.

For more information about hash APIs, see <TZ.APPS>/qtee_tas/sdk/latest/external/inc/qsee/qsee_hash.h.

Heap APIs

The heap memory is used for dynamic memory allocations. The heap APIs enable you to allocate and de-allocate the heap memory.

qsee_malloc()

Description

This function allocates a block of size bytes from the heap. If the size is zero, then it returns NULL.

```
void* qsee_malloc (size_t size)
```

Parameters

Parameters		Description
in	size	Number of bytes to allocate from the heap.

qsee_realloc()

Description

This function resizes a block of memory previously allocated using qsee_malloc() or qsee_realloc(). If the size is zero, then it returns NULL.

```
void qsee_realloc ( void  ptr, size_t size )
```

Parameters

Parameters		Description
in	ptr	Pointer to a previously allocated block. If NULL is passed, this function is akin to qsee_malloc().
in	size	Indicates a new block size.

Returns

Returns a pointer to the newly allocated block, or NULL when the block could not be allocated.

qsee_free()

Description

This function de-allocates a block of memory referenced by a memory pointer.

```
void qsee_free ( void  ptr )
```

Parameters

Parameters		Description
in	ptr	Pointer to the block that will be freed.

For more information about heap APIs, see <TZ.APPS>/qtee_tas/sdk/latest/external/inc/qsee/qsee_heap.h.

HMAC APIs

The HMAC verifies that the data being received is authentic and from reliable sources. The qsee_hmac() API uses the specified hash algorithm per the keyed HMAC (FIPS PUB 198-1).

qsee_hmac()

Description

This function creates an HMAC according to FIPS PUB 198-1 using the specified hash algorithm.

```
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 )
```

Parameters

Parameters		Description
in	alg	HMAC algorithm that can be used.
in	msg	Pointer to the message that provides authentication.
in	msg_len	Length of the message in bytes.

Parameters		Description
in	key	Pointer to the input key for the HMAC algorithm.
in	key_len	Length of input key (in bytes).
out	msg_digest	Pointer to the message digest (memory provided by the caller).

Returns

- SUCCESS - The function executes successfully.
- FAILURE - Any error encountered during the HMAC creation.

For more information about HMAC APIs, see <TZ.APPS>/qtee_tas/sdk/latest/external/inc/qsee/qsee_hmac.h.

KDF APIs

The key derivation function (KDF) is essential for secure key derivation.

qsee_kdf()

Description

This function derives keys to be used as the inputs to the encryption algorithm such as AES.

```
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 )
```

Parameters

Parameters		Description
In	key	Pointer to the key derivation key.
In	key_len	Length of the key derivation key in bytes.
In	label	Pointer to the key derivation label.
In	label_len	Length of the key derivation label in bytes.

Parameters		Description
In	context	Pointer to the key derivation context.
In	context_len	Length of the key derivation context in bytes.
Out	output	Pointer to the derived key.
in	output_len	Length of the derived key in bytes.

The software is a three-level stack:

1. First level: AES
2. Second level: CMAC algorithm from NIST SP 800-38B
3. Third level: Counter-based algorithm from NIST SP 800-108 (called KDF in implementation). The inputs are:
   - Key derivation key (key, key_len)
   - Label (label, label_len)
   - Context (context, context_len) The output is output_len bytes long. All sensitive data is set to zero before return.

   When a key is set to NULL, key_len is ignored. Qualcomm TEE sets a 32-byte key (inaccessible outside TrustZone) as the input key.

Returns

- QSEE_KDF_SUCCESS - The function executes successfully.
- QSEE_KDF_INVALID - An invalid parameter.
- QSEE_KDF_FAILURE - All other errors encountered during the key derivation process.

For more information about KDF APIs, see <TZ.APPS>/qtee_tas/sdk/latest/external/inc/qsee/qsee_kdf.h.

Logging APIs

These APIs enable to log and print messages in the Qualcomm TEE log file.

qsee_log()

Description

This function collects a set of logs in the internal buffers and flushes the logs to a rolling log file at predetermined thresholds.

```
void qsee_log ( uint8_t pri, const char  fmt, ... )
```

Parameters

Parameters		Description
in	pri	The priority of the message that is to be logged.
in	fmt	Pointer to a string describing the message format.
in	..	Indicates a variable argument list.

The maximum length of the trusted application messages allowed for Qualcomm TEE logging is 120. If TA passes a message with a length more than 120 characters, the message is trimmed down to 120 characters and printed in `qsee_log()`. This API is referred by `tz_app_init()` and `tz_app_shutdown()`.

qsee_log_get_mask()

Description

This function retrieves the bitmask for the log levels set currently.

```
uint8_t qsee_log_get_mask ( void )
```

Returns

- This function returns the bitmask value of the log levels set.
- Referenced by `tz_app_init()`.

qsee_log_set_mask()

Description

This function allows the user to set logs using a bitmask defined by the following levels:

- 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 the range QSEE_LOG_MSG_LOW to QSEE_LOG_MSG_FATAL. Any value outside this range is ignored.

```
void qsee_log_set_mask ( uint8_t pri_flags )
```

Parameters

Parameters		Description
in	pri_flags	OR(ed) bitmask for the required log levels.

For more information about logging APIs, see <TZ.APPS>/qtee_tas/sdk/latest/external/inc/qsee/qsee_log.h.

Pseudo-random number generator APIs

The pseudo-random number generators (PRNG) APIs are essential for generating random values.

qsee_prng_getdata()

Description

This function releases all the resources with a specified PRNG context. It generates a random number of a specified length.

```
uint32 qsee_prng_getdata (uint8 *out, uint32 out_len)
```

Parameters

Parameters		Description
out	out	Output data buffer.
in	out_len	The length of the output data. out_len must be at most QSEE_MAX_PRNG bytes.

Returns

This function returns the number of bytes read.

For more information about PRNG APIs, see TZ.APPS/qtee_tas/sdk/latest/external/inc/qsee/qsee_prng.h.

RSA APIs

The Rivest-Shamir-Adelmann (RSA) is a public key crypto system that's used for secure data transmission. The RSA APIs provide various functions that can be performed on the RSA private/public key.

qsee_rsa_encrypt()

Description

This function performs PKCS #1 v1.5 padding followed by encryption.

```
int qse_rsa_encrypt ( QSEE_RSA_KEY  key, QSEE_RSA_PADDING_TY PE
padding_type, void  padding_info, const unsigned char  msg, int
msglen, unsigned char  cipher, int  cipherlen )
```

Parameters

Parameters		Description
In	key	RSA key, using which the encryption is performed.
In	padding_type	Indicates the padding type.
In	padding_info	Optimal asymmetric encryption padding (OAEP) parameters.
In	msg	Plain text.
In	msglen	Length of the plain text (octets).
Out	cipher	Indicates cipher text.

Parameters		Description
In,out	cipherlen	The maximum size and resulting size of the cipher text.

Returns

- CE_SUCCESS - The function executes successfully.
- CE_ERROR_NOT_SUPPORTED - The feature isn't supported.
- CE_ERROR_INVALID_PACKET - An invalid packet.
- CE_ERROR_BUFFER_OVERFLOW - There isn't enough space for the output.
- CE_ERROR_NOP - The software crypto self-test has failed.

qsee_rsa_key_gen()

Description

This function generates an RSA private/public key pair as per FIPS 186-4 standards.

```
int qsee_rsa_key_gen ( QSEE_RSA_KEY key, int keylen, unsigned char
pub_exp, int pub_exp_len )
```

Parameters

Parameters		Description
out	key	Public/private RSA key.
in	keylen	Length of the RSA key (in bytes).
in	pub_exp	Public exponent array.
in	pub_exp_len	Length of the public exponent array.

Returns

- CE_SUCCESS - The function executes successfully.
- CE_ERROR_FAILURE - A generic error.
- CE_ERROR_NOT_SUPPORTED - The feature isn't supported.
- CE_ERROR_INVALID_ARG - A generic invalid argument.

- CE_ERROR_BUFFER_OVERFLOW - There isn't enough space for the output.
- CE_ERROR_NO_MEMORY - Out of memory.
- CE_ERROR_INVALID_SIGNATURE - An invalid signature.

qsee_rsa_decrypt()

Description

This function performs the PKCS #1 decryption, followed by a v1.5 depad.

```
int qsee_rsa_decrypt ( QSEE_RSA_KEY  key, QSEE_RSA_PADDING_T YPE
padding_type, void  padding_info, unsigned char  cipher, int
cipherlen, unsigned char  msg, int  msglen )
```

Parameters

Parameters		Description
in	key	RSA key, using which the decryption is performed.
in	padding_type	Indicates the padding type.
in	padding_info	OAEP padding parameters.
in	cipher	Cipher text.
in	cipherlen	Length of the cipher text (octets).
out	msg	Plain text.
in,out	msglen	The maximum size and resulting size of the plain text.

Returns

- CE_SUCCESS - The function executes successfully.
- CE_ERROR_NOT_SUPPORTED - The feature isn't supported.
- CE_ERROR_INVALID_PACKET - An invalid packet.
- CE_ERROR_BUFFER_OVERFLOW - There isn't enough space for the output.
- CE_ERROR_NOP - The software crypto self-test has failed.

qsee_rsa_sign_hash()

Description

This function performs the PKCS #1 padding and signs the signature.

```
int qsee_rsa_sign_hash ( QSEE_RSA_KEY  key, QSEE_RSA_PADDING_T YPE
padding_type, void  padding_info, QSEE_HASH_IDX hashidx, const
unsigned char  hash, int hashlen, unsigned char  signature, int
siglen )
```

Parameters

Parameters		Description
in	key	RSA key, using which the encryption is performed.
in	padding_type	Indicates the padding type.
in	padding_info	OAEP padding parameters.
in	hashidx	Required hash index.
in	hash	Hash to sign (octets).
in	Hashlen	Length of the hash to sign.
out	signature	Signature.
in,out	siglen	The maximum size and the resulting size of the signature.

Returns

- CE_SUCCESS - The function executes successfully.
- CE_ERROR_NOT_SUPPORTED - The feature isn't supported.
- CE_ERROR_INVALID_ARG - A generic invalid argument.
- CE_ERROR_BUFFER_OVERFLOW - There isn't enough space for the output.
- CE_ERROR_NO_MEMORY - Out of memory.
- CE_ERROR_NOP - The software crypto self-test has failed.

qsee_rsa_verify_signature()

Description

This function performs the PKCS #1 padding and verifies the signature.

```
int qsee_rsa_verify_signature ( QSEE_RSA_KEY  key, QSEE_RSA_PADDI NG_
TYPE padding_type, void  padding_info, QSEE_HASH_IDX hashidx,
unsigned char  hash, int hashlen, unsigned char  sig, int siglen )
```

Parameters

Parameters		Description
in	key	RSA key, using which the encryption is performed.
in	padding_type	Indicates the padding type.
in	padding_info	OAEP padding parameters.
in	hashidx	Required hash index.
in	hash	Hash to sign (octets).
in	Hashlen	Length of the hash to sign.
in	signature	Signature.
in	siglen	The maximum size and the resulting size of the signature.

Returns

- CE_SUCCESS - The function executes successfully.
- CE_ERROR_NOT_SUPPORTED - The feature isn't supported.
- CE_ERROR_INVALID_ARG - A generic invalid argument.
- CE_ERROR_BUFFER_OVERFLOW - There isn't enough space for the output.
- CE_ERROR_NO_MEMORY - Out of memory.
- CE_ERROR_NOP - The software crypto self-test has failed.

For more information about RSA APIs, see TZ.APPS/qtee_tas/sdk/latest/external/inc/qsee/qsee_rsa.h.

SFS APIs

The SFS APIs provide various functions to manage secure storage on a device. The SFS files are encrypted and only the application that created them can access them.

qsee_sfs_open()

Description

This function opens an SFS file. The file mode in the flag parameter specifies the options.

```
int qsee_sfs_open ( const char  path, int flags )
```

In SFS, opening a file doesn't perform any action in the file system. If the file (along with the associated file segments) exists and the file is created with the O_TRUNC mode, the associated sub files are deleted. The first segment is created only when the new bytes to be written begin arriving.

Note: The base directory must exist; otherwise, the API returns NULL.

Parameters

Parameters		Description
in	path	Pointer to a fully qualified path of the file name to be opened.
in	flags	The bitmask fields used to specify the file modes are: • O_RDONLY - Open for read only access. • O_READWRITE - Open for read-write access. • O_CREAT - Creates a file when it does not exist. • O_TRUNC - Truncates file to size 0 after opening. • O_APPEND - Write operations occur at the end of the file.

Returns

- Nonzero - A valid file descriptor.
- Zero - An error occurred while opening the file.

qsee_sfs_read()

Description

This function reads the bytes from an encrypted SFS file that was previously opened using a call to `qsee_sfs_open()`.

```
int qsee_sfs_read ( int fd, char  buf, int nbytes )
```

The `nbytes` are read from the current file position, and the file position advances by the number of read bytes. The SFS performs the necessary cipher and verification operations when the bytes are read from the file.

Parameters

Parameters		Description
in	`fd`	File descriptor.
in	`buf`	Pointer to the buffer to hold the read bytes.
in	`nbytes`	Number of bytes to read from the file.

Returns

- The number of bytes read from an SFS file.
- 1 if an error is encountered.

qsee_sfs_write()

Description

This function writes the bytes to a previously opened encrypted SFS file using a call to `qsee_sfs_open()`.

```
int qsee_sfs_write ( int fd, const char  buf, int nbytes )
```

Returns

- The number of bytes written to an SFS file.
- 1 if an error is encountered.

qsee_sfs_close()

Description

This function closes an open SFS file. It releases all the resources used by the file.

```
int qsee_sfs_close ( int fd )
```

Parameters

Parameters		Description
in	Fd	File descriptor.

Returns

- E_SUCCESS - Closed the file successfully.
- E_FAILURE - An error is encountered while closing the file.

For more information about SFS APIs, see TZ.APPS/qtee_tas/sdk/latest/external/inc/qsee/qsee_sfs.h.

Object-based Qualcomm TEE service interfaces

These APIs encrypt, decrypt, and update the input/output text (plain or cipher) and the input/output buffers.

method decrypt()

Description

This function decrypts the passed cipher text message using the specified algorithm.

```
method decrypt ( in buffer cipher, out buffer plain )
```

Parameters

Parameters		Description
in	cipher	Input for a cipher text buffer
out	plain	Output for a plain text buffer

The memory allocated for plain text must be large enough to hold the cipher text equivalent.

If a padding scheme is selected, the plain text output length may be up to one block size smaller than the cipher text length. If the output buffer isn't large enough to hold the decrypted results, an

error is returned.

Note: This API doesn't support the Galois counter mode (GCM) and GCM_STRM modes that must use the `update_aad/update/final` APIs.

Returns

- `Object_OK` - Successful
- `Object_ERROR_INVALID` - Not a multiple of block length
- `Object_ERROR` - Any other error encountered

method encrypt()

Description

This function encrypts the passed plain text message using the specified algorithm.

```
method encrypt ( in buffer plain, out buffer cipher )
```

Parameters

Parameters		Description
`in`	`plain`	Input for a plain text buffer
`out`	`cipher`	Output for a cipher text buffer

The memory allocated for the cipher text must be large enough to hold the plain text equivalent. If a padding scheme is selected, the cipher text buffer length should be up to one block size larger than the plain text length.

If the output buffer isn't large enough to hold the encrypted results, an error is returned.

Note: This API doesn't support GCM and GCM_STRM modes, which must use update_ aad/`update/final` APIs.

Returns

- `Object_OK` - Successful
- `Object_ERROR_INVALID` - Not a multiple of block length
- `Object_ERROR` - Any other error generated

method final()

Description

This function encrypts/decrypts the last segment of the input buffer that's saved in a cipher context, and saves the cipher/plain text to the output buffer `[obuf]`. This API updates the size of the output on a successful return in `obuf_lenout`.

```
method final ( out buffer obuf )
```

Parameters

Parameters		Description
out	obuf	Output buffer

Returns

- `Object_OK` - Successful
- `Object_ERROR` - Any error encountered

method update()

Description

This function encrypts/decrypts the input buffer [ibuf], saves the cipher/plain text to the output buffer [obuf], and saves the cipher context for further updates. This API updates the size of the output on a successful return in `obuf_lenout`.

```
method update ( in buffer ibuf, out buffer obuf )
```

Parameters

Parameters		Description
in	ibuf	Input buffer
out	obuf	Output buffer

Returns

- `Object_OK` - Successful
- `Object_ERROR` - Any error encountered

method update_aad()

Description

This function updates additional authentication data when the authenticated cipher mode (example, CCM/GCM) is selected. This API returns a failure for the other cipher modes. This API is called before calling the `update()` and `final()` APIs.

```
method update_aad ( in buffer aad, out buffer obuf )
```

Parameters

Parameters		Description
`in`	`aad`	An input for an attempt algorithm designator (AAD) buffer.
`out`	`obuf`	A dummy buffer for the output data. The crypto hardware engine requires that for all the AAD data that is sent to the hardware, the same length of data must be read out. Therefore, the AAD data must be read back from the hardware engine using the `obuf` dummy buffer. The size of `obuf` should not be less than the length of AAD. If you want to use the same buffer for both the in/out buffers, ensure that the buffer is not constant.

Returns

- `Object_OK` - Successful
- `Object_ERROR` - Any error encountered

method setParamAsData()

Description

This function modifies the parameters for a cipher context.

```
method setParamAsData ( in int32 paramID, in buffer param )
```

Parameters

Parameters		Description
in	paramID	Parameter to modify
in	param	Parameter value to set

Returns

- Object_OK - Successful
- Object_ERROR_INVALID - An invalid parameter has been encountered
- Object_ERROR - Any other error encountered

method setParamAsObject()

Description

This function modifies the parameters for a cipher context.

```
method setParamAsObject ( in int32 paramID, in interface param )
```

Parameters

Parameters		Description
in	paramID	Parameter to modify
in	param	Parameter value to set

Returns

- Object_OK - Successful
- Object_ERROR_INVALID - An invalid parameter encountered
- Object_ERROR - Any other error encountered

For more object-based Qualcomm TEE service interfaces APIs, see TZ.APPS/qtee_tas/sdk/latest/external/inc/idl/ICipher.idl.

Fuse APIs

These APIs read and write the row data from the specified QFPROM row address.

qsee_fuse_write()

Description

This function writes the row data from the specified QFPROM row address.

```
uint32_t   row_address,
int32_t    addr_type,
uint32_t   row_data[2],
uint32_t*  qfprom_api_status
```

Parameters

Parameters		Description
in	row_address	The row address in the QFPROM region from which the row data is read.
in	addr_type	Row (uncorrected) or FEC-corrected data.
out	row_data[]	Array of the data (size 2 x 32 bits) to be read.
out	qfprom_api_status	Pointer to return value from the QFPROM API.

qsee_fuse_read()

Description

This function reads the row data from the specified QFPROM row address.

```
uint32_t    raw_row_address,
uint32_t    row_data[2],
uint32_t    bus_clk_khz,
uint32_t*   qfprom_api_status
```

Parameters

Parameters		Description
in	raw_row_address	The row address in the QFPROM region to which the row data is to be written.
in	ow_data[]	Array of the data (size 2 x 32 bits) to write into the QFPROM region.
in	bus_clk_khz	The bus clock frequency (in kHz) is connected to the QFPROM region. This value is ignored.
out	qfprom_api_status	Pointer to return value from the QFPROM API.

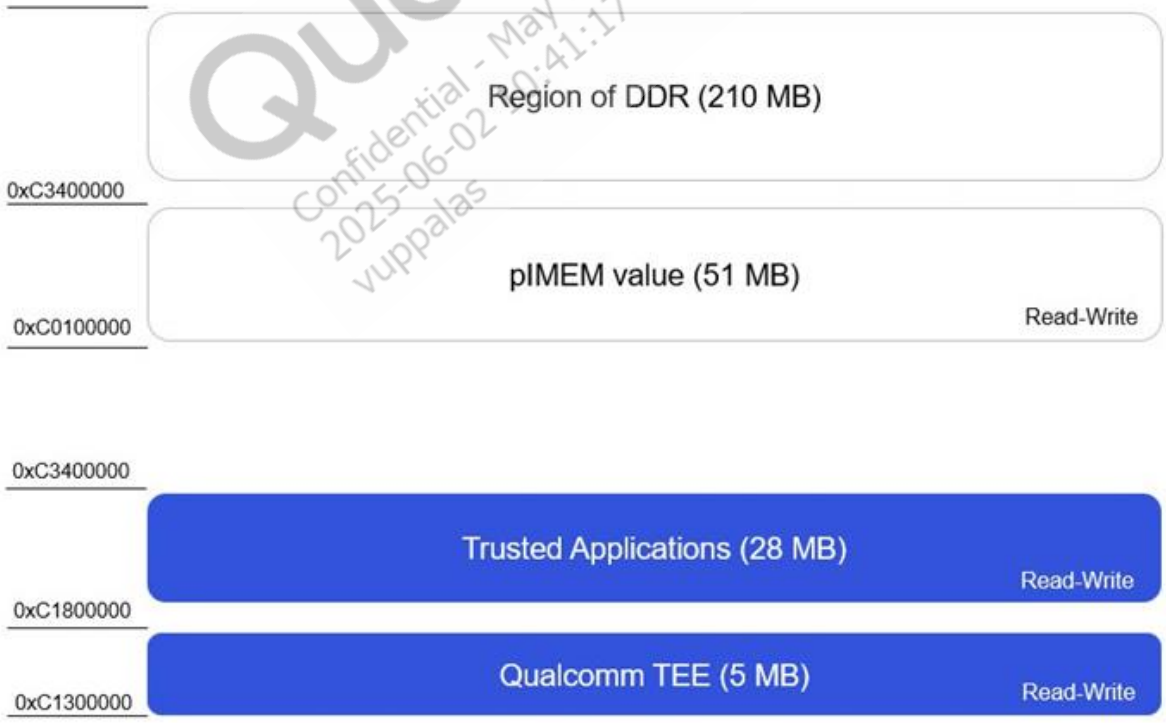
3 Customize memory for trusted applications

Customization is supported for memory and SEPolicy. For a large-size trusted application, you may need to customize the memory regions.

3.1 Customize memory

The trusted application memory is set to 28 MB in the DDR memory map by default. However, you can customize these settings to increase the trusted application memory.

The figure shows the placement of the trusted application memory in a DDR memory map, located under the internal memory (pIMEM) vault:



If the number of trusted applications is high, or if the trusted application memory exceeds the allocated memory, there is a risk of loading failure due to insufficient memory.

For debugging procedures, see Qualcomm Linux Security Guide → *DebugQualcommTEEandsecuredevices* → *Debugtrustedandclientapplications*.

In such cases, it's recommended to expand the additional trusted application memory in DDR.

For example, if you want to add an extra 10 MB to the existing 28 MB memory map, you must modify the following build files:

1. Go to `BOOT.MXF.1.0.c1-00026-KODIAKLA-1boot_imagesbootQcomPkgSocPkgKodiakCommonuefiplat.cfg`.
   a. Make the following changes as highlighted.
   b. Recompile the XBL image (`xbl.elf`).

```
-   0xC1800000, **0x01C00000**, "TZApps Reserved",
HobOnlyNoCacheSetting, MEM_RES, UNCACHEABLE, Reserv,
UNCACHED_UNBUFFERED_XN
      + 0xC1800000, **0x02600000**, "TZApps Reserved",
HobOnlyNoCacheSetting, MEM_RES, UNCACHEABLE,
Reserv, UNCACHED_UNBUFFERED_XN
```

2. Go to `TZ.XF.5.0-07756-KODIAKAAAAANAAZT-1trustzone_imagesssgsecuremsmtrustzoneqseeminkoemconfigkodiakoem_config.xml`.
   a. Make the following changes as highlighted.
   b. Recompile the `devcfg.mbn` file.

      For instructions on build and compilation, see Qualcomm Linux Build Guide → *GitHubworkflow(firmwareandextras)*.

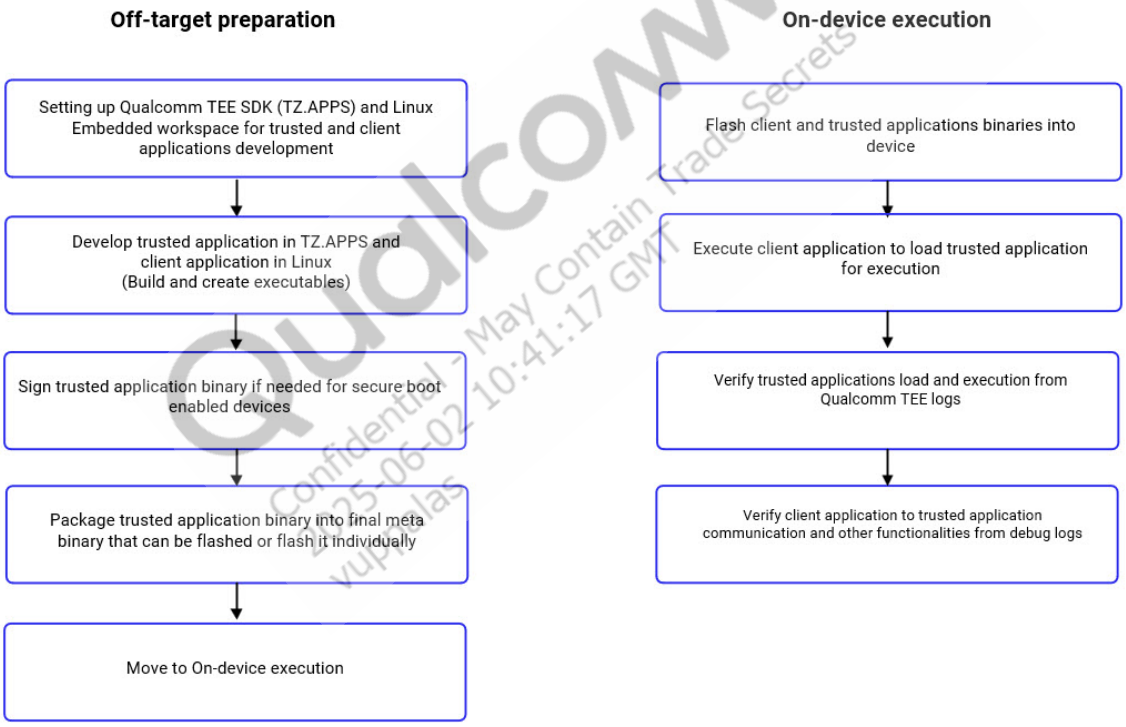
```
<props name="OEM_pil_secure_app_load_region_size"
type=DALPROP_ATTR_TYPE_UINT32>
          **  -0x01C00000
            +0x02600000**
</props>
```

      The limit of trusted application memory is determined by the available memory region in DDR.

4 Develop trusted and client applications

You can develop and run trusted and client applications using the default files placed in the global platform interfaces.

The following workflow helps you to understand the off-target preparation and the on-device execution.



4.1 Develop trusted and client applications

This section provides detailed steps for developing trusted and client applications.

The supported types of trusted and client applications are:

- SMC invoke-based applications

- Global platform-based applications

Configure trusted applications

While developing the trusted applications, you can refer to the available configurations and set them according to your requirement.

- File limit: By default, Qualcomm TEE can support up to 60 SFS files per trusted applications. If a new metadata entry `totalStorageFiles` is defined, the limit can be increased.
- Heap requirements: Memory used by the runtime environment for implementing SFS is from the `.bss` sections of trusted applications to the heap. The default heap size accommodates the default number of files supported by SFS.
- The trusted applications that define a non-default heap size must increase their heap definitions by 8 kB to account for shifting the memory use of the SFS to the heap. Trusted applications that use more than the default SFS file limit must increase their heap to align with the trusted application-specific file limit.
- SFS file/directory: By default, the files created in SFS are the global platform interfaces, which persist even after a factory reset. This behavior can be modified on a per trusted application basis using the metadata configuration of the trusted applications in the trusted applications SConscript file.

  A metadata property, `storageFilesNoPersist`, is defined and set to `True` to configure the storage of the trusted applications to be deleted during the factory reset. The persistence of data over the factory reset is implemented by switching the root path in the file system of the rich operating system. The SFS root path is configured in the Linux at:

  ```
  sources/securemsm/daemon/gpfspath_qclinux_oem_config.xml

  <gp_persist_path> /var/persist/qtee_supplicant/ </gp_
  persist_path>
  ```

- Encoded file and directory names: SFS and persistent object implementations use encoded directory and index file naming.
- File permissions: By default, the SFS and persistent object implementations support the read and write permissions for all the files.

4.2 Develop SMC invoke trusted applications

The SMC invoke-based trusted application is a nonsecure, Linux client interface to communicate with the client applications in Qualcomm TEE.

The communication occurs through SMC invoke user space APIs and driver interfaces. For the SMC invoke-based client application sample code, see SMC invoke skeleton C++ trusted applications source listing.

Sync build in Linux workspace

1. To ensure that the host machine is set up, see Qualcomm Linux Build Guide → $GitHubworkflowforunregisteredusers$.
2. To sync the non-Linux build with the prebuilts, see Qualcomm Linux Build Guide → $GitHubworkflow(firmwareandextras)$.

The following is the directory structure:

```
cd qualcomm-linux-spf-1-0_ap_standard_oem_nomodem
ls
about.html  ADSP.HT.5.5.c8  AOP.HO.3.6  BOOT.MXF.1.0.c1  BTFW.
MOSELLE.1.1.0  CDSP.HT.2.5.c3  CPUCP.FW.1.0  LE.QCLINUX.1.0.r1  LKP.
QCLINUX.1.0.r1  QCM6490.LE.1.0  TZ.APPS.1.0  TZ.XF.5.0  WLAN.MSL.2.0.
c2
```

Build trusted applications

1. Go to the TrustZone application directory using the command:

   ```
   cd TZ.APPS.1.0
   ```

2. Copy the `qtee_tas/sdk/latest/external/samples/smcinvoke_skeleton/TA` directory to the `qtee_tas/apps/securemsm/trustzone/qsapps/smcinvoke_skeleton/TA` directory.

   This step copies the source (src) directory and SConscript from the original location.

3. Copy the `qtee_tas/sdk/latest/external/samples/smcinvoke_skeleton/common` directory to the `qtee_tas/apps/securemsm/trustzone/qsapps/smcinvoke_skeleton/common` directory.

   This copies the INC and IDL files from the original location.

4. Copy the SConstruct file from `qtee_tas/apps/securemsm/trustzone/qsapps/sample_ta/` to `qtee_tas/apps/securemsm/trustzone/qsapps/smcinvoke_skeleton/TA/src`.

   a. Ensure that the SConscript and SConstruct files are in the same location. The directory structure should look like:

   ```
   ├── common
   │   ├── idl
   │   │   ├── CSMCIExample_open.h
   │   │   └── ISMCIExample.idl
   │   └── inc
   │       └── stringl.h
   ```

```
└── src
    ├── smci_ta_main.c
    ├── CSMCIExample.c
    ├── SConscript
    ├── SConstruct
```

5. Move to the root directory workspace. This means that running the `ls` command must display the `/qtee_tas` directory, which is the root directory for the trusted application compilation.

The following are the build trusted applications:

1. Ensure you have the following tools:

Table : Prerequisites to build trusted applications

Software/tool	Description
Python v3.10	-
SCons v3.0.1 or later	-
Qualcomm® Snapdragon™ LLVM toolchain 16.0.7	Check `sdk_config_lnx.cfg` for any version changes.

Software/tool	Description

Configure Sectools environment
`cd TZ.APPS.1.0`
`export SECTOOLS=<Meta Path>/common/ sectoolsv2/ext/Linux/ sectools`
	Go to the TrustZone application directory with `cdTZ.APPS.1.0..` Set the Sectools environment variable to the path of the Sectools directory. For example:
	For QCM6490/QCM5430:
	`export SECTOOLS=/ local/mnt/ workspace/ qualcomm-linux-spf-1-0_ap_ standard_oem_ nomodem/ QCM6490. LE.1.0/common/ sectoolsv2/ext/ Linux/sectools`
	For QCS9075:
	`export SECTOOLS=export SECTOOLS=/ local/mnt/workspace/ qualcomm-linux-spf-1-0_ ap_standard_oem_nm-qimpsdk/QCS9100. LE.1. 0/ common/sectoolsv2/ ext/Linux/sectools```

2. To build trusted applications, run the following commands:

```
python3 <scons> -C <TA SConstruct location from root dir> QTEE_
SDK=qtee_tas/sdk/latest/external BUILD_ROOT=. CHIPSET=<chipset>
qtee_sdk_version=latest <ta alias name mentioned in Sconscript>
```

For example:

- For QCM6490/QCM5430:

```
python3 ../TZ.XF.5.0/trustzone_images/tools/build/scons/
SCons/scons -C qtee_tas/apps/securemsm/trustzone/qsapps/
smcinvoke_skeleton/TA/src QTEE_SDK=qtee_tas/sdk/latest/
external BUILD_ROOT=. CHIPSET=kodiak qtee_sdk_version=latest
smcinvoke_skeleton_ta
```

- For QCS9075:

```
python3 ../TZ.XF.5.0/trustzone_images/tools/build/scons/
SCons/scons -C qtee_tas/apps/securemsm/trustzone/qsapps/
smcinvoke_skeleton/TA/src QTEE_SDK=qtee_tas/sdk/latest/
external BUILD_ROOT=. CHIPSET=lemans qtee_sdk_version=latest
smcinvoke_skeleton_ta
```

Note: Install scon in your workspace if it doesn't work in the command.

3. To clean the build, run the following command:

```
python3 <scons> -C <TA SConstruct location from root dir> QTEE_
SDK=qtee_tas/sdk/latest/external BUILD_ROOT=. CHIPSET=<chipset>
qtee_sdk_version=latest <ta alias name mentioned in Sconscript>
-c
```

For example:

- For QCS6490/QCS5430:

```
python3 ../TZ.XF.5.0/trustzone_images/tools/build/scons/
SCons/scons -C qtee_tas/apps/securemsm/trustzone/qsapps/
smcinvoke_skeleton/TA/src QTEE_SDK=qtee_tas/sdk/latest/
external BUILD_ROOT=. CHIPSET=kodiak qtee_sdk_version=latest
smcinvoke_skeleton_ta -c
```

- For QCS9075:

```
python3 ../TZ.XF.5.0/trustzone_images/tools/build/scons/
SCons/scons -C qtee_tas/apps/securemsm/trustzone/qsapps/
smcinvoke_skeleton/TA/src QTEE_SDK=qtee_tas/sdk/latest/
external BUILD_ROOT=. CHIPSET=lemans qtee_sdk_version=latest
smcinvoke_skeleton_ta -c
```

To check build ID mapping, see `qtee_tas/sdk/latest/external/config/sdk_config_kodiak.cfg`.

After the compilation, the trusted application image is created in the out-directory with the following files:

- `qtee_tas/build/ms/bin/<Build ID>/<appName>.mbn`
  - Check the build ID mapping: `qtee_tas/sdk/latest/external/config/sdk_config_kodiak.cfg`
  - Build ID = IAGAANAA

- `qtee_tas/build/ms/bin/unsigned/<appName>.mbn`

  An unsigned image when that's required to sign manually.

A `metadata.c` file is autogenerated when a particular trusted application is compiled.

- The inputs for metadata controlled from SConscript are heap, stack, and accept_buffer.
- When no explicit configuration is done in SConscript, the default value is picked.
- The application name is a mandatory metadata file that must be updated in the trusted application SConscript.
- The output directory location is: `<build>/qtee_tas/apps/bsp/trustzone/qsapps/smcinvoke_skeleton_ta/build/IAGAANAA/smcinvoke_skeleton_ta64_metadata`

4. If any Qualcomm TEE services are used, copy `I<ClassName>.h` and `C<ClassName>.h` into the directories for the respective trusted applications.

   The header file for the Qualcomm TEE-exposed service is generated when any trusted application is compiled at `qtee_tas/sdk/latest/external/inc/idl`.

   For example: If the trusted application must map the memory object associated physical address, then the `IMemSpace.h` header must be included.

   The `object.h` header location: `qtee_tas/sdk/latest/external/inc/util`.

4.3 Develop SMC invoke client applications

The SMC invoke-based client application is a nonsecure, Linux client interface to communicate with the trusted applications in Qualcomm TEE.

This communication occurs through SMC invoke user space APIs and driver interfaces. For a sample of an SMC invoke-based client application, see SMC invoke skeleton client applications source listing.

Sync build in Linux workspace

1. To ensure that the host machine is set up, see Qualcomm Linux Build Guide → $GitHubworkflowforunregisteredusers$.
2. To sync the Linux build with the prebuilts, see Qualcomm Linux Build Guide → $GitHubworkflow(firmwareandextras)$.

   Upon successful completion, the `<WORKSPACE DIR>/layers` directory is created.

3. Create the client application directory under `<current_workspace>/layers/meta-qcom-hwe/recipes-security/<CA_direcotry>`.

   This directory must contain the recipe file along with the source code in the following format:

```
├── <CA_directory>
│   ├── <CA-Recipe>.bb
│   │   files
│   │       ├── src
│   ├── inc
│   ├── makefile.am
│   ├── configure.ac
```

   For a sample code, see SMC invoke interface-based client applications.

4. Ensure that build-time dependencies on the `securemsm` and `dmabuf` recipes are marked as follows:

```
DEPENDS = "minkipc securemsm-features securemsm-headers qcom-
libdmabufheap"
```

   Marking the dependencies allow you to use the -l flags in the Makefile.am and include any of the headers.

5. Import the headers from the `securemsm-headers` package to your package by adding the following to the `configure.ac` file in the `<current_workspace>LE_HLOSlayersmeta-qcom-hwerecipes-security<CA_direcotry>filesconfigure.ac` path:

```
PKG_CHECK_MODULES([SECUREMSMHEADERS], [securemsm-headers])
AC_SUBST([SECUREMSMHEADERS_CFLAGS])
AC_SUBST([SECUREMSMHEADERS_LIBS])
```

6. Register the `Makefile.am` file in your configure.ac file as follows:

```
AC_CONFIG_FILES([ \<Path to your Makefile.am>/Makefile \])
```

7. In your `Makefile.am` file, import these headers through the following:

```
AM_CPPFLAGS = @SECUREMSMHEADERS_CFLAGS@ \
And add linkages to libraries:
<binary_name>_LDFLAGS += -lqcbor -lminkdescriptor -ldmabufheap
```

For sample makefile and configure.ac files, see SMC invoke interface-based client applications and SMC invoke skeleton client applications source listing.

8. Ensure that you add the recipe name at: <current_workspace>/layers/meta-qcom-hwe/recipes-security/packagegroups/packagegroup.bb.

```
SUMMARY = "CA test app"
LICENSE = "Qualcomm-Technologies-Inc.-Proprietary"
PACKAGE_ARCH = "${MACHINE_ARCH}"

inherit packagegroup

PROVIDES = "${PACKAGES}"

PACKAGES = "${PN}"

RDEPENDS:${PN} += " \
 securemsm-headers \
 securemsm-features \
 minkipc \
 qcom-libdmabufheap \
"
```


Build client applications

When the client application source code and the recipe are complete and ready, follow these steps:

1. Run the following command to build the client application:

```
export SHELL=/bin/bash
```

Set up the build environment. For instructions, see Qualcomm Linux Build Guide → $GitHubworkflowforunregisteredusers$.

```
bitbake <recipe_file_name>
```

2. Run the following command to clean the build:

```
bitbake -f -c cleanall smci-skeleton-client
```

The output location is:

<Current_workspace>/build-qcom-wayland/tmp-glibc/work/qcm6490-qcom-linux/<recipe_name>/1.0-r0/image/usr/bin/<CA_name>.

3. Load the client application to the device and explicitly push the client application to the device:

    a. Connect to the device using SSH. For instructions, see Qualcomm Linux Build Guide → *Howto* → *SigninusingSSH*.

    b. Run the following command:

    mount -o rw,remount / scp <CA-bin> root@<IP_address>:/usr/bin/ chmod + x <CA-bin>

    When all the images are flashed, the binary files become available on the device in the `/usr/bin` directory.

4.4 Load trusted applications on device

You can load the trusted applications on the device by remounting the file system and by pushing the trusted application to the device.

To load the trusted application onto the device, follow these steps:

1. To gain the root access, connect to the device using SSH. For instructions, see Qualcomm Linux Build Guide → *Howto* → *SigninusingSSH*.
2. To remount the file system with read-write permissions, run the following command:

```
mount -o rw,remount /
```

3. To push the trusted application to the device, run the following command:

```
scp <TZ.APPS>/qtee_tas/build/ms/bin/IAGAANAA/smcinvoke_skeleton_ta.mbn root@<IP_address>:/lib/firmware/qcom/qcm6490
```

4. To restart the device, run the following command:

```
reboot
```

4.5 Develop global platform trusted applications

The procedure provides instructions to build and develop a global platform of trusted applications. For information on trusted applications, see Qualcomm Linux Security Guide → *QualcommTEE*.

The procedure provides instructions to build and develop a global platform of trusted applications. For information on trusted applications, see Qualcomm Linux Security Guide → *QualcommTEE*.

1. Go to `cd TZ.APPS.1.0`.
2. Copy the `qtee_tas/sdk/latest/external/samples/example_gpapp/GPTA` directory in the `qtee_tas/apps/securemsm/trustzone/qsapps/example_gpapp/GPTA` directory.

   The SRC directory and SConscript are copied from this location.
3. Copy the `qtee_tas/sdk/latest/external/samples/example_gpapp/common` directory in the `qtee_tas/apps/securemsm/trustzone/qsapps/example_gpapp/common` directory.

   The INC is copied from this location.
4. Copy the `qtee_tas/sdk/latest/external/samples/example_gpapp/shared_headers` to `qtee_tas/apps/securemsm/trustzone/qsapps/example_gpapp/shared_headers`
5. Copy SConstruct from the `qtee_tas/apps/securemsm/trustzone/qsapps/sample_ta/` directory to the `qtee_tas/apps/securemsm/trustzone/qsapps/example_gpapp/GPTA/src` directory.

   Ensure both the SConscript and SConstruct files are in the same location. Your directory structure should look like:

```
── common
│    └── inc
│        └── stringl.h
└── inc
    ├──gp_basic.h
        ├──gp_time.h
└── src
    ├── ta_main.c
    ├── gp_time.c
    ├──gp_basic.c
    ├── SConscript
    ├── SConstruct
└──shared_headers
       ├──common.h
```

6. Move to the root directory workspace. The `ls` command displays the `/qtee_tas` directory, which is the root directory for trusted application compilation.

The following are the build instructions:

Table : Prerequisites to build trusted applications

Software/tool	Description
Python	v3.10
SCons	v3.0.1 or later
Qualcomm® Snapdragon™ LLVM toolchain 16.0.7	Check `sdk_config_lnx.cfg` for any version changes.

Configure Sectools environment:
`$ cd TZ.APPS.1.0`
`$ export SECTOOLS=<Meta Path>/common/ sectoolsv2/ ext/Linux/sectools`
	Go to the directory: `cd TZ.APPS.1.0`.
	Set the `SECTOOLS` environment variable to the path of the Sectools directory in your workspace. For example:
	For QCS6490/QCS5430:
	`export SECTOOLS=/local/mnt/ workspace/ qualcomm-linux-spf-1-0_ap_ standard_oem_nomodem/ QCM6490.LE.1.0/common/ sectoolsv2/ext/Linux/ sectools`
	For QCS9075:
	`export SECTOOLS=export SECTOOLS=/ local/mnt/workspace/ qualcomm-linux-spf-1-0_ ap_ standard_oem_nm-qimpsdk/ QCS9100.LE.1.0/ common/ sectoolsv2/ext/Linux/ sectools`

Run the following commands to build trusted applications:

```
$ python3 <scons> -C <TA SConstruct location from root dir> QTEE_
SDK=qtee_tas/sdk/latest/external BUILD_ROOT=. CHIPSET=<chipset> qtee_
sdk_version=latest <ta alias name mentioned in Sconscript>
```

For example:

- For QCS6490/QCS5430:

```
python3 ../TZ.XF.5.0/trustzone_images/tools/build/scons/SCons/
scons -C qtee_tas/apps/securemsm/trustzone/qsapps/example_gpapp/
GPTA/src/ QTEE_SDK=qtee_tas/sdk/latest/external BUILD_ROOT=.
CHIPSET=kodiak qtee_sdk_version=latest example_gpapp_ta
```

- For QCS9075:

```
python3 ../TZ.XF.5.0/trustzone_images/tools/build/scons/SCons/
scons -C qtee_tas/apps/securemsm/trustzone/qsapps/example_gpapp/
GPTA/src/ QTEE_SDK=qtee_tas/sdk/latest/external BUILD_ROOT=.
CHIPSET=lemans qtee_sdk_version=latest example_gpapp_ta
```

Note: Install scon in your workspace if it doesn't work in the command.

4.6 Develop global platform client applications

The global platform-based client application is a nonsecure, Linux client interface to communicate with the global platform-based trusted applications in Qualcomm TEE. The communication takes place through the global platform APIs.

1. To ensure that the host machine is set up, see Qualcomm Linux Build Guide → $GitHubworkflowforunregisteredusers$.
2. To sync the Linux build with the prebuilts, see Qualcomm Linux Build Guide → $GitHubworkflow(firmwareandextras)$.

   Upon successful completion, the `<WORKSPACE DIR>/layers` directory is created.
3. Create the client application directory under `<current_workspace>layersmeta-qcom-hwerecipes-security<CA_direcotry>`.

   This directory must contain the recipe file along with the source code in the following format:

```
├── <CA_directory>
│   ├── <CA-Recipe>.bb
│   │   files
│   │       ├── src/inc files
│   ├── makefile.am
│   ├── configure.ac
```

   For sample code, see Global platform skeleton client applications source listing.
4. Ensure that build-time dependencies on the `securemsm` and `dmabuf` recipes are marked as follows:

```
DEPENDS :=securemsm-headers securemsm-features minkipc
```

   Marking the dependencies allow you to use the -l flags in the Makefile.am and include any of the headers.
5. Import the headers from the `securemsm-headers` package to your package by adding the following to the `configure.ac` file in the `<current_workspace>LE_`

HLOSlayersmeta-qcom-hwerecipes-security<CA_directory>filesconfigure.ac path:

```
PKG_CHECK_MODULES([SECUREMSMHEADERS], [securemsm-headers])
AC_SUBST([SECUREMSMHEADERS_CFLAGS])
AC_SUBST([SECUREMSMHEADERS_LIBS])
```

6. Register the Makefile.am file in your configure.ac file as follows:

```
AC_CONFIG_FILES([ \<Path to your Makefile.am>/Makefile \])
```

7. In your Makefile.am file, import these headers through the following:

```
AM_CPPFLAGS = @SECUREMSMHEADERS_CFLAGS@ \
And add linkages to libraries:
<binary_name>_LDFLAGS += -lqcbor -lminkdescriptor -ldmabufheap
```

   For sample makefile and configure.ac files, see Global platform skeleton client applications source listing.

8. Ensure that you add the recipe name at: <current_workspace>layersmeta-qcom-hwerecipes-securitypackagegroupspackagegroup.bb.

Build client applications

When the client application source code and the recipe are complete and ready, follow these steps:

1. Build the client application:

```
export SHELL=/bin/bash
```

   Set up the build environment. For instructions, see Qualcomm Linux Build Guide → $GitHubworkflowforunregisteredusers.$

```
bitbake <recipe_file_name>
```

2. Clean the build:

```
bitbake -f -c cleanall smci-skeleton-client
```

   The output location is:

   <Current_workspace>/build-qcom-wayland/tmp-glibc/work/qcm6490-qcom-linux/<recipe_name>/1.0-r0/image/usr/bin/<CA_name>.

3. Load the client application to the device and explicitly push the client application to the device:

1. Connect to the device using SSH.

   For instructions, see Qualcomm Linux Build Guide → *Howto* → *SigninusingSSH*.

2. Run the following command:

```
mount -o rw,remount /
scp <CA-bin> root@<IP_address>:/usr/bin/
chmod + x <CA-bin>
```

When all the images are flashed, the binary will be made available on the device in the `/usr/bin` directory.

4.7 Execute client and trusted applications

The procedures provide steps to run client and trusted applications.

The following is an example of how to run the `smcinvoke_skeleton_client` binary to the `smcinvoke_skeleton_ta64` trusted applications.

```
smcinvoke_skeleton_client /lib/firmware/qcom/qcm6490/smcinvoke_
skeleton_ta64.mbn 1
```

1. Verify the client and trusted applications execution from the logs.

   To check the variable/log/messages for the client application execution prints, run the following command:

```
cat var /log/messages
```

   Sample log:

```
Jan  6 04:53:25 qcm6490 user.debug smcinvoke_skeleton_client:
DBG:appPath is /lib/firmware/qcom/qcm6490/smcinvoke_skeleton_
ta64.mbn
Jan  6 04:53:25 qcm6490 user.notice smcinvoke_skeleton_client:
INFO:load /lib/firmware/qcom/qcm6490/smcinvoke_skeleton_ta64.
mbn, size 36286, buffer 0x158350f0
Jan  6 04:53:25 qcm6490 user.notice smcinvoke_skeleton_client:
INFO:Load app /lib/firmware/qcom/qcm6490/smcinvoke_skeleton_
ta64.mbn succeeded
Jan  6 04:53:25 qcm6490 user.notice smcinvoke_skeleton_client:
INFO:Loading the application succeeded.
```

2. Check the TrustZone diag log for the trusted applications loading prints. See Qualcomm Linux Security Guide → *Debug* for TrustZone diag log collection.

Sample log:

```
 [2ebd707d6 / 653.472531](TZBSP_MINK_APP_LOAD_START
 App_start: Application load start: Name:
0x736d63696e766f6b655f736b656c65746f6e5f7461363400000000

[2ebd75fc7 / 653.473703](TZBSP_MINK_APP_LOAD_COMPLETED
App_load: Application loaded: Name:
0x736d63696e766f6b655f736b656c65746f6e5f7461363400000000
```

Note: The trusted application's name string is hexadecimal-encoded. For ASCII conversion, `0x736d63696e766f6b655f736b656c65746f6e5f7461363400000000` translates to `smcinvoke_skeleton_ta64`.

5 Security sevices examples

You can see the examples in this addendum to load CA and TA using different interfaces and run security services. All sample references from TrustZone are at `<TZ.APPS build>/qtee_tas/sdk//latest/external/samples/`

5.1 SMC invoke interface-based client applications

To load the trusted application, see the following sample function in the Linux user space client application.

Create directory to build client application

1. Create the `smci_skeleton_recipe` directory under `flayersmeta-qcom-hwerecipes-security` , which must have the following structure:

```
Smci_skeleton_recipe
        |__files
            |_src/
              |__ smci_ca_main.c
              |__ smcinvoke_skeleton.c
            |__inc/
                |__ ISMCIExample.h
                |__ smcinvoke_skeleton.h
                |__ utils.h
                        |__configure.ac
          |__makefile.am
      |__smci-skeleton-client_1.0.bb
```

Note: The same file contents can be reused for a quick client application validation.

2. Run the following commands to compile the client application (See Develop SMC invoke client applications).

```
$MACHINE=qcm6490 DISTRO=qcom-wayland source setup-
environment
$bitbake smci-skeleton-client
```

The client application binaries are available at:
`./tmp-glibc/work/qcm6490-qcom-linux/smci-skeleton-client/1.0-r0/image/usr/bin/smcinvoke_skeleton_client`.

Sample source code

```
smci-skeleton-client_1.0.bb :
inherit autotools-brokensep pkgconfig deploy python3native

DESCRIPTION = "SMCInvoke based Skeleton Client Application"

LICENSE = "Qualcomm-Technologies-Inc.-Proprietary"
LIC_FILES_CHKSUM = "file://${QCOM_COMMON_LICENSE_DIR}/${LICENSE};
md5=58d50a3d36f27f1a1e6089308a49b403"

DEPENDS = "minkipc securemsm-features securemsm-headers qcom-
libdmabufheap"

SRC_URI = "file://configure.ac \
           file://Makefile.am \
           file://src/smci_ca_main.c \
           file://src/smcinvoke_skeleton.c \
           file://inc/ISMCIExample.h \
           file://inc/smcinvoke_skeleton.h \
           file://inc/utils.h"

PACKAGE_ARCH = "${MACHINE_ARCH}"

S = "${WORKDIR}"

RM_WORK_EXCLUDE = "${PN}"

FILES:${PN} += "/usr/bin/*"
FILES:${PN} += "${bindir}/*"
```

files/configure.ac

```
AC_PREREQ(2.61)
AC_INIT([smci_skeleton_client], 1.0.0)
AM_INIT_AUTOMAKE([-Wall gnu foreign subdir-objects])
AM_MAINTAINER_MODE
AC_CONFIG_HEADER([config.h])
AC_CONFIG_MACRO_DIR([m4])

AC_PROG_CC
AM_PROG_CC_C_O
AM_PROG_AR
AM_PROG_AS
AC_PROG_LIBTOOL
AC_PROG_AWK
AC_PROG_CPP
AC_PROG_INSTALL
AC_PROG_LN_S
AC_PROG_MAKE_SET
AC_PROG_CXX

PKG_PROG_PKG_CONFIG

PKG_CHECK_MODULES([SECUREMSMHEADERS], [securemsm-headers])
AC_SUBST([SECUREMSMHEADERS_CFLAGS])
AC_SUBST([SECUREMSMHEADERS_LIBS])

AC_ARG_WITH([glib],
      AC_HELP_STRING([--with-glib],
         [enable glib, building HLOS systems which use glib]))

if (test "x${with_glib}" = "xyes"); then
        AC_DEFINE(ENABLE_USEGLIB, 1, [Define if HLOS systems uses
glib])

        PKG_CHECK_MODULES(GTHREAD, gthread-2.0 >= 2.16, dummy=yes,
                                AC_MSG_ERROR(GThread >= 2.16 is
required))
        PKG_CHECK_MODULES(GLIB, glib-2.0 >= 2.16, dummy=yes,
                                AC_MSG_ERROR(GLib >= 2.16 is
required))
        GLIB_CFLAGS="$GLIB_CFLAGS $GTHREAD_CFLAGS"
        GLIB_LIBS="$GLIB_LIBS $GTHREAD_LIBS"
        AC_SUBST(GLIB_CFLAGS)
```

```
        AC_SUBST(GLIB_LIBS)
fi

AC_CONFIG_FILES([Makefile])

AC_OUTPUT
```

files/Makefile.am

```
AM_CFLAGS = -I./inc \
            @SECUREMSMHEADERS_CFLAGS@ \
            -DOE

c_sources = src/smci_ca_main.c \
            src/smcinvoke_skeleton.c

bin_PROGRAMS = smcinvoke_skeleton_client
smcinvoke_skeleton_client_CC = @CC@

pkgconfigdir = $(libdir)/pkgconfig

smcinvoke_skeleton_client_SOURCES = $(c_sources)
smcinvoke_skeleton_client_CFLAGS = $(AM_CFLAGS)
smcinvoke_skeleton_client_LDFLAGS = -ldl -ldmabufheap -
lminkdescriptor -lqcbor
```

files/src/smci_ca_main.c

```
/*************************************************************
Copyright (c) 2023-2024 Qualcomm Technologies, Inc.
All Rights Reserved.
Confidential and Proprietary - Qualcomm Technologies, Inc.
*************************************************************
/

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

#include "smcinvoke_skeleton.h"
#include "object.h"
```

```
#include "alog_wrapper.h"

#define ARG_HAS_ITERATION 3

static void usage(void)
{
  printf("************************************************
**********\n");
  printf("************     SMCINVOKE_SKELETON CLIENT
**************\n");
  printf("************************************************
**********\n");
  printf("\n"
         "Runs the user space tests specified by the TEST_
TYPE\n"
         "OPTION can be:\n"
         "h : Print this help message and exit\n\n\n"
         "\t- adb push smcinvoke_skeleton_ta64.mbn to /lib/
firmware on device from TZ_APPS CRM\n"
         "\t- Connect to device: From command shell, do 'adb
shell'\n"
         "\t- Run smcinvoke_skeleton_client:\n"
         "\t- do './smcinvoke_skeleton_client <appPath>
<Iterations>'\n"
         "\t- Exmaple :-\n"
         "\t- smcinvoke_skeleton_client /lib/firmware/
smcinvoke_skeleton_ta64.mbn 1\n"
         "-----------------------------------------
-------\n\n\n");
}


int main(int argc, char *argv[])
{
  int32_t exampleRet = Object_ERROR;
  char *appPath = NULL;
  int32_t optind = 1;
  int32_t test_iterations = 0;

  usage();
  if (argc < ARG_HAS_ITERATION) {
    printf("Arguments passed are less than expected\n");
    return -1;
  }
```

```
  if (argv == NULL) {
    printf("No arguments to process, exiting! \n");
    return -1;
  }

  /* read the appPath from cmd line arguments */
  appPath = argv[1];
  printf("appPath %s \n", appPath);
 // LOGD("LOGD appPath %s \n", appPath);
 // LOGE("LOGE appPath %s \n", appPath);
  ALOGD("ALOGD appPath %s \n", appPath);
  ALOGE("ALOGE appPath %s \n", appPath);
  /* Iterations */
  test_iterations = atoi(argv[2]);
  if (test_iterations < 1) {
    printf("Iteration passed is less than 1!!\n");
    return -1;
  }
  for(int32_t i = 0; i < test_iterations; i++) {
    printf("Running for iteration %d\n", i);
    exampleRet = run_smcinvoke_ta_example(appPath);
  }

  if (exampleRet) {
    printf("Errors were encountered during execution: %d!",
exampleRet);
  } else {
    printf("CA executed successfully.\n");
  }
  return exampleRet;
}
```

```
/*************************************************************
Copyright (c) 2023-2024 Qualcomm Technologies, Inc.
All Rights Reserved.
Confidential and Proprietary - Qualcomm Technologies, Inc.
*************************************************************/

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

```
#include "object.h"
#include "IAppController.h"
#include "IAppLoader.h"
#include "IClientEnv.h"
#include "ISMCIExample.h"
#include "TZCom.h"
#include "CAppLoader.h"
#include "utils.h"
#include "alog_wrapper.h"

#define TEST_OK(input)                                          \
  {                                                             \
    if(input) {                                                 \
      ALOGE("%s failed\n",__func__);                            \
      return 0;                                                 \
    }                                                           \
  }

/* This function demonstrates how to open a Trusted Application (TA)
using SMCInvoke APIs. */
int32_t run_smcinvoke_ta_example(char *appPath)
{
  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;

  /* A ClientEnv object is required before one can establish
SMCInvoke based
```

```
   *  transport with the Qualcomm Trusted Execution Environment
(QTEE). */
  ret = TZCom_getClientEnvObject(&clientEnv);
  if (Object_isERROR(ret)) {
    ALOGE("Failed to obtain clientenv from TZCom with ret = %d\n",
ret);
    clientEnv = Object_NULL;
    goto cleanup;
  }

  /* Using the ClientEnv object we retrieved, we now obtain an
AppLoader object
   * by specifying its UID, which allows us to request loading of a
Trusted
   * Application within QTEE */
  ret = IClientEnv_open(clientEnv, CAppLoader_UID, &appLoader);
  if (Object_isERROR(ret)) {
    ALOGE("Failed to get apploader object with %d!\n", ret);
    appLoader = Object_NULL;
    goto cleanup;
  }

  ALOGI("Succeeded in getting apploader object.\n");
  ALOGD("appPath is %s\n", appPath);
  /* Load the Trusted Application, and obtain an AppController object
which
   * serves as a reference to the Application */
  ret = load_app(appLoader, appPath, &appController);
  if (Object_isERROR(ret)) {
    ALOGE("Loading the application failed with %d!\n", ret);
    appController = Object_NULL;
    goto cleanup;
  }
  ALOGI("Loading the application succeeded.\n");

  /* Finally, obtain an AppObj, which will act as an interface to our
loaded
   * Trusted Application, allowing us to send commands to it over the
   * SMCInvoke based transport */
  ret = IAppController_getAppObject(appController, &appObj);
  if (Object_isERROR(ret)) {
    ALOGE("Getting the application object failed with %d!\n", ret);
    appObj = Object_NULL;
    goto cleanup;
```

```
  }
  ALOGI("Getting the application object succeeded.\n");

  /* 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!\n", ret);
  } else {
    ALOGI("Add result: %d\n", addResult);
    printf("Add result: %d\n", addResult);
  }

cleanup:
  Object_ASSIGN_NULL(appObj);
  if (!Object_isNull(appController)) {
    TEST_OK(IAppController_unload(appController));
    Object_ASSIGN_NULL(appController);
  }
  Object_ASSIGN_NULL(appLoader);
  Object_ASSIGN_NULL(clientEnv);
  return ret;
}
```

files/inc/smcinvoke_skeleton.h

```
/*************************************************************
Copyright (c) 2023-2024 Qualcomm Technologies, Inc.
All Rights Reserved.
Confidential and Proprietary - Qualcomm Technologies, Inc.
*************************************************************/

#include <stdio.h>

int32_t run_smcinvoke_ta_example(char *appPath);
```

files/inc/ISMCIExample.h

```
/*************************************************************
Copyright (c) 2023-2024 Qualcomm Technologies, Inc.
All Rights Reserved.
Confidential and Proprietary - Qualcomm Technologies, Inc.
*************************************************************/
/**
* Interface to the SMCInvoke skeleton_app functionality.
*/
/** @cond */
#pragma once
#include <stdint.h>
#include "object.h"

#define ISMCIExample_OP_add 0

static inline int32_t
ISMCIExample_release(Object self)
{
  return Object_invoke(self, Object_OP_release, 0, 0);
}

static inline int32_t
ISMCIExample_retain(Object self)
{
  return Object_invoke(self, Object_OP_retain, 0, 0);
}

static inline int32_t
ISMCIExample_add(Object self, uint32_t val1_val, uint32_t val2_val,
uint32_t *result_ptr)
{
  ObjectArg a[2]={{{0,0}}};
  struct {
    uint32_t m_val1;
    uint32_t m_val2;
  } i;
  a[0].b = (ObjectBuf) { &i, 8 };
  i.m_val1 = val1_val;
  i.m_val2 = val2_val;
  a[1].b = (ObjectBuf) { result_ptr, sizeof(uint32_t) };

  return Object_invoke(self, ISMCIExample_OP_add, a, ObjectCounts_
```

```
pack(1, 1, 0, 0));
}
```

files/inc/utils.h

```
/*************************************************************
Copyright (c) 2023-2024 Qualcomm Technologies, Inc.
All Rights Reserved.
Confidential and Proprietary - Qualcomm Technologies, Inc.
*************************************************************/
#include <stdio.h>
#include <stdint.h>
#include <stddef.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/mman.h>

#include "IAppLoader.h"
#include "object.h"
#include "CAppLoader.h"
#include "alog_wrapper.h"

static int get_file_size(const char* filename) {
  FILE* file = NULL;
  int size = 0;
  int ret = 0;

  do {
    file = fopen(filename, "r");
    if (file == NULL) {
      ALOGE("Failed to open file %s: %s (%d)\n", filename,
strerror(errno), errno);
      size = -1;
      break;
    }

    ret = fseek(file, 0L, SEEK_END);
    if (ret) {
      ALOGE("Error seeking in file %s: %s (%d)\n", filename,
strerror(errno), errno);
      size = -1;
      break;
```

```
    }

    size = ftell(file);
    if (size == -1) {
      ALOGE("Error telling size of file %s: %s (%d)\n", filename,
strerror(errno), errno);
      size = -1;
      break;
    }
  } while (0);
  if (file) {
    fclose(file);
  }
  return size;
}

static int read_file(const char* filename, size_t size, uint8_t*
buffer) {
  FILE* file = NULL;
  size_t readBytes = 0;
  int ret = 0;
  do {
    file = fopen(filename, "r");
    if (file == NULL) {
      ALOGE("Failed to open file %s: %s (%d)\n", filename,
strerror(errno), errno);
      ret = -1;
      break;
    }
    readBytes = fread(buffer, 1, size, file);
    if (readBytes != size) {
      ALOGE("Error reading the file %s: %zu vs %zu bytes: %s (%d)\n",
                filename,
                readBytes,
                size,
                strerror(errno),
                errno);
      ret = -1;
      break;
    }
      ret = size;
  } while (0);
  if (file) {
    fclose(file);
```

```
  }
  return ret;
}

int32_t load_app(Object appLoader, char *path, Object *appController)
{
  int32_t ret = Object_OK;
  size_t size       = 0;
  uint8_t * buffer  = NULL;

  do {
    ret = get_file_size(path);
    if (ret <= 0) {
      ret = -1;
      break;
    }
    size = (size_t)ret;
    buffer = malloc(sizeof(uint8_t[size]));
    if (!buffer) {
      ALOGE("Malloc failed while allocating memory to buffer\n");
      ret = Object_ERROR_KMEM;
      break;
    }

    ret = read_file(path, size, buffer);

    if (ret < 0) break;
    ALOGI("load %s, size %zu, buffer %p\n", path, size, buffer);
    ret = IAppLoader_loadFromBuffer(appLoader, buffer, size,
appController);

    } while (0);

    if (buffer) free(buffer);

    if (!ret && !Object_isNull(*appController)) {
      ALOGI("Load app %s succeeded\n", path);
    }
    else {
      ALOGE("Load app %s failed: %d\n", path, ret);
      if (Object_isNull(*appController)) {
        ALOGE("appController is NULL!\n");
        ret = -1;
      }
```

```
    }

    return ret;
}

}

// dma buffer alocation
dmaFd = DmabufHeapAlloc(bufferAllocator, "qcom,qseecom-ta",
appBufSize, 0, 0);

// memory object creation, API is exposed from libminkdescriptor
Object appElfSMO = Object_NULL;
int dup_dmaFd = dup(dmaFd);
ret = TZCom_getFdObject(dup_dmaFd, &appElfSMO); if (Object_
isERROR(ret)) {
// Failure scenario, interface doesn't guarantee out Object is
untouched so reinitialize.
appElfSMO = Object_NULL; close(dup_dmaFd); goto cleanup;
}
dup_dmaFd = -1 // to avoid accidental close in success scenario as
ownership is taken by memory object

// To copy content into dma buffer, get dam buffer from associated
dma fd dmaBuf = (unsigned char *)mmap(NULL, len, PROT_READ | PROT_
WRITE, MAP_SHARED, dmaFd, 0);
// Read TA image file into dmaBuf,

// TA load requested using shared memory object (SMO)
ret = IAppLoader_loadFromRegion(appLoader, appElfSMO, appController);
if (Object_isERROR(ret)) {
ALOGE("Loading the application failed with %d!\n", ret);
appController = Object_NULL;
goto cleanup;
}

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

cleanup: Object_ASSIGN_NULL(appObj); if (!Object_
isNull(appController))
```

```
{ TEST_OK(IAppController_unload(appController)); Object_ASSIGN_
NULL(appController);
}
Object_ASSIGN_NULL(appElfSMO); Object_ASSIGN_NULL(appLoader); Object_
ASSIGN_NULL(clientEnv); return ret;
}
```

5.2 SMC invoke interface-based trusted applications

A sample code provided for developing trusted applications. This is a C++ example.

CSMCIExample.cpp:

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

*/



#include <cstring>
#include <cstddef>
#include <cstdint>
#include <stringl.h>
#include "ISMCIExample_invoke.hpp" extern "C" {
#include "qsee_heap.h"
#include "qsee_log.h"
}

#include "object.h"

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

virtual int32_t bufferExample(const void *in_msg_ptr, size_t in_msg_
len,

void *out_msg_ptr, size_t out_msg_len, size_t *out_msg_lenout);

virtual int32_t add(int32_t int1_val, int32_t int2_val, int32_t
```

```
*result_ptr); void retain();

bool release();



private: int refs; int idx;

/* we can use this index to identify specific instances of
ISMCIExample. */ static int index;

};



int CSMCIExample::index = 0; CSMCIExample::CSMCIExample() : refs(1),
idx(index

++) {}



/* 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. Overwriting the release and
retain functions
*   is only shown here to demonstrate that it is actually possible;
CSMCIExample's
*   destructor should normally handle releasing of resources. */ bool
CSMCIExample::release()

{

if (--refs == 0) {

qsee_log(QSEE_LOG_MSG_DEBUG, "Freeing last instance of index: %d",
idx); return true;

}

return false;
```

```
}



/* When retain is called, this ISMCIExample object's reference count
is
*   incremented. This would be called when keeping a new reference to
this
*   object. */

void CSMCIExample::retain()

{




refs++;

}



/* An example showing how to copy a message into a buffer for
delivery back to

* the CA. */

int32_t CSMCIExample::bufferExample(const void *in_msg_ptr, size_t
in_msg_len,

void *out_msg_ptr, size_t out_msg_len, size_t *msg_lenout)

{

if (in_msg_len == 0) {

qsee_log(QSEE_LOG_MSG_ERROR, "Supplied message length was 0.\n");
return CSMCIExample::ERROR_INPUT_BUFFER_TOO_SMALL;

}
```

```
qsee_log(QSEE_LOG_MSG_DEBUG, "Message received from CA: %s", in_msg_
ptr);



/* create a message to send back to the CA; copy our message to the
msg_ptr
*   pointer, and set msg_lenout to the return value of memscpy (the
acutal
*   size of the copied data) */

const char out_message[] = "Hello from secure side!"; if (out_msg_len
== 0) {



qsee_log(QSEE_LOG_MSG_ERROR, "Supplied output buffer length was 0.\n
"); return CSMCIExample::ERROR_OUTPUT_BUFFER_TOO_SMALL;
}
*msg_lenout = memscpy(out_msg_ptr, out_msg_len, out_message,
sizeof(out_message)); return Object_OK;
}

/* A simple example showing how to add two values inside the TA and
give the
* result back to the CA. */
int32_t CSMCIExample::add(int32_t int1_val, int32_t int2_val, int32_t
*result_ptr)
{
*result_ptr = int1_val + int2_val; return Object_OK;
}

int32_t CSMCIExample_open(Object *objOut)
{
CSMCIExample *me = new CSMCIExample(); if (!me) { qsee_log(QSEE_LOG_
MSG_ERROR, "Memory allocation for CSMCIExample failed!"); return
Object_ERROR_KMEM;
}

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

5.3 Load trusted applications using SMC invoke

A native Linux client application can load the trusted application using SMC invoke.

All invocations from a Linux client into the trusted environment begin with an `IClientEnv` object. A native Linux client can use `TZCom_getClientEnvObject()` to obtain an `IClientEnv` object.

To load a trusted application using SMC invoke for native Linux clients, follow these steps:

1. To load the user space trusted application, run the following code to:

```
int32_t load_ta_example(char *appPath)
{
 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.

 /* Before we can obtain an AppLoader object, a ClientEnv object
is required */
 ret = TZCom_getClientEnvObject(&clientEnv);
 if (Object_isERROR(ret)) {
  ALOGE("Failed to obtain clientenv from TZCom with ret = %d\n",
ret);
  clientEnv = Object_NULL;
  goto cleanup;
 }

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

```
  goto cleanup;
 }

 // dma buffer alocation
 dmaFd = DmabufHeapAlloc(bufferAllocator, "qcom,qseecom-ta",
appBufSize, 0, 0);

 // memory object creation, API is exposed from
libminkdescriptor
 Object appElfSMO = Object_NULL;
 int dup_dmaFd = dup(dmaFd);
 ret = TZCom_getFdObject(dup_dmaFd, &appElfSMO);
 if (Object_isERROR(ret)) {
   // Failure scenario, interface doesn't guarantee out Object
is untouched so reinitialize.
   appElfSMO = Object_NULL;
   close(dup_dmaFd);
   goto cleanup;
 }
 dup_dmaFd = -1 // to avoid accidental close in success scenario
as ownership is taken by memory object

 // To copy content into dma buffer, get dam buffer from
associated dma fd
 dmaBuf = (unsigned char *)mmap(NULL, len, PROT_READ | PROT_
WRITE, MAP_SHARED, dmaFd,  0);
 // Read TA image file into dmaBuf,

 // TA load requested using shared memory object (SMO)
 ret = IAppLoader_loadFromRegion(appLoader, appElfSMO,
appController);
 if (Object_isERROR(ret)) {
  ALOGE("Loading the application failed with %d!\n", ret);
  appController = Object_NULL;
  goto cleanup;
 }

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

```
cleanup:
 Object_ASSIGN_NULL(appObj);
 if (!Object_isNull(appController)) {
  TEST_OK(IAppController_unload(appController));
  Object_ASSIGN_NULL(appController);
 }
 Object_ASSIGN_NULL(appElfSMO);
 Object_ASSIGN_NULL(appLoader);
 Object_ASSIGN_NULL(clientEnv);
 return ret;
}
```

2. Transfer the signed trusted application binary into the native client process context buffer or DMA heap. If you are using a DMA heap, it must be converted into the memory object and shared with Qualcomm TEE using an SMC invoke interface to load the trusted application from the DMA heap.
3. Use the functions to obtain the following:
   - An `IClientEnv` object, which can be obtained using `TZCom_getClientEnvObject()`.
   - An `IAppLoader` object, which can be obtained using `IClientEnv_open()`.
4. Use the `IAppLoader` object to load the trusted application and obtain an `IAppController` object.
5. Use the `IAppController` object to retrieve the trusted application object, specifically, the object returned from `app_getAppObject()`.

5.4 SMC invoke skeleton C++ trusted applications source listing

Sample code for `CSMCIExample.cpp`.

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

#include <cstring>
#include <cstddef>
#include <cstdint>
#include <stringl.h>
```

```
#include "ISMCIExample_invoke.hpp" extern "C" {
#include "qsee_heap.h"
#include "qsee_log.h"
}
#include "object.h"

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

virtual int32_t bufferExample(const void *in_msg_ptr,
size_t in_msg_len,
void *out_msg_ptr, size_t out_msg_len, size_t *out_msg_lenout);
virtual int32_t add(int32_t int1_val, int32_t int2_val, int32_t
*result_ptr); void retain();
bool release();

private: int refs; int idx;
/* we can use this index to identify specific instances of
ISMCIExample. */ static int index;
};

int CSMCIExample::index = 0; CSMCIExample::CSMCIExample() : refs(1),
idx(index++) {}
/* 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. Overwriting the release and
retain functions
*   is only shown here to demonstrate that it is actually possible;
CSMCIExample's
*   destructor should normally handle releasing of resources. */ bool
CSMCIExample::release()
{
if (--refs == 0) {
qsee_log(QSEE_LOG_MSG_DEBUG, "Freeing last instance of index: %d",
idx); return true;
}
return false;
}

/* When retain is called, this ISMCIExample object's reference count
```

```
is
*   incremented. This would be called when keeping a new reference to
this
*   object. */
void CSMCIExample::retain()
{



refs++;
}

/* An example showing how to copy a message into a buffer for
delivery back to
* the CA. */
int32_t CSMCIExample::bufferExample(const void *in_msg_ptr,
size_t in_msg_len,
void *out_msg_ptr, size_t out_msg_len, size_t *msg_lenout)
{
if (in_msg_len == 0) {
qsee_log(QSEE_LOG_MSG_ERROR, "Supplied message length was 0.\n");
return CSMCIExample::ERROR_INPUT_BUFFER_TOO_SMALL;
}

qsee_log(QSEE_LOG_MSG_DEBUG, "Message received from CA: %s", in_msg_
ptr);

/* create a message to send back to the CA; copy our message to the
msg_ptr
*   pointer, and set msg_lenout to the return value of memscpy (the
acutal
*   size of the copied data) */
const char out_message[] = "Hello from secure side!";
if (out_msg_len == 0) {
qsee_log(QSEE_LOG_MSG_ERROR, "Supplied output buffer length was 0.\n
"); return CSMCIExample::ERROR_OUTPUT_BUFFER_TOO_SMALL;
}
*msg_lenout = memscpy(out_msg_ptr, out_msg_len, out_message,
sizeof(out_message)); return Object_OK;
}

/* A simple example showing how to add two values inside the TA and
give the
* result back to the CA. */
```

```
int32_t CSMCIExample::add(int32_t int1_val, int32_t int2_val, int32_t
*result_ptr)
{
*result_ptr = int1_val + int2_val; return Object_OK;
}

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

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

5.5 SMC invoke skeleton client applications source listing

Sample code for `smci_ca_main.c`.

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

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

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

#include "TZCom.h"
#include "alog.h"
#include "ca_paths.h" // Required for 'TA_PATH'
#include "map_library.h"
```

```
#include "qtee_init.h" // QTEE off-target environment initialization
#include "stringl.h"

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

/* This function demonstrates how to send a buffer to the
ISMCIExample
*   implementation, and how to also receive one back. */ static
int32_t run_buffer_example(Object *appObj)
{
int32_t ret;

const char inMsg[] = "Hello from CA side!"; const size_t inMsgLen =
sizeof(inMsg);

char outMsg[256];
const size_t outMsgLen = sizeof(outMsg); size_t outMsgLenOut;

ret = ISMCIExample_bufferExample(*appObj, inMsg, inMsgLen, outMsg,
outMsgLen, &outMsgLenOut); if (Object_isERROR(ret)) {
ALOGE("Call returned error %d!\n", ret); return ret;
}

if (outMsgLenOut == 0 || outMsgLenOut >= outMsgLen) { ALOGE("Out Msg
has an improper length %zu!\n", outMsgLenOut); return ret;
}

outMsg[outMsgLenOut] = '\0'; ALOGD("%s successful.\n",  func );
ALOGV("outMsg: %s\n", outMsg); return ret;
}

static int32_t run_addition_example(Object *appObj)
{
int32_t ret; int32_t val1 = 2;



int32_t val2 = 5; int32_t addResult;
ret = ISMCIExample_add(*appObj, val1, val2, &addResult); if (Object_
isERROR(ret)) {
ALOGE("Addition returned error %d!\n", ret);
```

```
} else {
ALOGD("Add result: %d\n", addResult);
}

return ret;
}

/* This function builds the path to our TA using strlcpy(), using the
TA name
* MINKIPC_TA_NAME - this is defined in the TA SConscript. */ static
char *create_image_path(void)
{
/* ensure PATH_MAX is defined */
#if !defined(PATH_MAX)
#if defined(MAX_PATH)
#define PATH_MAX MAX_PATH
#else
#define PATH_MAX 260
#endif
#endif

static const char testTAName[] = MINKIPC_TA_NAME; static char
imageName[PATH_MAX];

size_t index = strlcpy(imageName, TA_PATH, sizeof(imageName));
index += strlcpy(&imageName[index], testTAName, sizeof(imageName) -
index); index += strlcpy(&imageName[index], ".so", sizeof(imageName)
- index); return imageName;
}

/* This function demonstrates how to open a TA using SMCInvoke APIs.
*/ static int32_t run_smcinvoke_ta_example(void)
{
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.

void *imageBuffer = NULL;   // A pointer to the TA image size_t
imageSize = 0;  // It's size
int taFileDescriptor = -1; // A file descriptor referencing the TA

/* 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 smci_cleanup;
}

/* Before we can load our TA, we have to load it into a buffer. */
const char *imageName = create_image_path();

/* Here we call map_lib, which maps the TA specified in the path we
created
*   into a buffer 'imageBuffer', which is referenced by a file
descriptor



*   taFileDescriptor. */
taFileDescriptor = map_lib(imageName, &imageBuffer, &imageSize);

if (taFileDescriptor < 0) { ALOGE("Error mapping TA!"); ret = Object_
ERROR;
goto smci_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!\n", ret); appLoader =
Object_NULL;
goto smci_cleanup;
}
```

```
ALOGV("Succeeded in getting apploader object.\n");

/* load the application */
ret = IAppLoader_loadFromBuffer(appLoader, imageBuffer, imageSize, &
appController);
if (Object_isERROR(ret)) {
ALOGE("Loading the application failed with %d!\n", ret);
appController = Object_NULL;
goto smci_cleanup;
}

ALOGV("Loading the application succeeded.\n");

ret = IAppController_getAppObject(appController, &appObj); if
(Object_isERROR(ret)) {
ALOGE("Getting the application object failed with %d!\n", ret);
appObj = Object_NULL; goto smci_cleanup;
}
ALOGV("Getting the application object succeeded.\n"); ret = run_
buffer_example(&appObj);
if (Object_isERROR(ret)) {
ALOGE("Running buffer example failed with %d!\n", ret); goto smci_
cleanup;
}

ret = run_addition_example(&appObj); if (Object_isERROR(ret)) {
ALOGE("Running addition example failed with %d!\n", ret);
}

smci_cleanup: Object_RELEASE_IF(appObj); Object_RELEASE_
IF(appController); Object_RELEASE_IF(appLoader); Object_RELEASE_
IF(clientEnv);

if (taFileDescriptor >= 0) { unmap_lib(taFileDescriptor, imageBuffer,
imageSize);
}

return ret;
}

int main(void)
{
int ret;    // Return value for qtee_sdk calls int exampleRet; //
```

```
Return value for these examples

/* Initialize the QTEE SDK by calling qtee_sdk_init - this has to be
done



* in order to properly use the off target environment. */ ret = qtee_
sdk_init();
if (ret) {
ALOGE("QTEE SDK initialization failed with %d!\n", ret); return ret;
}

ALOGV("QTEE SDK Initialized.\n");
/*Run the SMCInvoke TA example, and then deinit the SDK after it
returns. */ exampleRet = run_smcinvoke_ta_example();
if (exampleRet) {
ALOGE("Errors were encountered during execution: %d!\n", exampleRet);
} else {
ALOGD("CA executed successfully.\n");
}

ret = qtee_sdk_deinit(); if (ret) {
ALOGE("Error occurred during sdk deinit: %d!\n", ret);
}
return (ret || exampleRet);
}
```


5.6 Global platform skeleton client applications source listing

A sample code for global platform client applications.

Run the following commands to compile the client application. See, Develop global platform client applications.

```
$MACHINE=qcm6490 DISTRO=qcom-wayland source setup-environment
$bitbake gptee-sample-ca
```

gptee-sample-ca/gptee-sample-ca_1.0.bb

```
inherit autotools-brokensep pkgconfig deploy python3native

DESCRIPTION = "GPTEE based Client Application"

LICENSE = "Qualcomm-Technologies-Inc.-Proprietary"
LIC_FILES_CHKSUM = "file://${QCOM_COMMON_LICENSE_DIR}/${LICENSE};
md5=58d50a3d36f27f1a1e6089308a49b403"

DEPENDS = "minkipc securemsm-features securemsm-headers"

SRC_URI = "file://configure.ac \
           file://Makefile.am \
           file://GPTEE_Sample_client.h \
           file://GPTEE_Sample_client.cpp"

PACKAGE_ARCH = "${MACHINE_ARCH}"

S = "${WORKDIR}"

RM_WORK_EXCLUDE = "${PN}"

FILES:${PN} += "/usr/bin/*"
FILES:${PN} += "${bindir}/*"
```

gptee_sample_ca\files\configure.ac

```
AC_PREREQ(2.61)
AC_INIT([gptee_sample_client], 1.0.0)
AM_INIT_AUTOMAKE([-Wall gnu foreign subdir-objects])
AM_MAINTAINER_MODE
AC_CONFIG_HEADER([config.h])
AC_CONFIG_MACRO_DIR([m4])

AC_PROG_CC
AM_PROG_CC_C_O
AM_PROG_AR
AM_PROG_AS
AC_PROG_LIBTOOL
AC_PROG_AWK
AC_PROG_CPP
AC_PROG_INSTALL
```

```
AC_PROG_LN_S
AC_PROG_MAKE_SET
AC_PROG_CXX

PKG_PROG_PKG_CONFIG

PKG_CHECK_MODULES([SECUREMSMHEADERS], [securemsm-headers])
AC_SUBST([SECUREMSMHEADERS_CFLAGS])
AC_SUBST([SECUREMSMHEADERS_LIBS])

AC_ARG_WITH([glib],
      AC_HELP_STRING([--with-glib],
         [enable glib, building HLOS systems which use glib]))

if (test "x${with_glib}" = "xyes"); then
        AC_DEFINE(ENABLE_USEGLIB, 1, [Define if HLOS systems uses
glib])

        PKG_CHECK_MODULES(GTHREAD, gthread-2.0 >= 2.16, dummy=yes,
                                AC_MSG_ERROR(GThread >= 2.16 is
required))
        PKG_CHECK_MODULES(GLIB, glib-2.0 >= 2.16, dummy=yes,
                                AC_MSG_ERROR(GLib >= 2.16 is
required))
        GLIB_CFLAGS="$GLIB_CFLAGS $GTHREAD_CFLAGS"
        GLIB_LIBS="$GLIB_LIBS $GTHREAD_LIBS"
        AC_SUBST(GLIB_CFLAGS)
        AC_SUBST(GLIB_LIBS)
fi

AC_CONFIG_FILES([Makefile])

AC_OUTPUT
```

gptee_sample_ca\files\makefile.am

```
ACLOCAL_AMFLAGS = -I m4

AM_CPPFLAGS = @SECUREMSMHEADERS_CFLAGS@ \
              -DOE

CPP_SOURCES =  GPTEE_Sample_client.cpp

bin_PROGRAMS = GPTEE_Sample_client
GPTEE_Sample_client_CC = @CC@
GPTEE_Sample_client_SOURCES = $(CPP_SOURCES)
GPTEE_Sample_client_CPPFLAGS = $(AM_CPPFLAGS)
GPTEE_Sample_client_LDADD = -lc -lGPTEE -lpthread
pkgconfigdir = $(libdir)/pkgconfig
library_includedir = $(pkgincludedir)

all-locat: $(bin_PROGRAMS)
```

gptee_sample_ca\files\GPTEE_Sample_client.cpp

```
/**
 * Copyright (c) 2022-2024 Qualcomm Technologies, Inc.
 * All Rights Reserved.
 * Confidential and Proprietary - Qualcomm Technologies, Inc.
 */
#include "GPTEE_Sample_client.h"

static int QTEEC_TEST_initialize_context(void)
{
  TEEC_Context context = {};
  TEEC_Result result = 0xFFFFFFFF;

  result = TEEC_InitializeContext(NULL, &context);
  if (result != TEEC_SUCCESS)
  {
    ALOGE("TEEC_InitializeContext failed in test: %s\n", __func__);
    return result;
  }

  ALOGD("TEEC_InitializeContext passed in test: %s\n", __func__);
  TEEC_FinalizeContext(&context);
```

```
  return result;
}

static int QTEEC_TEST_open_close_session(void) {
  /* Allocate TEE Client structures on the stack. */
  TEEC_Context context = {};
  TEEC_Session session = {};
  TEEC_Result result = 0xFFFFFFFF;
  uint32_t returnOrigin = 0xFFFFFFFF;
  TEEC_UUID uuid = {};

  memscpy(&uuid, sizeof(TEEC_UUID), &gpSample2UUID, sizeof(TEEC_
UUID));

  /* [1] Connect to TEE */
  result = TEEC_InitializeContext(NULL, &context);
  if (result != TEEC_SUCCESS)
  {
    return result;
  }

  /* [2] Open a Session with the TEE application. */
  /* No connection data needed for TEEC_LOGIN_USER. */
  /* No payload, and do not want cancellation. */
  result = TEEC_OpenSession(&context, &session, &uuid, TEEC_LOGIN_
USER, NULL, NULL, &returnOrigin);
  if (result != TEEC_SUCCESS)
  {
    TEEC_FinalizeContext(&context);
    return result;
  }

  /* [3] Close the Session with the TEE application. */
  TEEC_CloseSession(&session);
  /* [4] Tidy up resources */
  TEEC_FinalizeContext(&context);

  return result;
}

static int QTEEC_TEST_multiple_session(void) {
  /* Allocate TEE Client structures on the stack. */
  TEEC_Context context = {};
  TEEC_Session session_array[10];
```

```
  TEEC_Result result = 0xFFFFFFFF;
  uint32_t returnOrigin = 0xFFFFFFFF;
  int i = 0, j = 0;
  TEEC_UUID uuid = {};

  memscpy(&uuid, sizeof(TEEC_UUID), &gpSample2UUID, sizeof(TEEC_
UUID));
  /* [1] Connect to TEE  */
  result = TEEC_InitializeContext(NULL, &context);
  if (result != TEEC_SUCCESS) goto cleanup1;
  for (i = 0; i < 2; i++) {
    /* [2] Open a Session with the TEE application. */
    /* No connection data needed for TEEC_LOGIN_USER. */
    /* No payload, and do not want cancellation. */
    for (j = 0; j < 3; j++) {
      result = TEEC_OpenSession(&context, &session_array[j], &uuid,
TEEC_LOGIN_USER, NULL, NULL,
                                &returnOrigin);

      if (result != TEEC_SUCCESS) goto cleanup2;
    }
    for (j = 0; j < 3; j++) {
      /* [3] Close the Session with the TEE application. */
      TEEC_CloseSession(&session_array[j]);
    }
  }
/* [4] Tidy up resources */
cleanup2:
  TEEC_FinalizeContext(&context);
cleanup1:
  return result;
}

static void print_text(char const* const intro_message, void const*
text_addr, unsigned int size) {
  ALOGD("%s @ address = %p\n", intro_message, text_addr);
  for (unsigned int i = 0; i < size; i++) {
    ALOGD("%2x ", ((uint8_t const*)text_addr)[i]);
    if ((i & 0xf) == 0xf) ALOGD("\n");
  }
  ALOGD("\n");
}

static int QTEEC_TEST_TEEC_AllocateSharedMemory(void) {
```

```
  /* Allocate TEE Client structures on the stack. */
  TEEC_Context context = {};
  TEEC_Session session = {};
  TEEC_Operation operation = {};
  TEEC_Result result = 0xFFFFFFFF;
  TEEC_SharedMemory commsSM = {};
  uint32_t returnOrigin = 0xFFFFFFFF;
  TEEC_UUID uuid = {};
  uint32_t command = GP_SAMPLE_BUFFER_MULTIPLY_TEST;
  int cmpBufferRet = -1;
  uint8_t checkBuffer[BUFFERSIZE] = {};

  memscpy(&uuid, sizeof(TEEC_UUID), &gpTestUUID, sizeof(TEEC_UUID));
  /* [1] Connect to TEE */
  result = TEEC_InitializeContext(NULL, &context);

  if (result != TEEC_SUCCESS) goto cleanup1;

  /* [2] Open session with TEE application */
  /* Open a Session with the TEE application. */
  /* No connection data needed for TEEC_LOGIN_USER. */
  /* No payload, and do not want cancellation. */
  result = TEEC_OpenSession(&context, &session, &uuid, TEEC_LOGIN_
USER, NULL, NULL, &returnOrigin);
  if (result != TEEC_SUCCESS) goto cleanup2;

  /* [3] Initialize the Shared Memory buffers */
  /* [a] Communications buffer. */
  commsSM.size = BUFFERSIZE;
  commsSM.flags = TEEC_MEM_INPUT | TEEC_MEM_OUTPUT;
  /* Use TEE Client API to allocate the underlying memory buffer. */

  result = TEEC_AllocateSharedMemory(&context, &commsSM);
  if (result != TEEC_SUCCESS) goto cleanup3;
  /* Initialize buffer to all 1's */
  memset(commsSM.buffer, 0x1, commsSM.size);
  /* Initialize checkBuffer to all 1's */
  memset(checkBuffer, 0x1, commsSM.size);

  /* [4] Issue commands to operate on allocated and registered
buffers  */
  /* [a] Set the parameter types */
  operation.paramTypes = 0;
  operation.paramTypes =
```

```
      TEEC_PARAM_TYPES(TEEC_VALUE_INPUT, TEEC_MEMREF_PARTIAL_INOUT,
TEEC_NONE, TEEC_NONE);
  /* [b] Set the multiplier value input param[0]
  and buffer param[1] to operate on */
  operation.params[0].value.a = 42;
  operation.params[1].memref.parent = &commsSM;
  operation.params[1].memref.offset = 0;
  operation.params[1].memref.size = BUFFERSIZE;

  print_text("Initial data buffer", commsSM.buffer, 128);

  /* [c] Issue command to multiply commsSM by
  value set  in first parameter. */
  result = TEEC_InvokeCommand(&session, command, &operation, &
returnOrigin);
  if (result != TEEC_SUCCESS) goto cleanup4;

  /* do the multiply locally */
  for (size_t cnt = 0; cnt < commsSM.size; ++cnt) {
    *(checkBuffer + cnt) *= operation.params[0].value.a;
  }
  cmpBufferRet = memcmp(commsSM.buffer, checkBuffer, commsSM.size);

  print_text("Modified data buffer", commsSM.buffer, 128);

/* [5] Tidy up resources */
cleanup4:
  TEEC_ReleaseSharedMemory(&commsSM);
cleanup3:
  TEEC_CloseSession(&session);
cleanup2:
  TEEC_FinalizeContext(&context);
cleanup1:
  return (result || cmpBufferRet);
}

static int QTEEC_TEST_TEEC_RegisterSharedMemory(void) {
  /* Allocate TEE Client structures on the stack. */
  TEEC_Context context = {};
  TEEC_Session session = {};
  TEEC_Operation operation = {};
  TEEC_Result result = 0xFFFFFFFF;
  unsigned char test_buffer[BUFFERSIZE] = {};
  TEEC_SharedMemory commsSM = {};
```

```
  uint32_t returnOrigin = 0xFFFFFFFF;
  TEEC_UUID uuid = {};
  uint32_t command = GP_SAMPLE_BUFFER_MULTIPLY_TEST ;  // CMD_BASIC_
MULT_DATA
  int cmpBufferRet = -1;
  uint8_t checkBuffer[BUFFERSIZE] = {};

  memscpy(&uuid, sizeof(TEEC_UUID), &gpTestUUID, sizeof(TEEC_UUID));
  /* [1] Connect to TEE */
  result = TEEC_InitializeContext(NULL, &context);

  if (result != TEEC_SUCCESS) goto cleanup1;

  /* [2] Open session with TEE application */
  /* Open a Session with the TEE application. */
  /* No connection data needed for TEEC_LOGIN_USER. */
  /* No payload, and do not want cancellation. */
  result = TEEC_OpenSession(&context, &session, &uuid, TEEC_LOGIN_
USER, NULL, NULL, &returnOrigin);
  if (result != TEEC_SUCCESS) goto cleanup2;

  /* [3] Initialize the Shared Memory buffers */
  /* [a] Communications buffer. */
  commsSM.size = BUFFERSIZE;
  commsSM.flags = TEEC_MEM_INPUT | TEEC_MEM_OUTPUT;
  commsSM.buffer = test_buffer;

  /* Initialize buffer to all 1's */
  memset(commsSM.buffer, 0x1, commsSM.size);
  /* Initialize checkBuffer to all 1's */
  memset(checkBuffer, 0x1, commsSM.size);

  /* Use TEE Client API to register the underlying memory buffer. */
  result = TEEC_RegisterSharedMemory(&context, &commsSM);
  if (result != TEEC_SUCCESS) goto cleanup3;

  /* [4] Issue commands to operate on allocated and registered
buffers  */
  /* [a] Set the parameter types */
  operation.paramTypes = 0;
  operation.paramTypes =
      TEEC_PARAM_TYPES(TEEC_VALUE_INPUT, TEEC_MEMREF_PARTIAL_INOUT,
TEEC_NONE, TEEC_NONE);
  /* [b] Set the multiplier value input param[0]
```

```
  and buffer param[1] to operate on */
  operation.params[0].value.a = 42;
  operation.params[1].memref.parent = &commsSM;
  operation.params[1].memref.offset = 0;
  operation.params[1].memref.size = BUFFERSIZE;

  print_text("Initial data buffer", test_buffer, 128);

  /* [c] Issue command to multiply commsSM by
  value set  in first parameter. */
  result = TEEC_InvokeCommand(&session, command, &operation, &
returnOrigin);
  if (result != TEEC_SUCCESS) goto cleanup4;

  /* do the multiply locally */
  for (size_t cnt = 0; cnt < commsSM.size; ++cnt) {
    *(checkBuffer + cnt) *= operation.params[0].value.a;
  }
  cmpBufferRet = memcmp(test_buffer, checkBuffer, commsSM.size);

  print_text("Modified data buffer", test_buffer, 128);

/* [5] Tidy up resources */
cleanup4:
  TEEC_ReleaseSharedMemory(&commsSM);
cleanup3:
  TEEC_CloseSession(&session);
cleanup2:
  TEEC_FinalizeContext(&context);
cleanup1:
  return (result || cmpBufferRet);
}

static void print_UUID(const TEEC_UUID uuid) {
  ALOGD("%X-%X-%X-%X%X%X%X%X%X%X%X\n", uuid.timeLow, uuid.timeMid,
uuid.timeHiAndVersion,
         uuid.clockSeqAndNode[0], uuid.clockSeqAndNode[1], uuid.
clockSeqAndNode[2],
         uuid.clockSeqAndNode[3], uuid.clockSeqAndNode[4], uuid.
clockSeqAndNode[5],
         uuid.clockSeqAndNode[6], uuid.clockSeqAndNode[7]);
}

static TEEC_Result QTEEC_TEST_Open_Close_Session_Ex(TEEC_UUID uuid)
```

```
{
  /* Allocate TEE Client structures on the stack. */
  TEEC_Context context = {};
  TEEC_Session session = {};
  TEEC_Result result = 0xFFFFFFFF;
  uint32_t returnOrigin = 0xFFFFFFFF;
  /* [1] Connect to TEE */
  result = TEEC_InitializeContext(NULL, &context);
  if (result != TEEC_SUCCESS) goto cleanup1;

  /* [2] Open a Session with the TEE application. */
  /* No connection data needed for TEEC_LOGIN_USER. */
  /* No payload, and do not want cancellation. */
  result = TEEC_OpenSession(&context, &session, &uuid, TEEC_LOGIN_
USER, NULL, NULL, &returnOrigin);
  if (result != TEEC_SUCCESS) goto cleanup2;

  /* [3] Close the Session with the TEE application. */
  TEEC_CloseSession(&session);

/* [4] Tidy up resources */
cleanup2:
  TEEC_FinalizeContext(&context);
cleanup1:
  return result;
}

static int QTEEC_TEST_open_close_session_gpsample_10times(void) {
  TEEC_UUID uuid = {};
  TEEC_Result result = 0xFFFFFFFF;
  memscpy(&uuid, sizeof(TEEC_UUID), &gpSample2UUID, sizeof(TEEC_
UUID));
  for (size_t i = 0; i < 10; i++) {
    ALOGD("Test Open/Close Session for gpsample: %lu th: ", i+1);
    print_UUID(uuid);
    result = QTEEC_TEST_Open_Close_Session_Ex(uuid);
    if (result)
    {
      return result;
    }
  }

  return result;
}
```

```
static int QTEEC_TEST_load_unknown_UUID(void) {
  TEEC_Result expectedResult = 0xFFFF0000;
  TEEC_Result result = 0;
  for (size_t i = 0; i < sizeof(unknownUUIDs) /
sizeof(unknownUUIDs[0]); i++) {
    TEEC_UUID uuid = {};
    memscpy(&uuid, sizeof(TEEC_UUID), &unknownUUIDs[i], sizeof(TEEC_
UUID));
    ALOGD("Test Open/Close Session for unknownUUIDs: ");
    print_UUID(uuid);
    result = QTEEC_TEST_Open_Close_Session_Ex(uuid);
    if (result != expectedResult)
    {
      return -1;
    }
  }
  return 0;
}

static void* send_cancel_request(void* arg)
{
  ALOGD("IN CANCEL THREAD\n");
  usleep(100);
  TEEC_RequestCancellation((TEEC_Operation *)arg);
  return 0;
}

static int QTEEC_TEST_invoke_cmd_cancellation_test(void)
{
  /* Allocate TEE Client structures on the stack. */
  TEEC_Context context = {};
  TEEC_Session session = {};
  TEEC_Result result = 0xFFFFFFFF;
  uint32_t returnOrigin = 0;
  TEEC_Operation operation = {};
  uint32_t command = GP_SAMPLE_WAIT_TEST;
  pthread_t reqcancel_thread;

  operation.paramTypes =
      TEEC_PARAM_TYPES(TEEC_NONE, TEEC_NONE, TEEC_NONE, TEEC_NONE);

  operation.started = 0;
```

```
  result = TEEC_InitializeContext(NULL, &context);
  if (result != TEEC_SUCCESS) {
    ALOGE("TEEC_InitializeContext failed in test: %s\n", __func__);
    return result;
  }

  result = TEEC_OpenSession(&context, &session, &gpSample2UUID, TEEC_
LOGIN_USER, NULL, NULL, &returnOrigin);
  if (result != TEEC_SUCCESS) {
    ALOGE("TEEC_OpenSession failed in test: %s, retval: 0x%x\n", __
func__, result);
    TEEC_FinalizeContext(&context);
    return result;
  }

  /* create a new thread to cancel the command */
  result = pthread_create(&reqcancel_thread,
                          NULL,
                          send_cancel_request,
                          &operation);
  if ( 0 != result ) {
    ALOGE("Error: Creating a pthread in gp_reqcancel_start is failed!
\n");
    TEEC_CloseSession(&session);
    TEEC_FinalizeContext(&context);
    return -1;
  }
  operation.started = 0;

  result = TEEC_InvokeCommand(&session, command, &operation, &
returnOrigin);
  if (result != TEEC_ERROR_CANCEL) {
    ALOGE("TEEC_InvokeCommand failed to be canceled in test: %s,
retval: 0x%x\n", __func__, result);
  }

  pthread_join(reqcancel_thread, NULL);

  TEEC_CloseSession(&session);
  TEEC_FinalizeContext(&context);
  return result;
}

static int QTEEC_TEST_invoke_cmd_test(GP_SAMPLE_TESTS_IDS command_id)
```

```
{
  /* Allocate TEE Client structures on the stack. */
  TEEC_Context context = {};
  TEEC_Session session = {};
  TEEC_Result result = TEEC_ERROR_GENERIC;
  uint32_t returnOrigin = 0;
  TEEC_Operation operation = {};
  uint32_t command = (uint32_t)command_id;

  operation.paramTypes =
      TEEC_PARAM_TYPES(TEEC_NONE, TEEC_NONE, TEEC_NONE, TEEC_NONE);

  operation.started = 0;

  result = TEEC_InitializeContext(NULL, &context);
  if (result != TEEC_SUCCESS) {
    ALOGE("TEEC_InitializeContext failed in test: %s\n", __func__);
    return result;
  }

  result = TEEC_OpenSession(&context, &session, &gpSample2UUID, TEEC_
LOGIN_USER, NULL, NULL, &returnOrigin);
  if (result != TEEC_SUCCESS) {
    ALOGE("TEEC_OpenSession failed in test: %s, retval: 0x%x\n", __
func__, result);
    TEEC_FinalizeContext(&context);
    return result;
  }

  result = TEEC_InvokeCommand(&session, command, &operation, &
returnOrigin);
  if (result != TEEC_SUCCESS) {
    ALOGE("TEEC_InvokeCommand failed in test: %s,commad_id: 0x%x,
retval: 0x%x\n", __func__,command,result);
  }

  TEEC_CloseSession(&session);
  TEEC_FinalizeContext(&context);
  return result;
}

void usage(void) {
    printf("\n\n");
    printf("The currently implemented tests are:\n");
```

```
    printf("\n");
    printf("    Input 1:    TEST CASE NUMBER\n");
    printf("   ------------------------------\n");
    printf("    0 ------->  Dispaly Menu\n");
    printf("    1 ------->  Test Initialize/Deinitialize Context \n
");
    printf("    2 ------->  Test Open/Close Session \n");
    printf("    3 ------->  Test Open Multiple Sessions/Close All\n
");
    printf("    4 ------->  Test Open/Close Session 10 Iterations \n
");
    printf("    5 ------->  Test Invoke Comamnd with
AllocateSharedMemory API\n");
    printf("    6 ------->  Test Invoke Comamnd with
RegisterSharedMemory API\n");
    printf("    7 ------->  Test Request Cancellation \n");
    printf("    8 ------->  Test Open Session for Unknown UUIDs \n");
    printf("    9 ------->  Test Crypto API \n");
    printf("    10 ------>  Test SFS API \n");
    printf("    11 ------>  Test TA-TA internal API \n");
}

int main(int argc, char **argv) {
  int retval = 0;
  uint32_t passed_tests = 0;
  uint32_t failed_tests = 0;
  if (argc != 2) {
        usage();
        return -255;
  }

    int test_number = atoi(argv[1]);

    printf("Running test case %d\n", test_number);
    switch (test_number) {
        case 0:
            usage();
            break;
        case 1:
            TEST(QTEEC_TEST_initialize_context,TEEC_SUCCESS);
            break;
        case 2:
            TEST(QTEEC_TEST_open_close_session,TEEC_SUCCESS);
            break;
```

```
        case 3:
            TEST(QTEEC_TEST_multiple_session,TEEC_SUCCESS);
            break;
        case 4:
            TEST(QTEEC_TEST_open_close_session_gpsample_10times,TEEC_
SUCCESS);
            break;
        case 5:
            TEST(QTEEC_TEST_TEEC_AllocateSharedMemory,TEEC_SUCCESS);
            break;
        case 6:
            TEST(QTEEC_TEST_TEEC_RegisterSharedMemory,TEEC_SUCCESS);
            break;
        case 7:
            TEST(QTEEC_TEST_invoke_cmd_cancellation_test,TEEC_ERROR_
CANCEL);
            break;
        case 8:
            TEST(QTEEC_TEST_load_unknown_UUID,TEEC_SUCCESS);
            break;
        case 9:
            TEST_CMD(QTEEC_TEST_invoke_cmd_test,GP_SAMPLE_CRYPTO_
TEST,TEEC_SUCCESS);
            break;
        case 10:
            TEST_CMD(QTEEC_TEST_invoke_cmd_test,GP_SAMPLE_PERSISTENT_
OBJ_BASIC_TEST,TEEC_SUCCESS);
            break;
        case 11:
            TEST_CMD(QTEEC_TEST_invoke_cmd_test,GP_SAMPLE_TA_TA_TEST,
TEEC_SUCCESS);
            break;
        default:
            usage();
            retval = -1000;
            break;
    }

    return retval;
}
```

gptee_sample_ca\files\GPTEE_Sample_client.h

```
/**
 * Copyright (c) 2022-2024 Qualcomm Technologies, Inc.
 * All Rights Reserved.
 * Confidential and Proprietary - Qualcomm Technologies, Inc.
 */

#include <stdio.h>
#include <string.h>
#include <pthread.h>
#include <unistd.h>
#include <stdlib.h>
#include "TEE_client_api.h"
#include "alog_wrapper.h"

#define BUFFERSIZE 4096

#define TEST(test_name, expected_result)  retval = test_name(); \
  if (retval != expected_result) \
  { \
    ALOGE("%s failed\n", #test_name); \
    ++failed_tests; \
  } \
  else \
  { \
    ALOGV("%s passed\n", #test_name); \
    ++passed_tests; \
  } \

#define TEST_CMD(test_name,command_id,expected_result)  retval =
test_name(command_id); \
  if (retval != expected_result) \
  { \
    ALOGE("%s failed\n", #test_name); \
    ++failed_tests; \
  } \
  else \
  { \
    ALOGV("%s passed\n", #test_name); \
    ++passed_tests; \
  } \

//GP Sample Tests Case IDs
```

```
typedef enum {
  GP_INVALID_TEST = 0x00,
  GP_SAMPLE_BUFFER_MULTIPLY_TEST = 1,
  GP_SAMPLE_WAIT_TEST = 2,
  GP_SAMPLE_CRYPTO_TEST = 3,
  GP_SAMPLE_PERSISTENT_OBJ_BASIC_TEST = 4,
  GP_SAMPLE_TA_TA_TEST = 5,
} GP_SAMPLE_TESTS_IDS;

    /* UUID for gptest app */
const TEEC_UUID gpSample2UUID = {
    0x5AF8C3E6, 0xD9DF, 0x446E, {0x4F, 0xF2, 0xB2, 0xB3, 0x6C, 0x6A,
0x82, 0x79}};
const TEEC_UUID gpTestUUID = {
    0xCAD10542, 0X34E4, 0X452D, {0X61, 0X56, 0XE9, 0X79, 0XAA, 0X6E,
0X61, 0XBC}};

    /* Unknown UUIDs for test purpose */
const TEEC_UUID unknownUUIDs[] = {
    {0x3fd852b0, 0x5563, 0x11e9, {0x5F, 0x54, 0x49, 0x4D, 0x45, 0x41,
0x50, 0x49}},
    {0x3fd856ac, 0x5563, 0x11e9, {0x5F, 0x54, 0x49, 0x4D, 0x45, 0x41,
0x50, 0x49}},
    {0x3fd85968, 0x5563, 0x11e9, {0x5F, 0x54, 0x49, 0x4D, 0x45, 0x41,
0x50, 0x49}},
    {0x3fd85be8, 0x5563, 0x11e9, {0x5F, 0x54, 0x49, 0x4D, 0x45, 0x41,
0x50, 0x49}},
    {0x3fd8617e, 0x5563, 0x11e9, {0x5F, 0x54, 0x49, 0x4D, 0x45, 0x41,
0x50, 0x49}},
    {0x3fd86412, 0x5563, 0x11e9, {0x5F, 0x54, 0x49, 0x4D, 0x45, 0x41,
0x50, 0x49}},
    {0x3fd86692, 0x5563, 0x11e9, {0x5F, 0x54, 0x49, 0x4D, 0x45, 0x41,
0x50, 0x49}},
    {0x3fd86926, 0x5563, 0x11e9, {0x5F, 0x54, 0x49, 0x4D, 0x45, 0x41,
0x50, 0x49}},
    {0x3fd86be2, 0x5563, 0x11e9, {0x5F, 0x54, 0x49, 0x4D, 0x45, 0x41,
0x50, 0x49}},
    {0x3fd86e58, 0x5563, 0x11e9, {0x5F, 0x54, 0x49, 0x4D, 0x45, 0x41,
0x50, 0x49}},
};

static inline size_t memscpy(void *dst, size_t dst_size, const void
*src, size_t src_size) {
      size_t  copy_size = (dst_size <= src_size) ? dst_size : src_
```

```
size;
      memcpy(dst, src, copy_size);
      return copy_size;
}
```

5.7 Use Qualcomm TEE service APIs

A sample code where the TA invokes an SFS interface.

```
#include "object.h"
#include "qsee_log.h"
#include "qsee_ta_entry.h"
#include <stdint.h>
#include "qsee_heap.h"
#include "qsee_sfs.h"
#include "qsee_fs.h"

#define SFS_TEST_DATA "write dummy to file"

/* 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");
}

int SFS_Test(void)
{
   int fd = -1;
   int ret = 0;
   uint8_t *buf = NULL;

   buf = qsee_malloc(strlen(SFS_TEST_DATA) + 1);
   if (buf == NULL)
   {
```

```
      qsee_log(QSEE_LOG_MSG_ERROR, "Failed to create buffer! qsee_
malloc() failed!");
      return -1;
   }
   fd = qsee_sfs_open("myfile", O_RDWR | O_CREAT | O_TRUNC);
   if (fd == 0)
   {
      ret = qsee_sfs_error(fd);
      qsee_log(QSEE_LOG_MSG_ERROR, "qsee_sfs_open() failed errno = %d
", ret);
     return ret;
   }
   ret = qsee_sfs_write(fd, SFS_TEST_DATA, strlen(SFS_TEST_DATA));
   if (ret != strlen(SFS_TEST_DATA))
   {
      ret = qsee_sfs_error(fd);
      qsee_log(QSEE_LOG_MSG_ERROR, "qsee_sfs_write() failed errno =
%d", ret);
     return ret;
   }

   ret = qsee_sfs_read(fd, (char*)(uintptr_t)buf, strlen(SFS_TEST_
DATA));
   if ((ret == -1) || (ret != strlen(SFS_TEST_DATA)))
   {
     qsee_log(QSEE_LOG_MSG_ERROR, "qsee_sfs_read() failed errno = %d
", ret);
     return ret;
   }

   ret = qsee_sfs_close(fd);
   if (ret != 0)
   {
      ret = qsee_sfs_error(fd);
      qsee_log(QSEE_LOG_MSG_ERROR, "qsee_sfs_close() failed errno =
%d", ret);
      return ret;
   }

return ret;
}


void tz_app_cmd_handler(void *req, unsigned int reqlen, void *rsp,
```

```
unsigned int rsplen)
{
   int ret = 0;
   ret = SFS_Test();

}


/* 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");
}
```

5.8 Use IDL/object-based Qualcomm TEE service APIs

A sample code where the trusted application invokes the cipher interface for AES operation (CCipherAES128_UID/CCipherAES256_UID).

```
#include "CSMCIExample_open.h"
#include "object.h"
#include "qsee_log.h"
#include "qsee_ta_entry.h"
#include <stdint.h>
#include "qsee_env.h"
#include "qsee_heap.h"
#include "CCipher.h"
#include "ICipher.h"
#include "qsee_uf_aes.h"

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");
}
```

```
static int tz_app_ICipher_aes_func(SW_Cipher_Alg_Type alg, SW_
CipherModeType mode,
                                uint8_t *pt, uint32_t pt_len, uint8_t
*key,
                                uint32_t key_len, uint8_t *iv,uint8_t
*ct,
                                uint8_t verify)
{
  CryptoCntxHandle *cntx = 0;
  uint8_t *ct_tmp = 0;
  uint8_t *pt_tmp = 0;
  int status = -1;

  IovecListType   IovecListIn;
  IovecListType   IovecListOut;
  IovecType       IovecIn = {NULL, 0};
  IovecType       IovecOut = {NULL, 0};
  SW_CipherEncryptDir dir = SW_CIPHER_ENCRYPT;

  //Object CipherSwEnvObject = CryptoSw_getCipherObject();
  Object CipherObject;
  uint32_t IMode;

  /* Open Object Handle based on Key size */
  if(key_len == 16) //Use AES 128
  {
     //status = IEnv_open(CipherSwEnvObject, CCipherAES128Sw_UID, &
CipherObject);
     status = qsee_open(CCipherAES128_UID, &CipherObject); //Open
service to object CipherObject
  }
  else //Use AES 256
  {
     //status = IEnv_open(CipherSwEnvObject,CCipherAES256Sw_UID, &
CipherObject);
     status = qsee_open(CCipherAES256_UID, &CipherObject); //Open
service to object CipherObject
  }

  if( status != 0)
  {
     qsee_log(QSEE_LOG_MSG_ERROR,"CIPHER IEnv_open Failed: %d",status
);
```

```
     goto end;
  }
  if ((pt = qsee_malloc(pt_len)) == NULL)
     {
        qsee_log(QSEE_LOG_MSG_ERROR,"CIPHER Malloc for pt Failed");
        status = -1;
        goto end;
     }

  /* If verify is 0,  it means we are not using the predefined test
case and
  cannot verify the encrypted packet.
  Else allocate buffer to ct_tmp */
  if (verify == 0)
  {
     ct_tmp = ct;
     if (ct_tmp == NULL)
     {
        qsee_log(QSEE_LOG_MSG_ERROR,"CIPHER Input ct = NULL. Failed
");
        status = -1;
        goto end;
     }
     qsee_log(QSEE_LOG_MSG_DEBUG,"CIPHER Input ct = nonNULL. Success
");
  }
  else
  {
     if ((ct_tmp = qsee_malloc(pt_len)) == NULL)
     {
        qsee_log(QSEE_LOG_MSG_ERROR,"CIPHER Malloc for ct_tmp Failed
");
        status = -1;
        goto end;
     }
     qsee_log(QSEE_LOG_MSG_DEBUG,"CIPHER Malloc for ct_tmp Success");
     memset(ct_tmp, 0, pt_len);
  }

  /* Set key for encryption */
  if (0 != (status = ICipher_setParamAsData(CipherObject, ICipher_
PARAM_KEY, key, key_len)))
  {
     qsee_log(QSEE_LOG_MSG_ERROR,"CIPHER ICipher_setParamAsData for
```

```
KEY failed: %d", status);
     status = -1;
     goto end;
  }

  /* Set AES mode */
  switch(mode)
  {
     case SW_CIPHER_MODE_ECB :  IMode =  ICipher_MODE_ECB; break;
     case SW_CIPHER_MODE_CBC :  IMode =  ICipher_MODE_CBC; break;
     case SW_CIPHER_MODE_CTR :  IMode =  ICipher_MODE_CTR; break;
     case SW_CIPHER_MODE_CCM :  IMode =  ICipher_MODE_CCM; break;
     case SW_CIPHER_MODE_CTS :  IMode =  ICipher_MODE_CTS; break;
     default                 :  IMode =  status = -1; goto end;
  }

  if (0 != (status = ICipher_setParamAsU32(CipherObject, ICipher_
PARAM_MODE, IMode)))
  {
     qsee_log(QSEE_LOG_MSG_ERROR,"CIPHER ICipher_setParamAsData for
MODE Failed: %d", status);
     status = -1;
     goto end;
  }

  /* Set IV only if not NULL */
  if (mode != SW_CIPHER_MODE_ECB && iv != NULL)
  {
     if (0 != (status = ICipher_setParamAsData(CipherObject, ICipher_
PARAM_IV, iv, ICipher_AES_BLOCK_SZ)))
     {
        qsee_log(QSEE_LOG_MSG_ERROR,"CIPHER ICipher_setParamAsData
for IV Failed: %d", status);
        status = -1;
        goto end;
     }
  }

  /* Now encrypt the data */
  if (0 != (status = ICipher_encrypt(CipherObject, pt, pt_len, ct_
tmp, pt_len, (size_t *)&pt_len)))
  {
     qsee_log(QSEE_LOG_MSG_ERROR,"CIPHER ICipher_encrypt Failed: %d",
status);
```

```
     status = -1;
     goto end;
  }

  /* If NULL key pointer then we are using HW key so don't compare
encrypted result */
  if (verify == 1)
  {
     if (0 != memcmp(ct, ct_tmp, pt_len))
     {
        qsee_log(QSEE_LOG_MSG_ERROR,"CIPHER memcmp after encryption
Failed");
        status = -1;
        goto end;
     }
  }

  /* If verify is 0,  it means we are not using the predefined test
case and
  cannot verify the encrypted packet
  Else allocate memory to pt_tmp */
  if (verify == 0)
  {
     pt_tmp = pt;
     if (pt_tmp == NULL)
     {
        qsee_log(QSEE_LOG_MSG_ERROR, "CIPHER Input pt = NULL Failed
");
        status = -1;
        goto end;
     }
  }
  else
  {
     if ((pt_tmp = qsee_malloc(pt_len)) == NULL)
     {
        qsee_log(QSEE_LOG_MSG_ERROR, "CIPHER Malloc for pt_tmp Failed
");
        status = -1;
        goto end;
     }
     memset(pt_tmp, 0, pt_len);
  }
```

```
  /* We must set parameters again so we can do the decrypt */
  if (0 != (status = ICipher_setParamAsData(CipherObject, ICipher_
PARAM_KEY, key, key_len)))
  {
     qsee_log(QSEE_LOG_MSG_ERROR,"CIPHER ICipher_setParamAsData for
KEY Failed: %d", status);
     status = -1;
     goto end;
  }

  /* Set AES mode */
  if (0 != (status = ICipher_setParamAsU32(CipherObject, ICipher_
PARAM_MODE, IMode)))
  {
     qsee_log(QSEE_LOG_MSG_ERROR,"CIPHER ICipher_setParamAsData for
MODE Failed: %d", status);
     status = -1;
     goto end;
  }

  /* Set IV if not NULL */
  if (mode != SW_CIPHER_MODE_ECB)
  {
     if (0 != (status = ICipher_setParamAsData(CipherObject, ICipher_
PARAM_IV, iv, ICipher_AES_BLOCK_SZ)))
     {
        qsee_log(QSEE_LOG_MSG_ERROR,"CIPHER ICipher_setParamAsData
for IV Failed: %d", status);
        status = -1;
        goto end;
     }
  }

  /* Now decrypt the data */
  if (0 != (status = ICipher_decrypt(CipherObject, ct_tmp, pt_len,
pt_tmp, pt_len, (size_t *)&pt_len)))
  {
     qsee_log(QSEE_LOG_MSG_ERROR,"CIPHER ICipher_decrypt Failed: %d",
status);
     status = -1;
     goto end;
  }

  /* Now compare decrypted results */
```

```
  if (verify == 1)
  {
     if (0 != memcmp(pt, pt_tmp, pt_len))
     {
        qsee_log(QSEE_LOG_MSG_ERROR,"CIPHER memcmp after decryption
Failed");
             status = -1;
        goto end;
     }
  }

end:
  /* Free object */
  if (!Object_isNull(CipherObject))
  {
     ICipher_release(CipherObject);
  }

  /* Free malloc data */
  if (verify == 1)
  {
     if (ct_tmp)
     {
        qsee_free(ct_tmp);
        ct_tmp = 0;
     }

     if (pt_tmp)
     {
        qsee_free(pt_tmp);
        pt_tmp = 0;
     }
  }
  return status;
}

void tz_app_cmd_handler(void *req, unsigned int reqlen, void *rsp,
unsigned int rsplen)
{
   int ret = 0;
   uint8_t *pt = (uint8_t *)req;
   uint32_t pt_len = (uint32_t)reqlen;
   pt_len = 16;
   int status = -1;
```

```
   uint8_t *ct = (uint8_t *)rsp;
   uint8_t key[] = { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F};
   memset(pt, 0x01, pt_len);
   ret = tz_app_ICipher_aes_func(SW_CIPHER_ALG_AES192 ,SW_CIPHER_
MODE_ECB, pt, pt_len, key, sizeof(key), NULL,ct, 1);

}


void tz_app_shutdown(void)
{
  qsee_log(QSEE_LOG_MSG_DEBUG, "App shutdown");
}
```

6 References

6.1 Related documents

Title	Number
Qualcomm Technologies, Inc.	
Qualcomm Linux Security Guide	80-70018-11
Qualcomm Linux Build Guide	80-70018-254
Resources	
GlobalPlatform Technology TEE Internal Core API Specification Version	https://globalplatform.org/wp-content/uploads/2018/06/GPD_TEE_Internal_Core_API_Specification_v1.1.2.50_PublicReview.pdf

6.2 Acronyms and terms

Acronym or term	Definition
AAD	Attempt algorithm designator
BIMC	Bus-integrated memory controller
CMAC	Cipher-based message authentication code
CPZ	Content protected zone
DMA	Direct memory access
DRM	Digital rights management
ECC	Elliptic curve cryptography
ECDH	Elliptic curve diffie-hellman
ECDSA	Elliptic curve digital signature algorithm
ECIES	Elliptic curve integrated encryption scheme
EL0, EL1, EL2, and EL3	Exception levels

Acronym or term	Definition
ELF	Executable and linking format
GCM	Galois counter mode
GPCE	General-purpose cryptographic engine
HAL	Hardware abstraction layer
HDCP	High-bandwidth digital content protection
HLOS	High-level operating system
HMAC	Hashed message authentication code
I2C	Inter integrated circuit
ICE	Inline crypto engine
KDF	Key derivation function
MAC	Message authentication code
MINK	Mini kernel
MPU	Memory protection unit
OCIMEM	On-chip internal memory
OEM	Original equipment manufacturer
PIL	Peripheral image loader
pIMEM	Protected memory
PRNG	Pseudo-random number generator
QRNG	Qualcomm random number generator
Qualcomm TEE	Qualcomm Trusted Execution Environment
RMA	Returned material for analysis
RPMB	Replay protected memory block
RSA	Rivest-Shamir-Adelmann
SEL0 and SEL1	Secure exception levels
SFS	Secure file system
SKU	Stock keeping unit
SMC	Secure monitor call
SNoC	System-on-network chip
SPI	Serial peripheral interface
SSL	Secure sockets layer
SWI	Software interface
TZBSP	TrustZone board support package
UFS	Universal flash storage
UUID	Universal unique identifier
VMID	Virtual machine ID
XBL	eXtensible boot loader
xPU	External protection unit

LEGAL INFORMATION

Your access to and use of this material, along with any documents, software, specifications, reference board files, drawings, diagnostics and other information contained herein (collectively this "Material"), is subject to your (including the corporation or other legal entity you represent, collectively "You" or "Your") acceptance of the terms and conditions ("Terms of Use") set forth below. If You do not agree to these Terms of Use, you may not use this Material and shall immediately destroy any copy thereof.

1) Legal Notice.

This Material is being made available to You solely for Your internal use with those products and service offerings of Qualcomm Technologies, Inc. ("**Qualcomm Technologies**"), its affiliates and/or licensors described in this Material, and shall not be used for any other purposes. If this Material is marked as "**Qualcomm Internal Use Only**", no license is granted to You herein, and You must immediately (a) destroy or return this Material to Qualcomm Technologies, and (b) report Your receipt of this Material to qualcomm.support@qti.qualcomm.com. This Material may not be altered, edited, or modified in any way without Qualcomm Technologies' prior written approval, nor may it be used for any machine learning or artificial intelligence development purpose which results, whether directly or indirectly, in the creation or development of an automated device, program, tool, algorithm, process, methodology, product and/or other output. Unauthorized use or disclosure of this Material or the information contained herein is strictly prohibited, and You agree to indemnify Qualcomm Technologies, its affiliates and licensors for any damages or losses suffered by Qualcomm Technologies, its affiliates and/or licensors for any such unauthorized uses or disclosures of this Material, in whole or part.

Qualcomm Technologies, its affiliates and/or licensors retain all rights and ownership in and to this Material. No license to any trademark, patent, copyright, mask work protection right or any other intellectual property right is either granted or implied by this Material or any information disclosed herein, including, but not limited to, any license to make, use, import or sell any product, service or technology offering embodying any of the information in this Material.

THIS MATERIAL IS BEING PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESSED, IMPLIED, STATUTORY OR OTHERWISE. TO THE MAXIMUM EXTENT PERMITTED BY LAW, QUALCOMM TECHNOLOGIES, ITS AFFILIATES AND/OR LICENSORS SPECIFICALLY DISCLAIM ALL WARRANTIES OF TITLE, MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, SATISFACTORY QUALITY, COMPLETENESS OR ACCURACY, AND ALL WARRANTIES ARISING OUT OF TRADE USAGE OR OUT OF A COURSE OF DEALING OR COURSE OF PERFORMANCE. MOREOVER, NEITHER QUALCOMM TECHNOLOGIES, NOR ANY OF ITS AFFILIATES AND/OR LICENSORS, SHALL BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY EXPENSES, LOSSES, USE, OR ACTIONS HOWSOEVER INCURRED OR UNDERTAKEN BY YOU IN RELIANCE ON THIS MATERIAL.

Certain product kits, tools and other items referenced in this Material may require You to accept additional terms and conditions before accessing or using those items.

Technical data specified in this Material may be subject to U.S. and other applicable export control laws. Transmission contrary to U.S. and any other applicable law is strictly prohibited.

Nothing in this Material is an offer to sell any of the components or devices referenced herein.

This Material is subject to change without further notification.

In the event of a conflict between these Terms of Use and the *Website Terms of Use* on www.qualcomm.com, the *Qualcomm Privacy Policy* referenced on www.qualcomm.com, or other legal statements or notices found on prior pages of the Material, these Terms of Use will control. In the event of a conflict between these Terms of Use and any other agreement (written or click-through, including, without limitation any non-disclosure agreement) executed by You and Qualcomm Technologies or a Qualcomm Technologies affiliate and/or licensor with respect to Your access to and use of this Material, the other agreement will control.

These Terms of Use shall be governed by and construed and enforced in accordance with the laws of the State of California, excluding the U.N. Convention on International Sale of Goods, without regard to conflict of laws principles. Any dispute, claim or controversy arising out of or relating to these Terms of Use, or the breach or validity hereof, shall be adjudicated only by a court of competent jurisdiction in the county of San Diego, State of California, and You hereby consent to the personal jurisdiction of such courts for that purpose.

2) Trademark and Product Attribution Statements.

Qualcomm is a trademark or registered trademark of Qualcomm Incorporated. Arm is a registered trademark of Arm Limited (or its subsidiaries) in the U.S. and/or elsewhere. The Bluetooth® word mark is a registered trademark owned by Bluetooth SIG, Inc. Other product and brand names referenced in this Material may be trademarks or registered trademarks of their respective owners.

Snapdragon and Qualcomm branded products referenced in this Material are products of Qualcomm Technologies, Inc. and/or its subsidiaries. Qualcomm patented technologies are licensed by Qualcomm Incorporated.