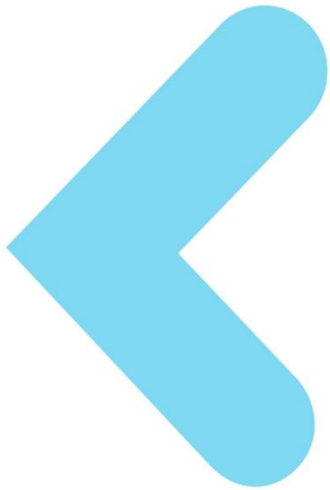
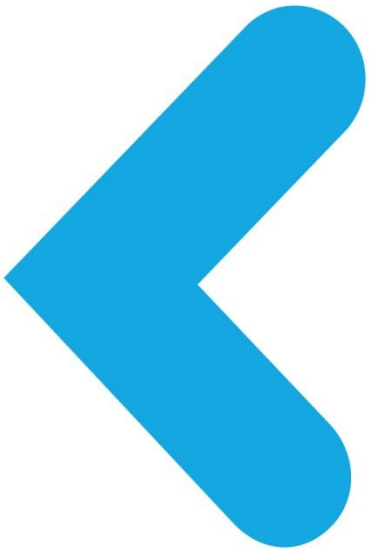
TRUSTONIC



Trustonic Application Protection Developer Manual



PREFACE

VERSION HISTORY

Version	Date	Modification
1.0	December 2016	First version for TAP 1.0 release
1.1	March 2017	Updates for TAP 1.1 release; including What's New, setting the logging/debug level, TUI API info, and makefile keywords/flags.
1.2	June 2017	Updates for TAP 1.2 release; including key sharing API, and Protected Client Library.
1.3	December 2017	Updates for TAP 1.3 release
1.4	March 2018	Updates for TAP 1.4 release, including AES-GCM and RSA support

Document code: v1.4, rev1—15/03/18

TABLE OF CONTENTS

1 INTRODUCTION

Welcome to the Trustonic Application Protection (TAP) Developer Manual.

This guide is for software developers writing TAP applications. It provides information about TAP API support. Topics include:

- TAP API overview
- TAP API support
- Using THPAgent
- Using the TUI layer library
- Fingerprint support for TAs
- Makefile keywords and flags

For an overview of Trustonic Application Protection (TAP) and the documentation, for a list of what's new in this release, and for information about designing a TA and choosing the type of protection (TEE or SWP), see the *Introduction to TAP.*

2 TAP API OVERVIEW

This chapter provides an overview of the Trustonic Application Protection (TAP) API. It introduces the TEE Internal API (used to develop TAs), and the TEE Client API (used by CAs to call a TA), and describes the interface between the TA and CA.

Trustonic Application Protection offers developers a set of APIs which are based on the GlobalPlatform TEE standard for developing Trusted Applications. Trusted Applications can be isolated and protected through a Trusted Execution Environment or by using the Software Protection mechanism and the corresponding Client, or Client Application. The Client Application is a standard Android or iOS application which calls the Trusted Application to perform security-sensitive operations.

A Trusted Application is command-oriented. Clients access a Trusted Application by opening a session with the Trusted Application and invoking commands within the session. When a Trusted Application receives a command, it parses the messages associated with the command, performs any required processing, and then sends a response back to the client.

The Trusted Application is developed using the TEE Internal API which defines the API for using features such as cryptography, secure storage of data, memory management, etc.

The Client part (the Client Application) uses the TEE Client API for calling the Trusted Application.

This section outlines the main principles of the Internal API and Client API. For further details, refer to the GlobalPlatform specifications.

For more information about determining what parts of your application should be incorporated within a TA, see the *Introduction to TAP.* After determining the split between the secure and non-secure parts, define the state in the TA and the interface between the TA and the Client Application (CA). Your design must take into consideration the restrictions of the TA environment and the CA-TA interfaces.

2.1 CALLING THE TRUSTED APPLICATION

To call and use a Trusted Application, the Client Application calls the Client API and performs the following actions:

- Creates a context
- Opens a session on the Trusted Application identified by its unique UUID
- Sends a command to the Trusted Application along with parameters and data payload and reads the response. This can be repeated several times.
- Closes the session
- Destroys the context

For passing data between the Client Application and the Trusted Application, you can either pass values as parameters or provide memory buffers containing operation payload. These memory buffers are referred to as Shared Memory.

A Shared Memory block is a region of memory allocated in the context of the Client Application memory space that can be used to transfer data between that Client Application and a Trusted Application. A Shared Memory block can be either an existing Client Application memory which is subsequently registered with the TEE Client API, or memory which is allocated on behalf of the Client Application using the TEE Client API. A Shared Memory block can be registered or allocated once and then used in multiple Commands, even in multiple sessions provided they exist within the scope of the TEE Context in which the Shared Memory was created. Typically, this pre-registration is more efficient than registering a block of memory using temporary registration if that memory buffer is used in more than one command invocation.

2.2 IMPLEMENTING THE TRUSTED APPLICATION

Each Trusted Application implements the following entry point functions which are called when a Client calls the Trusted Application:

- TA_CreateEntryPoint: this entry point is called when the Trusted Application is instantiated.
- TA_DestroyEntryPoint: this entry point is called when the Trusted Application instance is being destroyed
- TA_OpenSessionEntryPoint: this entry point is called when a Client opens a session with the Trusted Application
- TA_CloseSessionEntryPoint: this entry point is called when a Client closes a session with the Trusted Application
- TA_InvokeCommandEntryPoint: this entry point is called when the Client invokes a command of the Trusted Application, optionally passing parameters and data to the Trusted Application.

When implementing the entry points, use the Internal API to allocate memory, perform cryptographic computation, store persistent data securely, etc.

Note: Use only the Internal API in the Trusted Application code. Any other third-party API is not guaranteed to work or to offer the appropriate level of security.

2.2.1 Trusted Storage

The Internal API defines Trusted Storage for keys or general-purpose data. This API provides access to a storage space for general-purpose data and key material with guarantees on the confidentiality and integrity of the data stored and atomicity of the operations that modify the storage.

The objects in this storage space are accessible only to the TA that created them and are not visible to other TAs.

For more information, see the Trusted Storage API for Data and Keys chapter of the Global Platform TEE Internal API Specification.

2.2.2 Cryptographic Operations API

The Internal API provides the following cryptographic facilities:

- Generation and derivation of keys and key-pairs
- Support for the following types of cryptographic algorithms:
  - Digests
  - Symmetric Ciphers
  - Message Authentication Codes (MAC)
  - Authenticated Encryption algorithms such as AES-CCM and AES-GCM
  - Asymmetric Encryption and Signature
  - Key Exchange algorithms

For more information, see the Cryptographic Operations API chapter of the Global Platform TEE Internal API Specification.

2.2.2.1 High-speed AES support for stream ciphers

You can use high-speed AES for stream encryption.

Note: high-speed AES support for stream ciphers is available for SWP only.

Because high-speed AES is not as secure as the default high-security version, Trustonic do not recommend using it unless it is necessary for performance.

To enable high-speed AES in the Trusted Application's makefile, set:

```
CRYPTO_ALGORITHMS := HIGHSPEED_AES
```

Once enabled, high-speed AES is used for everything including the Secure Storage implementation. The default high-security AES and the high-speed AES versions cannot be used within the same Trusted Application. If both versions are needed by the application, you must implement two different Trusted Applications.

2.3 DEPLOYING A TA IN PRODUCTION

This section provides information you need to be aware of when deploying TAs into production.

If you're deploying TAs commercially on Samsung devices, you must complete a validation process. To do this, you send information to Trustonic, including the package name and public key from your signed APK. Trustonic then handles the validation process with Samsung directly. Once approved, Samsung issues a certificate and license file which is bundled with the application. For more information, see: https://trustonic.zendesk.com/hc/en-us/articles/200686791. For information about getting support and accessing zendesk, see the *Introduction to TAP*.

Tip! Before deploying in a production environment, remember to set the debugging level to ERROR by calling the `setLogLevel(LogLevel.level)` method. For more information, see **Setting the debug level**.

2.4 USING THE PROTECTED CLIENT LIBRARY

The TAP SDK contains a protected version of libTee.so (a TAP Client library). This differs from the standard library in the following ways:

- It is code-protected with a high-protection level (obfuscation, integrity checking, anti-debug features)
- SWP choice (`TEEC_CHOICE=SWP`) is disabled

Note: This is experimental functionality for customers who want to have a protected FALLBACK-mechanism.

To use the protected library in your TAP application build, use the following commands (the example below assumes that the current working directory is <`TAP_SDK_ROOT`>`/device`):

```
# Standard library back up
cp ./internal/ca_build/Bin/arm64-v8a/Release/libTee.so
./internal/ca_build/Bin/arm64-v8a/Release/libTee.so.backup
cp ./internal/ca_build/Bin/armeabi-v7a/Release/libTee.so
./internal/ca_build/Bin/armeabi-v7a/Release/libTee.so.backup

# Protected library use
cp ./swp/libs/libTee/arm64-v8a/Release/libTee.so.restricted
./internal/ca_build/Bin/arm64-v8a/Release/libTee.so
cp ./swp/libs/libTee/armeabi-v7a/Release/libTee.so.restricted
./internal/ca_build/Bin/armeabi-v7a/Release/libTee.so
```

3 TAP API SUPPORT

This chapter describes the set of TAP API supported when using Kinibi TEE or Software Protection. Most of this API is defined by the GlobalPlatform standard:

- The GlobalPlatform TEE Client API
- The GlobalPlatform TEE Internal Core API

Refer to the GlobalPlatform specifications for details about the GlobalPlatform APIs and how to use these APIs: http://www.globalplatform.org/specificationsdevice.asp.

Note: In TAP version 1 there is no proper API Level versioning—the one for Kinibi TEE is used. Ensure that you set the API Level to, at least, the value of **5** in Client and Trusted Application makefiles.

3.1 GLOBALPLATFORM TEE CLIENT API SUPPORT

This section outlines TAP support for the GlobalPlatform TEE Client API specification.

Function	**Kinibi TEE support**	**SWP support**
TEEC_InitializeContext	All Kinibi versions	From TAP version 1
TEEC_FinalizeContext		
TEEC_RegisterSharedMemory		
TEEC_AllocateSharedMemory		
TEEC_ReleaseSharedMemory		
TEEC_OpenSession		
TEEC_CloseSession		
TEEC_InvokeCommand		
TEEC_RequestCancellation		Not supported

3.2 GLOBALPLATFORM TEE INTERNAL CORE API SUPPORT

This section outlines TAP support for the GlobalPlatform TEE Internal Core API specifications.

Note: for information about the header file, common data types, and constants, see the TEE Internal header file.

3.2.1 Trusted Core Framework API

Memory Management Functions	**Kinibi TEE support**	**SWP support**
TEE_GetPropertyAsString	All Kinibi versions. See ***Note**.	From TAP version 1. See ****Note**.
TEE_GetPropertyAsBool		
TEE_GetPropertyAsU32		
TEE_GetPropertyAsUUID		
TEE_GetPropertyAsIdentity		
TEE_GetPropertyAsBinaryBlock	Kinibi versions with API Level 7, and later. See ***Note**.	Not supported.
TEE_GetCancellationFlag	All Kinibi versions	Not supported
TEE_UnmaskCancellation		
TEE_MaskCancellation		
TEE_CheckMemoryAccessRights		From TAP version 1. The function verifies only if a buffer is mapped completely to the memory of the TA.
TEE_Panic		From TAP version 1
TEE_SetInstanceData		
TEE_GetInstanceData		
TEE_Malloc		
TEE_Realloc		
TEE_Free		
TEE_MemMove		
TEE_MemCompare		
TEE_MemFill		

***Note**: Subset of properties depends on Kinibi TEE version.

****Note**: Currently supported properties in gpd.tee namespace: apiversion=1.0, description=TAP, cryptography=true, arith.maxBigIntSize=0; in gpd.ta namespace: singleInstance=true, multiSession=false, instanceKeepAlive=false, dataSize, stackSize, appID=TA UUID.

3.2.2 Trusted Storage API for Data and Keys

Generic Object Functions	**Kinibi TEE support**	**SWP support**
TEE_GetObjectInfo	All Kinibi versions.	From TAP version 1.
TEE_RestrictObjectUsage		
TEE_CloseObject		
TEE_GetObjectBufferAttribute		From TAP version 1. Not protected attributes.
TEE_GetObjectValueAttribute		
TEE_GetObjectInfo1	Kinibi versions with API Level 7, and later.	From TAP version 1, when built with API Level 7, or later.
TEE_RestrictObjectUsage1		
Transient Object Functions	**Kinibi TEE support**	**SWP support**
TEE_AllocateTransientObject	All Kinibi versions.	From TAP version 1.
TEE_FreeTransientObject		
TEE_ResetTransientObject		
TEE_PopulateTransientObject		Not protected attributes.
TEE_InitRefAttribute, TEE_InitValueAttribute		From TAP version 1.
TEE_CopyObjectAttributes	All Kinibi versions. See: ***Note**.	
TEE_CopyObjectAttributes1	Kinibi versions with API Level 7, and later. See: ***Note**.	
TEE_GenerateKey	All Kinibi versions except with API level 11. See: ****Note**.	From TAP version 1. TEE_TYPE_RSA_KEYPAIR not supported.
Persistent Object Functions	**Kinibi TEE support**	**SWP support**
TEE_OpenPersistentObject	All Kinibi versions.	From TAP version 1.
TEE_CreatePersistentObject		
TEE_CloseAndDeletePersistentObject		
TEE_CloseAndDeletePersistentObject1	Kinibi versions with API Level 7, and later.	
TEE_RenamePersistentObject		

Persistent Object Enumeration Functions	**Kinibi TEE support**	**SWP support**
TEE_AllocatePersistentObjectEnumerator	Kinibi versions with API Level 7, and later.	From TAP version 1.
TEE_FreePersistentObjectEnumerator		
TEE_ResetPersistentObjectEnumerator		
TEE_StartPersistentObjectEnumerator		
TEE_GetNextPersistentObject		
Data Stream Access Functions	**Kinibi TEE support**	**SWP support**
TEE_ReadObjectData	All Kinibi versions.	From TAP version 1.
TEE_WriteObjectData		
TEE_TruncateObjectData		
TEE_SeekObjectData		

***Note**: not supported for ECDSA and ECDH key types before API level 11 in TEE mode; the workaround is to use TEE_GetObjectBufferAttribute() to populate the other Object, or to build the TA using GpTRIC.

****Note**: TEE_TYPE_DES and TEE_TYPE_DES3 are not supported on Kinibi versions with API level 11; see note about DES/DES3 key generation in release notes.

3.2.2.1 SWP extension for TEE_OpenPersistentObject

`storageID equal to TEE_STORAGE_PERSO` is used to obtain the handle to the keys provisioned at build time. The following object types are currently supported for the build time provisioning:

- `TEE_TYPE_RSA_KEYPAIR` (from 1024 to 2048 bits including)
- `TEE_TYPE_ECDSA_KEYPAIR`
- All symmetric key types

Note: A `storageID` of TEE_STORAGE_PERSO can also be used by a TEE TA to access data personalized by the TAM Server plugin and triggered by THP Agent `personalizeTA()`.

3.2.3 Cryptographic Operations API

Generic Operation Functions	**Kinibi TEE support**	**SWP support**
TEE_AllocateOperation	All Kinibi versions.	From TAP version 1.
TEE_FreeOperation		
TEE_GetOperationInfo		
TEE_SetOperationKey		
TEE_GetOperationInfoMultiple	Not supported.	Not supported.

TEE_ResetOperation		
TEE_SetOperationKey2		
TEE_CopyOperation		
Message Digest Functions	**Kinibi TEE support**	**SWP support**
TEE_DigestUpdate	All Kinibi versions.	From TAP version 1.
TEE_DigestDoFinal		
Symmetric Cipher Functions	**Kinibi TEE support**	**SWP support**
TEE_CipherInit	All Kinibi versions.	From TAP version 1.
TEE_CipherUpdate		
TEE_CipherDoFinal		
MAC Functions	**Kinibi TEE support**	**SWP support**
TEE_MACInit	All Kinibi versions.	From TAP version 1.
TEE_MACUpdate		
TEE_MACComputeFinal		
TEE_MACCompareFinal		
Authenticated Encryption Functions	**Kinibi TEE support**	**SWP support**
TEE_AEInit	When TA is built using TAP 1.4 GpTRIC: all Kinibi versions. Otherwise: Kinibi versions with API level 11, and later.	TAP 1.4 and later, when built with API Level 11, or later. Only the AES-GCM algorithm is supported. The only nonce length supported is 12 bytes.
TEE_AEUpdateAAD TEE_AEUpdate		
TEE_AEEncryptFinal		
TEE_AEDecryptFinal		
Asymmetric Functions	**Kinibi TEE support**	**SWP support**
TEE_AsymmetricEncrypt, TEE_AsymmetricDecrypt	All Kinibi versions.	From TAP version 1.
TEE_AsymmetricSignDigest		
TEE_AsymmetricVerifyDigest		
Key Derivation Function	**Kinibi TEE support**	**SWP support**

TEE_DeriveKey	Kinibi versions with API level API Level 7, and later.	From TAP version 1, when built with API Level 7, or later.
Random Data Generation Function	**Kinibi TEE support**	**SWP support**
TEE_GenerateRandom	All Kinibi versions.	From TAP version 1.

Note: For more information about the cryptographic algorithms, key types, and key parts supported in the Cryptographic Operations API, see the GlobalPlatform TEE Internal Core API: http://www.globalplatform.org/specificationsdevice.asp.

3.2.3.1 SWP extension for TEE_AllocateOperation

Software Protection extends Table 6-4: TEE_AllocateOperation Allowed Modes of the GlobalPlatform TEE Internal Core API specification:

Algorithm	**Possible Modes**
TEE_TT_ALG_UNWRAP_AES_ECB_NOPAD	TEE_TT_MODE_UNWRAP
TEE_TT_ALG_UNWRAP_AES_CBC_NOPAD	
TEE_TT_ALG_UNWRAP_AES_CTR	
TEE_TT_ALG_UNWRAP_AES_CBC_XMLENC	
TEE_TT_ALG_KDF_NIST_800_108_COUNTER_CMAC_AES128	TEE_TT_MODE_KDF
TEE_TT_ALG_KDF_NIST_800_108_COUNTER_CMAC_AES128_L16BIT	
TEE_TT_ALG_KDF_NO_CONVERSION	
TEE_TT_ALG_KDF_SHA_1	
TEE_TT_ALG_KDF_SHA_256	
TEE_TT_ALG_KDF_SHA_384	

3.3 SUPPORTED CRYPTOGRAPHIC ALGORITHMS

The following algorithms are supported in TEE mode or in Software Protection mode:

Algorithm	**Supported Key Size**
TEE_ALG_AES_ECB_NOPAD	128, 192 or 256 bits. (192 bit only when deployed on Kinibi with API Level 11, or with TA built using GpTRIC with APIO Level 11.)
TEE_ALG_AES_CBC_NOPAD	
TEE_ALG_AES_CTR	
TEE_ALG_AES_GCM	128, 192 or 256 bits.

TEE_ALG_DES_ECB_NOPAD	Always 64 bits including the parity bits. 2. See: ***Note.
TEE_ALG_DES_CBC_NOPAD	
TEE_ALG_DES3_ECB_NOPAD	128 or 192 bits including the parity bits. See: ***Note.
TEE_ALG_DES3_CBC_NOPAD	
TEE_ALG_RSASSA_PKCS1_V1_5_MD5	256, 512, 768, 1024, 1536, 2048. See: *Note.
TEE_ALG_RSASSA_PKCS1_V1_5_SHA1	
TEE_ALG_RSASSA_PKCS1_V1_5_SHA224	
TEE_ALG_RSASSA_PKCS1_V1_5_SHA256	
TEE_ALG_RSASSA_PKCS1_V1_5_SHA384	
TEE_ALG_RSASSA_PKCS1_V1_5_SHA512	
TEE_ALG_RSASSA_PKCS1_PSS_MGF1_SHA1	
TEE_ALG_RSASSA_PKCS1_PSS_MGF1_SHA224	
TEE_ALG_RSASSA_PKCS1_PSS_MGF1_SHA256	
TEE_ALG_RSASSA_PKCS1_PSS_MGF1_SHA384	
TEE_ALG_RSASSA_PKCS1_PSS_MGF1_SHA512	
TEE_ALG_RSAES_PKCS1_V1_5	
TEE_ALG_RSAES_PKCS1_OAEP_MGF1_SHA1	
TEE_ALG_RSAES_PKCS1_OAEP_MGF1_SHA224	
TEE_ALG_RSAES_PKCS1_OAEP_MGF1_SHA256	
TEE_ALG_RSAES_PKCS1_OAEP_MGF1_SHA384	
TEE_ALG_RSAES_PKCS1_OAEP_MGF1_SHA512	
TEE_ALG_RSA_NOPAD	
TEE_ALG_DSA_SHA1	From 1024 bits, multiple of 64 bits.
TEE_ALG_DSA_SHA224	2048 or 3072 bits.
TEE_ALG_DSA_SHA256	
TEE_ALG_MD5	NA

TEE_ALG_SHA1	
TEE_ALG_SHA224	
TEE_ALG_SHA256	
TEE_ALG_SHA384	
TEE_ALG_SHA512	
TEE_ALG_HMAC_MD5	Between 64 and 512 bits, multiple of 8 bits.
TEE_ALG_HMAC_SHA1	Between 80 and 512 bits, multiple of 8 bits.
TEE_ALG_HMAC_SHA224	Between 112 and 512 bits, multiple of 8 bits.
TEE_ALG_HMAC_SHA256	Between 192 and 1024 bits, multiple of 8 bits.
TEE_ALG_HMAC_SHA384	Between 256 and 1024 bits, multiple of 8 bits.
TEE_ALG_HMAC_SHA512	Between 256 and 1024 bits, multiple of 8 bits.
TEE_ALG_ECDSA_P192	192 bits. See: ****Note**.
TEE_ALG_ECDSA_P224	224 bits. See: ****Note**.
TEE_ALG_ECDSA_P256	256 bits. See: ****Note**.
TEE_ALG_ECDSA_P384	384 bits. See: ****Note**.
TEE_ALG_ECDSA_P521	521 bits. See: ****Note**.
TEE_ALG_ECDH_P192	192 bits. See: ****Note**.
TEE_ALG_ECDH_P224	224 bits. See: ****Note**.
TEE_ALG_ECDH_P256	256 bits. See: ****Note**.
TEE_ALG_ECDH_P384	384 bits. See: ****Note**.
TEE_ALG_ECDH_P521	521 bits. See: ****Note**.

***Note**: for the TEE_ALG_RSA… algorithms listed above, support is provided for 3K and 4K bit from TAP 1.4, when deployed on Kinibi with API Level 11 or when the TA is built using GPTRIC with API Level 11.

****Note**: for the TEE_ALG_EC… algorithms listed above, support is provided from Kinibi API Level 7 onwards. If you are using GPTRIC, set the API Level in the TA to 7, even if the targeted device is lower.

***Note**: See note about DES/DES3 key generation in release notes.

The following algorithms and key size are supported only in TEE mode:

Algorithm	**Supported Key Size**
TEE_ALG_AES_CCM	128, 192 or 256 bits.
TEE_ALG_AES_CTS	
TEE_ALG_AES_XTS	
TEE_ALG_DSA_SHA1	From 1024 bits, multiple of 64 bits.
TEE_ALG_DSA_SHA224	2048 or 3072 bits.
TEE_ALG_DSA_SHA256	
TEE_ALG_DH_DERIVE_SHARED_SECRET	From 256 to 2048 bits.
All RSA algorithms above 2048 bits	Greater than 2048 bits.

The following algorithms and key size are supported only in Software Protection mode:

Algorithm	**Supported Key Size**
TEE_TT_ALG_UNWRAP_AES_ECB_NOPAD	128, 192 or 256 bits.
TEE_TT_ALG_UNWRAP_AES_CBC_NOPAD	
TEE_TT_ALG_UNWRAP_AES_CTR	
TEE_TT_ALG_UNWRAP_AES_CBC_XMLENC	
TEE_TT_ALG_KDF_NIST_800_108_COUNTER_CMAC_AES128	NA
TEE_TT_ALG_KDF_NIST_800_108_COUNTER_CMAC_AES128_L16BIT	
TEE_TT_ALG_KDF_NO_CONVERSION	
TEE_TT_ALG_KDF_SHA_1	
TEE_TT_ALG_KDF_SHA_256	
TEE_TT_ALG_KDF_SHA_384	

3.4 EXTENDED PROPRIETARY API

The following functions provide extended features and can be called from Trusted Applications developed with the TEE Internal API. The header file for the Extended Proprietary API is tee_internal_api_ext.h.

Generic Operation Functions	**Kinibi TEE support**	**SWP support**
TEE_LogPrintf	All Kinibi versions.	From TAP version 1.
TEE_LogvPrintf		
TEE_DbgPrintf		
TEE_DbgPrintf		
TEE_TBase_UnwrapObject		Not supported.
TEE_TBase_DeriveKey		
TEE_TBase_AddEntropy		
SetDeviceId	Not supported.	From TAP version 1.
TEE_TT_Unwrap		
TEE_TT_KDF		
TEE_TBase_TUI_GetScreenInfo	All Kinibi versions.	Not supported.
TEE_TBase_TUI_OpenSession		
TEE_TBase_TUI_CloseSession		
TEE_TBase_TUI_SetImage		
TEE_TBase_TUI_GetTouchEvent		
TEE_TBase_DRM_OpenSession		
TEE_TBase_DRM_ProcessContent		
TEE_TBase_DRM_ProcessContentEx	Kinibi API Level 7, and later.	
TEE_TBase_DRM_CloseSession	All Kinibi versions.	
TEE_TBase_DRM_CheckLink		

3.4.1 Logging

3.4.1.1 TEE_LogPrintf

```
void TEE_LogPrintf (const char* fmt,...);
```

This proprietary function allows writing formatted traces for logging purposes.

3.4.1.2 TEE_LogvPrintf

```
void TEE_LogvPrintf (const char* fmt, va_list args);
```

This proprietary function allows writing formatted traces for logging purposes.

3.4.1.3 TEE_DbgPrintf

```
void TEE_DbgPrintf (const char* fmt,...);
```

This proprietary function allows writing formatted traces for debugging purposes. It is only compiled in on debug builds of the TA.

3.4.1.4 TEE_DbgvPrintf

```
void TEE_DbgvPrintf (const char* fmt, va_list args);
```

This proprietary function allows writing formatted traces for debugging purposes. It is only compiled in on debug builds of the TA.

3.4.2 TEE_TBase_UnwrapObject

```
TEE_Result TEE_TBase_UnwrapObject(
        void                *src,
        size_t              srcLen,
        void                *dest,
        size_t              destLen);
```

Description

Unwraps a secure object.

Note: this API is TEE only.

Decrypts and verifies the checksum of given object for the context indicated in the secure object's header.

Verifies and decrypts a secure object and stores the user data (plain data and the decrypted data) to a given location.

After this operation, the source address contains the decrypted secure object (whose user data starts immediately after the secure object header), or the attempt of the decryption, which might be garbage, in case the decryption failed (due to a wrong context, for instance).

If `dest` is not NULL, copies the decrypted user data part to the specified location, which may overlap with the memory occupied by the original secure object.

Parameters

- in, out `src`: [in] Encrypted secure object, [out] decrypted secure object i.e. the secure object header data the plain data and the decrypted data (which was earlier encrypted by the wrapper function).
- in `srcLen`: Length of source buffer i.e. the length of the secure object.
- in, out `dest`: Address of user data or NULL if no extraction of user data is desired. Note that this buffer has to be statically allocated (which is the reason it is also set as an input parameter). The tlApiWrapObjectExt does not allocate the buffer, it only writes to the buffer from the starting address and maximum of destLen (see parameter below).

- in, out `destLen`: [in] Length of destination buffer. [out] Length of user data. The length of the statically allocated buffer is sent as input for copying the userdata after the decryption of the secure object. The length of the userdata is returned.

Return Code

- `TLAPI_OK` if operation was successful.
- `E_TLAPI_INVALID_INPUT` if an input parameter is invalid.
- `E_TLAPI_CR_WRONG_OUPUT_SIZE` if the output buffer is too small.
- `E_TLAPI_SO_WRONG_VERSION` if the version of the secure object is not supported.
- `E_TLAPI_SO_WRONG_TYPE` if secure object type is not supported.
- `E_TLAPI_SO_WRONG_LIFETIME` if the kind of lifetime of the secure object is not supported.
- `E_TLAPI_SO_WRONG_CONTEXT` if the kind of context of the secure object is not supported.
- `E_TLAPI_SO_WRONG_CHECKSUM` if (after decryption) the checksum over the whole secure object (header and payload) is wrong. This is usually an indication that the secure object has been tampered with, or that the client calling unwrap is not allowed to unwrap the secure object.
- `E_TLAPI_UNMAPPED_BUFFER` if one buffer is not entirely mapped in Trusted Application address space.

Panic Reasons

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

3.4.3 TEE_TBase_DeriveKey

```
TEE_Result TEE_TBase_DeriveKey(
        const void          *salt,
        size_t              saltLen,
        void                *dest,
        size_t                destLen);
```

This function is equivalent to `tlApiDeriveKey()` with context set to `MC_SO_CONTEXT_TLT` and lifetime set to `MC_SO_LIFETIME_PERMANENT`.

Description

Derives a new key from the hardware master key. The key derivation function used by the implementation may vary across the implementations.

Different salt values provide different keys.

The resulting key is expanded to `destLen` bytes using RFC5869 expansion.

The derived key can be diversified between Trusted Applications or Service Providers depending on the context parameter.

The derived key can also be diversified between sessions or powercycles depending on the lifetime parameter.

Parameters

- `salt` [in] Salt value for key derivation.
- `saltLen` [in] Length of salt value.
- `dest` [out] Resulting key.
- `destLen` [in] Length of desired key.

Return Code

- `TEE_SUCCESS`: On success

- An error code

Panic Reasons

- If `dest` or salt buffers is not accessible
- If the Implementation detects any other error which is not explicitly associated with a defined return code for this function

3.4.4 TEE_TBase_AddEntropy

```
TEE_Result TEE_TBase_AddEntropy(
        uint8_t* buf,
        uint32_t buflen);
```

Description

Reseeds the Random Number Generator with the entropy supplied in parameter.

Parameters

- `buf` Buffer containing the additional entropy.
- `buflen` Length of buffer.

Return Code

- `TLAPI_OK` if operation was successful.

3.4.5 SetDeviceId

```
TEE_TT_SetDeviceId (void* buffer, size_t length);
```

Description

The function binds the Secure Storage to a specific device ID.

Note: this API is for SWP only.

Parameters

- `buffer`: Pointer to the byte array containing the device ID. This byte array has to be generated based on some hardware details or other environment-specific parameters.
- `length`: Number of bytes in the `buffer` parameter. The device ID can be of arbitrary length. If the size is 0, the previously set device ID will be removed.

Panic Reasons

- If the Implementation detects any error.

3.4.6 TEE_TT_Unwrap

```
TEE_Result TEE_TT_Unwrap(
        TEE_OperationHandle operation,
        void* wrappedKey, size_t wrappedKeyLen,
        TEE_Attribute* attrs, uint32_t attrCount,
        TEE_ObjectHandle unwrappedKey);
```

'UNWRAPPING' cryptographic algorithm must be enabled (CRYPTO_ALGORITHMS := UNWRAPPING).

Description

The function unwraps a buffer that represents a wrapped key and converts it to an object handle.

Note: this API is for SWP only.

The `TEE_TT_Unwrap` function can only be used with algorithms defined in the table below:

Algorithm	Operation Parameters
TEE_TT_ALG_UNWRAP_AES_ECB_NOPAD	Unwrap algorithm based on AES cipher in ECB mode without padding
TEE_TT_ALG_UNWRAP_AES_CBC_NOPAD	Unwrap algorithm based on AES cipher in CBC mode without padding
TEE_TT_ALG_UNWRAP_AES_CTR	Unwrap algorithm based on AES cipher in CTR mode
TEE_TT_ALG_UNWRAP_AES_CBC_XMLENC	Unwrapping algorithm based on AES cipher using XML Encryption Padding (http://www.w3.org/TR/xmlenc-core)

TEE_TTK_Unwrap currently supports only the following object types (wrapped keys):

- TEE_TYPE_RSA_KEYPAIR (from 1024 to 2048 bits including)
- TEE_TYPE_ECDSA_KEYPAIR
- All symmetric key types

Parameters

- operation: A handle on an opened cipher operation setup with a key
- wrappedKey, wrappedKeyLen: Input buffer that contains the wrapped key
- attrs, attrCount: Array of object public attributes.
- unwrappedKey: Handle on an uninitialized transient object to be filled with the unwrapped key

Return Code

- TEE_SUCCESS: On success
- TEE_ERROR_BAD_FORMAT: If an incorrect or inconsistent input wrapped key is detected.

Panic Reasons

- operation is not a valid operation handle of class TEE_TT_OPERATION_UNWRAP
- No key is programmed in the operation
- The operation mode is not TEE_TT_MODE_UNWRAP
- wrappedKey is equal to NULL
- derivedKeySize is not supported or is too large
- unwrappedKey is not a valid opened object handle that is transient and uninitialized
- A mandatory parameter is missing
- Hardware or cryptographic algorithm failure
- If the Implementation detects any other error which is not explicitly associated with a defined return code for this function.

3.4.7 TEE_TT_KDF

Note: this API is available to SWP mode TAs only; for TEE mode TAs, employ #ifdefs to code around it.

According to the GP specification, the output of the TEE_DeriveKey function, derivedKey, MUST refer to an object with type TEE_TYPE_GENERIC_SECRET. It cannot be used as the transient object (the key). Use this key by retrieving its TEE_ATTR_SECRET_VALUE attribute using TEE_GetObjectBufferAttribute and then populate to a new object with TEE_PopulateTransientObject. Since the key retrieval operation is not secure for SWP, a new API has been introduced that:

- takes the TEE_TYPE_GENERIC_SECRET (or any object type) and transforms it to another symmetric type

- performs a key derivation using a particular derivation algorithm. Since the raw output of Diffie–Hellman key exchange algorithm is considered as a weak secret, we recommend you perform an additional conversion afterwards

```
TEE_Result TEE_TT_KDF(
        TEE_OperationHandle operation,
        TEE_ObjectHandle    derivedKey,
        uint32_t            derivedKeySize,
        TEE_Attribute*      params,
        uint32_t            paramCount);
```

'KDF' cryptographic algorithm must be enabled (CRYPTO_ALGORITHMS := KDF).

Description

The key derivation function derives a key object with specified key size from the key object set for the operation and values passed in parameters.

The TEE_TT_KDF function can only be used with algorithms defined in the table below:

Algorithm	Operation Parameters
TEE_TT_ALG_KDF_NIST_800_108_COUNTER_CMAC_AES128	TEE_TT_ATTR_KDF_NIST_800_LABEL: label, a binary buffer that identifies the purpose for the derived key, as defined by the NIST Special Publication 800-108 TEE_TT_ATTR_KDF_NIST_800_CONTEXT: the context, a binary buffer containing the information related to the derived key, as defined by the NIST Special Publication 800-108
TEE_TT_ALG_KDF_NIST_800_108_COUNTER_CMAC_AES128_L16BIT	
TEE_TT_ALG_KDF_SHA_1	
TEE_TT_ALG_KDF_SHA_256	TEE_TT_ATTR_KDF_SHA256_PLAIN1, TEE_TT_ATTR_KDF_SHA256_PLAIN2: buffers of bytes that should be prepended or appended to the key value before calculating the SHA-256 hash value
TEE_TT_ALG_KDF_SHA_384	

Parameters

- operation: A handle on an opened cipher operation setup with a key
- derivedKey: Handle on an uninitialized transient object to be filled with the derived key
- derivedKeySize: Requested key size. MUST be less than or equal to the maximum key size specified when the object container was created. MUST be a valid value as defined in Table 5-9: TEE_AllocateTransientObject Object Types and Key Sizes of GP specification
- params, paramCount: Parameters for the key derivation as in the table above. The values of all parameters are copied into the object so that the params array and all the memory buffers it points to may be freed after this routine returns without affecting the object.

Return Code

- TEE_SUCCESS: On success
- TEE_ERROR_BAD_PARAMETERS: If an incorrect or inconsistent attribute is detected.

Panic Reasons

- `operation` is not a valid operation handle of class `TEE_TT_OPERATION_KDF`
- No key is programmed in the operation
- The operation mode is not `TEE_TT_MODE_KDF`
- `derivedKey` is not a valid opened object handle that is transient and uninitialized
- `derivedKeySize` is not supported or is too large
- A mandatory parameter is missing
- Hardware or cryptographic algorithm failure
- If the Implementation detects any other error which is not explicitly associated with a defined return code for this function

3.4.8 Key sharing API

Use the following proprietary APIs to exchange keys between two different TAs that exist in different processes. The TAs do not need to have the same UUID. However, to ensure compatibility with the exported format, the TAs must be built using the same SDK.

Do not use this API to exchange keys between a TA and another entity, such as a server.

3.4.8.1 TEE_TT_Export

```
TEE_Result TEE_TT_Export([in] TEE_ObjectHandle object,
[in(objectIdLen)] void  *objectID, size_t objectIDLen,
[outbuf] void *buffer, size_t *size);
```

'KDF' cryptographic algorithm must be enabled (CRYPTO_ALGORITHMS := KDF).

Description

The `TEE_TT_Export` API allows a TA to export the key materials of a persistent or transient object passed in a parameter as an object into the output buffer pointed to by a buffer. This buffer can then be passed in input to `TEE_TT_Import()` to recreate the object.

Note: this API is for SWP only.

The output buffer does not reveal any attributes of the object; private attributes are exported into a protected form that are bound to the SKB export_key.

When a persistent object is passed, it must contain crypto attributes, it cannot be a pure data object.

Parameters

- `object`: Handle of the object to export.
- `objectID`, `objectIDLen`: A buffer containing the object identifier. The identifier contains arbitrary bytes, including the zero byte. The identifier length MUST be less than or equal to `TEE_OBJECT_ID_MAX_LEN` and can be zero. The buffer containing the object identifier cannot reside in shared memory.
- `buffer`, `size`: Output buffer to get the exported object blob.

Return Code

- `TEE_SUCCESS` : On success.
- `TEE_ERROR_BAD_PARAMETERS` : If a pure data persistent object is passed.
- `TEE_ERROR_SHORT_BUFFER` : If size value is equal to 0, or if the buffer is not large enough to contain the exported object.
- `TEE_ERROR_OUT_OF_MEMORY` : If there is not enough memory to complete the operation.
- `TEE_ERROR_GENERIC`: if any of the internal crypto operations fail unexpectedly.

Panic Reasons

- `object` is not a valid handle on an initialized object that contains crypto attributes.
- `objectIDLen` is greater than `TEE_OBJECT_ID_MAX_LEN`.
- If the implementation detects any other error which is not explicitly associated with a defined return code for this function.

3.4.8.2 TEE_TT_ Import

```
TEE_Result TEE_TT_Import([out] TEE_ObjectHandle *object,
[in(objectIdLen)] void *objectID, size_t objectIDLen,
[inbuf] void *buffer, size_t size)
```

'KDF' cryptographic algorithm must be enabled (CRYPTO_ALGORITHMS := KDF).

Description

The `TEE_TT_Import` API creates a transient object from a data blob passed as buffer previously created with `TEE_TT_Export`. It returns a handle object that can be used to access the object's attributes.

Note: this API is for SWP only.

Parameters

- `object`: A pointer to the handle, which contains the opened handle upon successful completion. If this function fails for any reason, the value pointed to by the object is set to `TEE_HANDLE_NULL`. When the object handle is no longer required, it MUST be closed using a call to the `TEE_CloseObject` function.
- `objectID`, `objectIDLen`: The object identifier. Note that this cannot reside in shared memory.
- `buffer`, `size`: Input buffer which contains the exported object obtained with `TEE_TT_Export`.

Return Code

- `TEE_SUCCESS` : On success
- `TEE_ERROR_OUT_OF_MEMORY` : If there is not enough memory to complete the operation.
- `TEE_ERROR_CORRUPT_OBJECT`: If the exported object structure is incorrect or of an incorrect size, if the exported object's integrity is found invalid, or if the exported key cannot be imported.
- `TEE_ERROR_GENERIC`: if any of the internal crypto operations fail unexpectedly.
- `TEE_ERROR_ITEM_NOT_FOUND`: If object with corresponding `objectID` cannot be imported from the input buffer

Panic Reasons

- `objectIDLen` is greater than `TEE_OBJECT_ID_MAX_LEN`.
- If the implementation detects any other error which is not explicitly associated with a defined return code for this function.

3.4.9 Trusted User Interface

Note: this API is TEE only.

3.4.9.1 Error Codes

Constant Name	Value	API Level	Definition
`E_TLAPI_TUI_NO_SESSION`	`0x00000501`	3 and higher	The session to TUI driver cannot be found. It was not opened or has been closed.

E_TLAPI_TUI_BUSY	0x00000502	3 and higher	TUI driver is busy. Another session may be open.
E_TLAPI_TUI_NO_EVENT	0x00000503	3 and higher	No TUI event has occurred since the session started or the last call of get event.
E_TLAPI_TUI_OUT_OF_DISPLAY	0x00000504	3 and higher	The coordinates/size of a displayable object are at least partially out of the of display area.
E_TLAPI_TUI_IMG_BAD_FORMAT	0x00000505	3 and higher	Some data found when parsing are related to a feature that is not supported.
E_TLAPI_TUI_MISSED_EVENTS	0x00000506	6 and higher	TUI event queue is overflowed.
E_TLAPI_TUI_INVALID_CONTEXT	0x00000507	6 and higher	The graphic context is invalid.

3.4.9.2 Types

3.4.9.2.1 tlApiTuiScreenInfo_t

```
typedef struct {
      uint32_t    grayscaleBitDepth;
      uint32_t    redBitDepth;
      uint32_t    greenBitDepth;
      uint32_t    blueBitDepth;
      uint32_t    width;
      uint32_t    height;
      uint32_t    wDensity;
      uint32_t    hDensity;
} tlApiTuiScreenInfo_t, *tlApiTuiScreenInfo_ptr;
```

General information about the screen.

The fields of the structure are:

- `grayscaleBitDepth`: Available grayscale depth.
- `redBitDepth`: Available red bit depth.
- `greenBitDepth`: Available green bit depth.
- `blueBitDepth`: Available blue bit depth.
- `width`: Width of the screen in pixel.
- `height`: Height of the screen in pixel.
- `wDensity`: Density of the screen in pixel-per-inch.
- `hDensity`: Density of the screen in pixel-per-inch.

3.4.9.2.2 tlApiTuiTouchEventType_t

Type of touch event.

```
typedef enum {
    TUI_TOUCH_EVENT_RELEASED = 0,
```

```
    TUI_TOUCH_EVENT_PRESSED  = 1,
} tlApiTuiTouchEventType_t;
```

Enumerator

TUI_TOUCH_EVENT_RELEASED: A pressed gesture has finished.

TUI_TOUCH_EVENT_PRESSED: A pressed gesture has occurred.

3.4.9.2.3 tlApiTuiImage_t

```
typedef struct {
      void*       imageFile;
      uint32_t    imageFileLength;
} tlApiTuiImage_t, *tlApiTuiImage_ptr;
```

Image file.

The fields of the structure are:

- `imageFile`: a buffer containing the image file.
- `imageFileLength`: size of the buffer.

3.4.9.2.4 tlApiTuiCoordinates_t

```
typedef struct {
    uint32_t    xOffset;
    uint32_t    yOffset;
} tlApiTuiCoordinates_t, *tlApiTuiCoordinates_ptr;
```

Coordinates.

These are related to the top-left corner of the screen.

The fields of the structure are:

- `xOffset`: x coordinate.
- `xOffset`: y coordinate.

3.4.9.2.5 tlApiTuiTouchEvent_t

```
typedef struct {
    tlApiTuiTouchEventType_t   type;
    tlApiTuiCoordinates_t      coordinates;
} tlApiTuiTouchEvent_t, *tlApiTuiTouchEvent_ptr;
```

Touch event data.

The fields of the structure are:

- `type`: type of touch event.
- `coordinates`: coordinates of the touch event in the screen.

3.4.9.2.6 tlApiTuiImageInfo_t

```
typedef enum {
    TUI_IMAGE_TYPE_INVALID = 0,
    TUI_IMAGE_TYPE_PNG
} tlApiTuiImageType_t;

typedef enum {
```

```
    TUI_IMAGE_PIXEL_FORMAT_GREYSCALE = 0,
    TUI_IMAGE_PIXEL_FORMAT_TRUECOLOR,
    TUI_IMAGE_PIXEL_FORMAT_GREYSCALE_ALPHA,
    TUI_IMAGE_PIXEL_FORMAT_TRUECOLOR_ALPHA,
    TUI_IMAGE_PIXEL_FORMAT_INDEXED
} tlApiTuiImagePixelFormat_t;

typedef struct {
    uint32_t type;
    uint32_t width;
    uint32_t height;
    uint32_t pixelFormat;
    uint32_t colorDepth;
    uint32_t decodingSize;
} tlApiTuiImageInfo_t, *tlApiTuiImageInfo_ptr;
```

Since API level 6.

The structure `tlApiTuiImageInfo_t` is filled by the `tlApiTuiGetImageInfo()` function.

The fields of the structure are:

- `type`: Image format. `TUI_IMAGE_TYPE_XXX`
- `width`: Image width in pixels
- `height`: Image height in pixels
- `pixelFormat`: Pixel format: `TUI_IMAGE_PIXEL_FORMAT_XXX`
- `colorDepth`: Color depth in bits
- `decodingSize`: Size of the buffer to be allocated by the TA to decode the image using `tlApiTuiDecodeImage()`

3.4.9.2.7 tlApiTuiRectangle_t

```
typedef struct {
    int32_t x;
    int32_t y;
    uint32_t width;
    uint32_t height;
} tlApiTuiRectangle_t, *tlApiTuiRectangle_ptr;
```

Since API level 6.

The fields of the structure are:

- `x`: X coordinate of the top-left corner of the rectangle.
- `y`: Y coordinate of the top-left corner of the rectangle.
- `width`: Width of the rectangle in pixels
- `height`: Height of the rectangle in pixels.

3.4.9.2.8 tlApiTuiRawImage_t

```
typedef struct {
    void        *data;
    uint32_t    size;
    uint32_t    pixelFormat;
    uint32_t    stride;
    uint32_t    width;
    uint32_t    height;
```

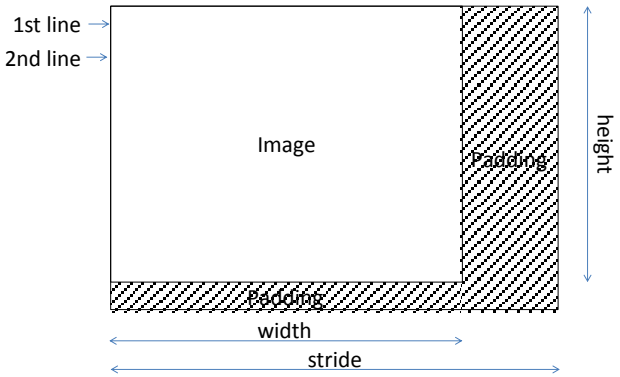
```
} tlApiTuiRawImage_t, *tlApiTuiRawImage_ptr;

typedef enum {
    TUI_RAW_PIXEL_FORMAT_RGBA_8888 = 0,
} tlApiTuiRawPixelFormat_t;
```

Since API level 6.

The fields of the structure are:

- `data`: A buffer containing the pixels.
- `size`: Size of the buffer.
- `pixelFormat`: Pixel format, only TUI_RAW_PIXEL_FORMAT_RGBA_8888 is supported.
- `stride`: Image stride (number of bytes in the buffer between the beginning of a line and the beginning of the next line). This value must be multiple of 4 (the pixel size is 4 bytes) and greater than 4 times the image width. It is usually equal to the width multiplied by the number of bytes per pixel.
- `width`: Image width in pixels.
- `height`: Image height in pixels.



Memory representation of a raw image

3.4.9.2.9 tlApiTuiGraphicContext_t

```
typedef struct __tlApiTuiGraphicContext_t *tlApiTuiGraphicContext_t;
```

Since API level 6.

Opaque data type representing a graphic context.

A graphic context holds some parameters related to a drawing area. The parameters include a drawing buffer and clipping parameters. The parameters apply to any operation made using the graphic context.

3.4.9.3 Functions

3.4.9.3.1 TEE_TBase_TUI_GetScreenInfo

```
TEE_Result TEE_TBase_TUI_GetScreenInfo(
      tlApiTuiScreenInfo_ptr screenInfo);
```

Description

Get screen information.

Parameters

- `screenInfo`: screen information.

Return Code

- `TLAPI_OK` if operation was successful.
- `E_TLAPI_DRV_NO_SUCH_DRIVER` if the TUI driver cannot be found.
  `E_TLAPI_NULL_POINTER` if one parameter is a null pointer.

3.4.9.3.2 TEE_TBase_TUI_OpenSession

```
TEE_Result TEE_TBase_TUI_OpenSession(void);
```

Description

Open a session to the TUI driver.

Return Code

- `TLAPI_OK` if operation was successful.
- `E_TLAPI_DRV_NO_SUCH_DRIVER` if the TUI driver cannot be found.
- `E_TLAPI_TUI_BUSY` if the TUI driver cannot be opened.

3.4.9.3.3 TEE_TBase_TUI_CloseSession

```
TEE_Result TEE_TBase_TUI_CloseSession(void);
```

Description

Close the session to the TUI driver.

Return Code

- `TLAPI_OK` if operation was successful.
- `E_TLAPI_DRV_NO_SUCH_DRIVER` if the TUI driver cannot be found.
- `E_TLAPI_TUI_NO_SESSION` if the TUI driver session cannot be found. It was not opened or has been closed.

3.4.9.3.4 TEE_TBase_TUI_SetImage

```
TEE_Result TEE_TBase_TUI_SetImage (
      tlApiTuiImage_ptr image,
      tlApiTuiCoordinates_t coordinates);
```

Description

Draw an image in secure display.

Unlike other drawing functions, this function draws directly to the front framebuffer of the screen and is subject to tearing.

Only non-interlaced PNG images can be displayed. The **Error! Reference source not found.** gives the c apabilities of the PNG decoder.

PNG Image Type	Allowed bit depth	t-base decoder	API level
Greyscale	1, 2, 4, 8	Supported	3 and higher
Greyscale	16	Not supported	3

Greyscale	16	Supported	4
Truecolor	8	Supported	3 and higher
Truecolor	16	Not supported	3 and higher
Indexed-color	1, 2, 4, 8	Not supported	3 and higher
Greyscale with alpha	8, 16	Not supported	3
Greyscale with alpha	8, 16	Supported	4
Truecolor with alpha	8	Not supported	3
Truecolor with alpha	8	Supported	4
Truecolor with alpha	16	Not supported	3 and higher

Parameters

- `image`: image to be displayed.
- `coordinates`: coordinates where to display the image in the screen, related to the top left corner defined by `tlApiTuiGetSreenInfo`.

Return Code

- `TLAPI_OK` if operation was successful.
- `E_TLAPI_DRV_NO_SUCH_DRIVER` if the TUI driver cannot be found.
- `E_TLAPI_TUI_NO_SESSION` if the TUI driver session cannot be found. It was not opened or has been closed.
- `E_TLAPI_NULL_POINTER` if one parameter is a null pointer.
- `E_TLAPI_ INVALID_INPUT` if one parameter is not valid.
- `E_TLAPI_TUI_IMG_BAD_FORMAT` if the image file cannot be recognized as a valid file.
- `E_TLAPI_NOT_IMPLEMENTED` if some data found when parsing the image file are related to a feature that is not supported.
- `E_TLAPI_TUI_OUT_OF_DISPLAY` if the image or a part of the image is out of the display area.

3.4.9.3.5 TEE_TBase_TUI_GetTouchEvent

```
TEE_Result TEE_TBase_TUI_GetTouchEvent(
      tlApiTuiTouchEvent_ptr touchEvent);
```

Description

Get a touch event from TUI driver.

This is non-blocking call. It shall be called when the TL is notified.

The touch events are actually queued in the TUI driver. In case the queue overflowed, the application has lost events and should call `tlApiTuiGetTouchEvent` and process events ASAP to avoid losing more.

Parameters

- `touchEvent`: the touch event that occurred.

Return Code

- `TLAPI_OK` if operation was successful.

- E_TLAPI_DRV_NO_SUCH_DRIVER if the TUI driver cannot be found.
- E_TLAPI_TUI_NO_SESSION if the TUI driver session cannot be found. It was not opened or has been closed.
- E_TLAPI_TUI_NO_EVENT if no event has occurred since the session started or the last call of this function.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer.
- E_TLAPI_INVALID_RANGE if the pointed touch event structure is not within the secure memory range.
- E_TLAPI_TUI_MISSED_EVENTS if the TUI event queue is overflowed. In that case the value of touchEvent is not accurate.

3.4.10DRM API

Note: this API is TEE only.

3.4.10.1Structures

3.4.10.1.1 tlApiDrmOffsetSizePair

```
typedef struct
{
    uint32_t nSize;                 /* Size of encrypted region */
    uint32_t nOffset;               /* offset to encrypted region */
} tlApiDrmOffsetSizePair_t;
```

Structure containing the offset and size of an encrypted data section within a buffer, potentially one of many sections within the buffer.

3.4.10.1.2 tlApiDrmAlg

```
typedef enum {
    TLAPI_DRM_ALG_NONE,
    TLAPI_DRM_ALG_AES_ECB,
    TLAPI_DRM_ALG_AES_CBC,
    TLAPI_DRM_ALG_AES_CTR32,
    TLAPI_DRM_ALG_AES_CTR64,
    TLAPI_DRM_ALG_AES_CTR96,
    TLAPI_DRM_ALG_AES_CTR128,
    TLAPI_DRM_ALG_AES_XTS,
    TLAPI_DRM_ALG_AES_CBCCTS
} tlApiDrmAlg_t;
```

Enum containing list of cryptographic algorithms available.

3.4.10.1.3 tlApiDrmLink

```
typedef enum {
    TLAPI_DRM_LINK_HDCP_1,
    TLAPI_DRM_LINK_HDCP_2,
    TLAPI_DRM_LINK_AIRPLAY,
    TLAPI_DRM_LINK_DTCP,
#if TBASE_API_LEVEL >= 7
    TLAPI_DRM_LINK_HDCP_2_1,
    TLAPI_DRM_LINK_HDCP_2_2,
    TLAPI_DRM_LINK_HDCP_1_0 = TLAPI_DRM_LINK_HDCP_1,
    TLAPI_DRM_LINK_HDCP_2_0 = TLAPI_DRM_LINK_HDCP_2,
```

```
#endif /* TBASE_API_LEVEL >= 7 */
} tlApiDrmLink_t;
```

Structure containing the type of output link that needs to be protected and checked according to license.

3.4.10.1.4 tlApiDrmInputSegmentDescriptor

```
typedef struct
{
 uint32_t nTotalSize;        /** size of buffer (plain + encrypted) */
 uint32_t nNumBlocks;                    /* No. of encrypted regions */
 tlApiDrmOffsetSizePair_t  aPairs[TLAPI_DRM_INPUT_PAIR_NUMBER];  /*  Array  of
offset/size pairs */
}tlApiDrmInputSegmentDescriptor;
```

Structure containing the number of encrypted regions in the buffer and their offset/size information.

3.4.10.1.5 tlApiDrmDecryptContext

The crypto context to contain all IV, key and algorithm information required to decrypt the content.

```
/**
 * For DRM cipher/copy operations
 *
 * Parameters
 * @param  key          [in]  content key
 * @param  key_len      [in]  key length in bytes (16,24,32)
 * @param  iv           [in]  initialization vector. Always 16 bytes.
 * @param  ivlen        [in]  length initialization vector.
 * @param  alg          [in]  algorithm
 * @param  outputoffset [in]  output data offset
 *
 */
typedef struct tlApiDrmDecryptContext
{
    uint8_t               *key;
    int32_t               keylen;
    uint8_t               *iv;
    uint32_t            ivlen;
    tlApiDrmAlg_t         alg;
    uint32_t              outputoffet;
}tlApiDrmDecryptContext;
```

3.4.10.1.6 tlApiDRM_headerV1

Since API level 7

This structure is used for the tlApiDrmProcessContentEx().

```
/**
 * Extended function for DRM cipher/copy operations
 *
 * Parameters
 * @param  size        [in]  Declared size of the structure to pass to the driver
 * @param  decryptCtx   [in]  DRM Cipher data
 * @param  input        [in]  virtual address of contiguous memory
```

```
 * @param  inputDesc    [in]  number of blocks and offset data in the input array
to be decrypted.
 * @param  processmode  [in]  content. E.g., encrypted/plain text
 * @param  output       [in/out]  Reference to the address of output decrypted
content. Also use for driver to return an address
 * @param  rfu          [in]  Used to pass additional parameters to driver
 * @param  rfuLen       [in]  Size of the variable pointed by rfu
 *
 */
    uint32_t                              size;
    tlApiDrmDecryptContext_t              decryptCtx;
    void                                  *input;
    tlApiDrmInputSegmentDescriptor_t      inputDesc;
    uint32_t                              processMode;
    uint64_t                              output;
    void                                  *rfu;
    uint32_t                              rfuLen;
} tlApiDRM_headerV1;
```

3.4.10.2Constants

Constant Name	Value	Definition
TLAPI_DRM_KEY_SIZE_128	16	Key size supported for AES Cipher.
TLAPI_DRM_KEY_SIZE_192	24	Key size supported for AES Cipher.
TLAPI_DRM_KEY_SIZE_256	32	Key size supported for AES Cipher.
TLAPI_DRM_PROCESS_ENCRYPTED_DATA	1	Indicates encrypted data is being passed to the driver.
TLAPI_DRM_PROCESS_DECRYPTED_DATA	2	Indicates decrypted data is being passed to the driver.
TLAPI_DRM_INPUT_PAIR_NUMBER	10	Number of offset/size pair in input descriptor

3.4.10.3Errors

Constant Name	Value	Definition
E_TLAPI_DRM_OK	0	No Error.
E_TLAPI_DRM_INVALID_PARAMS	0x601	Invalid parameter for Cipher
E_TLAPI_DRM_INTERNAL	0x602	Internal error in AES
E_TLAPI_DRM_MAP	0x603	Driver mapping error
E_TLAPI_DRM_PERMISSION_DENIED	0x604	Permission Denied
E_TLAPI_DRM_REGION_NOT_SECURE	0x605	If the output address is not protected.
E_TLAPI_DRM_SESSION_NOT_AVAILABLE	0x606	If a single session implementation is already active, or a multi session implementation has no free sessions.

E_TLAPI_DRM_INVALID_COMMAND	0x607	If the command ID received is unrecognized.
E_TLAPI_DRM_ALGORITHM_NOT_SUPPORTED	0x608	If the requested algorithm is not supported by the driver. If this error is thrown the trusted application must decipher the content itself.
E_TLAPI_DRM_DRIVER_NOT_IMPLEMENTED	0x609	If the functions have not been implemented.

3.4.10.4 Functions

3.4.10.4.1 TEE_TBase_DRM_OpenSession

```
TEE_Result TEE_TBase_DRM_OpenSession (
uint8_t *sHandle);
```

Description

If multiple session support is required it must be managed first here, this function is also required to set up any initial requirements in the hardware prior to decryption for example (if required according to platform and chose framework architecture):

- Initialize the Crypto Hardware
- Initialize the Media Framework
- Authenticate the decoder firmware.
- Enable Firewalls.

Parameters

`sHandle`: [out] Session Handle of the new TA session.

Return Code

- `E_TLAPI_DRM_OK` if operation was successful.
- `E_TLAPI_DRM_INTERNAL` general error in case of crypto problem
- `E_TLAPI_DRM_MAP` in case of error mapping memory to driver.
- `E_TLAPI_DRM_PERMISSION_DENIED` in case of rights access related issue
- `E_TLAPI_DRM_SESSION_NOT_AVAILABLE` in case the driver is busy and cannot open a session.
- `E_TLAPI_DRM_DRIVER_NOT_IMPLEMENTED` in case the function is not implemented.

3.4.10.4.2 TEE_TBase_DRM_ProcessContent

```
TEE_Result TEE_TBase_DRM_ProcessContent (
    uint8_t                         sHandle,
    tlApiDrmDecryptContext          decryptCtx,
    uint8_t                         *input,
    tlApiDrmInputSegmentDescriptor   inputDesc,
    uint16_t                        processMode,
    uint8_t                         *output);
```

Description

Processes the specified content.

If the algorithm is supported by the driver this function is used to decrypt the encrypted data into a protected buffer. If the algorithm is not supported it will respond in an error. In this case the decryption should be done by the Trusted Application and the decrypted data shall be copied to the protected buffer using this function with `processMode` set to `TL_DRM_PROCESS_DECRYPTED_DATA`.

The parameter processMode is a constant value indicating whether encrypted or decrypted data is being treated from the TA.

If multiple sessions are supported, the sHandle parameter is used to identify the session requested for decryption.

Input, key and iv data provided within the tlApiDrmDecryptContext structure indicates the context of the cryptographic operation.

If a frame consists of multiple encrypted areas, the tlApiDrmInputSegmentDescriptor structure must hold the offsets and lengths of the encrypted regions, the offsets will also correspond to the offsets in the output buffer. If the input is merely one encrypted buffer, this will be indicated by the structure. If the input buffer to the TA contains both clear and encrypted data then both clear and encrypted data must be passed to the driver using this function.

The output parameter holds a reference to a protected output location for the decrypted data. The actual type of reference may vary between platforms, it may be an identifier, address, ION handle, but needs to be consistent between NW media framework component that retrieves the reference and the DRM driver that handles it. It will be passed through from media framework to driver without modification by the NW and SW intermediate components.

If processmode is TL_DRM_PROCESS_DECRYPTED_DATA the structure elements: key, keylen and iv are ignored as they are irrelevant for the copy.

Parameters

- sHandle: [in] Session Handle of the current TA session.
- decryptCtx: [in] Contains the IV, Key, Key length and all necessary crypto information for the requested algorithm.
- input: [in] Address to the start of a block of encrypted data, or if required multiple sections of encrypted and decrypted segments the offsets and lengths of which are described in the next parameter.
- inputDesc: [in] Structure containing offsets and lengths of data to be decrypted if multiple segments are present within the same buffer, if not it will contain the offset and length of the only encrypted segment.
- processMode: [in] states whether the incoming data is decrypted or encrypted, which infers a secure copy, or a decrypt operation is required.
- output: [in] holds a reference to an output address, can be a handle, identifier or address.

Return Code

- E_TLAPI_DRM_OK if operation was successful.
- E_TLAPI_DRM_INVALID_PARAMS incorrect parameters in input.
- E_TLAPI_DRM_INTERNAL general Error in case of crypto problem
- E_TLAPI_DRM_MAP in case of error mapping memory to driver.
- E_TLAPI_DRM_PERMISSION_DENIED in case of rights access related issue
- E_TLAPI_DRM_REGION_NOT_SECURE if the memory for output is not protected
- E_TLAPI_DRM_ALGORITHM_NOT_SUPPORTED in case the algorithm is not supported.
- E_TLAPI_DRM_DRIVER_NOT_IMPLEMENTED in case the function is not implemented.

3.4.10.4.3 TEE_TBase_DRM_ProcessContentEx

```
TEE_Result TEE_TBase_DRM_ProcessContentEx(
        uint8_t                         sHandle,
        tlApiDRM_headerV1               *DRM_header);
```

Description

Since API level 7
Processes the specified content.
Extended function for tlApiDrmProcessContent().
More parameters can be transmitted to the DRM driver with the RFU fields of the tlApiDRM_headerV1 structure.

Parameters

- sHandle: [in] Session Handle of the current TA session.
- DRM_header: [in] Contains parameters to be sent to the driver.

Return Code

- E_TLAPI_DRM_OK if operation was successful.
- E_TLAPI_DRM_INVALID_PARAMS incorrect parameters in input.
- E_TLAPI_DRM_INTERNAL general Error in case of crypto problem
- E_TLAPI_DRM_MAP in case of error mapping memory to driver.
- E_TLAPI_DRM_PERMISSION_DENIED in case of rights access related issue
- E_TLAPI_DRM_REGION_NOT_SECURE if the memory for output is not protected
- E_TLAPI_DRM_ALGORITHM_NOT_SUPPORTED in case the algorithm is not supported.
- E_TLAPI_DRM_DRIVER_NOT_IMPLEMENTED in case the function is not implemented.

3.4.10.4.4 TEE_TBase_DRM_CloseSession

```
TEE_Result TEE_TBase_DRM_CloseSession (uint8_t sHandle);
```

Description

Closes the DRM session.

It operates on the session indicated by the session handle passed in the function. If multi session is not supported, this value is ignored.

Parameters

- sHandle: [in] Session Handle of the current TA session.

Return Code

- E_TLAPI_DRM_OK if operation was successful.
- E_TLAPI_DRM_INTERNAL in case of failure.
- E_TLAPI_DRM_DRIVER_NOT_IMPLEMENTED in case the function is not implemented.

3.4.10.4.5 TEE_TBase_DRM_CheckLink

```
TEE_Result TEE_TBase_DRM_CheckLink (
      uint8_t sHandle,
      tlApiDrmLink_t link);
```

Description

This function is used to check the protected external link information like HDCPv1, HDCPv2, AirPlay and DTCP.

It operates on the session indicated by the session handle passed in the function. If multi-session is not supported, this value is ignored.

Parameters

- sHandle: [in] Session Handle of the current TA session.

- `link`: [in] external link information like HDCPv1, HDCPv2, AirPlay, and DTCP.

Return Code

- `E_TLAPI_DRM_OK` if operation was successful.
- `E_TLAPI_DRM_INTERNAL` in case of failure.
- `E_TLAPI_DRM_DRIVER_NOT_IMPLEMENTED` in case the function is not implemented.

4 USING THP AGENT

This chapter describes how to use the THP Agent to install or upgrade a TA.

The THP Agent is responsible for communication between Android and Trustonic. You use the THP Agent to interface with TAs, and to perform commands such as installing, updating, and uninstalling TAs.

Tip! For information about using THP Agent to personalize TAs, see *Developing a TAP Application for Android*.

4.1 INSTALLING OR UPGRADING A TA

Use the THP Agent command `installOrUpdateTA()` to install a TA or upgrade an already-installed TA.

TAs are installed at application runtime. To load a TA, the THP Agent method `installOrUpdateTA()` is called when the application starts. When this method is called for the very first time in the lifetime of the application, it triggers a network connection to the TAM server. For subsequent calls, no network connection is required.

The android sample provided with the TAP SDK demonstrates how to use the THP Agent interface to install TAs. For more information about the android sample, see *Developing a TAP Application for Android.*

Example

The following example shows the code implemented on the Android side to send commands to the THP Agent.

First, we create a new THPAgent object to handle installing a TA.

```
THPAgent agent = new THPAgent(this);
```

Next, we set up the server. If using a secure connection, we also set up the public key certificate within the client that corresponds to the one set up on the server. For use with the Trustonic-hosted TAMv2 development server, this is included in the sample application.

```
   try {
// Require a certificate for https
agent = agent.setServerCA("-----BEGIN CERTIFICATE-----\n" +
activity.getString(R.string.ca_cert_raw) + "\n-----END CERTIFICATE-----
");
agent = agent. setServerBaseUrl
(activity.getString(R.string.tam_server_url));
}
catch(CertificateException e){
System.out.println("Certificate Exception" + e.getMessage());
}
```

The default values used above for the Trustonic-hosted development server and its public key certificate are defined in values/strings.xml as follows:

```
<string name="tam_server_url">https://tam.cgbe.trustonic.com/tam-
server<string>
<string
name="ca_cert_raw">MIIDcjCCAlqgAwIBAgIEEjRWADANBgkqhkiG9w0BAQsFADA/MQswCQ
YDVQQGEwJVSzEaMBgGA1UECgwRVHJ1c3RvbmljIExpbWl0ZWQxFDASBgNVBAMMC1RMUyBSb29
0IENBMB4XDTEzMDIyNzE2MzY1MVoXDTM4MDIyMTE2MzY1MVowPzELMAkGA1UEBhMCVUsxGjAY
BgNVBAoMEVRydXN0b25pYyBMaW1pdGVkMRQwEgYDVQQDDAtUTFMgUm9vdCBDQTCCASIwDQYJK
oZIhvcNAQEBBQADggEPADCCAQoCggEBAMaJYI6y7hxdxBoInTkiYhL2qhBtD0Kcmfx+NTiUUk
```

```
O+9u9qSyzbsN7kfgpsLO8Eq3AGg72zEjayXDfzmlHW1CnO/0nWDiM4b4hDhpJRspE2BCnKvAH
AvcQeGpzX5hhZLYh51Zrn/pOLCvoR9XV1inlw6M0+M9A5n11l6tEEWGbgWRnna+LJFX+bSGni
hP1Be2nHsVnqcIDMo/hzCj2ZX2G95e+UVPIc9JK/SVqFzYHltNjUyOFG7jGOncDhdG9vgHUoi
ikr6ZF7NHaovqxhIOiiK2tDVos//3/PgjrVwPkAJlVenMLpfEdxCtwyHO2frQpmkHc1eWFD5N
Gd8vzh2qECAwEAAaN2MHQwHQYDVR0OBBYEFE9Csq2lSvztAMSjVdll4sxTPwvbMB8GA1UdIwQ
YMBaAFE9Csq2lSvztAMSjVdll4sxTPwvbMA8GA1UdEwEB/wQFMAMBAf8wDgYDVR0PAQH/BAQD
AgIEMBEGCWCGSAGG+EIBAQQEAwICBDANBgkqhkiG9w0BAQsFAAOCAQEAQrnIh4jpJtNf6hqnC
pmwmQFD4456gFh0B3cmQnVvkfDCApJ+9G3xSsaL8LJRvK6c/pAV9p+0pvh3ftV4MFSw9AytZr
ihsrVykxlI1UGRHJmDO1hRh5QlfbMVfstHI0W8ec2Al41g3C9pM+FgBBKIoG6ewpDlaUbMXk8
033jf/OIyF5HTeQYqr788/ykFY32Mz0lpC2GdIeRThlK4ka63WuffdtKAayyPcitMeZtajJpa
7s02MZF9Dd5shISypnUvXAN/BZXIwQXSAOqajTGEv3X/wLyasm3nkEX29IgDvknLBoqnTS9rD
2LQ4BnqNQubr5XROBOlwdkrHTveN4Y9pA==</string>
```

Next, we supply the file name of the TA we want to install as a string (this is the same name generated as output from the build_ta.py script).

We determine which TEE mode will be used for this installation, as specified in TEEC_CHOICE in the setup.ini file. The rot13 sample is installed in the following example:

```
agent.installOrUpdateTA(getString(R.string.taRot13_uuid),
        this,
        TEEMode.fromInt(
                Integer.parseInt(
                        getString(R.string.taRot13_teec_choice))),
        false);
```

This produces a call back—a method that is triggered when the installTA() method is completed:

```
    @Override
    public void onInstallTACompleted(Outcome outcome) {

       String taMode = outcome.getTeeClient().name();
```

The outcome object supplies useful information, such as whether the TA was installed with a Trustonic TEE or with SWP, and a return type to indicate whether installation was successful or not:

```
        switch(outcome.getEventType()){
            case SUCCESSFUL:
                Log.d(TAG, "onInstallTACompleted::SUCCESSFUL");

                if (!CARot13.open()) {
                    Log.e(TAG, "Failed to open session to Rot13 TA");
                } else {
                    Log.d(TAG, "Session to the Rot13 TA established");
                    installOK = true;
                }

                break;

            case ERROR:
                Log.e(TAG, "onInstallTACompleted::ERROR");
                //method returned error. To see what happened, inspect the
exception thrown and perform custom logic
                Throwable errorCause = outcome.getException();
                Log.e(TAG, errorCause.getMessage(), errorCause);
        }
```

To install our second TA, we make a new THP Agent to handle our new TA. Aes (within Rot13 TAInstalled callback method) is used in the example.

```
agent.installOrUpdateTA(getString(R.string.taAes_uuid),
        new SecondListener(),
        TEEMode.fromInt(
                Integer.parseInt(
                        getString(R.string.taAes_teec_choice))),
        false); //aes
```

We pass as a second argument a new listener, called SecondListener(). This handles the second TA's call back, in the same way as onInstallTACompleted above handles the first THPAgent's call back.

```
    }

    private class SecondListener implements InstallTAListener {

        @Override
        public void onInstallTACompleted(Outcome outcome) {


            String taMode = outcome.getTeeClient().name();



            switch (outcome.getEventType()) {
                case SUCCESSFUL:
                    Log.d(TAG, "onInstallTACompleted::SUCCESSFUL");


                    if (!CAAes.CaOpen()) {
                        Log.e(TAG, "Failed to open session to Aes TA");
                    } else {
                        Log.d(TAG, "Session to the Aes TA established");
                        installOKAES = true;
                    }

                    break;

                case ERROR:
                    Log.e(TAG, "onInstallTACompleted::ERROR");
                    //method returned error. To see what happened, inspect
the exception thrown and perform custom logic
                    Throwable errorCause = outcome.getException();
                    Log.e(TAG, errorCause.getMessage(), errorCause);
            }

        }
```

4.2 UNINSTALLING A TA

This section describes how to use THP Agent to uninstall a TA.

Uninstalling a TA works in a similar way to `personalizeTA`. You call the method `uninstallTA`, passing the TA bundle (as you do for personalization) and an `UninstallTAListener` implementation i.e. a callback (similar to `personalizeTA`, except in that case you pass a `PersonalizeTAListener` implementation).

Example

For example, if your THPAgent instance is called "agent", initiate the uninstall flow using:

```
agent.uninstallTA(taName,this);
```

Use `this` if your Android code implements the `UninstallTAListener`. This means you need to implement the method `onUninstallTACompleted(Outcome result)`.

The following shows a simple example implementation of the method:

```
@Override
    public void onUninstallTACompleted(Outcome outcome) {
        if(outcome.getEventType() == EventType.SUCCESSFUL){
            Log.i("UNINSTALLTEST","Uninstall successful");
        }
        else{
            outcome.getException().printStackTrace();
        }
    }
```

4.3 SETTING THE DEBUG LEVEL

To help with debugging an Android application, you can configure THPAgent to provide more detailed information during testing. Set the debug level by calling the `setLogLevel(LogLevel.level)` method, where `LogLevel.level` is one of the following:

Level	Description
`LogLevel.ERROR`	Show only errors. This is the default setting. Use this setting in a production environment.
`LogLevel.WARNING`	Show errors and warnings. For example, a warning such as "`connection to TAM server failed, retrying. . . .`"
`LogLevel.INFO`	Show errors, warnings and messages that give high-level progress status. For example, "`Going online to retrieve personalization commands`" or "`the L2 SD is currently blocked: it is necessary to go online and retrieve the unblock command`" or "`performing Update TA`".
`LogLevel.DEBUG`	Show errors, warnings, messages and low-level information such as the raw commands being passed down to SD-TA. Use this setting for development.

`LogLevel.TRACE`	Show all of the above, plus detailed intermediate information. Use this if you are developing an application and want to send information to Trustonic for support purposes.

5 USING THE TUI LAYER LIBRARY

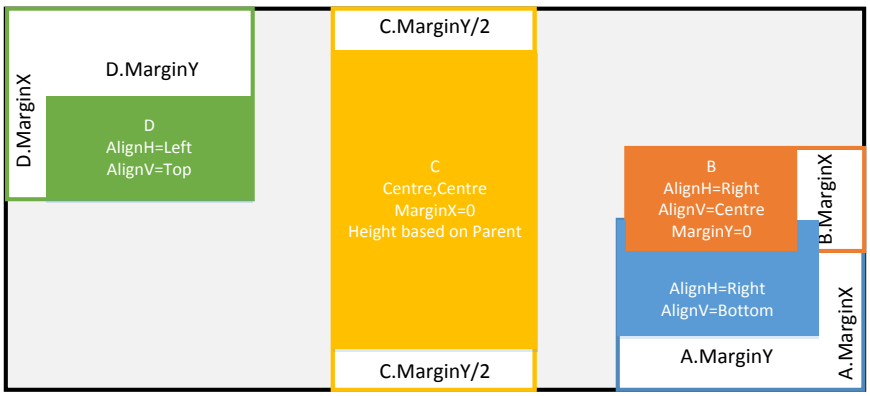
This chapter describes the TUI layer library. This is a simple layout manager and rendering engine that allows you to create UIs in the TUI using the industry-standard sketching tool.

The tool (referred to as the "box model" tool) consists of a structure of boxes, a set of static methods to assemble these and control rendering, and several independent renderers, each of which can render a single box based on some content; for example, a rectangle, image or text label.

5.1 LAYOUT MODES EXPLAINED

The following diagrams show how margins and alignment of children can be used to achieve different effects.

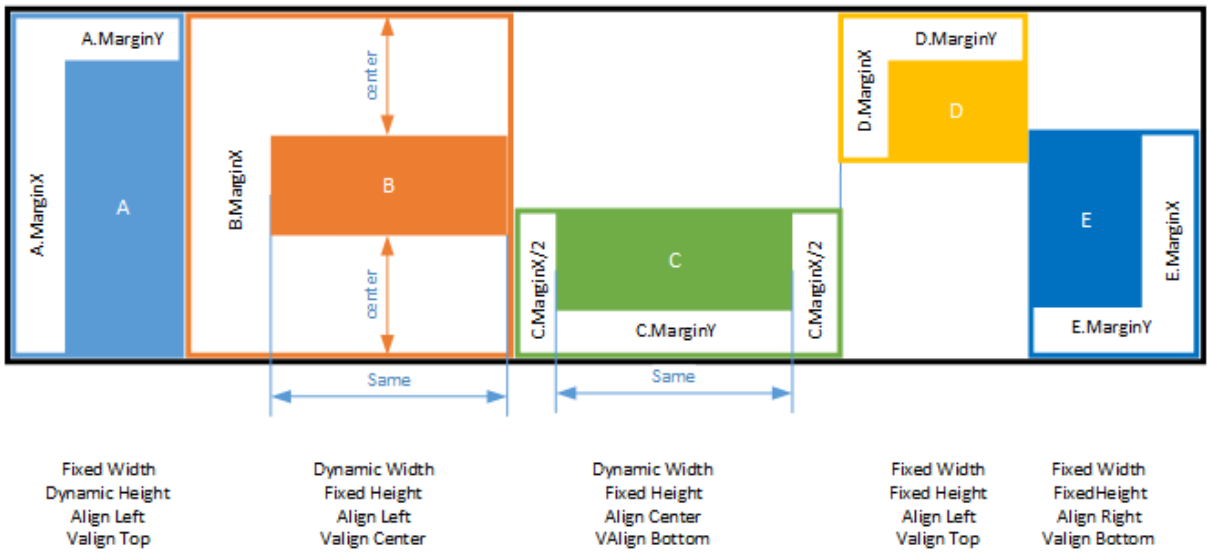
Example Layout within a parent Container with ChildLayout=None



In this example:

- Children are laid out independently, and may even overlap (they are rendered in first-last order, and receive events in last-first order)
- Boxes with height set to 'parent' take up the entire parent height – MarginY. Boxes with width set to parent are treated analogously.
- The parent box (the black box, in this example) can have its dimensions set to be 'based on children' in which case the dimension is set to be the maximum of the (child dimension + child margin) ignoring child boxes that are set to 'based on parent'.
- If a box's dimension is set to be based on content, then the size is determined by the renderer for that box. In practice, this is only useful for text areas
    - If the width is set to be 'based on content' then a single line of text is measured, regardless of length. If the height is also set to be based on content, it will be one line height.
    - If the height is set to be based on content, then the text is rendered within a rectangle of the specified width (whether explicit or based on children/parent) and the height is then set to the height necessary to render all the text.

Example Layout within a parent container with ChildLayout = Horizontal

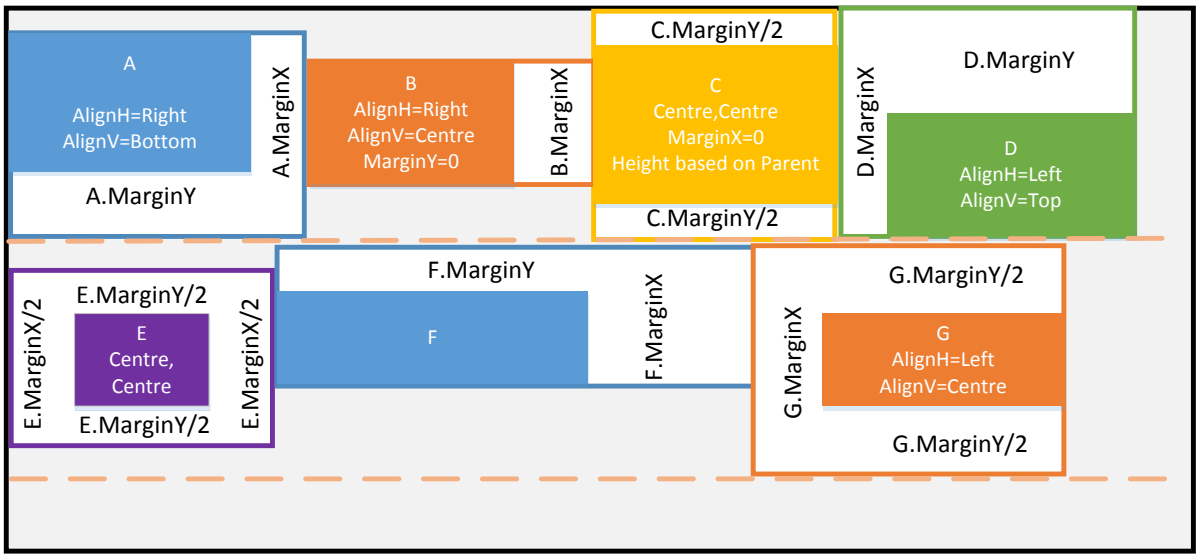


In the diagram, 'Dynamic' means 'based on Parent dimension'. In this example:

- Setting the 'flow' dimension to 'based on parent' shares the elastic space (horizontally, in this example), whereas setting the opposing dimension to 'based on parent' uses the whole space (vertically, in this example).
- Alignment can be used to position the shape in the opposing direction (vertically, in this case), and to position the margins in the flow direction (horizontally, in this case)
- When margins are applied to a centered box, half the margin is used on each side.
- The parent box (black box, in this example) can have its dimensions set to be 'based on children'. If the 'flow' dimension is set to be based on child dimensions (horizontal, in this case), then the parent width is set to the sum of the child margin values plus the sum of all children whose size is *not* set to 'based on parent'. If the opposing dimension is set to be based on child dimensions, then the parent dimension is set to be the size of the largest value of (child width + child margin), ignoring child sizes set to 'based on parent'.

The following diagram shows a complex flow example. In this 'flow' layout, all children have identical size and margins.

Example Layout within a parent Container with ChildLayout=Flow



In this example:

- The line height (shown dotted) for each line is based on the height of the tallest item on the line, including its HMargin
- The number of items on a line is not fixed, but is based on the number of items that will fit (with the first item that does not fit being moved to the next line)
- For calculation purposes, the child box considered is always (width+marginx,height+marginy). That box is then positioned based on AlignY for the child, and then the contents are positioned within that using marginX,marginY
- Width based on parent size is NOT supported with ChildLayout=Flow
- Height based on parent size is supported, but is interpreted to be the height of the current line, specifically (current line height – the box's margin). At least one box on the line must have a non-dynamic height.
- A box with ChildLayout=Flow may not have either its width or height set to be based on child size.

5.2 SCREEN SIZE ADAPTATION

5.2.1 Unit system overview

On Android, images assets and layout depends on both the screen *size* and the screen *density*. On layouts, measures and coordinates are not specified in physical pixels. Instead, they are usually specified in *dp* (device independent pixel) or *sp* (scaled pixel), which are not usable directly by the TUI.

On TUI, all units are physical screen pixels, so the library must handle the conversion from device-independent measurement to device-dependent measurement.

FreeType, which is the font renderer used by the TUI, needs a font size in points (1/72th of an inch) and a density in pixel/inch.

The TUI has access to this information through the API `TEE_TBase_TUI_GetScreenInfo()`, but may not be able to use this information directly (see below).

5.2.2 Android to TUI units conversion

The following section provides the conversion from Android units to units usable by the TUI and FreeType, and is based on the Android Documentation.

Unit name	Description
px	Actual device pixel. This is the only unit usable by the TUI API. Any dimension expressed in another unit must be converted to px before reaching the TUI API.
dp	*Device independent pixel* – 1/160th of an inch. Android defines it as: An abstract unit that is based on the physical density of the screen. These units are relative to a 160 dpi (dots per inch) screen, on which 1dp is roughly equal to 1px. Note that when converting this unit to **px**, Android does **not** use the device physical density information. It uses an approximate density, of the form *N*×160 where *N* is an integer. **convertion to px** $x_{px} = \texttt{metrics.density} \times x_{dp}$ **convertion for FreeType's `FT_Set_Char_Size` parameters**: `char_width` = $x_{dp} \times 72 \times 64 / 160$ `horz_resolution`=`metrics.density` × 160
sp	*Scale-independent Pixels* – similar to *dp* unit, but it is also scaled by the user's font size preference. **convertion to px:** $x_{px} = \texttt{metrics.density} \times x_{sp}$ **convertion for FreeType's `FT_Set_Char_Size` parameters:** `char_width` = $x_{sp} \times 72 \times 64 / 160$ `horz_resolution` = `metrics.scaledDensity` × 160

5.2.3 Android screen density buckets

On Android, image resources depend on the screen *density*. Android does not directly use the physical screen *density* measured in pixel/inch. Instead, it classifies the device screen into one of the following buckets:

bucket name	scaling factor (`metrics.density`)
`ldpi`	0.75
`mdpi`	1.0
`hdpi`	1.5

xhdpi	2.0
xxhdpi	3.0
xxxhdpi	4.0

Each bucket has a *scaling factor* which expresses the density relative to a reference resolution of 160 pixels/inch. For example, the Samsung Galaxy S7 Edge has 1440×2560 pixel display, for a screen measuring 2.86in×5.94in. The physical density is therefore approximatively 534 pixel/inch. Android classifies this phone in the `xxxhdpi` density bucket, so as far as Android is concerned, the phone has a density of 4×160 = 640 ppi. On this device, widget of 10×10 dp would take 40×40 screen pixels

The secure world does not have access to the scaling factor used by the normal world. The TUI reports the density in physical pixel/inch, but that does not give the bucket information. The secure world does not have access to the bucket information. The TA must obtain the scaling factor from the normal world.

The CA can get the screen density using the `DisplayMetrics()` API

```
DisplayMetrics metrics = new DisplayMetrics();
getWindowManager().getDefaultDisplay().getMetrics(metrics);
metrics.density; // scaling factor for the density
scaledDensity;   // scaling factor for adjusted by user preferences
```

5.2.4 Box model scaleX and scaleY

Since the box model applies the scaling factors `boxmodel.scaleX` and `boxmodel.scaleY` to all the box dimensions, the TA can express all dimension in the unit of its choosing (px, dp, sp), by setting the scale factor appropriately as summarized in the table below

Value of scaleX/scaleY	**Units in which boxes and text dimension are expressed**
1	px
Android's `metrics.density`	dp
Android's `metrics.scaledDensity`	sp

5.2.5 Layout

The layout used on the normal world depends on screen *size* measured in pixels. The TUI info returned by `TEE_TBase_TUIGetScreenInfo` is used for this.

5.3 EVENTS

Three types of events are supported:

1. Regular Events
    - These are identified events with an x,y coordinate which are passed to the 'closest' box and which bubble upwards through the box hierarchy until handled
        - Specifically, the top-level boxes are ordered top to bottom and processed in this order

        - For each visible root box, the box hierarchy is descended to find the most closely containing box
        - This box is passed the event. If it returns 'false' from its handler (or has none), then the event is passed to its parent, and their parent back up the tree
        - If no box under the current visual root handles the event, it is passed to the next lowest visual root, and the process continues
    - Typical uses for events of this type are touch, move when touching, or release
    - Custom events can be added, and arbitrary data passed to any event.
2. Secondary Events: Enter and Leave
    - These are generated automatically if more than one 'move' event is seen in a row, or a 'move' follows a 'touch' (any 'release' event will cancel)
    - If the previous event was outside a box, and the new event is inside the box, then an 'Enter' event is generated for that box. Similarly, if a previous event was inside a box, and the new event is outside the box, a 'Leave' event is generated.
    - Enter and Leave events DO NOT BUBBLE. They apply separately to each box entered/left.
3. Pseudo-events: Shown and Hidden
    - These are generated automatically *for every box* in a shown/hidden hierarchy when the show() or hide() methods are called.
    - These events also DO NOT BUBBLE.

Secondary and pseudo-events are relatively expensive to process, so can be disabled if not needed.

The box model also supports a simplified view of Touch/Release if the Move/Enter/Leave events are not used. In this mode, Release is registered at the same location as Touch, regardless of whether there was an intervening movement (in the more advanced mode, buttons will respond to Touch followed by any of Release/Leave/Hide).

5.4 TUI LAYER LIBRARY DEFINITIONS

5.4.1 Constant

5.4.1.1 Boxes dimension mode

These constants indicate how the box dimension is calculated. They must set in the field `widthMode` and `heightMode` of the `Box` structure.

Constant name	Value	Description
`EXPLICIT`	`0`	Dimension is explicitly set in the `width` or `height` field of the box. The function `BoxModel_Layout` will not change the `width` or `height` field
`FROM_PARENT`	`1`	Dimension is computed from the parent box dimension
`FROM_CONTENT`	`2`	Dimension is computed from the content. This applies to text box only
`FROM_CHILDREN`	`3`	Dimension is computed from the child nodes

5.4.1.2 Box state

These constants are bit flags indicating the state of the box.

Constant name	Value	Description
STATE_ACTIVE	1	The box is active.
STATE_DISABLED	2	The box is disabled.
STATE_PRESSED	4	The box is pressed.
STATE_INHERIT	8	The box inherits the state of its parent.

5.4.1.3 Child layout

These constants indicate how children boxes are laid-out in a parent box.

Constant name	Value	Description
CHILD_LAYOUT_NONE	0	Each child is laid out independently of each other.
CHILD_LAYOUT_HORIZONTAL	1	Layout children horizontally, from left to right, within the box.
CHILD_LAYOUT_VERTICAL	2	Layout children vertically, from top to bottom, within the box.
CHILD_LAYOUT_FLOW	3	Layout children first horizontally, from left to right, and then top to bottom, within the box.

5.4.1.4 Box alignment

The following constants indicate how a box is positioned relative to its parent.

5.4.1.4.1 Horizontal alignment

These flags set the alignment of the box relative to its parent. They also control where the margin of the box, if any, is placed. Refer to the description of the box flags `hAlign` and `vAlign` for details on how these flags are used.

Constant name	Value	Description
ALIGN_CENTER	0	Center the current box horizontally relatively to its parent
ALIGN_LEFT	1	Place the left border of the box on the left border of its parent.
ALIGN_RIGHT	2	Place the right border of the box on the right border of its parent.

5.4.1.4.2 Vertical alignment

Constant name	Value	Description
ALIGN_CENTER	0	Center the current box vertically relatively to its parent
ALIGN_TOP	1	Place the top border of the box on the top border of its parent.
ALIGN_BOTTOM	2	Place the bottom border of the box on the bottom border of its parent.

5.4.2 Data structures

5.4.2.1 Box Structure

Each Box is a C structure with the following members:

```
typedef struct Box
{
    char *db;

    uint32_t width; /* logical pixels, scale to device pixels using boxmodel.ScaleX */
    uint32_t height; /* logical pixels, scale to device pixels using boxmodel.ScaleY */

    uint32_t x; /* 0,0 is top left, moving right/down as values grows. logical pixels*/
    uint32_t y;

    /* offset for layout from natural position defined by layout flags */
    uint32_t marginX;
    uint32_t marginY;

    /* Flags - layout, state, etc */
    struct boxFlags flags;

    /* Structure */
    struct Box *parent;
    struct Box *nextSibling;
    struct Box *firstChild;

    /* Content */
    void *content;
    bool (*handleEvent)(struct Box *self, uint32_t x, uint32_t y, UIEventType type, void
*data);
} Box;
```

- `width, height` The size of the box in logical pixels. These are functions used by the application program, and rendering engines to manipulate boxes.
- `x, y` The absolute location of the top left corner of the box on the screen. For child boxes these values are set automatically by the layout manager. (0,0) is top left
- `parent` A pointer to the parent box that contains this box (or null for top level boxes/popups)
- `firstChild, nextSibling, parent` Internal references used to enable efficient graph walking. These should not be set explicitly; they are updated automatically as boxes are added to the graph.
- `content` A void * reference to content that is used by the rendering engine. The content and interpretation of the data pointed to depends on the box type. For example, it could be the colour information and/or text content. Content may be set to a structure representing a set of content pointers, and the rendering engines will choose the correct one based on the box's state. If this is used, the F flag must be

set. Note that the DYNAMIC_CONTENT structure may also be used to remember application data for a box, which may then be used in event handlers (for example).

- `flags` A set of flags described in Box Flags.
- `marginX, marginY` These values are used to adjust the position of the box during layout. Layout uses the size (width+marginx,height+marginy) and then positions the box inside this bounding box based on alignment. Margins apply on one side of a box, unless it is centred, where the margin is split.
- `Renderer` A reference to a function to render the box's content. The box model itself takes no part in rendering, other than working out which boxes to render, and in what order, and setting the clipping bounds appropriately.
- `handleEvent` A pointer to a function that will be sent any events related to this box, such as touch events. The event handling function is told the x,y location of the event, the event type and extra data (void*). It may return 'true' to indicate the event is handle, or 'false' in which case the event is offered to its parent.

5.4.2.2 Box Flags

The box flags are a stored in the C bitfield defined as follows:

```
struct boxFlags
{
    unsigned int widthMode : 2;
    unsigned int heightMode : 2;
    unsigned int state : 3;
    unsigned int dynamicContent : 1;

    unsigned int childLayout : 2;
    unsigned int hidden : 1;
    unsigned int opaque : 1;
    unsigned int hAlign : 2;
    unsigned int vAlign : 2;

    unsigned int childLayoutReversed : 1;
    unsigned int layoutAdjustment : 1;
    unsigned int _pad : 6;

    unsigned int renderer : 8;
};
```

- `Hidden` Used to indicate a box is Hidden and neither it, nor its children, should be drawn, but it will still take up space.
- `opaque` The box is fully opaque, so redrawing it will not require redrawing its parent beneath it.
- `childLayout` One of four layout schemes for child boxes under this box:
    1. `CHILD_LAYOUT_NONE`
    2. `CHILD_LAYOUT_HORIZONTAL` - Child boxes are placed in a horizontal row, one after the other, left to right (or right to left)
    3. `CHILD_LAYOUT_VERTICAL` - Child boxes are placed in a vertical column, one after the other, top to bottom (or bottom to top)
    4. `CHILD_LAYOUT_FLOW` - Child boxes are flowed left to right, line by line

- `widthMode, heightMode`
  Set the size of this boxes based on one of the following schemes. Horizontal and Vertical schemes can be set separately.
  1. `EXPLICIT`– size is given in pixels
  2. `FROM_PARENT` – size is based on parent size
  3. `FROM_CONTENT` – size is based on size of content (only applies to text)
  4. `FROM_CHILREN` – size is the smallest size needed to layout children with the specified scheme.
- `hAlign` When the parent `childLayout` field is set to either `CHILD_LAYOUT_NONE` or `CHILD_LAYOUT_VERTICAL`, this fields specify where the box is horizontally placed in its parent box. After positioning a logical box of size (width+marginX,height+marginY), in any parent's `childLayout` mode, position the box within this logical box based on the left/center/right alignment value so that the margin is on the left, for left alignment, on the right for right alignment and split 50/50 for centred alignment.
- `vAlign` When the parent `childLayout` field is set to either `CHILD_LAYOUT_NONE` or `CHILD_LAYOUT_VERTICAL`, this fields specify where the box is horizontally placed in its parent box. After positioning a logical box of size (width+marginX,height+marginY), in any parent's `childLayout` mode, position the box within this logical box based on the top/center/bottom alignment value so that the margin is on the top, for top alignment, on the bottom for bottom alignment and split 50/50 for centred alignment.
- `state` Used by renders and event handlers, but without direct influence on layout. State is one of:
  1. `NORMAL` Regular state (default)
  2. `PRESSED` E.g. a button currently being touched / depressed
  3. `ACTIVE` E.g. a radio button that is currently selected
  4. `DISABLED` E.g. a disabled button
  5. `INHERIT` The state should be read from the first ancestor box whose state is not itself inherit. Useful when a logical button is split into multiple sub-boxes for rendering.
  6. `DYNAMIC_CONTENT` Used to indicate that the content used for rendering (see below) is dependent on the box state. Renders should detect this state and read the correct content based on the box state.

5.4.2.3 Properties

Properties of the box model are stored in a global object of type `BoxModel`. This global object is accessible by the application under the name `boxmodel`.

```
typedef struct BoxModel
{
    int clipX;
    int clipY;
    int clipW;
    int clipH;

    float scaleX;
    float scaleY;

    Box *rootList;

    int registeredEvents;
```

```
    void(*render)(Box *element);
    void(*measure)(Box *element);
} BoxModel;

extern BoxModel boxmodel;
```

- `scaleX,scaleY`
  - Scaling factors. These do not affect the box model itself, but should be used by renders to scale the logical dimensions to physical dimensions. They can also be used to select appropriate assets.
- `clipX,clipY,clipW,clipH`
  - Clipping region. Rendering functions can *optionally* use this to avoid unnecessary processing. The box model itself uses these to avoid calling Render() on any box outside the clipping region
- `registeredEvents`
  - A bitset of events that will be propagated to boxes. Set to 'all' by default, it may be reduced to improve performance. In particular, 'leave', 'enter', 'shown' and 'hidden' events are relatively expensive and can be disabled if not in use.
- `rootList`
  - A tree of all boxes in the box model. Should not be used directly, but instead boxes should be declared by calling the Add function. Made accessible to allow static provisioning (to avoid need for malloc).

5.5 TUI LAYER LIBRARY FUNCTIONS

These are functions used by the Trusted Applications, and rendering engines to manipulate boxes.

5.5.1 Synopsis

```
Box *BoxModel_Register(Box *box);
void BoxModel_Layout(Box *element);
void BoxModel_Render(Box* element);
void BoxModel_RenderVisible(void);
UIEventType BoxModel_HandleButtonTouchRelease(Box *self, UIEventType type);
void BoxModel_RaiseEvent(short x, short y, UIEventType type, void *data);
void BoxModel_Hide(Box *el);
void BoxModel_Show(Box *el);
void *BoxModel_GetContent(Box* element);
```

5.5.2 BoxModel_Register

```
Box *BoxModel_Register(Box *box);
```

Description

Declare existence of a new box. All boxes must be declared. It adds the box provided as argument to the box graph and updates it (each parent links to the first child, and each child links to its next sibling).

Note. Top-level boxes (background, popups, etc) must be added in Z-order, lowest first. The relative order of boxes under a given parent dictates how they are laid out, but there is no need to add all children from one box before another.

Parameters

- box — The box to be added to the global box forest. The field parent must be set to the parent of the box, so this is added at the correct place in the graph. The other fields for the box relatives are automatically updated by the function.

Return value

Returns the box itself, to allow more compact code.

5.5.3 BoxModel_Layout

```
void BoxModel_Layout(Box *element);
```

Description

Layout the contents of a box (recursively). This function computes the position and dimension of all the boxes that are descendant of `element`. This call is typically only called once, unless the contents or position change. Note as the x,y coordinate of each box is stored in absolute coordinates, if a popup is moved, Layout() must be called again.

For scrolling boxes [future], boxes inside a scrolling container will have their coordinates stored relative to the region being scrolled *not* the screen. Layout will not need to be called if an area scrolls.

5.5.4 BoxModel_Render

```
void BoxModel_Render(Box* element);
```

Description

Render *the region covered by* the specified box. This may include items below or above the box in the visual stack. For example, if the image on a button is changed, Render(button) may be called to draw it showing 'pressed' state. The box model will take care of any boxes that overlap this, such as popups (by redrawing them), and will also take care of boxes under this one if it is not opaque.

5.5.5 BoxModel_RenderVisible

```
void BoxModel_RenderVisible(void);
```

Description

Render all visible boxes. This is less efficient than the Render() call below, as it redraws the entire UI

5.5.6 BoxModel_Show

```
void BoxModel_Show(Box *el);
```

Description

Set the box to Visible, raise a shown event, and display.

5.5.7 BoxModel_Hide

```
void BoxModel_Hide(Box *el);
```

Description

Set the box to Hidden, raise a hidden event and redraw region under the box.

5.5.8 BoxModel_RaiseEvent

```
void BoxModel_RaiseEvent(short x, short y, UIEventType type, void *data);
```

Description

Raise an event. The Event is routed to the top-most visible box at the coordinates (x,y). If this has no handler, or returns false from the handler function, then the event is passed to each parent in turn. If nothing in this stack handles the event at all, that 'stack' is ignored and the process restart with the next most visible stack.

5.5.9 BoxModel_HandleButtonTouchRelease

```
UIEventType BoxModel_HandleButtonTouchRelease(Box *self, UIEventType type);
```

Description

1. [Optionally] utility method for boilerplate event handling
2. Handles Touch/Release/Leave events for a box by checking the box is enabled, changing its state to pressed / released, and redrawing the box.
3. Will return EventType.Touch/EventType.Release for handled events, and EventType.None otherwise.

5.5.10 BoxModel_GetContent

```
void *BoxModel_GetContent(Box* element);
```

Description

Return the content for a box, taking into account potential dynamic content (content based on the box or ancestor box state).

Renderer should not read the content directly (`box->content`) unless willing to handle dynamic content themselves.

5.5.11 BoxModel_Free

```
void BoxModel_Free(Box* element);
```

Free the resources allocated for the box, either at by BoxModel_Render or BoxModel_Layout.

5.5.12 Renderers

Renderers are extensions to the box model to support rendering of individual boxes. Each renderer takes a box as an argument, and each box contains a pointer to the relevant function for that box (or null if the box itself is not rendered – for example if it is simply a container for other rendered boxes)

The renderer will typically use the following information

- box->X,Y,Width,Height
  - Screen location to render at, plus size of the box
- box->Flags
  - Flags for the box, in particular, the box's state.
- box->content
  - void* pointer to data related to this renderer (e.g. text, color, font etc.)
    This should always be accessed via GetContent() to enable dynamic state
- clipX,clipY,clipW,clipH
  - Clipping region, which can be used to avoid drawing unnecessary areas. Clipping is useful when there are multiple top level elements (e.g. popups).
  - A renderer does not have to limit itself to the clipping region (but MUST limit itself to the box's dimensions), however *if* scrolling support is added in future, renderers will have to respect clipping regions, so this is good practice.
- scaleX,scaleY

    - Scaling to be applied to coordinates used in the box model. This is to enable DPI independent UIs.
    - Note that the box model does not directly make use of scale – however Renders must take this into account and should scale box dimensions using these factors before applying to the underlying display.

5.5.13 Rendering Functions

Rendering functions used in the test framework (and proposed for production)

- Solid rectangle
- Simple border (hollow rectangle with line width = 1 pixel)
- Flowing text (left to right, top to bottom, with words fitted on a line broken at spaces).
    - For Japanese and similar languages, spaces can be used to hint at break points, but are not actually rendered.
- Image (PNG or Raw)
    - Note that there is no scaling support at present
    - For production, a table of assets may be supported to allow easy configuration of the set of image assets for a given device/resolution.
- Tiled Image (PNG or Raw)
    - Repeat image in X and Y directions. Useful for generating complex button faces by replicating a thin strip vertically or horizontally.
- Advanced Rectangle
    - Optional rounded corners (radius + list of corners to apply to)
    - Optional linear gradient (start/end colors + horizontal or vertical direction). Uses basic interpolation of each of A,R,G,B channels.
    - Note that there is limited/no anti-aliasing support, and “under the covers” the rectangle is created by a stack of smaller, solid rectangles.

5.5.14 Notes on Rendering Buttons

To render shaped buttons, try the following:

- Image based.
    - Use an image for the entire button.
    - Use alternate images for other states (e.g. pressed, disabled etc.)
- Advanced Rectangle
    - Rounded buttons and simple gradients are supported; however, the lack of advanced gradients or antialiasing means this may not work in all cases.
    - Alternate content can be used for other states.
- Slices/Tiled Images
    - The left and right of the button are supplied as images, and the centre as a third image that is tiled.
    - Alternate images can be used for the other states.
    - BoxModel has good support for buttons assembled in this way, including layout and state inheritance.
- Sliced Rectangles
    - Where a button face has a variable gradient, it may be assembled from several sub-rectangles, each using a linear gradient / single color

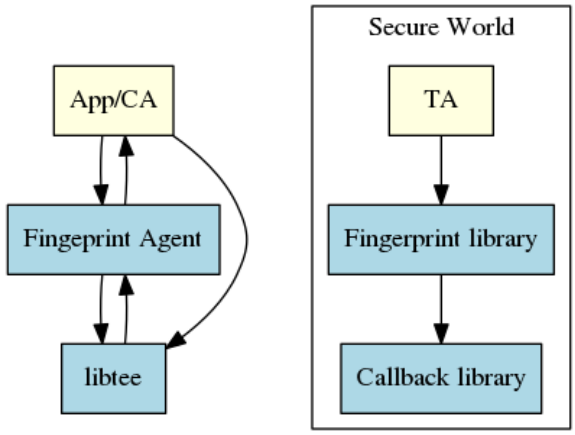
6 FINGERPRINT SUPPORT FOR TRUSTED APPLICATIONS

This chapter describes the API extensions that allow a TA to authenticate a user using their fingerprint.

6.1 ARCHITECTURE

Fingerprint authentication in the TA uses the Android API for fingerprint.

The fingerprint authentication in the TA is the joint operation of two additional components, as well as the Client and internal APIs.



- The fingerprint library is a static library for TAs. It provides the fingerprint authentication APIs to the TAs and handles the communication with the Android fingerprint API.
- The fingerprint Agent is an Android archive. It is a proxy between the fingerprint library and the Android fingerprint API. Additionally, it provides Java listeners so that the user application can handle the fingerprint authentication UI by itself.

6.2 USAGE

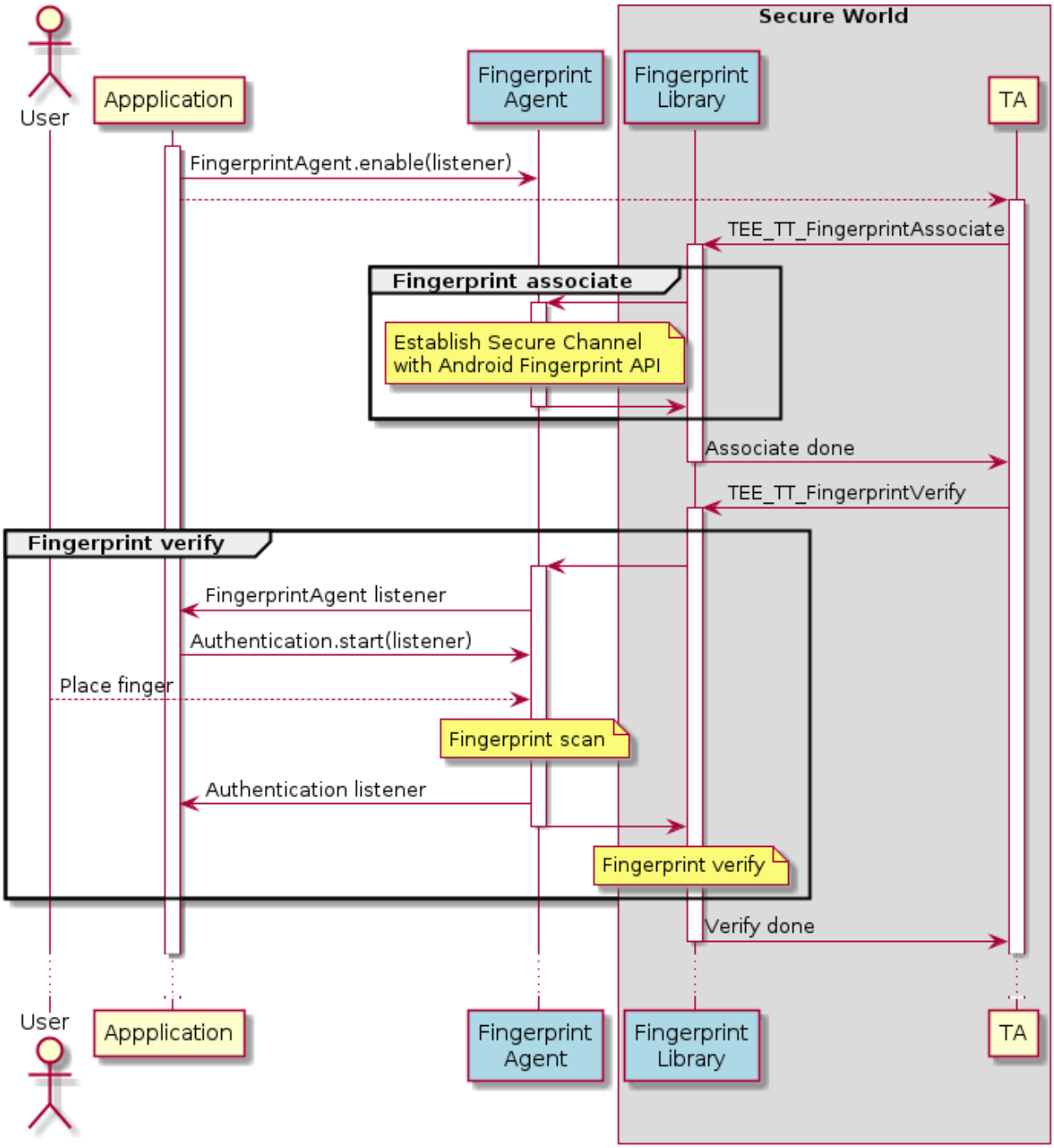
The level of service of the fingerprint API for TAs is similar to the Android fingerprint API. It offers a reduced set of operations:

- **Associate**/**Dissociate**: establish/destroy a secure channel with the Android fingerprint API. This operation does not require a fingerprint capture.
- **Verify**: launch a fingerprint authentication. This operation requires a fingerprint capture.

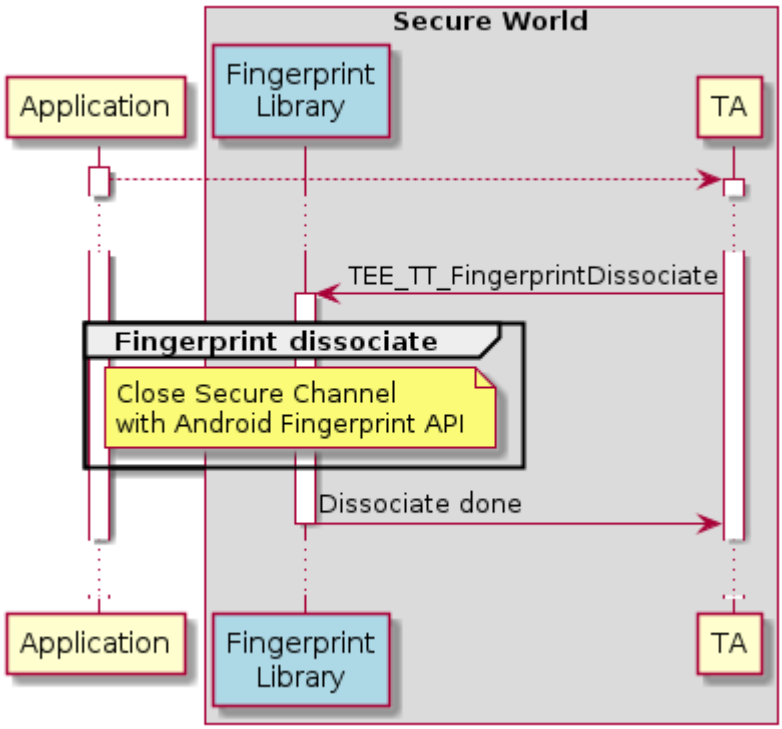
There is no enroll operation as the fingerprint API for TAs relies on templates enrolled in the Android security settings. Note that fingerprint templates must be registered, but enabling Android features based on top of these templates is not necessary.

The following diagrams show the typical sequence of operations to authenticate a user using fingerprints from a TA.

Authenticating:



Dissociating:



6.3 PREREQUISITES/LIMITATIONS

The TA targets devices that provide Android 6.0 API and fingerprint sensor hardware.

Android 6.0 does not allow enrolling a fingerprint using a user application but the settings application. Then the TA does not support the fingerprint enroll operation but relies on fingerprint templates registered by the Android security settings.

The current implementation does not offer any secure way to get the public key securely into the TA. It is directly recovered from the Android security API.

6.4 INTERNAL API EXTENSION

6.4.1 Functions

6.4.1.1 TEE_TT_FingerprintAssociate

This function gets an object handle associated to the fingerprint templates enrolled by the Android system.

```
TEE_Result TEE_TT_FingerprintAssociate(TEE_ObjectHandle *);
```

6.4.1.2 TEE_TT_FingerprintVerify

This function verifies a fingerprint against an object handle previously associated.

This function starts a fingerprint capture process including a user interaction.

```
TEE_Result TEE_TT_FingerprintVerify(TEE_ObjectHandle);
```

6.4.1.3 TEE_TT_FingerprintDissociate

This function dissociates an object handle from the fingerprint templates enrolled by the Android system.

```
TEE_Result TEE_TT_FingerprintDissociate(TEE_ObjectHandle);
```

6.5 CLIENT API EXTENSION

6.5.1 class FingerprintAgent

6.5.1.1 Constructor

```
public FingerprintAgent(OnCallback callback)
```

This creates a new instance of the fingerprint agent.

The callback parameter is a listener that must implement the OnCallback interface.

6.5.1.2 Methods

```
public void enable() throws TeeException
```

The enable method must be called to enable the fingerprint support. No fingerprint listeners can be invoked if not called.

```
public void disable()
```

The disable method disables the fingerprint support. No fingerprint listeners can be invoked once called.

6.5.1.3 Interface

```
public interface OnCallback {
    void onCallback();
}
```

This interface provides a listener for authentication notification. It is invoked each time a fingerprint capture is requested allowing the app to handle the UI.

6.5.2 class Authentication

6.5.2.1 Constructor

```
public Authentication(Context context, Callback callback)
```

This create a new instance of an object that is supposed to be active within a fingerprint capture.

6.5.2.2 Methods

```
public void start()
```

The start method enables the fingerprint capture. No fingerprint capture listeners can be invoked if not called.

```
public void cancel(String message)
```

The cancel method can be used to cancel a fingerprint capture. No fingerprint capture listeners can be invoked once called.

6.5.2.3 Interface

```
public interface Callback {
    void onDone(String message);
    void onErrorMessage(String message);
    void onHelpMessage(String message);
}
```

This interface provides a listener for fingerprint capture events. It indicates the progress in authentication allowing the app to refresh the UI.

The capture ends with `onDone is successful` or `onErrorMessage` if failing. `onHelpMessage` provides information enabling the user to better process the capture.

Appendix I. MAKEFILE KEYWORDS AND FLAGS

This appendix provides information about the keywords and flags used in the makefile.mk of the TA.

Keyword/flag	Description
SDL1_ALIAS	the name of the Company key, as provided to key_gen.py
SDL2_ALIAS	the name of the Group key, as provided to key_gen.py
TA_KEY_ALIAS	the name of the TA “App” key, as provided to key_gen.py
PIN	the password to enable use of the keys held in the keystore
TA_VERSION	increment this to allow THPAgent’s installOrUpdateTA() to upgrade the currently installed TA binary, when required
SRC_CPP, SRC_C, SRC_S	the filenames of the SP’s C++, C and Assembler source code files, respectively
GP_ENTRYPOINTS	use the GlobalPlatform TEE Client API (mandatory)
TA_INSTANCES	the maximum number of concurrently active instances
TBASE_API_LEVEL	the version of the Kinibi API being targeted. To use later Kinibi APIs or TEE features, set this level to one where those APIs are
HEAP_SIZE_INIT	the initial size of heap allocated during the loading of the TA
HEAP_SIZE_MAX	the maximum size of heap during the TA's lifetime (optional)
INCLUDE_DIRS	include paths. Relative paths are supported in TAP version 1.3 only
TA_STATIC_KEY_FILE	the path to the key description file for the build-time provisioned SWP keys

Cryptographic algorithm keywords/flags

The following lists the crypto algorithms required for software-protected Trusted Applications:

Keyword/flag	Description
HIGHSPEED_AES	the High-Speed AES algorithm set for the entire Trusted Application instead of the High-Security AES
RSA	RSA decryption, RSA signature and RSA key handling in Secure Storage
DES	DES symmetric encryption/decryption
HMAC	HMAC signing and verification (SHA-1, SHA-224, SHA-384, SHA-512, MD5)

ECDSA	ECDSA signing and ECC key pair generation using up to 264-bit and up to 528-bit prime curves, ECC key handling in Secure Storage
ECDH	ECDH and ECC key pair generation using up to 264-bit and up to 528-bit prime curves, ECC key handling in Secure Storage
DSA	DSA signing, DSA key generation and DSA key handling in Secure Storage
UNWRAPPING	key unwrapping for all key types
KDF	key derivation algorithm (for TEE_TT_KDF)

Reserved

Keyword/flag	**Description**
AUTH_TOOL_KEYSTORE	DO NOT USE. Allows the keystore to be provided elsewhere; however, this feature is not yet fully integrated
TA_SERVICE_TYPE	must be set to SYS
TA_BUILD_PATH	auxiliary variable, used as Trusted Application project root directory. DO NOT change as other TAP scripts rely on this
TRUSTED_APP_DIR	auxiliary variable, used as Trusted Application project source code directory. DO NOT change as other TAP scripts rely on this
TRUSTED_APP_BIN	auxiliary variable indicating the path to intermediate Trusted Application binaries. Do not change as other TAP scripts rely on this