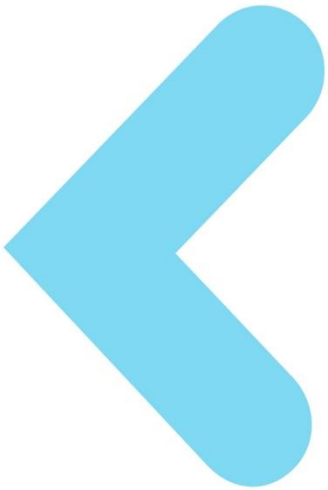
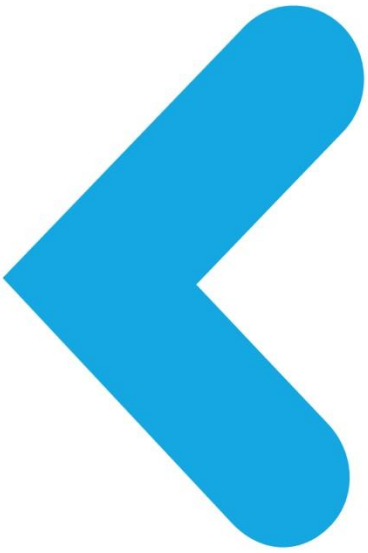
TRUSTONIC



Kinibi API Documentation

API Level 11



PREFACE

DOCUMENT HISTORY

Modification	Date
First version for API Level 1	February 1st, 2013
Updated for API Level 2	March 18th, 2013
Updated for API Level 3	November 18th, 2013
Updated for API Level 4	May 23rd, 2014

Updated for API Level 5:
Removed definitions of unsupported algorithms:
`TLAPI_ALG_MD2`
`TLAPI_ALG_MD5`
`TLAPI_ALG_SHA384`
`TLAPI_ALG_SHA512`
TEE_ALG_RSASSA_PKCS1_PSS_MGF1_SHA1 is supported since API level 3.
Introduced extended memory layout.
Added `tlApiAddEntropy()`
Added `TEE_TBase_AddEntropy()`
Added `tlApiGetSecureTimestamp`()
	December 1st, 2014

Clarify that the block allocated by tlApiMalloc/tlApiRealloc or freed by tlApiRealloc/tlApiFree is filled with zeroes.	December 19th, 2014
Updated for API Level 6: Added new APIs for Trusted User Interface (TUI).	June 17th, 2015

Updated for API Level 7	July 20th, 2015
Updated for API Level 8	October 29th, 2015
Updated for API Level 9 Added new API TEEC_TT_RegisterPlatformContext()	February 12, 2016
Updated for API Level 10 Added TEE_GetTAPersistentTime() and TEE_SetTAPersistentTime()	April 7th, 2016
Updated for API Level 11	September 19th, 2016
Added KinibiChecker API	February 2nd, 2017
Deprecate TLAPI_ALG_DES_MACXXX	March 24, 2017
Add printf supported formats	Feb 15, 2018
Add APK identity property	June 26th, 2019

TABLE OF CONTENTS

LIST OF TABLES

List of FIGURES

1 INTRODUCTION

This document specifies the API for developing Trusted Applications and Client Applications:

Kinibi provides two distinct set of APIs for developers:

- The Kinibi Legacy API to develop legacy Trusted Applications using the tlAPI and the mcClient API. Note that the mcClient API is also available at the kernel level.
- The GlobalPlatform TEE APIs to develop Trusted Applications as defined by GlobalPlatform using the GlobalPlatform TEE Client and Internal APIs.

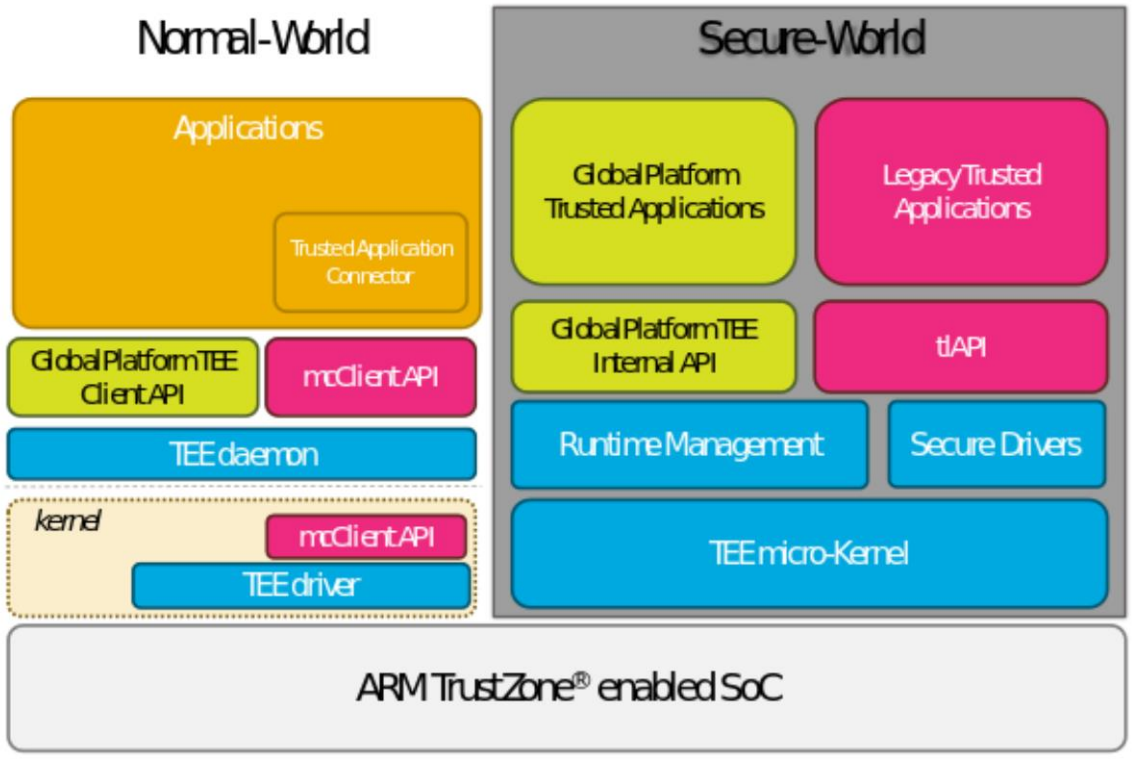


Figure 1: Kinibi API Overview.

For introduction and guidance on how to develop for Kinibi, please refer to the Kinibi Developer Guide.

2 API VERSION HISTORY

API Level	Change	First TEE Version Where New API Level is Introduced
Level 1	First Kinibi API	Kinibi-199
Level 2	Added mcOpenTrustlet() function Added tlApiGetVirtMemType() function Added tlApiIsNwdBufferValid() function mcOpenSession() is deprecated mcMallocWsm() is deprecated mcFreeWsm() is deprecated Added tlApiDeriveKey() function	Kinibi-200
Level 3	Added Trusted User Interface API Added DRM API Added GlobalPlatform TEE APIs Added tlApiEndorse() Added `tlApiGetSecureTimestamp()`	Kinibi-300
Level 4	Added support for DSA and ECDSA	Kinibi-301
Level 5	Extended memory layout Added tlApiAddEntropy() Added TEE_TBase_AddEntropy()	Kinibi-302A
Level 6	Added tlApiTuiDrawBuffer() Added tlApiTuiDrawImage() Added tlApiTuiFlipFramebuffers() Added tlApiTuiFillRectangle() Added tlApiTuiGetImageInfo() Added tlApiTuiDecodeImage() Added tlApiTuiGetClippingRectangle() Added tlApiTuiSetClippingRectangle() Added tlApiTuiGetScreenContext() Added TEE_TBase_TUI_DrawBuffer()	Kinibi-303A

	Added TEE_TBase_TUI_DrawImage() Added TEE_TBase_TUI_FlipFramebuffers() Added TEE_TBase_TUI_GetScreenContext() Added TEE_TBase_TUI_FillRectangle() Added TEE_TBase_TUI_GetImageInfo() Added TEE_TBase_TUI_DecodeImage() Added TEE_TBase_TUI_GetClippingRectangle() Added TEE_TBase_TUI_SetClippingRectangle()	
Level 7	Added tlApiDrmProcessContentEx() Added TEE_TBase_DRM_ProcessContentEx() Added support for MD5, SHA224, SHA384 and SHA512. Added support for RSA OAEP, PKCS1 and PSS for all the SHAx Added support for HMAC for all the SHAx. Added support for TEE_DeriveKey(). Added support for BigInt in GP API. Added support for DSA, DH, ECDSA and ECDH in GP API. Added support for object enumeration in GP API. Added support for TEE_GetPropertyAsIdentity() Added support for TEE_GetSystemTime() and TEE_GetREETime()	Kinibi-310A
Level 8	Added support for real timeouts for tlApiWaitNotification()	Kinibi-310B
Level 9	Added TEEC_TT_RegisterPlatformContext()	Kinibi-310C
Level 10	Added TEE_GetTAPersistentTime() Added TEE_SetTAPersistentTime()	Kinibi-311A
Level 11	Added TEE_AllocatePropertyEnumerator() Added TEE_FreePropertyEnumerator() Added TEE_StartPropertyEnumerator() Added TEE_ResetPropertyEnumerator() Added TEE_GetPropertyName() Added TEE_GetNextProperty() Added TEE_OpenTASession()	Kinibi-400A

	Added TEE_InvokeTACommand() Added TEE_CloseTASession() Added TEE_Wait() Added TEE_CopyOperation() Added TEE_GetOperationInfoMultiple() Added TEE_ResetOperation() Added TEE_SetOperationKey2() Added TEE_AEInit() Added TEE_AEUpdateAAD() Added TEE_AEUpdate() Added TEE_AEEncryptFinal() Added TEE_AEDecryptFinal()	

3 GLOBALPLATFORM TEE CLIENT API

Kinibi supports parts of the GlobalPlatform TEE Client API v1.0 available at:

- http://www.globalplatform.org/specificationsdevice.asp

Developers should refer to the GlobalPlatform specifications for details about the GlobalPlatform APIs and how to use these APIs.

This sections details the differences between this version of Kinibi and the GlobalPlatform specification.

3.1 HEADER FILE

The header file for the TEE Client API is "`tee_client_api.h`".

```
#include "tee_client_api.h"
```

3.2 IMPLEMENTATION NOTES

3.2.1 TEEC_InitializeContext

```
TEEC_Result TEEC_InitializeContext (
    const char*     name,
    TEEC_Context*   context)
```

This function is supported. The argument `name` is ignored.

3.2.2 TEEC_FinalizeContext

```
TEEC_Result TEEC_FinalizeContext (
    TEEC_Context*   context)
```

This function is supported.
It is a programmer error to call `TEEC_FinalizeContext` while there are still open sessions. In practice, in this case the sessions will be closed only when the process dies. That's an acceptable consequence of a programmer error.
The function will cause a segmentation fault if the parameter `context` is not valid. The function implementation does nothing if `context` is NULL.

3.2.3 TEEC_RegisterSharedMemory

```
TEEC_Result TEEC_RegisterSharedMemory (
    TEEC_Context*         context,
    TEEC_SharedMemory*    sharedMem)
```

This function is supported.
Kinibi uses a zero-copy shared memory system. The maximum buffer size is 1024 kB.

Up to Kinibi-311A, `TEEC_RegisterSharedMemory` essentially treats registered memory references exactly like temporary memory references. Since Kinibi-311B, registered memory references profit from lazy unmapping.

3.2.4 TEEC_AllocateSharedMemory

```
TEEC_Result TEEC_AllocateSharedMemory (
      TEEC_Context*         context,
      TEEC_SharedMemory*    sharedMem)
```

This function is supported.
Kinibi uses a zero-copy shared memory system. The maximum buffer size is 1024 kB.

3.2.5 TEEC_ReleaseSharedMemory

```
TEEC_Result TEEC_ReleaseSharedMemory (
      TEEC_SharedMemory*    sharedMem)
```

This function is supported.

3.2.6 TEEC_OpenSession

```
TEEC_Result TEEC_OpenSession (
      TEEC_Context*    context,
      TEEC_Session*    session,
      const TEEC_UUID* destination,
      uint32_t         connectionMethod,
      const void*      connectionData,
      TEEC_Operation*  operation,
      uint32_t*        returnOrigin)
```

This function is supported.
The destination UUID points to the filename of Kinibi System or Service Provider Trusted Application in the Kinibi registry.

connectionMethod	connectionData	API Level
TEEC_LOGIN_PUBLIC	connectionData SHOULD be NULL	3 and higher
TEEC_LOGIN_USER	connectionData SHOULD be NULL	7 and higher
TEEC_LOGIN_GROUP	4-byte connection data is either the egid of the process, its rgid or one of its supplementary groups	7 and higher
TEEC_LOGIN_APPLICATION	connectionData SHOULD be NULL	7 and higher
TEEC_LOGIN_USER_APPLICATION	connectionData SHOULD be NULL	7 and higher
TEEC_LOGIN_GROUP_APPLICATION	4-byte connection data is either the egid of the process, its rgid or one of its supplementary groups	7 and higher

3.2.7 TEEC_CloseSession

```
TEEC_Result TEEC_CloseSession (
     TEEC_Session*    session)
```

This function is supported.

3.2.8 TEEC_InvokeCommand

```
TEEC_Result TEEC_InvokeCommand (
     TEEC_Session*    session,
     uint32_t         commandID,
     TEEC_Operation*  operation,
     uint32_t*        returnOrigin)
```

This function is supported.
If the `operation` parameter references a memory region, the respective memory will be mapped to the Trusted Application for the duration of the function call.
The maximum buffer size is 1024 kB for each parameter.

3.2.9 TEEC_RequestCancellation

```
TEEC_Result TEEC_RequestCancellation (
     TEEC_Operation*  operation)
```

This function is supported.

3.3 EXTENDED API

3.3.1 TEEC_TT_RegisterPlatformContext

```
void TEEC_TT_RegisterPlatformContext(
    void                *globalContext,
    void                *localContext)
```

Since API level 9.

Let the Client Application register platform specific data for its context. This function is platform-specific.

On Android

The Client Application SHOULD call this function to provide both the current Java Virtual Machine (JVM) pointer (`JavaVM*`) as `globalContext` and an android.app.Context object as `localContext`, to be used to bind to the Android service(s) required for TEE operation. The caller must ensure that the context provided remains valid for any subsequent TEE Session. For this reason, the context provided MUST be the Application Context, returned by the Context.getApplicationContext() method, since this context is valid for the whole lifetime of an Android Client Application.

If the JVM pointer and android.app.Context are not registered before the call to either `TEEC_InitializeContext` (for GP applications) or `mcOpenDevice` (for legacy applications), some TEE functionally may not be available.

On any other platform

This function does nothing.

Parameters:

- in `globalContext`: Global context data. The content of this data is platform specific.
- in `localContext`: Local context data. The content of this data is platform specific.

3.4 KERNEL CLIENT API

Since API level 10, the GP TEE Client is available for the Linux Kernel.

In order to comply with the Linux coding rules, the functions and types have been renamed.

3.4.1 Header File

The header file for the TEE Client API is "`tee_client_api.h`".

```
#include "tee_client_api.h"
```

3.4.2 Implementation Notes

3.4.2.1 teec_initialize_context

```
u32 teec_initialize_context(
      const char *name,
      struct teec_context *context)
```

Equivalent to `TEEC_InitializeContext()`.

3.4.2.2 teec_finalize_context

```
void teec_finalize_context(
      struct teec_context *context)
```

Equivalent to `TEEC_FinalizeContext()`.

3.4.2.3 teec_register_shared_memory

```
u32 teec_register_shared_memory(
      struct teec_context *context,
      struct teec_shared_memory *shared_mem)
```

Equivalent to `TEEC_RegisterSharedMemory()`.

3.4.2.4 teec_allocate_shared_memory

```
u32 teec_allocate_shared_memory(
      struct teec_context *context,
      struct teec_shared_memory *shared_mem)
```

Equivalent to `TEEC_AllocateSharedMemory()`.

3.4.2.5 teec_release_shared_memory

```
void teec_release_shared_memory(
      struct teec_shared_memory *shared_mem)
```

Equivalent to `TEEC_ReleaseSharedMemory()`.

3.4.2.6 teec_open_session

```
u32 teec_open_session(
      struct teec_context *context,
      struct teec_session *session,
      const struct teec_uuid *destination,
      u32 connection_method,
      const void *connection_data,
      struct teec_operation *operation,
      u32 *return_origin)
```

Equivalent to `TEEC_OpenSession()`.

connection_method	connection_data	API Level
`TEEC_LOGIN_PUBLIC`	connection_data SHOULD be `NULL`	10 and higher

3.4.2.7 teec_close_session

```
void teec_close_session(
      struct teec_session *session)
```

Equivalent to `TEEC_CloseSession()`.

3.4.2.8 teec_invoke_command

```
u32 teec_invoke_command(
      struct teec_session *session,
      u32 command_id,
      struct teec_operation *operation,
      u32 *return_origin)
```

Equivalent to `TEEC_InvokeCommand()`.

3.4.2.9 teec_request_cancellation

```
Void teec_request_cancellation(
struct teec_operation *operation)
```

Equivalent to `TEEC_RequestCancellation()`.

4 GLOBALPLATFORM TEE INTERNAL API

Kinibi supports the GlobalPlatform TEE Internal API v1.1.1 available at:

- http://www.globalplatform.org/specificationsdevice.asp

Developers should refer to the GlobalPlatform specifications for details about the GlobalPlatform APIs and how to use these APIs.

This sections details the differences between this version of Kinibi and the GlobalPlatform specification.

4.1 HEADER FILE

The header file for the TEE Client API is "`tee_internal_api.h`".

```
#include "tee_internal_api.h"
```

4.2 IMPLEMENTATION NOTES

The following table lists all the functions of the TEE Internal API and indicates the supported API level for each function:

Function	Supported API level	Comment
Asymmetric		
TEE_AsymmetricDecrypt	Supported	
TEE_AsymmetricEncrypt	Supported	
TEE_AsymmetricSignDigest	Supported	
TEE_AsymmetricVerifyDigest	Supported	
Authenticated Encryption		
TEE_AEDecryptFinal	Supported since API level 11.	
TEE_AEEncryptFinal	Supported since API level 11.	
TEE_AEInit	Supported since API level 11.	
TEE_AEUpdate	Supported since API level 11.	
TEE_AEUpdateAAD	Supported since API level 11.	
Basic Arithmetic		
TEE_BigIntAdd	Supported since API level 7.	
TEE_BigIntDiv	Supported since API level 7.	
TEE_BigIntMul	Supported since API level 7.	

TEE_BigIntNeg	Supported since API level 7.	
TEE_BigIntSquare	Supported since API level 7.	
TEE_BigIntSub	Supported since API level 7.	
Cancellation		
TEE_GetCancellationFlag	Supported	
TEE_MaskCancellation	Supported	
TEE_UnmaskCancellation	Supported	
Converter		
TEE_BigIntConvertFromOctetString	Supported since API level 7.	
TEE_BigIntConvertFromS32	Supported since API level 7.	
TEE_BigIntConvertToOctetString	Supported since API level 7.	
TEE_BigIntConvertToS32	Supported since API level 7.	
Data Stream Access		
TEE_ReadObjectData	Supported	See sections 4.2.6, 5.12.5.1
TEE_SeekObjectData	Supported	See section 5.12.5.4
TEE_TruncateObjectData	Supported	See sections 4.2.6, 5.12.5.3
TEE_WriteObjectData	Supported	See sections 4.2.6, 5.12.5.2
Fast Modular Multiplication		
TEE_BigIntComputeFMM	Supported since API level 7.	
TEE_BigIntConvertFromFMM	Supported since API level 7.	
TEE_BigIntConvertToFMM	Supported since API level 7.	
Generic Object		
TEE_CloseObject	Supported	
TEE_GetObjectBufferAttribute	Supported	
TEE_GetObjectInfo	Supported	
TEE_GetObjectInfo1	Supported since API level 7.	
TEE_GetObjectValueAttribute	Supported	
TEE_RestrictObjectUsage	Supported since API level 7, but only the flag TEE_USAGE_EXTRACTABLE. Since API level 11, all flags are	See section 5.12.2.5

	supported	
TEE_RestrictObjectUsage1	Supported since API level 7, but only the flag TEE_USAGE_EXTRACTABLE. Since API level 11, all flags are supported.	See section 5.12.2.5
Generic Operation		
TEE_AllocateOperation	Supported	See section **Error! Reference source not found.**
TEE_CopyOperation	Supported since API level 11.	
TEE_FreeOperation	Supported	
TEE_GetOperationInfo	Supported	
TEE_ResetOperation	Supported since API level 11.	
TEE_SetOperationKey	Supported	
TEE_SetOperationKey2	Supported since API level 11.	
Initialization		
TEE_BigIntInit	Supported since API level 7.	
TEE_BigIntInitFMM	Supported since API level 7.	
TEE_BigIntInitFMMContext	Supported since API level 7.	
Internal Client API		
TEE_CloseTASession	Supported since API level 11.	
TEE_InvokeTACommand	Supported since API level 11.	
TEE_OpenTASession	Supported since API level 11.	
Key Derivation		
TEE_DeriveKey	Supported since API level 7.	
Logical Operation		
TEE_BigIntCmp	Supported since API level 7.	
TEE_BigIntCmpS32	Supported since API level 7.	
TEE_BigIntGetBit	Supported since API level 7.	
TEE_BigIntGetBitCount	Supported since API level 7.	
TEE_BigIntShiftRight	Supported since API level 7.	
MAC		

TEE_MACCompareFinal	Supported	
TEE_MACComputeFinal	Supported	
TEE_MACInit	Supported	
TEE_MACUpdate	Supported	
Memory Allocation and Size of Objects		
TEE_BigIntFMMContextSizeInU32	Supported since API level 7.	
TEE_BigIntFMMSizeInU32	Supported since API level 7.	
TEE_BigIntSizeInU32 (macro)	Supported since API level 7.	
Memory Management		
TEE_CheckMemoryAccessRights	Supported	
TEE_Free	Supported	
TEE_GetInstanceData	Supported	
TEE_Malloc	Supported	See section 4.2.3
TEE_MemCompare	Supported	See section 4.2.3
TEE_MemFill	Supported	See section 4.2.3
TEE_MemMove	Supported	See section 4.2.3
TEE_Realloc	Supported	
TEE_SetInstanceData	Supported	
Message Digest		
TEE_DigestDoFinal	Supported	
TEE_DigestUpdate	Supported	
Modular Arithmetic		
TEE_BigIntAddMod	Supported since API level 7.	
TEE_BigIntInvMod	Supported since API level 7.	
TEE_BigIntMod	Supported since API level 7.	
TEE_BigIntMulMod	Supported since API level 7.	
TEE_BigIntSquareMod	Supported since API level 7.	
TEE_BigIntSubMod	Supported since API level 7.	
Other Arithmetic		

TEE_BigIntComputeExtendedGcd	Supported since API level 7.	
TEE_BigIntIsProbablePrime	Supported since API level 7.	
TEE_BigIntRelativePrime	Supported since API level 7.	
Panic Function		
TEE_Panic	Supported	
Persistent Object		
TEE_CloseAndDeletePersistentObject	Supported	See sections 4.2.5, 5.12.3.4
TEE_CloseAndDeletePersistentObject1	Supported since API level 7.	See sections 4.2.5, 5.12.3.3
TEE_CreatePersistentObject	Supported	See sections 4.2.5, 5.12.3.2
TEE_OpenPersistentObject	Supported	See sections 4.2.5, 5.12.3.1
TEE_RenamePersistentObject	Supported since API level 7.	See section 5.12.3.5
Persistent Object Enumeration		
TEE_AllocatePersistentObjectEnumerator	Supported since API level 7.	See section 5.12.4.1
TEE_FreePersistentObjectEnumerator	Supported since API level 7.	See section 5.12.4.2
TEE_GetNextPersistentObject	Supported since API level 7.	See section 5.12.4.5
TEE_ResetPersistentObjectEnumerator	Supported since API level 7.	See section 5.12.4.3
TEE_StartPersistentObjectEnumerator	Supported since API level 7.	See section 5.12.4.4
Property Access		
TEE_AllocatePropertyEnumerator	Supported since API level 11.	
TEE_FreePropertyEnumerator	Supported since API level 11.	
TEE_GetNextProperty	Supported since API level 11.	
TEE_GetPropertyAsBinaryBlock	Supported	See section 4.2.1
TEE_GetPropertyAsBool	Supported	See section 4.2.1
TEE_GetPropertyAsIdentity	Supported since API level 7.	See section 4.2.1
TEE_GetPropertyAsString	Supported	See section 4.2.1
TEE_GetPropertyAsU32	Supported	See section 4.2.1
TEE_GetPropertyAsUUID	Supported	See section 4.2.1

TEE_GetPropertyName	Supported since API level 11.	
TEE_ResetPropertyEnumerator	Supported since API level 11.	
TEE_StartPropertyEnumerator	Supported since API level 11.	
Random Data Generation		
TEE_GenerateRandom	Supported	
Symmetric Cipher		
TEE_CipherDoFinal	Supported	
TEE_CipherInit	Supported	
TEE_CipherUpdate	Supported	
TA Interface		
TA_CloseSessionEntryPoint	Supported	
TA_CreateEntryPoint	Supported	
TA_DestroyEntryPoint	Supported	
TA_InvokeCommandEntryPoint	Supported	
TA_OpenSessionEntryPoint	Supported	
Time		
TEE_GetREETime	Supported since API level 7.	See section 4.2.8
TEE_GetSystemTime	Supported since API level 7.	Monotonic since API level 7.
TEE_GetTAPersistentTime	Supported since API level 10.	See section 4.2.8
TEE_SetTAPersistentTime	Supported since API level 10.	See section 4.2.8
TEE_Wait	Supported since API level 11.	
Transient Object		
TEE_AllocateTransientObject	Supported	See section 4.2.4
TEE_CopyObjectAttributes	Supported	
TEE_CopyObjectAttributes1	Supported since API level 7.	
TEE_FreeTransientObject	Supported	
TEE_GenerateKey	Supported	
TEE_InitRefAttribute	Supported	
TEE_InitValueAttribute	Supported	

TEE_PopulateTransientObject	Supported	
TEE_ResetTransientObject	Supported	

4.2.1 Properties

A property is an immutable value identified by a name, which is a Unicode string. The property value can be retrieved in a variety of formats: Unicode string, binary block, 32-bit integer, Boolean, and Identity.

Kinibi supports the full Properties API:

- TEE, TA and client properties are supported.
- Property conversion to string is supported since API level 11.
- Properties can be enumerated since API level 11

The property set is passed to each function in a pseudo-handle parameter. The following table lists the defined and supported property sets:

Pseudo-Handle	Meaning	Supported
TEE_PROPSET_CURRENT_TA	The configuration properties for the current Trusted Application.	Supported
TEE_PROPSET_CURRENT_CLIENT	The properties of the current client.	Supported since API level 7.
TEE_PROPSET_TEE_IMPLEMENTATION	The properties of the TEE Implementation itself	Supported

4.2.1.1 Implementation properties

For `TEE_PROPSET_TEE_IMPLEMENTATION` property set the following properties are supported:

Property name	Property type	Property value
gpd.tee.apiversion	String	“1.1” from Kinibi-400 “1.0” before
gpd.tee.description	String	$Buildtag, e.g. “t-base-Arndale-Android-400A-20160501_011308_9060_36620” from Kinibi-400 “<t-base” before
gpd.tee.deviceID	UUID	SUID obtained by tlApiGetSuid function
gpd.tee.systemTime.protectionLevel	UINT32	100 Supported since API level 10.

gpd.tee.TAPersistentTime.protectionLevel	UINT32	100 Supported since API level 11.
gpd.tee.arith.maxBigIntSize	UINT32	0xFFFF. Supported since API level 6.
gpd.tee.cryptography.ecc	Boolean	1 Supported since API level 6.
gpd.tee.trustedStorage. antiRollback.protectionLevel	UINT32	100 or 1000 depending on the MobiConfig by the OEM (automatically set to 1000 if RPMB SWd driver configured) Supported since API level 10.
gpd.tee.trustedos.implementation.version	String	“400A” Supported since API level 11.
gpd.tee.trustedos.implementation.binaryversion	String	$SVN_REVISION from buildTag. e.g. 36620 Supported since API level 11.
gpd.tee.trustedos.manufacturer	String	Trustonic
gpd.tee.firmware.implementation.version	String	Platform-specific value Supported since API level 11.
gpd.tee.firmware.implementation.binaryversion	String	Platform-specific value Supported since API level 11.
gpd.tee.firmware.manufacturer	String	Integration-specific value Supported since API level 11.
gpd.tee.modelID	UUID	SIP:SOC from SUID 16 byte-SUID with serial number set to 0. That is 8 byte zero at the end.
gpd.tee.tui.securityIndicator	Boolean	False Supported since API level 11.
gpd.tee.debug.DLM_available	Boolean	False Supported since API level 11.
gpd.tee.dbg_dlm.data_available	UINT32	0

		Supported since API level 11.
gpd.tee.dbg_pmr.data_available	UINT32	0 Supported since API level 11.
gpd.tee.debug.debug_unlock_properties	Boolean	False Supported since API level 11.

4.2.1.2 Trusted Application properties

For `TEE_PROPSET_CURRENT_TA` property set the following properties are supported:

Property name	Property type	Property value
gpd.ta.appID	UUID	GP UUID of a given TA
gpd.ta.singleInstance	Boolean	As specified in TA's properties.xml Interpreted as of Kinibi-400
gpd.ta.multiSession	Boolean	As specified in TA's properties.xml Interpreted as of Kinibi-400
gpd.ta.instanceKeepAlive	Boolean	As specified in TA's properties.xml Interpreted as of Kinibi-400
gpd.ta.dataSize	UINT32	Heap area size of a given TA
gpd.ta.stackSize	UINT32	Stack area size of a given TA
gpd.ta.parentSD	UUID	Parent SD according to TMF spec. Supported since API level 5
gpd.ta.version	String	As specified in TA's properties.xml Supported since API level 11
gpd.ta.description	String	As specified in TA's properties.xml Supported since API level 11
gpd.ta.session.ID	UUID	12 byte: zero. 2 byte: Session Handle 2 byte: Thread ID A client shall use appID to make full ID.
gpd.ta.dbg_dlm.data_available	UINT32	0
gpd.ta.dbg_pmr.data_available	UINT32	0

4.2.1.3 Client properties

For `TEE_PROPSET_CURRENT_CLIENT` property set the following properties are supported:

Property name	Property type	Property value
gpd.client.identity	Identity	Identity of the current REE client. Supported since API level 7.
com.trustonic.client.virtualMachine	String	String containing the Virtual Machine identifier

4.2.1.3.1 gpd.client.identity

Identity.login: uint32_t

- As specified in connectionMethod parameter to TEEC_OpenSession().

Identity.uuid: TEE_UUID

- Depends on connectionMethod and connectionData.
- Depends on REE integration.
- Implementation on Linux: SHA1 hash of a seed. Seed fields:
  - UID on `LOGIN_USER` and `LOGIN_USER_APPLICATION`.
  - GID on `LOGIN_GROUP` and `LOGIN_GROUP_APPLICATION`.
  - File path of the process executable on `LOGIN_APPLICATION`, `LOGIN_USER_APPLICATION and LOGIN_GROUP_APPLICATION`.
- Implementation on Android:
  - If the Client Application is a native binary; same as Linux implementation.
  - If the Client Application lives inside an Android application: truncated SHA1 hash of Application package name concatenated with the public key of application signing key.

    Android code used to generate SHA1 hash:

```
Signature[] signatures = packageInfo.signatures;
byte[] cert = signatures[0].toByteArray();
byte[] infos = null;
infos = new byte[packageName.getBytes().length + cert.length];
System.arraycopy(packageName.getBytes(), 0, infos, 0,
                 packageName.getBytes().length);
System.arraycopy(cert, 0, infos, packageName.getBytes().length, cert.length);
MessageDigest digest = MessageDigest.getInstance("SHA1");
digest.update(infos);
truncatedHash = Arrays.copyOfRange(digest.digest(), 0, 16);
```

4.2.2 TEE Capabilities

In addition to properties defined by GlobalPlatform, Kinibi supports the following proprietary properties. These are available from API level 11.

Property name	Property type	Property value
com.trustonic.tee.isa.name	String	“ARM” ISA stands for instruction set architecture and a field like this is required by GP Audit interface.

com.trustonic.tee.isa.arm.armv7	Boolean	False
com.trustonic.tee.isa.arm.armv8	Boolean	True if platform is ARMv8 Aarch32 or ARMv8 Aarch32
com.trustonic.tee.isa.arm.aarch32	Boolean	True if platform support ARMv8 Aarch32 TA/SWd Driver
com.trustonic.tee.isa.arm.aarch64	Boolean	True if platform support ARMv8 Aarch64 TA/Swd Driver.
com.trustonic.tee.isa.arm.lpae	Boolean	Are physical memory > 4GB supported by Kinibi? Dynamic at build time
com.trustonic.tee.isa.arm.neon	Boolean	True if Kinibi supports NEON and Hardware Floating Point for TA Dynamic at build time
com.trustonic.tee.isa.arm.crypto	Boolean	True if Kinibi supports AARCH32 crypto instructions for TA Dynamic at build time
com.trustonic.tee.trustedos.name	String	Kinibi-400A
com.trustonic.tee.trustedos.version	UINT32	400A
com.trustonic.tee.integration.name	String	Platform as in buildTag. e.g. EXYNOS64
com.trustonic.tee.integration.version	UINT32	The version of the SIP release, aka V001, V002 from buildTag. e.g. 1
com.trustonic.tee.trustedos.api_level	UINT32	11
com.trustonic.tee.mcVersionInfo As a replacement for: tlApiGetMobiCoreVersion (mcVersionInfo_t *info);	binary	struct { char productId[MC_PRODUCT_ID_LEN]; uint32_t versionMci uint32_t versionSo uint32_t versionMclf uint32_t versionContainer uint32_t versionMcConfig

		uint32_t versionTlApi uint32_t versionDrApi uint32_t versionCmp} mcVersionInfo_t
com.trustonic.tee.build_time	String	YYYY/MM/DD, HH:MM:SS from buildTag, e.g. 2016/05/01, 01:13:08
com.trustonic.tee.config_time	String	YYYY/MM/DD, HH:MM:SS from buildTag, e.g. 2016/05/11, 15:42:59
com.trustonic.tee.memory.free	UINT32	Global free memory, available for loading TA code, data and heap; in byte, page-size granularity, dynamic e.g. 1048576
com.trustonic.tee.memory.size	UINT32	Boot-time available memory for whole TEE; in byte, page-size granularity, boot-time dynamic e.g. 6291456
com.trustonic.tee.trustedapps.free	UINT32	Available slots for running TAs, e.g. 5.
com.trustonic.tee.tui.available	Bool	True if TUI is available Dynamic, will try to contact TUI driver
com.trustonic.tee.tui.screenInfo	binary	Get screen characteristics in the legacy format dynamic, will call tlApiTuiGetScreenInfo() 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,

4.2.3 Memory Management

4.2.3.1 TEE_Malloc

```
DECLARE_TRUSTED_APPLICATION_MAIN_HEAP( uint32_t staticSize);

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

To use TEE_Malloc, the developer must define the static size of the heap in the TA code through the DECLARE_TRUSTED_APPLICATION_MAIN_HEAP() macro. Otherwise the `TEE_Malloc` function returns NULL. The heap size is limited by the size of the TA address space. The `hint` parameter has to be 0, indicating a block of memory filled with zeros.

4.2.3.2 TEE_MemMove

The SDK uses the memmove() implementation of the compiler.

4.2.3.3 TEE_MemCompare

The SDK uses the memcmp() implementation of the compiler.

4.2.3.4 TEE_MemFill

The SDK uses the memset() implementation of the compiler.

4.2.4 Transient Objects

4.2.4.1 TEE_AllocateTransientObject

The following table lists the supported object types.

Table 1: TEE_AllocateTransientObject: Supported Object Sizes

Object Type	Possible Object Sizes
TEE_TYPE_AES	128 o 256 bits 192 since API level 11
TEE_TYPE_DES	Always 56 for API level < 11, since then always 64 bits including the parity bits. This gives an effective key size of 56 bits
TEE_TYPE_DES3	112 for API level < 11, then 128 or 192 bits including the parity bits. This gives effective key sizes of 112 or 168 bits

TEE_TYPE_HMAC_MD5	Between 64 and 512 bits, multiple of 8 bits. Supported since API level 7.
TEE_TYPE_HMAC_SHA1	Between 80 and 512 bits, multiple of 8 bits
TEE_TYPE_HMAC_SHA224	Between 112 and 512 bits, multiple of 8 bits. Supported since API level 7.
TEE_TYPE_HMAC_SHA256	Between 192 and 1024 bits, multiple of 8 bits
TEE_TYPE_HMAC_SHA384	Between 256 and 1024 bits, multiple of 8 bits. Supported since API level7.
TEE_TYPE_HMAC_SHA512	Between 256 and 1024 bits, multiple of 8 bits. Supported since API level7.
TEE_TYPE_RSA_PUBLIC_KEY	Key size up to 2048 bits
TEE_TYPE_RSA_KEYPAIR	Key size up to 2048 bits
TEE_TYPE_DSA_PUBLIC_KEY	Supported since API level 7. Depends on Algorithm: ALG_DSA_SHA1: Between 512 and 1024 bits, multiple of 64 bits ALG_DSA_SHA224: 2048 bits ALG_DSA_SHA256: 2048 or 3072 bits
TEE_TYPE_DSA_KEYPAIR	Same as for DSA public key size.
TEE_TYPE_DH_KEYPAIR	Supported since API level 7. From 256 to 2048 bits

TEE_TYPE_GENERIC_SECRET	Supported since API level 7. Multiple of 8 bits, up to 4096 bits. This type is intended for secret data that is not directly used as a key in a cryptographic operation, but participates in a key derivation.
TEE_TYPE_ECDSA_PUBLIC_KEY TEE_TYPE_ECDSA_KEYPAIR	Supported since API level 7. All the following curve sizes are supported: • TEE_ECC_CURVE_NIST_P192 • TEE_ECC_CURVE_NIST_P224 • TEE_ECC_CURVE_NIST_P256 • TEE_ECC_CURVE_NIST_P384 • TEE_ECC_CURVE_NIST_P521
TEE_TYPE_ECDH_PUBLIC_KEY TEE_TYPE_ECDH_KEYPAIR	Supported since API level 7. All the following curve sizes are supported: • TEE_ECC_CURVE_NIST_P192 • TEE_ECC_CURVE_NIST_P224 • TEE_ECC_CURVE_NIST_P256 • TEE_ECC_CURVE_NIST_P384 • TEE_ECC_CURVE_NIST_P521

4.2.5 Persistent Objects

The persistent objects API described in §5.12 is available to Trusted Applications compiled against the GlobalPlatform APIs.

Since API level 7, the persistent objects API is available in both tlApi and the GlobalPlatform API. See §5.12 for the documentation of the persistent objects API at level 7 and above.

This section documents limitations applicable in Kinibi versions 300–303, which implement API levels ≤6. These limitations do not apply in Kinibi 310 and above, regardless of the API version requested by the application.

4.2.5.1 TEE_OpenPersistentObject

The completion of this function depends on two elements:

- The heap size;
- The size of the buffer used for the Driver Communication Interface;

If one of these elements is full, a TEE_ERROR_OUT_OF_MEMORY is returned.

Currently, the access permission setup for persistent storage object is not implemented. The `flags` parameter is ignored by the implementation.

4.2.5.2 TEE_CreatePersistentObject

The completion of this function depends on two elements:

- The heap size;
- The size of the buffer used for the Driver Communication Interface;

If one of these elements is full, a TEE_ERROR_OUT_OF_MEMORY is returned.

Currently, the access permission setup for persistent storage object is not implemented. Only the flag TEE_DATA_FLAG_EXCLUSIVE in the `flags` parameter is used by the implementation.

4.2.5.3 TEE_CloseAndDeletePersistentObject

This function does not destroy all opened session associated with the underlined storage file but the one passed as argument `object`. In case multiple copy of storage file are opened, the developer should consider the possibility of concurrency and keep their file object up-to-date.

4.2.6 Data Stream Access

4.2.6.1 TEE_ReadObjectData, TEE_WriteObjectData, TEE_TruncateObjectData

The completion of these functions depends on two elements:

- The heap size;
- The size of the buffer used for the Driver Communication Interface;

If one of these elements is full, a TEE_ERROR_OUT_OF_MEMORY is returned.

4.2.6.1.1 Currently, the access permission setup for persistent storage object is not implemented.

4.2.7 Generic Operation Functions

4.2.7.1 TEE_AllocateOperation

The following table lists the supported algorithms.

Table 2: TEE_AllocateOperation: Supported Modes

Algorithm	Possible Modes
TEE_ALG_AES_ECB_NOPAD TEE_ALG_AES_CBC_NOPAD TEE_ALG_AES_CTR	TEE_MODE_ENCRYPT TEE_MODE_DECRYPT
TEE_ALG_AES_CTS (since API level 11) TEE_ALG_AES_XTS (since API level 11) TEE_ALG_AES_CCM (since API level 11) TEE_ALG_AES_GCM (since API level 11)	TEE_MODE_ENCRYPT TEE_MODE_DECRYPT
TEE_ALG_DES_ECB_NOPAD TEE_ALG_DES_CBC_NOPAD TEE_ALG_DES3_ECB_NOPAD TEE_ALG_DES3_CBC_NOPAD	TEE_MODE_ENCRYPT TEE_MODE_DECRYPT
TEE_ALG_DES_CBC_MAC_NOPAD (since API level 11) TEE_ALG_AES_CBC_MAC_NOPAD (since API level 11)	TEE_MODE_ENCRYPT TEE_MODE_DECRYPT

TEE_ALG_AES_CBC_MAC_PKCS5 (since API level 11) TEE_ALG_AES_CMAC (since API level 11) TEE_ALG_DES_CBC_MAC_PKCS5 (since API level 11) TEE_ALG_DES3_CBC_MAC_NOPAD (since API level 11) TEE_ALG_DES3_CBC_MAC_PKCS5 (since API level 11)	
TEE_ALG_RSASSA_PKCS1_V1_5_MD5 (since API level 7) TEE_ALG_RSASSA_PKCS1_V1_5_SHA1 TEE_ALG_RSASSA_PKCS1_V1_5_SHA224 (since API level 7) TEE_ALG_RSASSA_PKCS1_V1_5_SHA256 (since API level 7) TEE_ALG_RSASSA_PKCS1_V1_5_SHA384 (since API level 7) TEE_ALG_RSASSA_PKCS1_V1_5_SHA512 (since API level 7) TEE_ALG_RSASSA_PKCS1_PSS_MGF1_SHA1 TEE_ALG_RSASSA_PKCS1_PSS_MGF1_SHA224 (since API level 7) TEE_ALG_RSASSA_PKCS1_PSS_MGF1_SHA256 (since API level 7) TEE_ALG_RSASSA_PKCS1_PSS_MGF1_SHA384 (since API level 7) TEE_ALG_RSASSA_PKCS1_PSS_MGF1_SHA512 (since API level 7) TEE_ALG_DSA_SHA1 (since API level 7) TEE_ALG_DSA_SHA224 (since API level 7) TEE_ALG_DSA_SHA256 (since API level 7) TEE_ALG_ECDSA (since API level 7) TEE_ALG_ECDSA_SHA1 (since API level 11) TEE_ALG_ECDSA_SHA224 (since API level 11) TEE_ALG_ECDSA_SHA256 (since API level 11) TEE_ALG_ECDSA_SHA384 (since API level 11) TEE_ALG_ECDSA_SHA512 (since API level 11)	TEE_MODE_SIGN TEE_MODE_VERIFY
TEE_ALG_RSAES_PKCS1_V1_5 TEE_ALG_RSAES_PKCS1_OAEP_MGF1_SHA1(since API level 7) TEE_ALG_RSAES_PKCS1_OAEP_MGF1_SHA224(since API level 7) TEE_ALG_RSAES_PKCS1_OAEP_MGF1_SHA256(since API level 7) TEE_ALG_RSAES_PKCS1_OAEP_MGF1_SHA384(since API level 7) TEE_ALG_RSAES_PKCS1_OAEP_MGF1_SHA512(since API level 7) TEE_ALG_RSA_NOPAD	TEE_MODE_ENCRYPT TEE_MODE_DECRYPT

Algorithm	**Possible Modes**
TEE_ALG_DH_DERIVE_SHARED_SECRET (since API level 7) TEE_ALG_ECDH_DERIVE_SHARED_SECRET (since API level 11)	TEE_MODE_DERIVE
TEE_ALG_MD5 (since API level 7)	TEE_MODE_DIGEST

TEE_ALG_SHA1 TEE_ALG_SHA224 (since API level 7) TEE_ALG_SHA256 TEE_ALG_SHA384 (since API level 7) TEE_ALG_SHA512 (since API level 7)	
TEE_ALG_HMAC_MD5 (since API level 7) TEE_ALG_HMAC_SHA1 TEE_ALG_HMAC_SHA224 (since API level 7)TEE_ALG_HMAC_SHA256 TEE_ALG_HMAC_SHA384 (since API level 7) TEE_ALG_HMAC_SHA512 (since API level 7)	TEE_MODE_MAC

4.2.7.2 TEE_GetOperationInfoMultiple

Supported since API level 11.

The following table lists the supported operation states.

Table 3: TEE_GetOperationInfoMultiple: Supported States

Operation State
TEE_OPERATION_STATE_INITIAL TEE_OPERATION_STATE_ACTIVE

4.2.8 Time API

Using the function `TEE_GetREETime()` the TA can query the REE time. This field is updated whenever the REE kernel driver switches into the TEE.

Using the function `TEE_GetSystemTime()` the TA can query a secure timestamp. The System time is based on the REE time but guarantees monotonicity. Therefore the system implements a counter mode that is activated in three cases:

- When REE time is 0,
- When last System time is above year 2072,
- When REE time is smaller than System time.

Otherwise, the next System time is calculated as max(REE time, last System time). Kinibi implements the minimum reliability, `gpd.tee.systemTime.protectionLevel = 100`. The time is not maintained over a device reset.

The functions TEE_GetTAPersistentTime and TEE_SetTAPersistentTime can be used by the TA to set and retrieve a timestamp in trusted storage. They allow a TA to apply a device-reset offset to the System Time.

4.3 EXTENDED API

The following functions provide extended features and can be called from Trusted Applications developed with the TEE Internal API

4.3.1 Logging

4.3.1.1 TEE_LogPrintf

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

This proprietary function allows writing formatted traces for logging purposes.

4.3.1.2 TEE_LogvPrintf

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

This proprietary function allows writing formatted traces for logging purposes.

4.3.1.3 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.

4.3.1.4 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.

4.3.1.5 Supported formats

Kinibi internal printf function only support a subset of GNU printf function:

- %s: Print a ‘\0’ terminated string. NULL value emits "<NULL>".
- ‘%d’, ‘%i’: Print an integer as a signed decimal number.
- ‘%u’: Print an integer as an unsigned decimal number.
- ‘%x’, ‘%X’: Print an integer as an unsigned hexadecimal number.
- %llx: Print a 64bit value as unsigned hexadecimal number.
- %p: pointer (hex with fixed width of 8)
- %% outputs single %
- %s, %x, %d, and %u support width (example %5s). Width is interpreted as minimum number of characters.

4.3.2 TEE_TBase_UnwrapObject

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

This function is equivalent to `tlApiUnwrapObject()` with flags set to `TLAPI_UNWRAP_PERMIT_DELEGATED`.

4.3.3 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`.

4.3.4 TEE_TBase_AddEntropy

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

This function is equivalent to `tlApiAddEntropy()`.

4.3.5 Trusted User Interface API

The Trusted User Interface API is also available for Trusted Applications developed with GlobalPlatform APIs.

The following functions are the same as the tlApi, only the names of the functions change:

```
TEE_Result TEE_TBase_TUI_GetScreenInfo(
     tlApiTuiScreenInfo_ptr screenInfo)

TEE_Result TEE_TBase_TUI_OpenSession(void)

TEE_Result TEE_TBase_TUI_CloseSession(void)

TEE_Result TEE_TBase_TUI_SetImage (
     tlApiTuiImage_ptr image,
     tlApiTuiCoordinates_t coordinates)

#if TBASE_API_LEVEL >= 6
TEE_Result TEE_TBase_TUI_DrawBuffer(
    tlApiTuiGraphicContext_t graphicContext,
    const tlApiTuiRawImage_t *pRawImage,
    int32_t dx,
    int32_t dy)
```

```
TEE_Result TEE_TBase_TUI_DrawImage(
    tlApiTuiGraphicContext_t graphicContext,
    const tlApiTuiImage_t *pImage,
    int32_t dx,
    int32_t dy)

TEE_Result TEE_TBase_TUI_FlipFramebuffers(void)

TEE_Result TEE_TBase_TUI_SetImage(
    tlApiTuiImage_ptr image,
    tlApiTuiCoordinates_t coordinates)

TEE_Result TEE_TBase_TUI_GetScreenContext(
        tlApiTuiGraphicContext_t *pGraphicContext)

TEE_Result TEE_TBase_TUI_FillRectangle(
    tlApiTuiGraphicContext_t graphicContext,
    const tlApiTuiRectangle_t *pRectangle,
    uint32_t color)

TEE_Result TEE_TBase_TUI_GetImageInfo(
    const tlApiTuiImage_t *pImage,
    tlApiTuiImageInfo_t *pInfo)

TEE_Result TEE_TBase_TUI_DecodeImage(
    const tlApiTuiImage_t *pImage,
    tlApiTuiRawImage_t *pRaw)

TEE_Result TEE_TBase_TUI_GetClippingRectangle(
    tlApiTuiGraphicContext_t graphicContext,
    tlApiTuiRectangle_t *pRectangle)

TEE_TBase_TUI_SetClippingRectangle(
    tlApiTuiGraphicContext_t graphicContext,
    const tlApiTuiRectangle_t *pRectangle)

#endif /* TBASE_API_LEVEL >= 6 */
```

The following function is the equivalent of the the tlApi function `tlApiTuiGetTouchEvent`, except that it is a blocking call that returns only when an event TUI event occurred.

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

Please refer to section 5.8 for details about the Trusted User Interface API.

4.3.6 DRM API

The DRM API is also available for Trusted Applications developed with GlobalPlatform APIs.

The API is the same than the tlApi, only the names of the functions change:

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

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

TEE_Result TEE_TBase_DRM_CloseSession (
     uint8_t sHandle);

TEE_Result TEE_TBase_DRM_CheckLink (
     uint8_t sHandle,
     tlApiDrmLink_t link)

#if TBASE_API_LEVEL >= 7
TEE_Result TEE_TBase_DRM_ProcessContentEx(
        uint8_t                        sHandle,
        tlApiDRM_headerV1              *DRM_header);
#endif /* TBASE_API_LEVEL >= 7 */
```

Please refer to section 5.9 for details about the DRM API.

5 tlApi

The tlApi is the Kinibi legacy API for developing Trusted Applications

5.1 HEADER FILE

The header file for the tlApi is "`TlApi.h`".

```
#include "TlApi.h"
```

5.2 GENERIC CONSTANTS

5.2.1 Error Codes

Constant Name	Value	Definition
TLAPI_OK	0x00000000	TlApi return values. Returns on successful execution of a function.
E_TLAPI_NOT_IMPLEMENTED	0x00000101	Function not yet implemented.
E_TLAPI_UNKNOWN	0x00000102	Unknown error during TlApi usage.

	0x00000103	
E_TLAPI_UNKNOWN_FUNCTION	0x00000104	Function not known.
E_TLAPI_INVALID_INPUT	0x00000105	Input data is invalid.
E_TLAPI_INVALID_RANGE	0x00000106	If address/pointer has invalid range
E_TLAPI_BUFFER_TOO_SMALL	0x00000107	Buffer is too small.
E_TLAPI_INVALID_TIMEOUT	0x00000108	The chosen timeout value was invalid.
E_TLAPI_TIMEOUT	0x00000109	Timeout expired.
E_TLAPI_NULL_POINTER	0x0000010a	Null pointer.
E_TLAPI_INVALID_PARAMETER	0x0000010b	Invalid parameter.
E_TLAPI_UNMAPPED_BUFFER	0x0000010c	Specified buffer is not entirely mapped in Trusted Application address space.
E_TLAPI_UNALIGNED_POINTER	0x0000010d	Passed pointer is not word-aligned.
E_TLAPI_COM_WAIT	0x0000010e	Waiting for a notification failed.
E_TLAPI_COM_ERROR	0x0000010f	Internal communication error.
E_TLAPI_BUFFER_INCORRECT_TYPE	0x00000110	Passed buffer is not of correct type.
E_TLAPI_OUT_OF_MEMORY	0x00000111	Not enough memory to perform the operation. Since TBASE_API_LEVEL=3
E_TLAPI_DRV_UNKNOWN	0x00000201	Unspecified driver error.
E_TLAPI_DRV_NO_SUCH_DRIVER	0x00000202	Unknown driver, bad driver ID.
E_TLAPI_DRV_INVALID_PARAMETERS	0x00000203	Driver parameters invalid.
E_TLAPI_CR_HANDLE_INVALID	0x00000301	No running session is associated with this handle value or caller has not permission to access the session associated with this handle value.
E_TLAPI_CR_ALGORITHM_NOT_AVAILABLE	0x00000302	Algorithm is not available for the caller.
E_TLAPI_CR_ALGORITHM_NOT_SUPPORTED	0x00000303	The intended algorithm usage is not supported.
E_TLAPI_CR_WRONG_INPUT_SIZE	0x00000304	Input data (message or data to be encrypted or decrypted) is too short or too long.
E_TLAPI_CR_WRONG_OUPUT_SIZE	0x00000305	Provided Output buffer is of wrong size.
E_TLAPI_CR_INCONSISTENT_DATA	0x00000306	Inconsistency of data was determined.

E_TLAPI_CR_OUT_OF_RESOURCES	0x00000307	No more additional resources available.
E_TLAPI_CR_DRBG_ERROR	0x00000308	Error from DRBG. Since TBASE_API_LEVEL=5
E_TLAPI_SO_WRONG_CONTEXT	0x00000401	Illegal (unsupported) secure object context.
~~E_TLAPI_SO_WRONG_PADDING~~	~~0x00000402~~	~~Wrong padding of secure object.~~ This error code was not returned by the API and has been removed.
E_TLAPI_SO_WRONG_CHECKSUM	0x00000403	Wrong secure object checksum
E_TLAPI_SO_CONTEXT_KEY_FAILED	0x00000404	Derivation of context key failed.
E_TLAPI_SO_WRONG_LIFETIME	0x00000405	Illegal (unsupported) secure object lifetime.
E_TLAPI_SO_WRONG_VERSION	0x00000406	Illegal (unsupported) secure object version.
E_TLAPI_SO_WRONG_TYPE	0x00000407	Illegal (unsupported) secure object type.
E_TLAPI_SO_DELEGATED_NOT_PERMITTED	0x00000408	Unwrap does not permit delegated objects.
E_TLAPI_SO_CONTEXT_NOT_PERMITTED	0x00000409	Unwrap does not permit this context.

Table 1: Trusted Application API Error Codes

5.2.2 Macros

Macro Name	Definition
TLAPI_ERROR_DETAIL(ecode)	Get detail part of error code.
TLAPI_ERROR_MAJOR(ecode)	Get MAJOR part of error code.
TLAPI_ERROR_MAJOR_CODE(ecode)	Get MAJOR_CODE part of error code.
TLAPI_ERROR_MAJOR_COMPONENT(ecode)	Get MAJOR_COMPONENT part of error code.
TLAPI_INFINITE_TIMEOUT ((uint32_t)(-1))	Wait infinite time for a response of the MC.
TLAPI_NO_TIMEOUT	Do not wait for a response of the MC.
MC_SO_SIZE_F22(plainLen, encryptedLen)	Calculates the total size of a secure object. plainLen is the length of plain text part within secure object. encryptedLen is the length of encrypted part within secure object (excl. hash, padding). Returns the total (gross) size of the secure object or 0 if given parameters are illegal or would lead to a secure object of invalid size.

Table 2: Trusted Application API Macros

5.2.3 Other Constants

Constant Name	Value	Definition
MC_SO21_HASH_SIZE	24	Size of hash used for secure object v2.1.
MC_SO21_RND_SIZE	9	Size of random used for secure objects v2.1.
MC_SO22_HASH_SIZE	32	Size of hash used for secure object v2.2.
MC_SO22_RND_SIZE	16	Size of random used for secure objects v2.2.
MC_SO_ENCRYPT_BLOCK_SIZE	16	Block size of encryption algorithm used for secure objects.
MC_SO_HASH_SIZE	32	Size of hash used for secure objects v2.
MC_SO_MAX_PADDING_SIZE	(MC_SO_ENCRYPT_BLOCK_SIZE)	Maximum number of ISO padding bytes.
MC_SO_PAYLOAD_MAX_SIZE	1000000	Maximum size of the payload (plain length + encrypted length) of a secure object.

Table 3: Other Constants

5.3 GENERIC TYPES

5.3.1 Enumerations and types

5.3.1.1 mcSoContext_t

Secure object context.

A context defines which key to use to encrypt/decrypt a secure object.

Enumeration:

- MC_SO_CONTEXT_TLT Trusted Application context.
- MC_SO_CONTEXT_SP Service provider context.
- MC_SO_CONTEXT_DEVICE Device context.
- MC_SO_CONTEXT_DUMMY Dummy to ensure that enum is 32 bit wide.

5.3.1.2 mcSoLifeTime_t

Secure object lifetime.

A lifetime defines how long a secure object is valid.

Enumeration:

- MC_SO_LIFETIME_PERMANENT SO does not expire.
- MC_SO_LIFETIME_POWERCYCLE SO expires on reboot (coldboot).
- MC_SO_LIFETIME_SESSION SO expires when Trusted Application is closed.

- MC_SO_LIFETIME_DUMMY Dummy to ensure that enum is 32 bit wide.

5.3.1.3 mcSoType_t

Secure object type.

Enumeration:

- MC_SO_TYPE_REGULAR Regular secure object.
- MC_SO_TYPE_DUMMY Dummy to ensure that enum is 32 bit wide.

5.3.1.4 mcSpid_t

```
typedef uint32_t mcSpid_t ;
```

Service provider Identifier type.

Constant Name	Value	Definition
MC_SPID_FREE	0xFFFFFFFF	SPID value used as free marker in root containers.
MC_SPID_RESERVED	0	Reserved UUID.
MC_SPID_SYSTEM	0xFFFFFFFE	UUID for system applications.

Table 4 mcSpid_t Constants

5.3.2 mcUuid_t - Universally Unique Identifier.

Universally Unique Identifier (UUID) according to ISO/IEC 11578.

```
typedef struct {
    uint8_t value[16]; /**< Value of the UUID. */
} mcUuid_t, *mcUuid_ptr;
```

This structure can be initialized with 3 different values:

Constant Name	Value	Definition
MC_UUID_FREE	{ 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF }	UUID value used as free marker in service provider containers.
MC_UUID_RESERVED	{ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }	Reserved UUID.
MC_UUID_SYSTEM	{ 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFE }	UUID for system applications.
MC_SUID_LEN	16	Length of SUID.

Table 5: mcUuid_t Constants

5.3.3 mcSoHeader_t

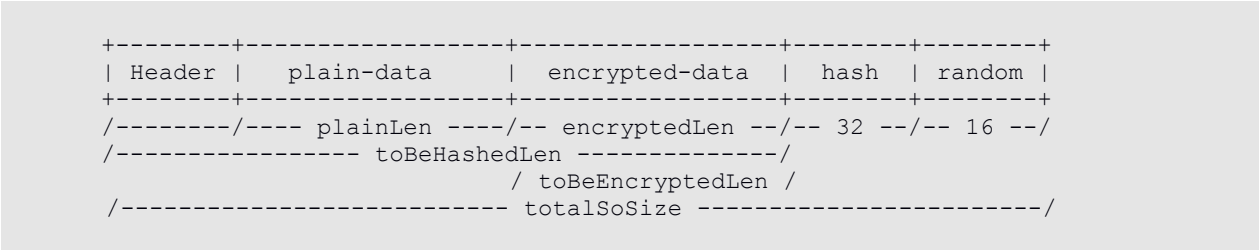
```
typdef struct
    uint32_t type;
    uint32_t version;
    mcSoContext_t context;
    mcSoLifeTime_t lifetime;
    tlApiSpTrustletId_t producer;
    uint32_t plainLen;
    uint32_t encryptedLen;
```

```
} mcSoHeader_t;
```

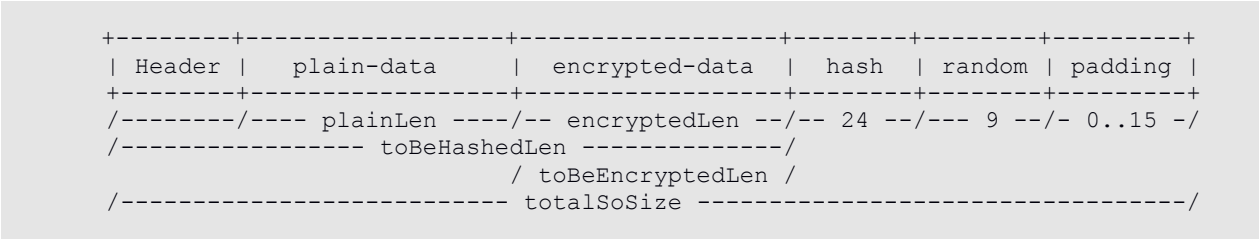
The fields of the structure are:

- type: Type of secure object.
- version: Secure object version.
- context: Secure object context.
- lifetime: Secure object lifetime.
- producer: Producer Trusted Application id.
- plainLen: Length of unencrypted user data (after the header).
- encryptedLen: Length of encrypted user data (after unencrypted data, excl. checksum and excl. padding bytes).

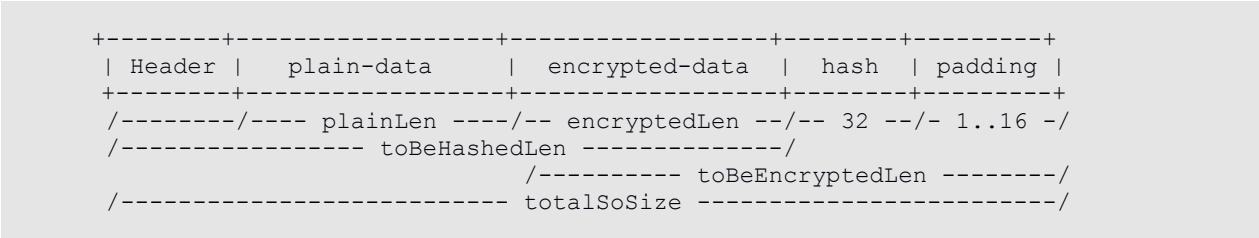
A secure object header introduces a secure object. Layout of a secure object:



Secure object header in v2.1 is:



Secure object header in v2.0 is:



5.3.4 mcSuid_t

```
typedef struct {
    uint32_t    sipId;
    suidData_t  suidData;
} mcSuid_t;
```

Universally Unique Identifier (UUID) according to ISO/IEC 11578.

The fields of the structure are:

- sipId : Silicon Provider ID to be set during build

- uint8_t suidData [12]: Value of the UUID.

5.3.5 suidData_t

```
/** Length of SUID. */
#define MC_SUID_LEN    16

typedef struct {
    uint8_t data[MC_SUID_LEN - sizeof(uint32_t)];
} suidData_t;
```

Platform specific device identifier (serial number of the chip).

5.3.6 tlApiRsaKey_t

```
typedef struct {
    tlApiLongInt_t  exponent;
    tlApiLongInt_t  modulus;
    tlApiLongInt_t  privateExponent;
    struct {
        tlApiLongInt_t Q;
        tlApiLongInt_t P;
        tlApiLongInt_t DQ;
        tlApiLongInt_t DP;
        tlApiLongInt_t Qinv;
    } privateCrtKey;

} tlApiRsaKey_t;
```

Maximum Key size	API level
2048 bit	1,2,3
4096 bit	4

The fields of the structure are:

- exponent: Pointer to public exponent.
- modulus: Modulus.
- privateExponent: Private exponent (if private key present).
- Q: Pointer to prime q (if private CRT key present).
- P: Pointer to prime p, with p>q (if private CRT key present).
- DQ: Pointer to DQ := d mod (Q-1) (if private CRT key present).
- DP: Pointer to DP := d mod (P-1) (if private CRT key present).
- Qinv: Pointer to Q inverse Qinv := q^{-1} mod p (if private CRT key present).

Note that p needs to be bigger than q as of RSA CRT requirements such that Qinv is unique.

5.3.7 tlApiDsaKey_t

Since API level 4

```
typedef struct {
    tlApiLongInt_t  p;
    tlApiLongInt_t  q
    tlApiLongInt_t  g;
    tlApiLongInt_t  x;
    tlApiLongInt_t  y;
} tlApiDsaKey_t;
```

The DSA key structure may contain public key, both public key and private key or private key only as described in the NIST.FIPS.186-4 standard.

The DSA key sizes from 512 to 3072 bits are supported.

The fields of the structure are:

- p: Prime modulus p.
- q: Prime q
- g: Generator
- x: Private key
- y: Public key

5.3.8 tlApiEcdsaKey_t

Since API level 4

```
typedef struct {
    uint32_t        curveType;
    tlApiLongInt_t  x;
    tlApiLongInt_t  y;
    tlApiLongInt_t  privKey;
} tlApiEcdsaKey_t;
```

The ECDSA key structure may contain public key, both public key and private key or private key only as described in the NIST.FIPS.186-4 standard.

In all cases, the curve type must be present.

The fields of the structure are:

- curveType: ECC_CURVE_NIST_P192, ECC_CURVE_NIST_P224, ECC_CURVE_NIST_P256, ECC_CURVE_NIST_P384 or ECC_CURVE_NIST_P521.
- x: Public key affine point x
- y: Public key affine point y
- privKey: private key data

5.3.9 tlApiKey_t

```
typedef union {
    tlApiSymKey_t *symKey;
    tlApiRsaKey_t *rsaKey;
#if TBASE_API_LEVEL >= 4
    tlApiDsaKey_t *dsaKey;
    tlApiDsaKey_t *ecdsaKey;
#endif
```

```
} tlApiKey_t;
```

Union of key structure pointers. Enables generic interfaces.

The fields of the structure are:

- symKey: Pointer to symmetric key.
- rsaKey: Pointer to RSA key.
- dsaKey: Pointer to DSA key.
- ecdsaKey: Pointer to ECDSA key.

5.3.10tlApiSymKey_t

```
typedef struct {
    uint8_t *key;
    uint32_t len;
 } tlApiSymKey_t;
```

Symmetric key structure.

The fields of the structure are:

- key: Pointer to the key.
- len: Byte length of the key.

5.3.11 tlApiKeyPair_t

```
typedef union {
    tlApiRsaKey_t *rsaKeyPair;
#if TBASE_API_LEVEL >= 4
    tlApiDsaKey_t *dsaKeyPair;
    tlApiEcdsaKey_t *ecdsaKeyPair;
#endif
} tlApiKeyPair_t;
```

Asymmetric key structure.

The fields of the structure are:

- rsaKeyPair: Pointer to RSA key structure.
- dsaKeyPair: Pointer to DSA key structure.
- ecdsaKeyPair: Pointer to ECDSA key structure.

5.3.12tlApiLongInt_t

```
typedef struct {
    uint8_t *value;
    uint32_t len;
} tlApiLongInt_t;
```

Symmetric key structure.

The fields of the structure are:

- value: Pointer to value. Byte array in big endian format.
- len: Byte length of value.

5.3.13tlApiPlatformInfo_t

```
typedef struct {
    uint32_t size;
    uint32_t manufacturerdId;
    uint32_t platformVersion;
    uint32_t platformInfoDataLength;
    uint8_t  platformInfoData[];
} tlApiPlatformInfo_t;
```

The platform information structure returns manufacturer specific information about the hardware platform.

The fields of the structure are:

- size: size of the structure.
- manufacturerdId: Manufacturer ID provided by Trustonic.
- platformVersion :Version of platform
- platformInfoDataLength : Length of manufacturer specific platform information
- platformInfoData: Manufacturer specific platform information data.

5.3.14tlApiSpTrustletId_t

```
typedef struct {
    mcSpid_t spid;
    mcUuid_t uuid; } tlApiSpTrustletId_t;
```

Service provider Trusted Application id.

The fields of the structure are:

- spid: Service Provider ID.
- uuid: Trusted Application UUID.

5.4 INTER-WORLD COMMUNICATION

Kinibi provides Trusted Applications with a set of functions for inter-world communication.

Communication is based on shared memory buffers and notifications without a message payload.

Message-formatting is application specific. Messages are interchanged on world shared memory buffers that the Trusted Application Connector (TLC) specifies in `mcOpenSession()` and `mcMap()`. The Kinibi driver and Kinibi OS set up this shared memory buffer between TLC and Trusted Application and forward notifications between the two.

5.4.1 tlApiNotify

```
_TLAPI_EXTERN_C tlApiResult_t tlApiNotify (void);
```

Notify the Trusted Application Connector about changes in the Trusted Application Communication Interface (TCI) message buffer.

Trusted Applications must use the **tlApiNotify()** function to inform the Kinibi TEE that the session channel contains information to be processed by its Trusted Application Connector. Trusted Applications must not make assumptions at what point in time the information will be processed by the Trusted Application Connector. It is up to Kinibi TEE to decide when to inform the Trusted Application Connector. **tlApiNotify()** returns when the notification is processed and the Trusted Application can continue.

Returns:

- TLAPI_OK if notification has been issued successfully.

5.4.2 tlApiWaitNotification

```
_TLAPI_EXTERN_C tlApiResult_t tlApiWaitNotification (uint32_t timeout)
```

Wait for a notification from the NWd.

Trusted Applications use the **tlApiWaitNotification()** to wait for a notification by their Trusted Application Connector. Calling **tlApiWaitNotification()** has two possible outcomes: If a notification is pending for the Trusted Application, it will return immediately. If no notification is pending, the function will block the Trusted Application until a notification dedicated to the Trusted Application session arrives. This function can be used in blocking mode, timeout set to TLAPI_INFINITE_TIMEOUT or in polling mode, timeout set to TLAPI_NO_TIMEOUT. If other timeout values are provided, E_TLAPI_INVALID_TIMEOUT is returned immediately.

Since Kinibi 310B corresponding to API LEVEL 8, real timeouts are supported.

Parameters:

- timeout: TLAPI_NO_TIMEOUT => direct return, TLAPI_INFINITE_TIMEOUT => wait infinitely
  - since API LEVEL 8 => milliseconds to wait, maximum 16.777.216 ms =4h40min

Returns:

- TLAPI_OK if notification has been received.

5.5 CRYPTOGRAPHY

Kinibi provides Trusted Applications with access to cryptographic primitives.

Depending on the platform, cryptographic functions are implemented in a software library or using a hardware cryptographic engine. Depending on the used cryptographic driver, certain functions may or may not exist.

5.5.1 Constants

Constant Name	API level	Value	Definition
AF_CIPHER	1 and higher	(1U << 24)	Algorithm ID is composed of group flags and a consecutive number. The upper 16bits are used for grouping, whereas the lower 16bits are available to distinguish algorithms within each group. Algorithm type flags.
AF_CIPHER_AES	1 and higher	(1U << 16)	Subgroups of cipher algorithms.
AF_SIG_DES	1 and higher	(1U << 16)	Subgroups of signature algorithms.
AF_SIG_AES	4	(8U << 16)	
AF_SIG_DSA	4	(16U << 16)	
AF_SIG_ECDSA	4	(32U << 16)	
ECC_CURVE_NIST_P192	4	1	Elliptic curve P192
ECC_CURVE_NIST_P224	4	2	Elliptic curve P224
ECC_CURVE_NIST_P256	4	3	Elliptic curve P256
ECC_CURVE_NIST_P384	4	4	Elliptic curve P384
ECC_CURVE_NIST_P521	4	5	Elliptic curve P521
CR_SID_INVALID	1 and higher	0xffffffff	Invalid crypto session id returned in case of an error.

Table 6 Cryptography Constants

5.5.2 Types

5.5.2.1 tlApiCrSession_t

```
typedef uint32_t tlApiCrSession_t;
```

Handle of a crypto session.

5.5.3 Enumerations

5.5.3.1 tlApiCipherAlg_t

List of Cipher algorithms:

- AES ciphers
- Triple-DES ciphers
- DES ciphers
- RSA ciphers

An algorithm in this list is to be interpreted as a combination of cryptographic algorithm, paddings, block sizes and other information.

Enumerator Name	API level	Definition
TLAPI_ALG_AES_128_CBC_NOPAD	1 and higher	AES with key size 128 in CBC mode, no padding.
TLAPI_ALG_AES_128_CBC_ISO9797_M1	1 and higher	AES with key size 128 in CBC mode, padding according to the ISO 9797 method 1 scheme.
TLAPI_ALG_AES_128_CBC_ISO9797_M2	1 and higher	AES with key size 128 in CBC mode, padding according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.
TLAPI_ALG_AES_128_CBC_PKCS5	1 and higher	AES with key size 128 in CBC mode, padding according to the PKCS#5 scheme.
TLAPI_ALG_AES_128_CBC_PKCS7	1 and higher	AES with key size 128 in CBC mode, padding according to the PKCS#7 scheme.
TLAPI_ALG_AES_128_ECB_NOPAD	1 and higher	AES with key size 128 in ECB mode, no padding.
TLAPI_ALG_AES_128_ECB_ISO9797_M1	1 and higher	AES with key size 128 in ECB mode, padding according to the ISO 9797 method 1 scheme.
TLAPI_ALG_AES_128_ECB_ISO9797_M2	1 and higher	AES with key size 128 in ECB mode, padding according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.
TLAPI_ALG_AES_128_ECB_PKCS5	1 and higher	AES with key size 128 in ECB mode, padding according to the PKCS#5 scheme.
TLAPI_ALG_AES_128_ECB_PKCS7	1 and higher	AES with key size 128 in ECB mode, padding according to the PKCS#7 scheme.
TLAPI_ALG_AES_128_CTR_NOPAD	1 and higher	AES with key size 128 in CTR mode, no padding.
TLAPI_ALG_AES_256_CBC_NOPAD	1 and higher	AES with key size 256 in CBC mode, no padding.
TLAPI_ALG_AES_256_CBC_ISO9797_M1	1 and higher	AES with key size 256 in CBC mode, padding according to the ISO 9797 method 1 scheme.
TLAPI_ALG_AES_256_CBC_ISO9797_M2	1 and higher	AES with key size 256 in CBC mode, padding according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.
TLAPI_ALG_AES_256_CBC_PKCS5	1 and higher	AES with key size 256 in CBC mode, padding according to

		the PKCS#5 scheme.
TLAPI_ALG_AES_256_CBC_PKCS7	1 and higher	AES with key size 256 in CBC mode, padding according to the PKCS#7 scheme.
TLAPI_ALG_AES_256_ECB_NOPAD	1 and higher	AES with key size 256 in ECB mode, no padding.
TLAPI_ALG_AES_256_ECB_ISO9797_M1	1 and higher	AES with key size 256 in ECB mode, padding according to the ISO 9797 method 1 scheme.
TLAPI_ALG_AES_256_ECB_ISO9797_M2	1 and higher	AES with key size 256 in ECB mode, padding according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.
TLAPI_ALG_AES_256_ECB_PKCS5	1 and higher	AES with key size 256 in ECB mode, padding according to the PKCS#5 scheme.
TLAPI_ALG_AES_256_ECB_PKCS7	1 and higher	AES with key size 256 in ECB mode, padding according to the PKCS#7 scheme.
TLAPI_ALG_AES_256_CTR_NOPAD	1 and higher	AES with key size 256 in CTR mode, no padding.
TLAPI_ALG_3DES_CBC_ISO9797_M1	1 and higher	Triple DES with key size 16 byte in outer CBC mode, padding according to the ISO 9797 method 1 scheme.
TLAPI_ALG_3DES_CBC_ISO9797_M2	1 and higher	Triple DES with key size 16 byte in outer CBC mode, padding according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.
TLAPI_ALG_3DES_CBC_NOPAD	1 and higher	Triple DES with key size 16 byte in outer CBC mode, no padding.
TLAPI_ALG_3DES_CBC_ PKCS5	1 and higher	Triple DES with key size 16 byte in outer CBC mode, padding according to the PKCS#5 scheme.
TLAPI_ALG_3DES_2KEY_CBC_ISO9797_M1	3	Triple DES with key size 16 byte in outer CBC mode, padding according to the ISO 9797 method 1 scheme.
TLAPI_ALG_3DES_2KEY_CBC_ISO9797_M2	3	Triple DES with key size 16 byte in outer CBC mode, padding according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.
TLAPI_ALG_3DES_2KEY_CBC_NOPAD	3	Triple DES with key size 16 byte in outer CBC mode, no padding.
TLAPI_ALG_3DES_2KEY_CBC_ PKCS5	3	Triple DES with key size 16 byte in outer CBC mode, padding according to the PKCS#5 scheme.
TLAPI_ALG_3DES_3KEY_CBC_ISO9797_M1	3	Triple DES with key size 24 byte in outer CBC mode, padding according to the ISO 9797 method 1 scheme.
TLAPI_ALG_3DES_3KEY_CBC_ISO9797_M2	3	Triple DES with key size 24 byte in outer CBC mode, padding according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.
TLAPI_ALG_3DES_3KEY_CBC_NOPAD	3	Triple DES with key size 24 byte in outer CBC mode, no

		padding.
TLAPI_ALG_3DES_3KEY_CBC_ PKCS5	3	Triple DES with key size 24 byte in outer CBC mode, padding according to the PKCS#5 scheme.
TLAPI_ALG_DES_CBC_ISO9797_M1	1 and higher	DES in CBC mode or triple DES in outer CBC mode, padding according to the ISO 9797 method 1 scheme.
TLAPI_ALG_DES_CBC_ISO9797_M2	1 and higher	DES in CBC mode or triple DES in outer CBC mode, padding according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.
TLAPI_ALG_DES_CBC_NOPAD	1 and higher	DES in CBC mode or triple DES in outer CBC mode, no padding.
TLAPI_ALG_DES_CBC_PKCS5	1 and higher	DES in CBC mode or triple DES in outer CBC mode, padding according to the PKCS#5 scheme.
TLAPI_ALG_DES_ECB_ISO9797_M1	1 and higher	DES in ECB mode, padding according to the ISO 9797 method 1 scheme.
TLAPI_ALG_DES_ECB_ISO9797_M2	1 and higher	DES in ECB mode, padding according to the ISO 9797 method 2 (ISO 7816-4, EMV'96) scheme.
TLAPI_ALG_DES_ECB_NOPAD	1 and higher	DES in ECB mode, no padding.
TLAPI_ALG_DES_ECB_PKCS5	1 and higher	DES in ECB mode, padding according to the PKCS#5 scheme.
TLAPI_ALG_RSA_ISO14888	1 and higher	RSA, padding according to the ISO 14888 scheme.
TLAPI_ALG_RSA_NOPAD	1 and higher	RSA, no padding.
TLAPI_ALG_RSA_PKCS1	1 and higher	RSA, padding according to the PKCS#1 (v1.5) scheme.
TLAPI_ALG_RSA_SHA1_OAEP	7	RSA, padding according to the OAEP scheme. (SHA-1 MGF1)
TLAPI_ALG_RSA_SHA224_OAEP	7	RSA, padding according to the OAEP scheme. (SHA-224 MGF1)
TLAPI_ALG_RSA_SHA256_OAEP	7	RSA, padding according to the OAEP scheme. (SHA-256 MGF1)
TLAPI_ALG_RSA_SHA384_OAEP	7	RSA, padding according to the OAEP scheme. (SHA-384 MGF1)
TLAPI_ALG_RSA_SHA512_OAEP	7	RSA, padding according to the OAEP scheme. (SHA-512 MGF1)
TLAPI_ALG_RSACRT_SHA1_OAEP	7	Same as TLAPI_ALG_RSA_SHA1_OAEP
TLAPI_ALG_RSACRT_SHA224_OAEP	7	Same as TLAPI_ALG_RSA_SHA224_OAEP
TLAPI_ALG_RSACRT_SHA256_OAEP	7	Same as TLAPI_ALG_RSA_SHA256_OAEP

TLAPI_ALG_RSACRT_SHA384_OAEP	7	Same as TLAPI_ALG_RSA_SHA384_OAEP
TLAPI_ALG_RSACRT_SHA512_OAEP	7	Same as TLAPI_ALG_RSA_SHA512_OAEP

Table 7 tlApiCipherAlg Enumeration

5.5.3.2 tlApiCipherMode_t

Enumerator Name	Value	Definition
TLAPI_MODE_ENCRYPT	0	Encryption mode.
TLAPI_MODE_DECRYPT	1	Decryption mode.

Table 8 tlApiCipherMode_t Enumeration

5.5.3.3 tlApiKeyPairType_t

List of Key Pair types.

Enumeration:

Enumerator Name	API level	Value	Definition
TLAPI_RSA	1 and higher	0x00000001	RSA public and RSA normal / CRT private key.
TLAPI_DSA	4	0x00000002	DSA public / private key
TLAPI_ECDSA	4	0x00000003	ECDSA public / private key

5.5.3.4 tlApiMdAlg_t

List of Message Digest algorithms.

Enumerator Name	API level	Value	Definition
TLAPI_ALG_MD5	7	AF_MD | 2	Message Digest algorithm MD5.
TLAPI_ALG_SHA1	1 and higher	AF_MD | 3	Message Digest algorithm SHA1.
TLAPI_ALG_SHA256	1 and higher	AF_MD | 4	Message Digest algorithm SHA256.
TLAPI_ALG_SHA224	7	AF_MD | 5	Message Digest algorithm SHA224.
TLAPI_ALG_SHA384	7	AF_MD | 6	Message Digest algorithm SHA384.
TLAPI_ALG_SHA512	7	AF_MD | 7	Message Digest algorithm SHA512.

Table 9 tlApiMdAlg_t Enumeration

5.5.3.5 tlApiRngAlg_t

List of Random Data Generation algorithms.

Enumerator Name	API level	Definition
TLAPI_ALG_SECURE_RANDOM	1 and higher	Random data which is considered to be cryptographically secure.
TLAPI_ALG_PSEUDO_RANDOM	5	Pseudo random data, most likely a returning pattern.
TLAPI_ALG_PLATFORM_RANDOM	5	Raw data from the platform's TRNG (not cryptographically secure). Note that this algorithm may not be supported. In this case tlApiRandomGenerateData() will return E_TLAPI_CR_ALGORITHM_NOT_AVAILABLE.

Table 10 tlApiRngAlg_t Enumeration

5.5.3.6 tlApiSigAlg_t

List of Signature algorithms.

An algorithm in this list is to be interpreted as a combination of cryptographic algorithm, paddings, block sizes and other information.

Enumerator Name	API level	Definition
TLAPI_ALG_DES_MAC4_NOPAD TLAPI_ALG_DES_MAC4_PKCS5 TLAPI_ALG_DES_MAC8_ISO9797_1_M2_ALG3 TLAPI_ALG_DES_MAC8_ISO9797_M1 TLAPI_ALG_DES_MAC8_ISO9797_M2 TLAPI_ALG_DES_MAC8_NOPAD TLAPI_ALG_DES_MAC8_PKCS5	1 and higher	Deprecated at API level 11.
TLAPI_ALG_HMAC_SHA_256	1 and higher	HMAC following the steps found in RFC 2104 using SHA-256 (SHA-2) as the hashing algorithm.
TLAPI_ALG_HMAC_SHA1	1 and higher	HMAC following the steps found in RFC 2104 using SHA-1 as the hashing algorithm.
TLAPI_ALG_HMAC_SHA224	7	HMAC following the steps found in RFC 2104 using SHA-224 as the hashing algorithm.
TLAPI_ALG_HMAC_SHA256	7	TLAPI_ALG_HMAC_SHA_256
TLAPI_ALG_HMAC_SHA384	7	HMAC following the steps found in RFC 2104 using SHA-384 as the hashing algorithm.
TLAPI_ALG_HMAC_SHA512	7	HMAC following the steps found in RFC 2104 using SHA-512 as the hashing algorithm.
TLAPI_ALG_HMAC_MD5	7	HMAC following the steps found in RFC 2104 using MD5 as the hashing algorithm.

TLAPI_SIG_RSA_SHA_ISO9796	1 and higher	20-byte SHA-1 digest, padded according to the ISO 9796-2 scheme as specified in EMV '96 and EMV 2000, encrypted using RSA.
TLAPI_SIG_RSA_SHA_ISO9796_MR	1 and higher	20-byte SHA-1 digest, padded according to the ISO9796-2 specification and encrypted using RSA.
TLAPI_SIG_RSA_SHA_PKCS1	1 and higher	20-byte SHA-1 digest, padded according to the PKCS#1 (v1.5) scheme, and encrypted using RSA.
TLAPI_SIG_RSA_SHA256_PSS	1 and higher	RSASSA-PSS according to PKCS#1 v2, Content digest SHA-256, MGF SHA-256.
TLAPI_SIG_RSA_SHA1_PSS	1 and higher	RSASSA-PSS according to PKCS#1 v2, Content digest SHA-1, MGF SHA-1.
TLAPI_SIG_RSA_SHA1_PKCS1	7	TLAPI_SIG_RSA_SHA_PKCS1
TLAPI_SIG_RSA_SHA224_PKCS1	7	28-byte SHA-224 digest, padded according to the PKCS#1 (v1.5) scheme, and encrypted using RSA.
TLAPI_SIG_RSA_SHA256_PKCS1	7	32-byte SHA-256 digest, padded according to the PKCS#1 (v1.5) scheme, and encrypted using RSA.
TLAPI_SIG_RSA_SHA384_PKCS1	7	48-byte SHA-384 digest, padded according to the PKCS#1 (v1.5) scheme, and encrypted using RSA.
TLAPI_SIG_RSA_SHA512_PKCS1	7	64-byte SHA-512 digest, padded according to the PKCS#1 (v1.5) scheme, and encrypted using RSA.
TLAPI_SIG_RSA_SHA224_PSS	7	RSASSA-PSS, ContentDigest-SHA224, MfgDigest-SHA224.
TLAPI_SIG_RSA_SHA384_PSS	7	RSASSA-PSS, ContentDigest-SHA384, MfgDigest-SHA384.
TLAPI_SIG_RSA_SHA512_PSS	7	RSASSA-PSS, ContentDigest-SHA512, MfgDigest-SHA512.
TLAPI_SIG_DSA_RAW	4	Obsolete – Instead, use TLAPI_SIG_DSA_HASHED which has the same value.
TLAPI_SIG_DSA_HASHED	4	Digital Signature Algorithm defined in FIPS PUB 186-4, no digest calculation
TLAPI_SIG_ECDSA_RAW	4	Obsolete – Instead, use TLAPI_SIG_ECDSA_HASHED which has the same value.
TLAPI_SIG_ECDSA_HASHED	4	Digital Signature Algorithm defined in FIPS PUB 186-4 section D.2, no digest calculation

Table 11 tlApiSigAlg_t Enumeration

5.5.3.7 tlApiSigMode_t

Main operation modes for signature.

Enumerator Name	Definition
TLAPI_MODE_SIGN	Signature generation mode.
TLAPI_MODE_VERIFY	Message and signature verification mode.

5.5.4 Functions

5.5.4.1 tlApiCipherDoFinal

```
TLAPI_EXTERN_C tlApiResult_t tlApiCipherDoFinal (
    tlApiCrSession_t sessionHandle,
    const uint8_t * srcData,
    size_t srcLen,
    uint8_t * destData,
    size_t * destLen)
```

Encrypts/decrypts the input data.

Processes data that has not been processed by previous calls to **tlApiCipherUpdate()** as well as data supplied in srcData. Completes the cipher session. Afterwards the session is closed and sessionHandle invalid. Input and output buffer may be the same (input data gets buffered block by block).

Parameters:

- sessionHandle: handle of a running Cipher session.
- srcData: reference to input data to be encrypted/decrypted.
- srcLen: byte length of input data to be encrypted/decrypted.
- destData: reference to result area.
- in,out destLen: [in] Byte length of buffer for output data. [out] Byte length of generated output.

Returns:

- TLAPI_OK if operation was successful.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer (session is not being closed).
- E_TLAPI_INVALID_RANGE if buffer is not within the drivers memory range (session is not being closed).
- E_TLAPI_CR_HANDLE_INVALID if session is invalid or not owned by req. client (session is not being closed).
- E_TLAPI_CR_ALGORITHM_NOT_AVAILABLE if algorithm is not supported.
- E_TLAPI_CR_WRONG_OUTPUT_SIZE if [in]destLen is inconsistent with algorithm requirements.
- E_TLAPI_INVALID_INPUT if RSA modulus length is invalid.
- E_TLAPI_DRV_UNKNOWN if some unknown error occurred.
- E_TLAPI_CR_INCONSISTENT_DATA if algorithm could not work with the input data (e.g. wrong padding).
- E_TLAPI_UNMAPPED_BUFFER if one buffer is not entirely mapped in Trusted Application address space.

5.5.4.2 tlApiCipherInit

```
_TLAPI_EXTERN_C tlApiResult_t tlApiCipherInit (
      tlApiCrSession_t * pSessionHandle,
      tlApiCipherAlg_t alg,
      tlApiCipherMode_t mode,
      const tlApiKey_t * key)
```

Initializes a new cipher session.

A handle for the new session is generated. The session is associated with the key. Mode and algorithm type are set. If this method does not return with TLAPI_OK then there is no valid handle value. No crypto session is opened in case of an error.

Parameters:

- pSessionHandle: output, reference to generated Cipher session handle (undefined in case of error).
- alg: see enum cipherMode_t.
- mode : TLAPI_MODE_ENCRYPT or TLAPI_MODE_DECRYPT.
- key: Key for this session. Key data is not copied but stored as reference. Must maintain unchanged during session!

Returns:

- TLAPI_OK if operation was successful.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer.
- E_TLAPI_INVALID_RANGE if buffer is not within the drivers memory range.
- E_TLAPI_INVALID_INPUT if provided mode is unknown or RSA cipher modulus length is zero.
- E_TLAPI_CR_OUT_OF_RESOURCES if no more session could be created.
- E_TLAPI_CR_ALGORITHM_NOT_AVAILABLE if combination of algorithm and mode and key length is not available.
- E_TLAPI_DRV_UNKNOWN if some unknown error occurred.
- E_TLAPI_UNMAPPED_BUFFER if one buffer is not entirely mapped in Trusted Application address space.

5.5.4.3 tlApiCipherInitWithData

```
_TLAPI_EXTERN_C tlApiResult_t tlApiCipherInitWithData (
      tlApiCrSession_t * pSessionHandle,
      tlApiCipherAlg_t alg,
      tlApiCipherMode_t mode,
      const tlApiKey_t * key,
      const uint8_t * buffer,
      size_t bufferLen)
```

Initializes a new cipher session with an initialization vector (IV).

A handle for the new session is generated. The session is associated with the key. Mode and algorithm type are set. If this method does not return with TLAPI_OK then there is no valid handle value. No crypto session is opened in case of an error.

Note:

The IV is ignored for asymmetric ciphers.

If the IV bufferLen is 0 a default value IV = 0 is taken and NO error is returned.

For AES CTR mode a different 16 bytes IV should be used for each encryption.

Parameters:

- out pSessionHandle: reference to generated Cipher session handle (undefined in case of error).
- alg: See enum cipherMode_t.
- mode : TLAPI_MODE_ENCRYPT or TLAPI_MODE_DECRYPT.
- key: Key for this session. Key data is not copied but stored as reference. Must maintain unchanged during session!
- buffer: pointer to the initialization vector.
- bufferLen: length of IV, this has to match the block cipher size, i.e. 16 bytes for AES and 8 bytes for DES.

Returns:

- TLAPI_OK if operation was successful.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer.
- E_TLAPI_INVALID_RANGE if buffer is not within the drivers memory range.
- E_TLAPI_INVALID_INPUT if provided mode is unknown or RSA cipher modulus length is zero.
- E_TLAPI_CR_OUT_OF_RESOURCES if no more session could be created.
- E_TLAPI_CR_ALGORITHM_NOT_AVAILABLE if combination of algorithm and mode and key length is not available.
- E_TLAPI_DRV_UNKNOWN if some unknown error occurred.
- E_TLAPI_CR_INCONSISTENT_DATA if key type doesn't match the algorithm.
- E_TLAPI_UNMAPPED_BUFFER if one buffer is not entirely mapped in Trusted Application address space.

5.5.4.4 tlApiCipherUpdate

```
_TLAPI_EXTERN_C tlApiResult_t tlApiCipherUpdate (
      tlApiCrSession_t sessionHandle,
      const uint8_t * srcData,
      size_t srcLen,
      uint8_t * destData,
      size_t * destLen)
```

Encrypts/decrypts the input data.

Input data does not have to be multiple of block size. Subsequent calls to this method are possible. Unless one or several calls of this function have supplied sufficient input data no output is generated. Input and output buffer may be the same (input data gets buffered block by block).

Parameters:

- sessionHandle: handle of a running Cipher session.
- srcData: reference to input data to be encrypted/decrypted.
- srcLen: byte length of input data to be encrypted/decrypted.
- destData: reference to result area.
- in,out destLen: [in] Byte length of output buffer. [out] Byte length of generated output data.

Returns:

- TLAPI_OK if operation was successful.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer (session is not being closed).
- E_TLAPI_INVALID_RANGE if buffer is not within the drivers memory range (session is not being closed).
- E_TLAPI_CR_HANDLE_INVALID if session invalid or not owned by req. client (session is not being closed).
- E_TLAPI_CR_ALGORITHM_NOT_AVAILABLE if during tlApiCipherInit() provided RSA padding is not available (function needs to check that input data does not exceed block size of RSA cipher).
- E_TLAPI_CR_WRONG_INPUT_SIZE if [in]srcLen is inconsistent with algorithm requirements.
- E_TLAPI_CR_WRONG_OUPUT_SIZE if [in]destLen is inconsistent with algorithm requirements.
- E_TLAPI_DRV_UNKNOWN if some unknown error occurred.
- E_TLAPI_CR_INCONSISTENT_DATA if key type doesn't match the algorithm.
- E_TLAPI_UNMAPPED_BUFFER if one buffer is not entirely mapped in Trusted Application address space.

5.5.4.5 tlApiCrAbort

```
_TLAPI_EXTERN_C tlApiResult_t tlApiCrAbort (
    tlApiCrSession_t sessionHandle)
```

Aborts a crypto session.

Afterwards sessionHandle is not valid anymore.

Parameters:

- sessionHandle: Handle of session to be aborted.

Returns:

- TLAPI_OK if operation was successful.
- E_TLAPI_CR_HANDLE_INVALID if no valid handle was provided.

5.5.4.6 tlApiGenerateKeyPair

```
_TLAPI_EXTERN_C tlApiResult_t tlApiGenerateKeyPair (
      tlApiKeyPair_t * keyPair,
      tlApiKeyPairType_t type,
      size_t len)
```

Generates a key pair.

The key components are generated according to requested type and length.

The caller has to set addresses in the key pair structure and initialize the public key exponent. Generated key components are written to those addresses. It is the responsibility of the caller to provide sufficient space and set length parameters of each of the buffer length elements in key structures appropriately. Length information of generated components will be overwritten with the actual length of the generated key pair structure elements.

For RSA the length value identifies the length of the modulus in bytes. The buffer for the generated modulus and private exponent (if present) must have room for at least len bytes. If present, the buffers for the RSA CRT components P, Q, DP, DQ and QInv must have at least half the length of the modulus (rounded up). The generated modulus is a (len*8)-bit or (len*8-1)-bit number which is the product of two (len*4)-bit probable primes. Public exponent must have non-zero most significant byte. Also public exponent must be odd.

For ECDSA, the length value passed as a parameter doesn't have any meaning as the key length is determined based on the curve type, which is passed as a part of tlApiEcdsaKey_t structure.

For RSA, DSA and ECDSA, key data buffers are expected to be writable.

Parameters:

- in,out keyPair: Reference to key pair structure.
- type: See enum keyPairType_t.
- len: Requested byte length of keys.

Returns:

- TLAPI_OK if operation was successful.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer.
- E_TLAPI_INVALID_RANGE if buffer is not within the drivers range.
- E_TLAPI_CR_ALGORITHM_NOT_AVAILABLE if combination of algorithm and mode and key length is not available.
- E_TLAPI_CR_WRONG_OUPUT_SIZE if provided buffer length of one of the buffers is too small.
- E_TLAPI_DRV_UNKNOWN if some unknown error occurred.
- E_TLAPI_UNMAPPED_BUFFER if one buffer is not entirely mapped in Trusted Application address space.

5.5.4.7 tlApiMessageDigestDoFinal

```
_TLAPI_EXTERN_C tlApiResult_t tlApiMessageDigestDoFinal (
      tlApiCrSession_t sessionHandle,
      const uint8_t * message,
      size_t messageLen,
      uint8_t * hash,
      size_t * hashLen)
```

Hashes the message.

Finishes the message digest session. Afterwards the session is closed and sessionHandle invalid.

Parameters:

- sessionHandle: Handle of a running session Message Digest session.
- message: Reference to message to be hashed.
- messageLen: Byte length of message.
- hash: Reference to generated hash.
- in,out hashLen: [in] Byte length of hash buffer. [out] Byte length of generated hash data.

Returns:

- TLAPI_OK if operation was successful.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer (session is not being closed).
- E_TLAPI_INVALID_RANGE if buffer is not within the drivers memory range (session is not being closed).
- E_TLAPI_CR_HANDLE_INVALID if no valid handle was provided. (session is not being closed).
- E_TLAPI_CR_INCONSISTENT_DATA if hash operation failed.
- E_TLAPI_DRV_UNKNOWN if some unknown error occurred.
- E_TLAPI_UNMAPPED_BUFFER if one buffer is not entirely mapped in Trusted Application address space.

5.5.4.8 tlApiMessageDigestInit

```
_TLAPI_EXTERN_C tlApiResult_t tlApiMessageDigestInit (
      tlApiCrSession_t * pSessionHandle,
      tlApiMdAlg_t algorithm)
```

Initializes a new Message Digest session.

A handle for the new session is generated. The algorithm type is set. If this method does not return with TLAPI_OK, then there is no valid handle returned. No crypto session is opened in case of an error.

Parameters:

- pSessionHandle: Reference to generated Message Digest session handle (undefined in case of error).
- Algorithm: See enum mdAlg_t.

Returns:

- TLAPI_OK if operation was successful.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer.
- E_TLAPI_CR_OUT_OF_RESOURCES if no more session could be created.
- E_TLAPI_CR_ALGORITHM_NOT_AVAILABLE if algorithm is not supported.
- E_TLAPI_CR_INCONSISTENT_DATA if hash operation failed.
- E_TLAPI_DRV_UNKNOWN if some unknown error occurred.
- E_TLAPI_UNMAPPED_BUFFER if one buffer is not entirely mapped in Trusted Application address space.

5.5.4.9 tlApiMessageDigestInitWithData

```
_TLAPI_EXTERN_C tlApiResult_t tlApiMessageDigestInitWithData (
      tlApiCrSession_t * pSessionHandle,
      tlApiMdAlg_t alg,
      const uint8_t * buffer,
      const uint8_t lengthOfDataHashedPreviously[8])
```

Initializes a new Message Digest session.

A handle for the new session is generated. The algorithm type is set. Initializes a hash algorithm with a specified initialization vector. The initialization vector and the length of the previously hashed data needs to be provided to the function in big endian format. This may be used to calculate a part of the hash outside of the Kinibi TEE and then finish the hash in the secure world. If this method does not return with TLAPI_OK then there is no valid handle value. No crypto session is opened in case of an error.

Parameters:

- pSessionHandle: Reference to generated Message Digest session handle (undefined in case of error).
- alg: See mdAlg_t.
- buffer: Reference to previously calculated hash data.
- lengthOfDataHashedPreviously: Byte array in big endian format containing length of previously calculated hash.

Returns:

- TLAPI_OK if operation was successful.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer.
- E_TLAPI_CR_OUT_OF_RESOURCES if no more session could be created.
- E_TLAPI_CR_ALGORITHM_NOT_AVAILABLE if algorithm is not supported.
- E_TLAPI_CR_INCONSISTENT_DATA if hash operation failed.
- E_TLAPI_DRV_UNKNOWN if some unknown error occurred.
- E_TLAPI_UNMAPPED_BUFFER if one buffer is not entirely mapped in Trusted Application address space.

5.5.4.10 tlApiMessageDigestUpdate

```
_TLAPI_EXTERN_C tlApiResult_t tlApiMessageDigestUpdate (
      tlApiCrSession_t sessionHandle,
      const uint8_t * message,
      size_t messageLen)
```

Accumulates message data for hashing.

The message does not have to be blocksize aligned. Subsequent calls to this method are possible.

Parameters:

- sessionHandle: Handle of a running session Message Digest session.
- message: Reference to message to be hashed.
- messageLen: Byte length of input data to be hashed.

Returns:

- TLAPI_OK if operation was successful.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer (session is not being closed).
- E_TLAPI_INVALID_RANGE if buffer is not within the drivers memory range (session is not being closed).
- E_TLAPI_CR_HANDLE_INVALID if no valid handle was provided. (session is not being closed).
- E_TLAPI_CR_INCONSISTENT_DATA if hash operation failed.
- E_TLAPI_DRV_UNKNOWN if some unknown error occurred.
- E_TLAPI_UNMAPPED_BUFFER if one buffer is not entirely mapped in Trusted Application address space.

5.5.4.11 tlApiRandomGenerateData

```
_TLAPI_EXTERN_C tlApiResult_t tlApiRandomGenerateData (
      tlApiRngAlg_t alg,
      uint8_t * randomBuffer,
      size_t * randomLen)
```

Generates random data.

Parameters:

- alg: See enum randomDataGenerationAlg_t.
- randomBuffer: Reference to generated random data.
- in,out randomLen: [in] Byte length of desired random length. [out] Byte length of generated random data.

Returns:

- TLAPI_OK if operation was successful.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer.
- E_TLAPI_INVALID_RANGE if buffer is not within the drivers memory range.
- E_TLAPI_CR_ALGORITHM_NOT_AVAILABLE if algorithm is unknown.
- E_TLAPI_DRV_UNKNOWN for any other errors.
- E_TLAPI_UNMAPPED_BUFFER if one buffer is not entirely mapped in Trusted Application address space.

5.5.4.12 tlApiAddEntropy

```
_TLAPI_EXTERN_C tlApiResult_t tlApiAddEntropy(
    uint8_t  *buf,
    uint32_t buflen)
```

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

Parameters:

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

Returns:

- TLAPI_OK if operation was successful.

5.5.4.13 tlApiSignatureInit

```
_TLAPI_EXTERN_C tlApiResult_t tlApiSignatureInit (
      tlApiCrSession_t * pSessionHandle,
      const tlApiKey_t * key,
      tlApiSigMode_t mode,
      tlApiSigAlg_t alg)
```

Initializes a new signature session and returns the handle for the new session for further usage.

The session is associated with the key. Mode and algorithm type are set. If this method does not return with TLAPI_OK then there is no valid handle value. No crypto session is opened in case of an error.

Parameters:

- out pSessionHandle: Reference to generated Signatures session handle (undefined in case of error).
- key: Key for this session. Key data is not copied but stored as reference. Must maintain unchanged during session!
- Mode: TLAPI_MODE_SIGN or TLAPI_MODE_VERIFY.
- alg: see enum of algorithms.

Returns:

- TLAPI_OK if operation was successful.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer.
- E_TLAPI_INVALID_RANGE if buffer is not within the drivers memory range.
- E_TLAPI_INVALID_INPUT if provided mode is unknown.
- E_TLAPI_CR_OUT_OF_RESOURCES if no more session could be created.
- E_TLAPI_CR_ALGORITHM_NOT_AVAILABLE if algorithm is not supported.
- E_TLAPI_CR_INCONSISTENT_DATA if hash operation failed or key type doesn't match the algorithm.
- E_TLAPI_DRV_UNKNOWN if some unknown error occurred.
- E_TLAPI_UNMAPPED_BUFFER if one buffer is not entirely mapped in Trusted Application address space.

5.5.4.14 tlApiSignatureInitWithData

```
_TLAPI_EXTERN_C tlApiResult_t tlApiSignatureInitWithData (
      tlApiCrSession_t * pSessionHandle,
      const tlApiKey_t * key,
      tlApiSigMode_t mode,
      tlApiSigAlg_t alg,
      const uint8_t * buffer,
      size_t bufferLen)
```

Initializes a new signature session.

A handle for the new session is generated. The session is associated with the key. Mode and algorithm type are set. If this method does not return with TLAPI_OK then there is no valid handle value. No crypto session is opened in case of an error.

Note:

If the buffer/bufferLen contains invalid or inconsistent data (wrong length or length = 0, null, ...) a default value = 0 is taken and NO error is returned.

If the used algorithm doesn't use additional algorithm specific data, the given values are ignored and don't result in an error.

For PSS signatures and verification bufferLen is interpreted as salt length, and buffer may be NULL.

For TLAPI_SIG_RSA_SHA_ISO9796_MR verify this function begins the verification sequence by recovering the message encoded within the signature itself and initializing the internal hash function. Therefore, the signature data needs to be provided in the buffer!

Parameters:

- out pSessionHandle: Reference to generated Signatures session handle (undefined in case of error).
- key: Key for this session. Key data is not copied but stored as reference. Must maintain unchanged during session!
- mode: TLAPI_MODE_SIGN or TLAPI_MODE_VERIFY.
- alg: see enum of algorithms.
- buffer: Reference to algorithm specific data like seed for hash or salt for PSS.
- bufferLen: Length of buffer containing algorithm specific data.

Returns:

- TLAPI_OK if operation was successful.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer.
- E_TLAPI_INVALID_RANGE if buffer is not within the drivers memory range.
- E_TLAPI_INVALID_INPUT if provided mode is unknown.
- E_TLAPI_CR_OUT_OF_RESOURCES if no more session could be created.
- E_TLAPI_CR_ALGORITHM_NOT_AVAILABLE if algorithm is not supported.
- E_TLAPI_CR_INCONSISTENT_DATA if hash operation failed or key type doesn't match the algorithm.
- E_TLAPI_DRV_UNKNOWN if some unknown error occurred.
- E_TLAPI_UNMAPPED_BUFFER if one buffer is not entirely mapped in Trusted Application address space.

5.5.4.15 tlApiSignatureSign

```
_TLAPI_EXTERN_C tlApiResult_t tlApiSignatureSign (
      tlApiCrSession_t sessionHandle,
      const uint8_t * message,
      size_t messageLen,
      uint8_t * signature,
      size_t * signatureLen)
```

Signs the message.

Finishes the signature session. Afterwards the session is closed and sessionHandle invalid. message pointer may be NULL if messageLen = 0.

Parameters:

- sessionHandle: Handle of a running Signature session.
- message: Reference to message to be signed.
- messageLen: Byte length of message.
- in,out signature: Reference to generated signature.
- in,out signatureLen: [in] Byte length of signature buffer. [out] Byte length of generated signature.

Returns:

- TLAPI_OK if operation was successful.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer (session is not being closed).
- E_TLAPI_INVALID_RANGE if buffer is not within the drivers memory range (session is not being closed).
- E_TLAPI_CR_HANDLE_INVALID if no valid handle was provided (session is not being closed).
- E_TLAPI_CR_OUT_OF_RESOURCES if required subsession could not be created.
- E_TLAPI_CR_ALGORITHM_NOT_AVAILABLE if algorithm is not supported.
- E_TLAPI_CR_WRONG_OUTPUT_SIZE if [in]signatureLen is too short.
- E_TLAPI_INVALID_INPUT
- E_TLAPI_DRV_UNKNOWN if some unknown error occurred.
- E_TLAPI_CR_INCONSISTENT_DATA if the crypto library could not calculate a signature.
- E_TLAPI_UNMAPPED_BUFFER if one buffer is not entirely mapped in Trusted Application address space.

5.5.4.16 tlApiSignatureUpdate

```
_TLAPI_EXTERN_C tlApiResult_t tlApiSignatureUpdate (
      tlApiCrSession_t sessionHandle,
      const uint8_t * message,
      size_t messageLen)
```

Accumulates data for a signature calculation.

Input data does not have to be multiple of blocksize. Subsequent calls of this method are possible. **tlApiSignatureSign()** or **tlApiSignatureVerify()** have to be called to complete the signature operation.

Parameters:

- sessionHandle: Handle of a running Signature session.
- message: Reference to message to be signed/verified.
- messageLen: Byte length of message.

Returns:

- TLAPI_OK if operation was successful.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer (session is not being closed).
- E_TLAPI_INVALID_RANGE if buffer is not within the drivers memory range (session is not being closed).
- E_TLAPI_CR_HANDLE_INVALID if session invalid or not owned by req. client (session is not being closed).
- E_TLAPI_CR_WRONG_OUTPUT_SIZE
- E_TLAPI_CR_ALGORITHM_NOT_AVAILABLE if algorithm is not supported.
- E_TLAPI_CR_INCONSISTENT_DATA if there was a problem with the algorithm.
- E_TLAPI_DRV_UNKNOWN if some unknown error occurred.
- E_TLAPI_UNMAPPED_BUFFER if one buffer is not entirely mapped in Trusted Application address space.

5.5.4.17 tlApiSignatureVerify

```
_TLAPI_EXTERN_C tlApiResult_t tlApiSignatureVerify (
      tlApiCrSession_t sessionHandle,
      const uint8_t * message,
      size_t messageLen,
      const uint8_t * signature,
      size_t signatureLen,
      bool * validity)
```

Validates a signature for the supplied message.

Finishes the signature session. Afterwards the session is closed and sessionHandle invalid. Message pointer may be NULL if messageLen = 0. Null pointer ex.

Parameters:

- sessionHandle: Handle of a running Signature session.
- message: Reference to message to be verified.
- messageLen: Byte length of message.
- signature: Reference to signature to be verified.
- signatureLen: Byte length of signature.
- validity: Reference to verification result. TRUE if verified, otherwise FALSE.

Returns:

- TLAPI_OK if operation was successful.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer (session is not being closed).
- E_TLAPI_INVALID_RANGE if buffer is not within the drivers memory range (session is not being closed).
- E_TLAPI_CR_HANDLE_INVALID if no valid handle was provided (session is not being closed).
- E_TLAPI_CR_OUT_OF_RESOURCES if required subsession could not be created.
- E_TLAPI_CR_ALGORITHM_NOT_AVAILABLE if algorithm is not supported.
- E_TLAPI_CR_WRONG_OUTPUT_SIZE if [in]signatureLen is too short.
- E_TLAPI_INVALID_INPUT
- E_TLAPI_DRV_UNKNOWN if some unknown error occurred.
- E_TLAPI_UNMAPPED_BUFFER if one buffer is not entirely mapped in Trusted Application address space.

5.5.4.18 tlApiDeriveKey

```
_TLAPI_EXTERN_C tlApiResult_t tlApiDeriveKey(
    const void *salt,
    size_t saltLen,
    void *dest,
    size_t destLen,
    mcSoContext_t context,
    mcSoLifeTime_t lifetime)
```

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.
- context [in] Context for derived key. Possible values are:
  - MC_SO_CONTEXT_TLT: The key is diversified for each Trusted Application. This means a unique identifier of the calling Trusted Application is added to the salt.
  - MC_SO_CONTEXT_SP: The key is diversified for each Service Provider. This means a unique identifier of the Service Provider is added to the salt.
  - MC_SO_CONTEXT_DEVICE: The key is not diversified across Trusted Applications or Service Providers.
- lifetime [in] Lifetime for derived key. Possible values are:
  - MC_SO_LIFETIME_POWERCYCLE: the key is diversified for each powercycle. This means a unique identifier of the powercycle is added to the salt.
  - MC_SO_LIFETIME_SESSION: the key is diversified for each session of the Trusted Application. This means a unique identifier of the session is added to the salt.
  - MC_SO_LIFETIME_PERMANENT: the key is not diversified further across sessions or powercycles.

Returns:

TLAPI_OK if operation was successful.

5.5.4.19 tlApiEndorse

```
_TLAPI_EXTERN_C tlApiResult_t tlApiEndorse(
    const void *msg,
```

```
    size_t msgLen,
    void *dst,
    size_t *dstLen,
    mcScope_t scope);
```

Creates a device endorsement signature for a given message.

The endorsement can be exported to the service provider and then verified through Trustonic Directory server to prove that it originates from the right Trusted Application running on a genuine Kinibi-device.

This is typically used when sharing a public key with a server or generally for a service provider to trust that a request is coming from the its Trusted Application on a genuine Kinibi device.

The message is first signed with a key derived from the Hardware Unique Key and then encrypted with Trustonic public endorsement key.

Parameters:

- msg [in] The message to be endorsed.
- msgLen [in] Length of the message.
- dst [out] The endorsement message. If dst is NULL then destLen is updated with the required length of the buffer to store the endorsement.
- destLen [in, out] length of the endorsement.
- scope [in] The scope of the endorsement: If scope is `MC_SCOPE_TRUSTED_APPLICATION` the endorsement is unique to the calling TA. If scope is `MC_SCOPE_CONTAINER` the endorsement is unique to the container of the TA calling this function.

Returns:

TLAPI_OK if operation was successful.

5.6 SECURE OBJECTS

The secure object API provides integrity, confidentiality and authenticity for data that is sensitive but needs to be stored in untrusted (normal world) memory.

Secure objects provide device binding. Respective objects are only valid on a specific device.

There are two core operations of this API:

- wrap() which encloses user data in a secure object.
- unwrap() which extracts user data from a secure object.

The user data is divided into data that will remain as plain data and data that will be encrypted:

```
        +------------------+------------------------+
        |   plain-data     |  to-be-encrypted-data  |
        +------------------+------------------------+
        /-------------- user data -------------------/
        /- data that is    /-- data that will be
           not going to        encrypted. This data
           be encrypted        is encrypted by the
           and will remain     wrapper function ----/
           as cleartext ---/

        /---- plainLen ----/------ encryptLen -------/
```

A secure object looks like this:

```
+--------+--------------------+------------------+--------+---------+
| Header |      plain-data    |   encrypted-data |  hash  | random  |
+--------+--------------------+------------------+--------+---------+
/--------/---- plainLen -------/-- encryptedLen ---/-- 32 --/- 16 -/
/----------------- toBeHashedLen  -----------------/
                             /---------- toBeEncryptedLen ----------/
```

DATA INTEGRITY

A secure object contains a message digest (hash, random) that ensures data integrity of the user data. The hash value is computed and stored during the wrap() operation as well as adding a random number (before data encryption takes place) and recomputed and compared during the unwrap() operation (after the data has been decrypted).

CONFIDENTIALITY

Secure objects are encrypted using context-specific keys that are never exposed, neither to the normal world, nor to the Trusted Application. It is up to the user to define how many bytes of the user data are to be kept in plain text and how many bytes are to be encrypted. The plain text part of a secure object always precedes the encrypted part.

AUTHENTICITY

As a means of ensuring the trusted origin of secure objects, the wrap operation stores the Trusted Application id (SPID, UUID) of the calling Trusted Application in the secure object header (as producer). This allows Trusted Applications to only accept secure objects from certain partners. This is most important for scenarios involving secure object sharing.

CONTEXT

The concept of context allows for sharing of secure objects. At present there are three kinds of context:

- MC_SO_CONTEXT_TLT: Trusted Application context. The secure object is confined to a particular Trusted Application. This is the standard use case.
- PRIVATE WRAPPING: If no consumer was specified, only the Trusted Application that wrapped the secure object can unwrap it.
- DELEGATED WRAPPING: If a consumer Trusted Application is specified, only the Trusted Application specified as 'consumer' during the wrap operation can unwrap the secure object. Note that there is no delegated wrapping with any other contexts.
- MC_SO_CONTEXT_SP: Service provider context. Only Trusted Applications that belong to the same service provider can unwrap a secure object that was wrapped in the context of a certain service provider.
- MC_SO_CONTEXT_DEVICE: Device context. All Trusted Applications can unwrap secure objects wrapped for this context.

Default flag TLAPI_UNWRAP_DEFAULT permits only Trusted Application context and no delegation. Include flag TLAPI_UNWRAP_PERMIT_DELEGATED if you want to allow delegated objects. Include flags TLAPI_UNWRAP_PERMIT_CONTEXT_SP or TLAPI_UNWRAP_PERMIT_CONTEXT_DEVICE if you want to permit unwrapping with those context.

LIFETIME

The concept of a lifetime allows limiting how long a secure object is valid. After the end of the lifetime, it is impossible to unwrap the object. At present, three lifetimes are defined:

MC_SO_LIFETIME_PERMANENT: Secure Object does not expire.

MC_SO_LIFETIME_POWERCYCLE: Secure Object expires on reboot.

MC_SO_LIFETIME_SESSION: Secure Object expires when Trusted Application session is closed. The secure object is thus confined to a particular session of a particular Trusted Application. Note that session lifetime is only allowed for private wrapping in the Trusted Application context MC_SO_CONTEXT_TLT.

5.6.1 Types

5.6.1.1 tsSource_t

Real time sources in Kinibi.

Enumerator:

- TS_SOURCE_ILLEGAL Illegal counter source value.
- TS_SOURCE_SOFTCNT monotonic counter that is reset upon power cycle.
- TS_SOURCE_SECURE_RTC Secure real time clock that uses underlying hardware clock.

5.6.2 Functions

5.6.2.1 tlApiUnwrapObjectExt

```
_TLAPI_EXTERN_C tlApiResult_t tlApiUnwrapObjectExt (
   void * src,
   size_t srcLen,
   void * dest,
   size_t * destLen,
   uint32_t flags)
```

Unwraps a secure object.

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. For further details refer to tlApiWrapObject().

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.

It is recommended that the source and destination buffers for the unwrapping reside in secure memory.

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 allocated (which is the reason why it also is set as 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.
- in flags: See more explanation at the top, in the CONTEXT part.

Returns:

- 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.

- E_TLAPI_SO_DELEGATED_NOT_PERMITTED Delegated objects were not permitted.
- E_TLAPI_SO_CONTEXT_NOT_PERMITTED This context was not permitted.

5.6.2.2 tlApiWrapObjectExt

```
_TLAPI_EXTERN_C tlApiResult_t tlApiWrapObjectExt (
      const void * src,
      size_t plainLen, 3
      size_t encryptedLen,
      void * dest,
      size_t * destLen,
      mcSoContext_t context,
      mcSoLifeTime_t lifetime,
      const tlApiSpTrustletId_t * consumer,
      uint32_t flags)
```

Wraps user data given in the source buffer and creates a secure object in the destination buffer.

The required size of the destination buffer (total size of secure object) can be obtained through the MC_SO_SIZE() macro. The input to this macro is the length of the plain data and the length of the data that is to be encrypted.

It is recommended that the source and destination buffers for the wrapping reside in secure memory.

Example:

```
  secureObjectLength = MC_SO_SIZE(plainLength, encryptLength)
```

The MC_SO_SIZE() macro can be used for allocating memory with the size of the Secure Object. Example:

```
  uint8_t dataBuf[MC_SO_SIZE(plainLength, encryptLength)];

Source and destination addresses may overlap, thus the following code
is a typical usage pattern:

  union {       uint8_t src[100];       uint8_t dst[MC_SO_SIZE(30, 70)];
// 30 bytes plain, 70 bytes encrypted.  } buffer;


  size_t soLen = sizeof(buffer);

  // Fill source.  buffer.src[0] = 'H';  ...

  if  (TLAPI_OK  ==  tlApiWrapObject(            buffer.src,  30,  70,
buffer.dst, &soLen, MC_SO_CONTEXT_TLT, NULL)) {      ...  }
```

Parameters:

- in src: User data. The data which is created by the user. The user data is divided into two types i.e. data that will remain cleartext and will not be encrypted and the data that will be encrypted into the secure object. Note! It can be a good programming exercise/experiment to check the Secure Object data and there find out that some part of the SO is plain-text and therefore readable.
- In plainLen: Length of plain text user data (from beginning of src). This is the length of the userdata that is going to remain as plain text (plain data), i.e. not be encrypted.

- in encryptedLen: Length of to-be-encrypted user data (after plain text user data). This is the Length of the data that is going to be encrypted. The offset is after the last byte of the plain text.
- in, out dest: Destination buffer (secure object). Every secure object starts with the header, so pass the header into this function. Note that the pointer must be word-aligned.
- in, out destLen: [in] Length of the destination buffer. [out] Length of the secure object.
- in context: Key context.
- in lifetime: Expiry type of secure object.
  - MC_SO_LIFETIME_PERMANENT: Secure Object does not expire.
  - MC_SO_LIFETIME_POWERCYCLE: Secure Object expires on reboot.
  - MC_SO_LIFETIME_SESSION: Secure Object expires when Trusted Application session is closed.
- in consumer: NULL or Trusted Application/service provider identifier for delegated wrapping. Delegated wrapping makes it possible for other Trusted Applications to unwrap the secure object. Such scenario can be communication between Trusted Applications. It can be a service provider that is using several Trusted Applications which are communicating with each other.
- in flags: Use the TLAPI_WRAP_DEFAULT flag

Returns:

- TLAPI_OK if operation was successful.
- E_TLAPI_NULL_POINTER If a pointer input parameter is NULL.
- E_TLAPI_INVALID_INPUT If an input parameter is invalid, for instance if the maximum payload size is exceeded.
- E_TLAPI_CR_WRONG_OUPUT_SIZE If the output buffer is too small.
- E_TLAPI_UNALIGNED_POINTER If the secure object pointer is not word-aligned. E_TLAPI_UNMAPPED_BUFFER If one buffer is not entirely mapped in Trusted Application address space.

5.7 SYSTEM FUNCTIONS

The Kinibi system API interface provides system information and system functions to Trusted Applications.

5.7.1 Functions

5.7.1.1 tlApiExit

```
_TLAPI_EXTERN_C _TLAPI_NORETURN void tlApiExit (
      uint32_t exitCode)
```

Trusted Applications can use the tlApiExit() to terminate themselves and return an exit code. This can be used if the initialization fails or an unrecoverable error occurred. The Trusted Application will be terminated immediately and this function will not return.

Parameters:

exitCode: the exit code.

5.7.1.2 tlApiGetPlatformInformation

```
_TLAPI_EXTERN_C _TLAPI_NORETURN void tlApiGetPlatformInformation (
    tlApiPlatformInfo_t platformInfo)
```

Get information about the hardware platform.

The function tlApiGetPlatformInformation() provides information about the current hardware platform.

Parameters:

- platformInfo: pointer to tlApiPlatformInfo_t structure that receives the platform information.

Returns:

There is no return code, since the function will not return.

5.7.1.3 tlApiGetMobicoreVersion

```
_TLAPI_EXTERN_C tlApiResult_t tlApiGetMobicoreVersion (
    mcVersionInfo_t * mcVersionInfo)
```

Get information about the underlying Kinibi version.

The function **tlApiGetMobicoreVersion()** provides the Kinibi product id and version information about the various Kinibi interfaces as defined in mcVersionInfo.h

Parameters:

- mcVersionInfo: pointer to version information structure.

Returns:

- TLAPI_OK if version has been set
- E_TLAPI_NULL_POINTER if one parameter is a null pointer.
- E_TLAPI_UNMAPPED_BUFFER if one buffer is not entirely mapped in Trusted Application address space.

5.7.1.4 tlApiGetSuid

```
_TLAPI_EXTERN_C tlApiResult_t tlApiGetSuid (
    mcSuid_t * suid)
```

Get the System on Chip Universal Identifier.

Parameters:

- suid: pointer to Suid structure that receives the Suid data

Returns:

- TLAPI_OK if Suid has been successfully read.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer.
- E_TLAPI_UNMAPPED_BUFFER if one buffer is not entirely mapped in Trusted Application address space.

5.7.1.5 tlApiGetVirtMemType

```
_TLAPI_EXTERN_C tlApiResult_t tlApiGetVirtMemType(
        uint32_t *type,
        addr_t   addr,
        uint32_t size);
```

Get the virtual memory type

Parameters:

- in *type: pointer to address where type is returned
- in addr: start address of checked memory
- in size: size checked memory

Returns:

- TLAPI_VIRT_MEM_TYPE_SECURE The memory area is mapped as secure
- TLAPI_VIRT_MEM_TYPE_NON_SECURE The memory area is mapped as non-secure

5.7.1.6 tlApiIsNwdBufferValid

```
bool tlApiIsNwdBufferValid( addr_t addr, uint32_t size )
```

Helper to simplify NWd buffer testing.

Parameters:

- in addr: pointer to NWd buffer.
- In size: size of NWd buffer.

Returns:

- TLAPI_OK if buffer is valid.

5.7.1.7 tlApiGetVersion

```
_TLAPI_EXTERN_C tlApiResult_t tlApiGetVersion (
    uint32_t * tlApiVersion)
```

Gets information about the implementation of the Kinibi Trusted Application API version.

Parameters:

- tlApiVersion: pointer to tlApi version.

Returns:

- TLAPI_OK if version has been set
- E_TLAPI_NULL_POINTER if one parameter is a null pointer.
- E_TLAPI_UNMAPPED_BUFFER if one buffer is not entirely mapped in Trusted Application address space.

5.7.1.8 tlApiLogvPrintf

```
_TLAPI_EXTERN_C void tlApiLogvPrintf (
      const char * fmt,
      va_list args)
```

Formatted logging functions.

tlApiLogvPrintf, tlApiLogPrintf

Minimal printf-like function to print logging message to NWd log.

Supported formatters:

- %s String, NULL value emit "<NULL>".
- %x %X hex
- %p pointer (hex with fixed width of 8)
- %d i signed decimal
- %u unsigned decimal
- %t timestamp (if available in platform). NOTE: This does not consume any value in parameter list.
- %% outputs single %
- %s, %x, %d, and %u support width (example %5s). Width is interpreted as minimum number of characters. Hex number is left padded using '0' to desired width. Decimal number is left padded using ' ' to desired width. String is right padded to desired length.

Newline is used to terminate logging line.

Parameters:

- fmt: Formatter

5.8 TRUSTED USER INTERFACE

5.8.1 Header File

The header file for the Trusted User Interface (TUI) API is "`TlApiTui.h`".

```
#include "TlApiTui.h"
```

5.8.2 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.

Table 12: Trusted Application API Error Codes

5.8.3 Types

5.8.3.1 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.

5.8.3.2 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.

5.8.3.3 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.

5.8.3.4 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.

5.8.3.5 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.

5.8.3.6 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()`

5.8.3.7 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.

5.8.3.8 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.

Figure 2: Memory representation of a raw image.

5.8.3.9 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.

5.8.4 Functions

5.8.4.1 tlApiTuiGetScreenInfo

```
TLAPI_EXTERN_C tlApiResult_t tlApiTuiGetScreenInfo (
    tlApiTuiScreenInfo_ptr screenInfo)
```

Get screen information.

Parameters:

- screenInfo: screen information.

Returns:

- 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.

5.8.4.2 tlApiTuiOpenSession

```
_TLAPI_EXTERN_C tlApiResult_t tlApiTuiOpenSession (void)
```

Open a session to the TUI driver.

Returns:

- 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.

5.8.4.3 tlApiTuiCloseSession

```
_TLAPI_EXTERN_C tlApiResult_t tlApiTuiCloseSession (void)
```

Close the session to the TUI driver.

Returns:

- 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.

5.8.4.4 tlApiTuiSetImage – DEPRECATED

This function is deprecated – `tlApiTuiDrawImage` should be used instead.

```
_TLAPI_EXTERN_C tlApiResult_t tlApiTuiSetImage (
      tlApiTuiImage_ptr image,
      tlApiTuiCoordinates_t coordinates)
```

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 Table 13 gives the capabilities 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

Table 13 PNG support.

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`.

Returns:

- 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.

5.8.4.5 tlApiTuiGetTouchEvent

```
_TLAPI_EXTERN_C tlApiResult_t tlApiTuiGetTouchEvent (
    tlApiTuiTouchEvent_ptr touchEvent)
```

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.

Returns:

- 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.

5.8.4.6 tlApiTuiDrawBuffer

```
_TLAPI_EXTERN_C tlApiResult_t tlApiTuiDrawBuffer(
    tlApiTuiGraphicContext_t graphicContext,
    const tlApiTuiRawImage_t *pRawImage,
    int32_t dx,
    int32_t dy);
```

Since API level 6.

Draw a raw image buffer of pixels at (dx:dy) into a drawing buffer. This function requires a valid graphic context. The pixel format is fixed to RGBA_8888 (red channel first in memory).

Parameters:

- graphicContext: Graphic context.
- pRawImage: Raw image buffer to be drawn.
- dx: X coordinates of the top-left corner of the raw image in the drawing buffer.
- dy: Y coordinates of the top-left corner of the raw image in the drawing buffer.

Returns:

- TLAPI_OK if operation was successful or specific error.
- E_TLAPI_DRV_NO_SUCH_DRIVER if the TUI driver cannot be found.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer.
- E_TLAPI_INVALID_RANGE if the pointed raw image structure is not within the secure memory range.
- E_TLAPI_UNALIGNED_POINTER if the raw image buffer is not aligned accordingly to the given pixel format.
- E_TLAPI_ INVALID_INPUT if one parameter is not valid.
- E_TLAPI_ INVALID_CONTEXT if the graphical context is invalid.

5.8.4.7 tlApiTuiDrawImage

```
_TLAPI_EXTERN_C tlApiResult_t tlApiTuiDrawImage(
    tlApiTuiGraphicContext_t graphicContext,
    const tlApiTuiImage_t *pImage,
    int32_t dx,
    int32_t dy);
```

Since API level 6.

Draw an image at (dx:dy) in a drawing buffer. This function requires a valid graphic context.

This functionahas the same decoding capabilities as `tlApiTuiSetImage`.

Parameters:

- graphicContext: Graphic context.
- pImage: Image to be drawn.
- dx: X coordinates of the top-left corner of the raw image in the drawing buffer.
- dy: Y coordinates of the top-left corner of the raw image in the drawing buffer.

Returns:

- TLAPI_OK if operation was successful or specific error.
- E_TLAPI_DRV_NO_SUCH_DRIVER if the TUI driver cannot be found.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer.
- E_TLAPI_ INVALID_CONTEXT if the graphical context is invalid.
- E_TLAPI_INVALID_RANGE if the pointed image structure is not within the secure memory range.
- 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.

5.8.4.8 tlApiTuiFlipFramebuffers

```
_TLAPI_EXTERN_C tlApiResult_t tlApiTuiFlipFramebuffers(void);
```

Since API level 6.

Flip the front and the back framebuffers of the screen. This function requires an active TUI session.

This function must be called after calling one or more drawing functions. It causes the accumulated content to be actually displayed on the screen. An exception is the legacy function `tlApiTuiSetImage` which draws directly to the screen.

During a TUI session, two framebuffers are maintained. The front buffer is what is shown on the screen. All drawing functions (except `tlApiTuiSetImage`) draw to the back buffer when called with a graphic context of a screen. Calling `tlApiTuiFlipFramebuffers` causes the current content of the back buffer to be displayed, i.e. the back buffer effectively becomes the front buffer. The previous content of the front buffer is lost. Subsequent calls to drawing functions will modify the new back buffer. This two-step process allows multiple parts of an image to be shown at once and avoids tearing when the screen is refreshed in the middle of a display operation.

Returns:

- TLAPI_OK if operation was successful or specific error.
- 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.

5.8.4.9 tlApiTuiFillRectangle

```
_TLAPI_EXTERN_C tlApiResult_t tlApiTuiFillRectangle(
    tlApiTuiGraphicContext_t graphicContext,
    const tlApiTuiRectangle_t *pRectangle,
    uint32_t color);
```

Since API level 6.

Draw a rectangle filled with a particular color into a drawing buffer. This function requires a valid graphic context.

The color format is fixed to RGBA_8888 (red channel first in memory). Alpha blending is done with pixels present in the drawing buffer.

Parameters:

- graphicContext: Graphic context.
- pRectangle: Rectangle to be filled.
- Color: RGBA_8888 pixel color.

Returns:

- TLAPI_OK if operation was successful or specific error.
- E_TLAPI_DRV_NO_SUCH_DRIVER if the TUI driver cannot be found.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer.
- E_TLAPI_ INVALID_CONTEXT if the graphical context is invalid.
- E_TLAPI_INVALID_RANGE if the pointed rectangle structure is not within the secure memory range.
- E_TLAPI_ INVALID_INPUT if one parameter is not valid.

5.8.4.10tlApiTuiGetImageInfo

```
_TLAPI_EXTERN_C tlApiResult_t tlApiTuiGetImageInfo(
    const tlApiTuiImage_t *pImage,
    tlApiTuiImageInfo_t *pInfo);
```

Since API level 6.

Get image information. This function does not require an active TUI session.

Parameters:

- pImage: Image to be parsed.
- pInfo: Information recovered from the image.

Returns:

- TLAPI_OK if operation was successful or specific error.
- E_TLAPI_DRV_NO_SUCH_DRIVER if the TUI driver cannot be found.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer.
- E_TLAPI_INVALID_RANGE if the pointed image structure or the pointed info structure is not within the secure memory range.
- 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.

5.8.4.11 tlApiTuiDecodeImage

```
_TLAPI_EXTERN_C tlApiResult_t tlApiTuiDecodeImage(
    const tlApiTuiImage_t *pImage,
    tlApiTuiRawImage_t *pRaw);
```

Since API level 6.

Decode an image to a raw buffer of pixels. This function does not require an active TUI session.

This function decodes an image from TA memory to a raw buffer of pixels from TA memory. TAs must provide a big enough raw buffer. TAs should use the `tlApiTuiGetImageInfo()` to allocate the output raw buffer of pixels. The pixel format is RGBA_8888.

Parameters:

- pImage: Image to be decoded.
- pRaw: Output raw buffer of pixels.

Returns:

- TLAPI_OK if operation was successful or specific error.
- E_TLAPI_DRV_NO_SUCH_DRIVER if the TUI driver cannot be found.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer.
- E_TLAPI_INVALID_RANGE if the pointed image structures are not within the secure memory range.
- 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.

5.8.4.12 tlApiTuiGetClippingRectangle

```
_TLAPI_EXTERN_C tlApiResult_t tlApiTuiGetClippingRectangle(
        tlApiTuiGraphicContext_t graphicContext,
        tlApiTuiRectangle_t *pRectangle);
```

Since API level 6.

Get the current clipping rectangle. This function requires a valid graphic context.

If never explicitly set by `tlApiTuiSetClippingRectangle()` the default clipping rectangle is the full size of the drawing buffer of the related graphic context.

Parameters:

- graphicContext: Related graphic context.
- pRectangle: Current clipping rectangle.

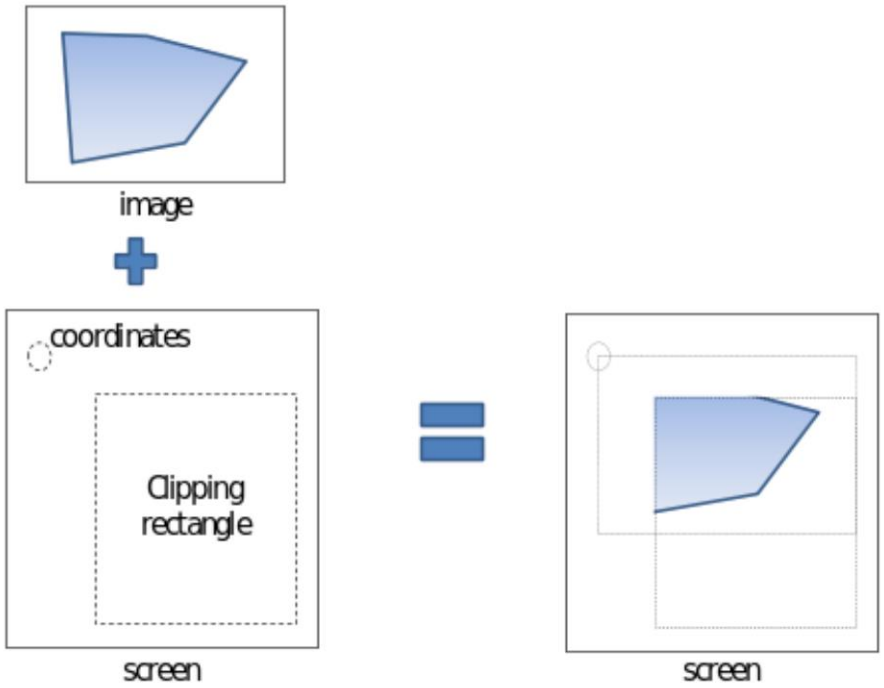


Figure 3: Clipping rectangle.

Returns:

- TLAPI_OK if operation was successful or specific error.
- E_TLAPI_DRV_NO_SUCH_DRIVER if the TUI driver cannot be found.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer.
- E_TLAPI_ INVALID_CONTEXT if the graphical context is invalid.
- E_TLAPI_INVALID_RANGE if the pointed rectangle structure is not within the secure memory range.
- E_TLAPI_ INVALID_INPUT if one parameter is not valid.

5.8.4.13 tlApiTuiSetClippingRectangle

```
_TLAPI_EXTERN_C tlApiResult_t tlApiTuiSetClippingRectangle(
        tlApiTuiGraphicContext_t graphicContext,
        const tlApiTuiRectangle_t *pRectangle);
```

Since API level 6.

Set the current clipping rectangle. This function requires a valid graphic context.

The clipping rectangle cannot be outside of the drawing area of the graphic context.

Parameters:

- graphicContext: Related graphic context.
- pRectangle: Rectangle defining the new clipping area.

Returns:

- TLAPI_OK if operation was successful or specific error.
- E_TLAPI_DRV_NO_SUCH_DRIVER if the TUI driver cannot be found.
- E_TLAPI_NULL_POINTER if one parameter is a null pointer.
- E_TLAPI_ INVALID_CONTEXT if the graphical context is invalid.
- E_TLAPI_INVALID_RANGE if the pointed rectangle structure is not within the secure memory range.
- E_TLAPI_ INVALID_INPUT if one parameter is not valid.
- E_TLAPI_TUI_OUT_OF_DISPLAY if the clipping rectangle or a part of the clipping rectangle is outside of the display area.

5.8.4.14 tlApiTuiGetScreenContext

```
_TLAPI_EXTERN_C tlApiResult_t tlApiTuiGetScreenContext(
        tlApiTuiGraphicContext_t *pGraphicContext);
```

Since API level 6.

Get the graphic context related to the screen. This function requires an active TUI session. The recovered graphic context is no longer valid after the session is closed.

Parameters:

- pGraphicContext: The graphic context.

Returns:

- TLAPI_OK if operation was successful or specific error.
- 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_RANGE if the pointed structure is not within the secure memory range.

5.9 DRM API

5.9.1 Structures

5.9.1.1 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.

5.9.1.2 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.

5.9.1.3 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.

5.9.1.4 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.

5.9.1.5 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;
```

5.9.1.6 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;
```

5.9.2 Constants

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

Table 14 Constants

5.9.3 Errors

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.

Table 15 Secure Playback Errors

5.9.4 Functions

5.9.4.1 tlApiDrmOpenSession

```
tlApiResult_t tlApiDrmOpenSession(
      uint8_t *sHandle);
```

If multiple session support is required it must be first managed 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.

Returns:

- `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.

5.9.4.2 tlApiDrmProcessContent

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

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.

Returns:

- `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.

5.9.4.3 tlApiDrmProcessContentEx

```
tlApiResult_t tlApiDrmProcessContentEx(
        uint8_t                         sHandle,
        tlApiDRM_headerV1               *DRM_header)
```

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.

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

5.9.4.4 tlApiDrmCloseSession

```
tlApiResult_t tlApiDrmCloseSession(
      uint8_t sHandle);
```

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.

Returns:

- `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.

5.9.4.5 tlApiDrmCheckLink

```
tlApiResult_t tlApiDrmCheckLink(
      uint8_t sHandle,
      tlApiDrmLink_t link)
```

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.

Returns:

- `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.

5.10 MEMORY MANAGEMENT

5.10.1 Header File

The header file for the Memory Management functions is "`TlApiHeap.h`".

```
#include "TlApiHeap.h"
```

5.10.2 Heap reservation for the legacy memory layout

The Trusted application has to reserve its heap using the following macro in one of its C-files.

```
DECLARE_TRUSTLET_MAIN_HEAP( uint32_t HeapSize);
```

Parameters:

- HeapSize: [in] the number of bytes to be reserved.

5.10.3 Heap reservation for the extended memory layout

The Trusted application has to reserve its heap using the following lines in the makefile.

```
TBASE_API_LEVEL := 5 ;(or higher to use the extended memory layout)
HEAP_SIZE_INIT := 4096 ;(for example)
HEAP_SIZE_MAX := 8192 ;(optional)
```

5.10.4 Functions

5.10.4.1 tlApiMalloc

```
void* tlApiMalloc(uint32_t size, uint32_t hint);
```

Allocates a block of memory from the heap.

The address of the allocated block is aligned on a 8-bytes boundary. A block allocated by `tlApiMalloc` must be freed by `tlApiFree`.

If the size of the space requested is zero, the value returned is still a non-NULL pointer that the Trusted Application must not attempt to access.

The returned block of memory is filled with zeroes.

Limitations with legacy memory layout:

The total size of the memory that can be allocated is tied to the `DECLARE_TRUSTED_APPLICATION_MAIN_HEAP()` macro. For example `DECLARE_TRUSTED_APPLICATION_MAIN_HEAP(60*1024)` gives 60k available heap. The address space for a Trusted Application is also a limiting factor; please refer to the developer guide for more information. Also, the memory reorganization functionality of libc malloc is not implemented because of code size limitations so alloc and free operations need to be synchronized carefully.

Parameters:

- `size`: [in] the number of bytes to be allocated.
- `hint`: [in] must be 0

Returns:

- Upon successful completion, with size not equal to zero, the function returns a pointer to the allocated space. Otherwise, a NULL pointer is returned.

5.10.4.2 tlApiRealloc

```
void* tlApiRealloc(void* buffer, uint32_t newSize);
```

Reallocates a block of memory from a heap.

This function allows resizing a memory block.

If buffer is NULL, tlApiRealloc is equivalent to tlApiMalloc.

If buffer is not NULL and newSize is 0, then tlApiRealloc is equivalent to tlApiFree and returns a non-NULL pointer that the Trusted Application must not attempt to access.

If newSize is less or equal to the current size of the block, the block is truncated, the content of the block is left unchanged and the function returns buffer.

If newSize is greater than the current size of the block, the size of the block is increased. The whole content of the block is copied at the beginning of the new block. If possible, the block is enlarged in place and the function returns buffer. If this is not possible, a new block is allocated with the new size, the content of the current block is copied, the current block is freed and the function returns the pointer on the new block. The extended space is filled with zeroes.

Parameters:

- buffer: [in] Pointer to the block of memory that the function reallocates. This value may be null or returned by an earlier call to tlApiMalloc or tlApiRealloc.
- newSize: [in] size of the memory block in bytes. This value may be zero.

Returns:

- A pointer to the reallocated memory block, a non-NULL pointer if the newSize is zero or NULL if an error is detected.

5.10.4.3 tlApiFree

```
void tlApiFree(void* buffer);
```

Frees a memory block allocated from a heap by tlApiMalloc or tlApiRealloc.

This function does nothing if buffer is NULL.
The memory block is zeroed before being freed.

Parameters:

- `buffer`: [in] Pointer to the block of memory to be freed.

5.11 TIME

5.11.1 tlApiGetSecureTimestamp

```
_TLAPI_EXTERN_C tlApiResult_t tlApiGetSecureTimestamp(
                  timestamp_ptr pTimestamp)
```

Get a timestamp value from the Secure-World in µs. Note that this function is not available on all the hardware platforms.

The timestamp is guaranteed to be monotonically increasing between resets, but it is not strictly monotonic. So, multiple consecutive calls may read the same value. This may happen in power saving states, where the platform reduces the clock update rate. Thus, this API must be used with care when used to measure a short duration.

Parameters:

- pTimestamp: [out] pointer to timestamp_t counter value

Returns:

- `TLAPI_OK` if the timestamp could be retrieved
- `E_TLAPI_INVALID_PARAMETER` if pTimestamp is NULL or pointer is invalid
- `E_TLAPI_NOT_IMPLEMENTED` if the platform does not support secure timestamps

5.12 PERSISTENT OBJECTS

Kinibi implements the persistent storage API as specified in the GlobalPlatform TEE Internal API Specification v1.0 and GlobalPlatform TEE Internal Core API Specification v1.1. Known deviations from the GlobalPlatform specifications are summarized in §5.12.2. Refer to the GlobalPlatform specifications for a full behavior description, a list of available constants for enumerations and flag masks, and a list of applicable error codes for each functions.

Since API level 7, these functions are available through tlApi. In previous API levels, these functions were only available through the GlobalPlatform API (§4).

5.12.1General considerations

5.12.1.1 Error codes

Note that the functions in this section use GlobalPlatform-style error values (type `TEE_Result`): `TEE_SUCCESS` to indicate success, `TEE_ERROR_xxx` to indicate failure.

5.12.1.2 Atomicity

All functions described in this section are atomic and resilient. If a function call is successful, other applications will observe the initial state up to some point, then the final state. If a function call fails, the state remains unchanged. If the system crashes during the operation, it is either completed or not started.

5.12.1.3 Memory management

Unless otherwise indicated, pointer arguments must point to a block of data that is in the Trusted Application's own memory, not in shared memory.

5.12.2 Known differences with the GlobalPlatform specification

5.12.2.1 Integer type sizes

Kinibi 310 is aligned with version 1.0 of the GlobalPlatform TEE Internal API specification with respect to the use of `size_t` vs. `uint32_t`.

5.12.2.2 Fields of the TEE_ObjectInfo structure

Kinibi 310 is aligned with version 1.0 of the GlobalPlatform TEE Internal API specification with respect to the field names in the `TEE_ObjectInfo` structure, with fields called `objectSize` and `maxObjectSize`.

5.12.2.3 Exclusive creation with TEE_CreatePersistentObject

Kinibi complies with the original semantics of the `TEE_DATA_FLAG_EXCLUSIVE` flag for **Error! Reference source not found.** defined by version 1.0 of the GlobalPlatform specification: overwriting an existing object is only allowed when this flag is absent.

The flag name `TEE_DATA_FLAG_OVERWRITE`, whose semantics are thus to prevent overwriting when present, is not defined. Use the name `TEE_DATA_FLAG_EXCLUSIVE`.

5.12.2.4 Concurrent access to crypto streams

An object may be accessed through different handles, either inside the same Trusted Application instance or from different instances, or where applicable by the security domain for personalization objects. In such cases, all instances see the same contents for the data stream: writing through one handle and subsequently reading through another handle is guaranteed to return the latest data.

For the crypto stream, Kinibi 310 does not provide such a guarantee. As of Kinibi 310, cryptographic data is read as soon as the object is open; subsequent changes to the object by calling `TEE_CreatePersistentObject` or `TEE_RestrictObjectUsage` or TEE_RestrictObjectUsage1 affect the stored content of the object and the handle through which the operation was performed, but are not reflected in other already-open handles to the same object.

Future versions of Kinibi may broadcast changes made to crypto objects to other handles. To reliably use a cryptographic object without being affected by modifications made through another handle, copy the object to a transient object with `TEE_CopyObjectAttributes`.

Note that it is generally not recommended to modify the cryptographic data of a persistent object anyway, because changes in the object type can cause uses of the object to panic in `TEE_SetOperationKey` or `TEE_SetOperationKey2`.

In order to protect concurrent access to cryptographic data, the following restriction applies to `TEE_CreatePersistentObject`: when overwriting an existing object, if the object is already open with the `TEE_DATA_FLAG_ACCESS_WRITE` flag but without the `TEE_DATA_FLAG_SHARE_WRITE` flag and the old version of the object had a crypto stream, the operation is rejected with `TEE_ERROR_ACCESS_CONFLICT`.

5.12.2.5 Persistent object flags for TEE_RestrictObjectUsage

As discussed in §5.12.2.4, as of Kinibi 310, changes made through one handle to a crypto stream of a persistent object are not reflected through another open handle on the same object.

In order to prevent collisions in modifications to usage flags, applications may only call `TEE_RestrictObjectUsage` or `TEE_RestrictObjectUsage1` if they have exclusive write access to the object, namely:

- if the object is a transient object;
- if the object is a persistent object opened with the TEE_DATA_FLAG_ACCESS_WRITE flag but without the TEE_DATA_FLAG_SHARE_WRITE flag;
- if the object is a persistent object opened with the TEE_DATA_FLAG_ACCESS_WRITE_META flag.

In all other cases, TEE_RestrictObjectUsage1 returns TEE_ERROR_ACCESS_DENIED; TEE_RestrictObjectUsage panics.

Note that it is generally not recommended to apply usage restrictions to a persistent object, because if the operation fails for any reason (temporarily unreachable storage, power failure), the persistent object remains in storage with the old restrictions. In particular, when creating a new object to store a cryptographic key, call TEE_RestrictObjectUsage or TEE_RestrictObjectUsage1 on the transient object before calling TEE_CreatePersistentObject to add it to persistent storage. This way, the persistent store never contains the unrestricted object.

5.12.2.6 Sharing flags in Kinibi 30x

In Kinibi versions below 310, the sharing and access flags other than TEE_DATA_FLAG_EXCLUSIVE passed to TEE_CreatePersistentObject or TEE_OpenPersistentObject are ignored; concurrent openings of a file are not detected. Since Kinibi 310, sharing and access flags follow the GlobalPlatform specification, except for the restriction on writing to open crypto streams noted in §5.12.2.4.

5.12.3 General persistent object functions

5.12.3.1 TEE_OpenPersistentObject

```
TEE_Result TEE_OpenPersistentObject(
                  unit32_t storageID,
                  void *objectID, uint32_t objectIDLen,
                  uint32_t flags,
                  TEE_ObjectHandle *object);
```

Open an existing persistent object. The object contains a data stream and may contain a crypto stream.

The following values are currently supported for storageID:

- TEE_STORAGE_PRIVATE: storage that is only available to instances of the calling Trusted Application.
- TEE_STORAGE_PERSO: storage that is read-only for instances of the calling Trusted Application and may be written by the managing Security Domain. Trusted Applications that have no managing Security Domain have no file under this storage ID.
- TEE_TT_STORAGE_PROTECTED: Factory Reset Protected storage that is only available to instances of the calling Trusted Application.

Other values may be added in future versions.

The flags argument is a mask of the following flags:

- TEE_DATA_FLAG_ACCESS_READ: read access — allow the use of TEE_ReadObjectData.
- TEE_DATA_FLAG_SHARE_READ: allow concurrent read access to the same object via other handles (in the same application instance or in a different one).
- TEE_DATA_FLAG_ACCESS_WRITE: write access — allow the use of TEE_WriteObjectData and TEE_TruncateObjectData.

- TEE_DATA_FLAG_SHARE_WRITE: allow concurrent write access to the same object via other handles (in the same application instance or in a different one).
- TEE_DATA_FLAG_ACCESS_WRITE_META: meta access — allow the use of TEE_CloseAndDeletePersistentObject, TEE_CloseAndDeletePersistentObject1, TEE_RenamePersistentObject.

The name of the object is given by objectID, which is an array of objectIDLen bytes. The minimum value of objectIDLen is 0 and the maximum value is TEE_OBJECT_ID_MAX_LEN == 64.

Upon success, a handle to the opened object is written in *object. Upon failure, *object is set to TEE_HANDLE_NULL.

See §5.12.2.4 regarding restrictions on write access to crypto streams.

Once the application no longer needs to access the object, it must call TEE_CloseObject, or .

5.12.3.2 TEE_CreatePersistentObject

```
TEE_Result TEE_CreatePersistentObject(
                  unit32_t storageID,
                  void *objectID, uint32_t objectIDLen,
                  uint32_t flags,
                  TEE_ObjectHandle attributes,
                  void *initialData, uint32_t initialDataLen,
                  TEE_ObjectHandle *object);
```

Create a persistent object, possibly overwriting the existing one of the same name. The object contains a data stream and may contain a crypto stream.

The parameters storageID, objectID, objectIDLen and object have the same meaning as for .

The same flags are allowed as for , plus:

- TEE_DATA_FLAG_EXCLUSIVE: If there is already an object by the given name, return TEE_ERROR_ACCESS_CONFLICT. Without this flag, if there is an object by that identifier, it is overwritten.

As discussed in §5.12.2.4, if the object already exists with a crypto stream, and the object is already open with TEE_DATA_FLAG_ACCESS_WRITE but without TEE_DATA_FLAG_SHARE_WRITE, then TEE_CreatePersistentObject returns TEE_ERROR_ACCESS_CONFLICT. This is supplementary to the data stream concurrent access rules described in the description of TEE_OpenPersistentObject which also apply. Beware that in compliance with the GlobalPlatform specification, this restriction does not apply to the data stream, therefore it is possible to overwrite an object that is open without TEE_DATA_FLAG_SHARE_WRITE by calling TEE_CreatePersistentObject.

The initial content of the data stream is initialDataLen bytes read starting at the address initialData. This address may be in any memory that is accessible to the application, including shared memory. Note that an object always contains a data stream; if initialData is null, you must pass initialDataLen=0.

The content of the object's crypto stream is given by attributes, which must be an initialized transient object. It is possible to create an object without a crypto stream by passing attributes=TEE_HANDLE_NULL.

See §5.12.2.4 regarding restrictions on write access to crypto streams.

5.12.3.3 TEE_CloseAndDeletePersistentObject1

This function is supported since API level 7.

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

Atomically close the specified persistent object and delete it. The object must have been opened with the flag `TEE_DATA_ACCESS_WRITE_META`.

5.12.3.4 TEE_CloseAndDeletePersistentObject

This function is deprecated. Use instead.

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

This function is identical to , except that if an error occurs, the application is panicked.

5.12.3.5 TEE_RenamePersistentObject

This function is supported since API level 7.

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

Rename `object`, which must be a persistent object opened with the flag `TEE_DATA_ACCESS_WRITE_META`. The same constraints on newObjectID and newObjectIDLen apply as described for .

5.12.4 Persistent object enumeration functions

This part of the API is available since API level 7.

5.12.4.1 TEE_AllocatePersistentObjectEnumerator

```
TEE_Result TEE_AllocatePersistentObjectEnumerator(
                  TEE_ObjectEnumHandle *enumerator);
```

Allocate a persistent object enumerator. A handle to the enumerator is placed in `*enumerator`.

Once the application no longer needs the enumerator, it must call .

5.12.4.2 TEE_FreePersistentObjectEnumerator

```
TEE_Result TEE_FreePersistentObjectEnumerator(
                  TEE_ObjectEnumHandle enumerator);
```

Free a persistent object enumerator allocated by .

5.12.4.3 TEE_ResetPersistentObjectEnumerator

```
void TEE_AllocatePersistentObjectEnumerator(
                  TEE_ObjectEnumHandle *enumerator);
```

Reset the persistent object enumerator to its initial state. This is equivalent to freeing the enumerator and allocating a new one, but cannot run out of memory.

5.12.4.4 TEE_StartPersistentObjectEnumerator

```
TEE_Result TEE_StartPersistentObjectEnumerator(
                  TEE_ObjectEnumHandle enumerator,
                  uint32_t storageID);
```

Attach the enumerator to the specified `storageID`. After calling this function, successive calls to will return the objects that can be opened with with the same `storageID` value.

5.12.4.5 TEE_GetNextPersistentObject

```
TEE_Result TEE_GetNextPersistentObject(
                  TEE_ObjectEnumHandle enumerator,
                  TEE_ObjectInfo *info,
                  void *objectID, uint32_t *objectIDLen);
```

Return the next object in the storage area attached to the enumerator with . If all objects have already been listed, return `TEE_ERROR_ITEM_NOT_FOUND`.

If has never been called or if was called afterwards, return `TEE_ERROR_ITEM_NOT_FOUND`.

If this function is called several times in succession after attaching a storage area to the enumerator, and no file is created, renamed or deleted in the storage area, then it is guaranteed that all objects in that storage area are returned exactly once, then all subsequent calls return `TEE_ERROR_ITEM_NOT_FOUND` (until the enumerator is next reattached).

If an object is created, renamed or deleted in the storage area, no guarantee is made regarding the objects listed by this function: it is possible for objects to be missed or listed twice.

5.12.5 Persistent object data stream access functions

5.12.5.1 TEE_ReadObjectData

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

Read from `object`'s data stream starting at the current position. `object` must be a persistent object opened with the flag `TEE_DATA_ACCESS_READ`. The data is placed in memory starting at `buffer`, which may point to shared memory.

This function reads `size` bytes, or up to the end of the data stream, whichever is smaller. The number of bytes that have been read is placed in `*count`. The position of the data stream is advanced by the same number.

5.12.5.2 TEE_WriteObjectData

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

Write `size` bytes to `object`'s data stream starting at the current position. `object` must be a persistent object opened with the flag `TEE_DATA_ACCESS_WRITE`. The data read from memory starting at `buffer`, which may point to shared memory.

If this function succeeds, the position of the data stream is advanced by `size`.

If the initial position is past the end of the data stream, null bytes are first written from the end of the stream to the initial position.

If this function fails, the content of the object and the position are not modified.

5.12.5.3 TEE_TruncateObjectData

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

Arrange for the data stream of `object` to contain exactly `size` bytes. `object` which must be a persistent object opened with the flag `TEE_DATA_ACCESS_WRITE`.

If the data stream initially contains less than `size` bytes, it is padded at the end with null bytes. If the data stream initially contains more than `size` bytes, bytes at offsets `size` and beyond are deleted. The position in the object handle is not modified.

If the operation fails, the object is unchanged.

5.12.5.4 TEE_SeekObjectData

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

Change the data stream position in the object handle `object`.

- If `whence==TEE_DATA_SEEK_SET`, set the position to `offset`.
- If `whence==TEE_DATA_SEEK_CUR`, set the position to `offset` plus the current position.
- If `whence==TEE_DATA_SEEK_END`, set the position to `offset` plus the current data stream size.

It is not an error to seek past the end of the data stream. Reads past the end of the stream return no data; writes past the end of the stream pad it with null bytes.

If the final position is negative, it is forced to 0.

If the final position is 2^{32} or greater, the position is not changed and this function returns `TEE_ERROR_OVERFLOW`.

6 mcClient API

The mcClient API is the Legacy API for developing Client Applications.

Note that the same API is also available at the kernel-level for kernel modules.

6.1 HEADER FILE

The header file for the mcClient API is "MobiCoreDriverApi.h".

```
#include "MobiCoreDriverApi.h"
```

6.2 DATA STRUCTURES

6.2.1 mcBulkMap_t Structure Reference

```
typedef struct {
    void *sVirtualAddr;
    uint32_t sVirtualLen;
} mcBulkMap_t;
```

Information structure about additional mapped Bulk buffer between the Trusted Application Connector (Nwd) and the Trusted Application (Swd). This structure is initialized from a Trusted Application Connector by calling mcMap(). In order to use the memory within a Trusted Application the Trusted Application Connector has to inform the Trusted Application about the content of this structure via the TCI.

The fields of the structure are:

- sVirtualAddr: The virtual address of the Bulk buffer regarding the address space of the Trusted Application, already includes a possible offset!
- sVirtualLen: Length of the mapped Bulk buffer

6.2.2 mcSessionHandle_t Structure Reference

Structure of Session Handle, includes the Session ID and the Device ID the Session belongs to. The session handle will be used for session-based Kinibi communication.

It will be passed to calls which address a communication end point in the Kinibi environment.

```
typedef struct {
    uint32_t sessionId;
    uint32_t deviceId;
} mcSessionHandle_t;
```

The fields of the structure are:

- sessionId: session ID

- deviceId: Device ID the session belongs to

6.3 ERROR CODES

Macro Name	Definition
MC_DRV_ERROR_MAJOR(ecode)	Get MAJOR part of error code.
MC_DRV_ERROR_MCP(ecode)	Get MCP part of error code.
MC_DRV_ERROR_DETAIL(ecode)	Get detail part of error code.

Table 16 Trusted Application API Error Codes

Constant Name	Definition
MC_DRV_OK	Function call succeeded.
MC_DRV_NO_NOTIFICATION	No notification available.
MC_DRV_ERR_NOTIFICATION	Error during notification on communication level.
MC_DRV_ERR_NOT_IMPLEMENTED	Function not implemented.
MC_DRV_ERR_OUT_OF_RESOURCES	No more resources available.
MC_DRV_ERR_INIT	Driver initialization failed.
MC_DRV_ERR_UNKNOWN	Unknown error.
MC_DRV_ERR_UNKNOWN_DEVICE	The specified device is unknown.
MC_DRV_ERR_UNKNOWN_SESSION	The specified session is unknown.
MC_DRV_ERR_INVALID_OPERATION	The specified operation is not allowed.
MC_DRV_ERR_INVALID_RESPONSE	The response header from the MC is invalid.
MC_DRV_ERR_TIMEOUT	Function call timed out.
MC_DRV_ERR_NO_FREE_MEMORY	Cannot allocate additional memory.
MC_DRV_ERR_FREE_MEMORY_FAILED	Free memory failed.
MC_DRV_ERR_SESSION_PENDING	Still some open sessions pending.
MC_DRV_ERR_DAEMON_UNREACHABLE	MC daemon not reachable
MC_DRV_ERR_INVALID_DEVICE_FILE	The device file of the kernel module could not be

	opened.
MC_DRV_ERR_INVALID_PARAMETER	Invalid parameter.
MC_DRV_ERR_KERNEL_MODULE	Error from Kernel Module, see DETAIL for more.
MC_DRV_ERR_BULK_MAPPING	Error during mapping of additional bulk memory to session.
MC_DRV_ERR_BULK_UNMAPPING	Error during un-mapping of additional bulk memory to session.
MC_DRV_INFO_NOTIFICATION	Notification received, exit code available.
MC_DRV_ERR_NQ_FAILED	Setup of NWd connection failed.
MC_DRV_ERR_DAEMON_VERSION	Wrong daemon version.
MC_DRV_ERR_CONTAINER_VERSION	Wrong container version.
MC_DRV_ERR_WRONG_PUBLIC_KEY	System Trusted Application public key is wrong.
MC_DRV_ERR_CONTAINER_TYPE_MISMATCH	Wrong container type(s).
MC_DRV_ERR_CONTAINER_LOCKED	Container is locked (or not activated).
MC_DRV_ERR_SP_NO_CHILD	SPID is not registered with root container.
MC_DRV_ERR_TL_NO_CHILD	UUID is not registered with SP container.
MC_DRV_ERR_UNWRAP_ROOT_FAILED	Unwrapping of root container failed.
MC_DRV_ERR_UNWRAP_SP_FAILED	Unwrapping of service provider container failed.
MC_DRV_ERR_UNWRAP_TRUSTLET_FAILED	Unwrapping of Trusted Application container failed.
MC_DRV_ERR_DEVICE_ALREADY_OPEN	Device is already open.
MC_DRV_ERR_SOCKET_CONNECT	MC daemon socket not reachable.
MC_DRV_ERR_SOCKET_WRITE	MC daemon socket write error.
MC_DRV_ERR_SOCKET_READ	MC daemon socket read error.
MC_DRV_ERR_SOCKET_LENGTH	MC daemon socket read error.
MC_DRV_ERR_DAEMON_SOCKET	MC daemon had problems with socket.
MC_DRV_ERR_DEVICE_FILE_OPEN	The device file of the kernel module could not be opened.
MC_DRV_ERR_NULL_POINTER	Null pointer passed as parameter.
MC_DRV_ERR_TCI_TOO_BIG	Requested TCI length is too high.

MC_DRV_ERR_WSM_NOT_FOUND	Requested TCI was not allocated with mallocWsm().
MC_DRV_ERR_TCI_GREATER_THAN_WSM	Requested TCI length is bigger than allocated WSM.
MC_DRV_ERR_TRUSTLET_NOT_FOUND	Trusted Application could not be found in mcRegistry.
MC_DRV_ERR_DAEMON_KMOD_ERROR	Daemon cannot use Kernel module as expected.
MC_DRV_ERR_DAEMON_MCI_ERROR	Daemon cannot use MCI as expected.
MC_DRV_ERR_MCP_ERROR	Control Protocol error. See MC_DRV_ERROR_MCP().
MC_DRV_ERR_INVALID_LENGTH	Invalid length.
MC_DRV_ERR_KMOD_NOT_OPEN	Device not open.
MC_DRV_ERR_BUFFER_ALREADY_MAPPED	Buffer is already mapped to this Trusted Application.
MC_DRV_ERR_BLK_BUFF_NOT_FOUND	Unable to find internal handle for buffer.
MC_DRV_ERR_DAEMON_DEVICE_NOT_OPEN	No device associated with connection.
MC_DRV_ERR_DAEMON_WSM_HANDLE_NOT_FOUND	Daemon could not find wsm.h
MC_DRV_ERR_DAEMON_UNKNOWN_SESSION	The specified session is unknown by the daemon
MAKE_MC_DRV_MCP_ERROR(mcpCode)	Macro used to build a MCP Error Code
MAKE_MC_DRV_KMOD_WITH_ERRNO(theErrno)	Macro used to build a Kernel Module Error Code
MC_DEVICE_ID_DEFAULT	The default device ID
MC_INFINITE_TIMEOUT	Wait infinite for a response of the MC.
MC_NO_TIMEOUT	Do not wait for a response of the MC.
MC_MAX_TCI_LEN	TCI/DCI must not exceed 1MiB

Table 17 mcClient API Constants

6.4 FUNCTIONS

6.4.1 mcOpenDevice

```
__MC_CLIENT_LIB_API mcResult_t mcOpenDevice (uint32_t deviceId)
```

Initializes all device specific resources required to communicate with an Kinibi instance located on the specified device in the system. If the device does not exist the function will return `MC_DRV_ERR_UNKNOWN_DEVICE`.

A mutex is locked during the execution to avoid concurrent accesses.

Parameters:

- in deviceId: Identifier for the Kinibi device to be used. MC_DEVICE_ID_DEFAULT refers to the default device.

Return:

- MC_DRV_OK if operation has been successfully completed.
- MC_DRV_ERR_DAEMON_UNREACHABLE when problems with daemon occur.
- MC_DRV_ERR_UNKNOWN_DEVICE when device Id is unknown.
- MC_DRV_ERR_INVALID_DEVICE_FILE if kernel module under /dev/mobicore-user cannot be opened

6.4.2 mcCloseDevice

```
__MC_CLIENT_LIB_API mcResult_t mcCloseDevice (uint32_t deviceId)
```

Close the connection to a Kinibi device.

When closing a device, active sessions have to be closed beforehand. Resources associated with the device will be released. The device may be opened again after it has been closed.

A mutex is locked during the execution to avoid concurrent accesses.

Parameters:

- in deviceId: Identifier for the Kinibi device to be used. MC_DEVICE_ID_DEFAULT refers to the default device.

Return:

- MC_DRV_OK if operation has been successfully completed.
- MC_DRV_ERR_UNKNOWN_DEVICE when device Id is unknown.
- MC_DRV_ERR_SESSION_PENDING when a session is still open.
- MC_DRV_ERR_DAEMON_UNREACHABLE when problems with daemon occur.

6.4.3 mcOpenTrustlet – for OTAv1 only

```
__MC_CLIENT_LIB_API mcResult_t mcOpenTrustlet(
    mcSessionHandle_t*      session,
    mcSpid_t                spid,
    uint8_t*                trustlet,
    uint32_t                tLen,
    uint8_t*                tci,
    uint32_t                tciLen
);
```

Opens a new session to a Service Provider Trusted Application.

`session.deviceId` must be set to the device id of a device opened with a call to `mcOpenDevice`.

The `trustlet` memory buffer must contain the Trusted Application encoded binary.

The `spid` structure must be filled to indicate the ID of the Service Provider of the Trusted Application. This parameter is ignored for System Trusted Applications.

The caller must allocate the `tci` communication buffer prior to calling this function. This buffer must not be freed until the session is closed through a call to `mcCloseSession`.

When this function returns MC_DRV_OK the `session` structure has been populated with any implementation-defined information necessary for subsequent operations within the session.

A mutex is locked during the execution to avoid concurrent accesses.

Parameters:

- in, out `session`: Before calling the function `session.deviceId` has to be filled with the device id of a previously opened device. On success, the required fields will be filled.
- in `spid`: Service Provider ID. Ignored for System Trusted Applications
- in `trustlet` : memory buffer containing the Trusted Application binary
- in `tlen` : length of the memory buffer containing the Trusted Application
- in `tci`: Communication buffer for communicating with the Trusted Application.
- in `tciLen`: Length of the TCI buffer. Maximum allowed value is `MC_MAX_TCI_LEN`.

Return:

- `MC_DRV_OK` if operation has been successfully completed.
- `MC_DRV_INVALID_PARAMETER` if session parameter is invalid.
- `MC_DRV_ERR_UNKNOWN_DEVICE` when device id is invalid.
- `MC_DRV_ERR_DAEMON_UNREACHABLE` when problems with daemon socket occur.
- `MC_DRV_ERR_UNKNOWN_DEVICE` when daemon returns an error.

6.4.4 mcOpenSession – for System TAs only

```
__MC_CLIENT_LIB_API mcResult_t mcOpenSession (
     mcSessionHandle_t*    session,
     const mcUuid_t*       uuid,
     uint8_t*              tci,
     uint32_t              tciLen)
```

Open a new session to a System Trusted Application.

The Trusted Application with the given UUID must be available in the flash filesystem.

Write MCP open message to buffer and notify Kinibi about the availability of a new command. Waits till the Kinibi responds with the new session ID (stored in the MCP buffer).

A mutex is locked during the execution to avoid concurrent accesses.

Parameters:

- in, out session: On success, the session data will be returned. Note that session.deviceId has to be the device id of an opened device.
- in uuid: UUID of the Trusted Application to be opened
- in tci: TCI buffer for communicating with the Trusted Application.
- in tciLen: Length of the TCI buffer. Maximum allowed value is MC_MAX_TCI_LEN.

Return:

- MC_DRV_OK if operation has been successfully completed.
- MC_DRV_INVALID_PARAMETER if session parameter is invalid.
- MC_DRV_ERR_UNKNOWN_DEVICE when device id is invalid.
- MC_DRV_ERR_DAEMON_UNREACHABLE when problems with daemon socket occur.
- MC_DRV_ERR_UNKNOWN_DEVICE when daemon returns an error.

6.4.5 mcCloseSession

```
__MC_CLIENT_LIB_API mcResult_t mcCloseSession (
    mcSessionHandle_t *session)
```

Close a Trusted Application session.

Close the specified Kinibi session. The call will block until the session has been closed.

A mutex is locked during the execution to avoid concurrent accesses.

Precondition:

Device deviceId has to be opened in advance.

Parameters:

- in session: pointer to the session to be closed.

Return:

- MC_DRV_OK if operation has been successfully completed.
- MC_DRV_INVALID_PARAMETER if session parameter is invalid.
- MC_DRV_ERR_UNKNOWN_SESSION when session id is invalid.
- MC_DRV_ERR_UNKNOWN_DEVICE when device id of session is invalid.
- MC_DRV_ERR_DAEMON_UNREACHABLE when problems with daemon occur.
- MC_DRV_ERR_INVALID_DEVICE_FILE when daemon cannot open Trusted Application file.

6.4.6 mcNotify

```
__MC_CLIENT_LIB_API mcResult_t mcNotify (mcSessionHandle_t *session)
```

Notifies the session end point about available message data. If the session parameter is correct, notify will always succeed. Corresponding errors can only be received by mcWaitNotification().

Precondition:

A session has to be opened in advance.

Parameters:

- in session: pointer to the session to notify.

Return:

- MC_DRV_OK if operation has been successfully completed.
- MC_DRV_INVALID_PARAMETER if session parameter is invalid.
- MC_DRV_ERR_UNKNOWN_SESSION when session id is invalid.
- MC_DRV_ERR_UNKNOWN_DEVICE when device id of session is invalid.

6.4.7 mcWaitNotification

```
__MC_CLIENT_LIB_API mcResult_t mcWaitNotification (
     mcSessionHandle_t*    session,
     int32_t               timeout)
```

Wait for a notification issued by the Kinibi for a specific session. The timeout parameter specifies the number of milliseconds the call will wait for a notification. If the caller passes 0 as timeout value the call will immediately return. If timeout value is below 0 the call will block until a notification for the session has been received.

Attention:

If timeout is below 0, call will block: Caller has to trust the other side to send a notification to wake him up again.

Parameters:

- in session: pointer to the session which receives the notification.
- in Timeout: Time in milliseconds to wait (MC_NO_TIMEOUT : direct return, >0 : milliseconds, MC_INFINITE_TIMEOUT : wait infinitely)

Returns:

- MC_DRV_OK if notification is available.
- MC_DRV_ERR_TIMEOUT if no notification arrived in time.
- MC_DRV_INFO_NOTIFICATION if a problem with the session was encountered. Get more details with mcGetSessionErrorCode().
- MC_DRV_ERR_NOTIFICATION if a problem with the socket occurred.
- MC_DRV_INVALID_PARAMETER if a parameter is invalid.
- MC_DRV_ERR_UNKNOWN_SESSION when session id is invalid.
- MC_DRV_ERR_UNKNOWN_DEVICE when device id of session is invalid.

6.4.8 mcMallocWsm – DEPRECATED

```
__MC_CLIENT_LIB_API mcResult_t mcMallocWsm (
      uint32_t  deviceId,
      uint32_t  align,
      uint32_t  len,
      uint8_t** wsm,
      uint32_t  wsmFlags)
```

This function is deprecated. Standard malloc() and free () functions should be used instead.

Allocate a block of world shared memory (WSM).

The MC driver allocates a contiguous block of memory which can be used as WSM.

Always returns a buffer aligned to 4KB.

Parameters:

- in deviceId: The ID of an opened device to retrieve the WSM from.
- in align: This parameter is ignored.
- in len: Length of the block in bytes. It does not have to be a multiple of 4KB.
- out **wsm: pointer to the virtual address of the world shared memory block.
- in wsmFlags: Platform specific flags describing the memory to be allocated.

Attention:

align and wsmFlags fields are currently ignored.

Returns:

- MC_DRV_OK if operation has been successfully completed.
- MC_DRV_INVALID_PARAMETER if a parameter is invalid.
- MC_DRV_ERR_UNKNOWN_DEVICE when device id is invalid.
- MC_DRV_ERR_NO_FREE_MEMORY if no more contiguous memory is available in this size or for this process.

6.4.9 mcFreeWsm – DEPRECATED

```
__MC_CLIENT_LIB_API mcResult_t mcFreeWsm (
      uint32_t deviceId,
      uint8_t* wsm)
```

This function is deprecated. Standard malloc() and free () functions should be used instead.

Free a block of world shared memory (WSM).

The MC driver will free a block of world shared memory (WSM) previously allocated with mcMallocWsm(). The caller has to assure that the address handed over to the driver is a valid WSM address.

A mutex is locked during the execution to avoid concurrent access to the shared memory being freed.

Parameters:

- in deviceId: The address to which the given address belongs.
- in wsm: Address of WSM block to be freed.

Returns:

- MC_DRV_OK if operation has been successfully completed.
- MC_DRV_INVALID_PARAMETER if a parameter is invalid.
- MC_DRV_ERR_UNKNOWN_DEVICE when device id is invalid.
- MC_DRV_ERR_FREE_MEMORY_FAILED on failures.

6.4.10mcMap

```
__MC_CLIENT_LIB_API mcResult_t mcMap (
     mcSessionHandle_t*    session,
     void*                 buf,
     uint32_t              len,
     mcBulkMap_t*          mapInfo)
```

Map additional bulk buffer between a Trusted Application Connector (TLC) and the Trusted Application (TL) for a session.

Memory allocated in user space of the TLC can be mapped as additional communication channel (besides TCI) to the Trusted Application. Limitation of the Trusted Application memory structure applies: only 4 chunks can be mapped with a maximum chunk size of 1 MB each.

A mutex is locked during the execution to avoid concurrent access to the shared memory being freed.

Attention:

It is up to the application layer (TLC) to inform the Trusted Application about the additional mapped bulk memory.

Parameters:

- in session: pointer to the session with information of the deviceId used with the sessionId. The given buffer is mapped to the session specified in the session-Handle.
- in buf: Virtual address of a memory portion (relative to TLC) to be shared with the Trusted Application, already includes a possible offset!
- in len: Length of the block in bytes.
- out mapInfo: Information structure about the mapped Bulk buffer between the TLC (Nwd) and the TL (Swd).

Returns:

- MC_DRV_OK if operation has been successfully completed.
- MC_DRV_INVALID_PARAMETER if a parameter is invalid.
- MC_DRV_ERR_UNKNOWN_SESSION when session id is invalid.
- MC_DRV_ERR_UNKNOWN_DEVICE when device id of session is invalid.
- MC_DRV_ERR_DAEMON_UNREACHABLE when problems with daemon occur.
- MC_DRV_ERR_BULK_MAPPING when buf is already used as bulk buffer or when registering the buffer failed.

6.4.11 mcUnmap

```
__MC_CLIENT_LIB_API mcResult_t mcUnmap (
      mcSessionHandle_t*    session,
      void*                 buf,
      mcBulkMap_t*          mapInfo)
```

Remove additional mapped bulk buffer between Trusted Application Connector (TLC) and the Trusted Application (TL) for a session.

Attention:

The bulk buffer will immediately be unmapped from the session context. The application layer (TLC) must inform the TL about un-mapping of the additional bulk memory before calling mcUnmap!

Parameters:

- in session: pointer to the session with information of the deviceId and the sessionId. The given buffer is mapped to the session specified in the session-Handle.
- in buf: Virtual address of a memory portion (relative to TLC) to be shared with the Trusted Application, already includes a possible offset!
- in mapInfo: Information structure about the mapped Bulk buffer between the TLC (Nwd) and the TL (Swd).

Attention:

The clientlib currently ignores the len field in mapInfo.

Returns:

- MC_DRV_OK if operation has been successfully completed.
- MC_DRV_INVALID_PARAMETER if a parameter is invalid.
- MC_DRV_ERR_UNKNOWN_SESSION when session id is invalid.
- MC_DRV_ERR_UNKNOWN_DEVICE when device id of session is invalid.
- MC_DRV_ERR_DAEMON_UNREACHABLE when problems with daemon occur.
- MC_DRV_ERR_BULK_UNMAPPING when buf was not registered earlier or when unregistering failed.

6.4.12 mcGetSessionErrorCode

```
__MC_CLIENT_LIB_API mcResult_t mcGetSessionErrorCode (
    mcSessionHandle_t*    session,
    int32_t*              lastErr)
```

Get additional error information of the last error that occurred on a session.

After the request the stored error code will be deleted.

Parameters:

- in session: pointer to the session.
- out lastErr: Pointer to the last error in given session:
  - If >0 Trusted Application has terminated itself with this value,
  - If <0 Trusted Application is dead because of an error within the Kinibi (e.g. Kernel exception). See also notificationPayload_t enum in MCI definition at "mcinq.h".

Returns:

- MC_DRV_OK if operation has been successfully completed.
- MC_DRV_INVALID_PARAMETER if a parameter is invalid.
- MC_DRV_ERR_UNKNOWN_SESSION when session id is invalid.
- MC_DRV_ERR_UNKNOWN_DEVICE when device id of session is invalid.

6.4.13 mcGetMobiCoreVersion

```
__MC_CLIENT_LIB_API mcResult_t mcGetMobiCoreVersion (
      uint32_t         deviceId,
      mcVersionInfo_t* versionInfo)
```

Parameters:

- in deviceId: the Kinibi deviceId.
- out versionInfo: The Kinibi version information.

7 Extended Class for Android

7.1 CLASS TEECLIENT

7.1.1 Description

On some specific devices, the security of Android is enforced and applications are not allowed to access to the TEE.

An additional Trustonic Service (Security Activator) must be downloaded from the Google Play Store in order to allow this access.

The TeeClient is a java class which provides an API to facilitate the access to the TEE thought this Trustonic Service.

To use the TeeClient class, please import TeeClient.aar into your android project.

7.1.1.1 Constructor

```
public TeeClient(Context applicationContext)
```

Creates a new TeeClient object with application context

Parameters :

- in applicationcontext : android.app.Context object

7.1.1.2 Method Summary

type	Method Summary
void	mcOpenDevice(int deviceId)
void	mcCloseDevice(int deviceId)
void	launchPlayStoreOnTeeProxyService(Activity activity)
boolean	isTeeProxyServiceInstalled()
boolean	isTuiAvailable()
EnumSet<TEEError>	getDeviceErrata()

7.1.2 Method Detail

7.1.2.1 TeeClient.mcOpenDevice

```
public void mcOpenDevice(int deviceId) throws TeeException
```

Java API for the mcOpenDevice mcClient API native function (see mcOpenDevice).

Parameters :

- in deviceId: Identifier for the Kinibi device to be used. MC_DEVICE_ID_DEFAULT refers to the default device.

Throws :

- TeeException – if mcOpenDevice failed.

7.1.2.2 TeeClient.mcCloseDevice

```
public void mcCloseDevice(int deviceId) throws TeeException
```

Java API for the mcCLoseDevice mcClient API native function (see mcCloseDevice).

Parameters :

- in deviceId: Identifier for the Kinibi device to be used. MC_DEVICE_ID_DEFAULT refers to the default device.

Throws :

- TeeException – if mcCloseDevice failed.

7.1.2.3 TeeClient.launchPlayStoreOnTeeProxyService

```
public void launchPlayStoreOnTeeProxyService(Activity activity)
```

Starts the Google Play Store application on the TeeProxyService (ie: Security Activator) download page.

Parameters :

- in activity: the activity of the calling application

7.1.2.4 TeeClient.isTeeProxyServiceInstalled

```
public boolean isTeeProxyServiceInstalled()
```

Checks if the TeeProxyService (ie: Security Activator) is installed in the device.

Return :

- Returns true if Proxy service is installed and running, otherwise returns false.

7.1.2.5 TeeClient. isTuiAvailable

```
public boolean isTuiAvailable ()
```

Checks if the Trusted User Interface (TUI) service is installed in the device.

Return :

- Returns true if TUI service is installed and running, otherwise returns false.

7.1.2.6 TeeClient.getDeviceErrata

```
public EnumSet<TEEError> getDeviceErrata()
```

Returns whether TEE is installed or not on this device. TeeClient object should be initiated before calling.

Return :

- TEE_NOT_SUPPORTED : TEE not supported on this device
- TEE_PROXY_INSTALLATION_REQUIRED: TeeProxyService is required on this device.
- TEE_PROXY_LICENSE_REQUIRED : License is needed to use TeeProxyService.
- TEE_PROXY_UNREACHABLE_ERROR : Not allowed to use TeeProxyService

- if the EnumSet returned is empty, it means that the TEE can be used on this device".

Example:

```
TeeClient tee = new TeeClient(this.getApplicationContext());

//KinibiChecker
try {
       EnumSet<TEEError> issues = tee.getDeviceErrata();

       if (!issues.isEmpty())
            Log.i(LOG_TAG, "Error");
       else
            Log.i(LOG_TAG, "TEE is supported");

} catch (LinkageError e) {
       Log.i(LOG_TAG, e.getMessage());
}
```