GLOBALPLATFORM™
THE STANDARD FOR MANAGING APPLICATIONS ON SECURE CHIP TECHNOLOGY

GlobalPlatform Device Technology
TEE Internal Core API Specification
Version 1.1.0.26 – Public Review
Please send to: tee-int-core-api-review@globalplatform.org

COMPANY: Open-TEE project (https://github.com/Open-TEE)
Name: Tanel Dettenborn, Brian McGillion
Date: 22.05.2014

#	Page#	Chapter / Paragraph (*)	Type(**)	Comment (Problem & Reason)	Proposed Resolution
1	80	4.11	T	New function. If TA maximum memory is defined, there should be a function for querying how much free memory there is available for a TA. When a TA knows the available memory, it could determine next available action.	Add new function: - uint32_t TEE_GetMemoryAvailable(); Return value is available memory as bytes.
2	89	5.1 § 2 – line 2	T	Unnecessary complexity. Persistent object 0-byte identifier length is a special case (perspective of TEE implementation) and gaining benefits over 0-byte length is arguable.	Persistent object identifier length is from 1 to 64 bytes.
3	90	5.1 § 7 – line 4	T	Unnecessary functionality. When TEE_ERROR_CORRUPT_OBJECT(_2) return code is returned; any attempt at continuing to use the corrupted object handle will cause a panic. This should not happen if the closed object handle is set to null. Use of a null handle might cause panic, but that panic is not related to corrupted handle. Related also #21.	Remove sentence “subsequent use of the handle SHALL cause a panic.”
4	90	5.1	T	Useful functionality. Persistent object might not be corrupt, when TEE_ERROR_CORRUPT_OBJECT is returned. For example the object handle might get corrupted yet the persistent object in storage is unaffected. The same could also occur with a reading error.	Introduce new return code TEE_ERROR_CORRUPT_HANDLE. Return code is only used with persistent object and only if persistent object data at storage is not corrupt. In such a case the handle is not closed nor persistent object deleted from storage. It would then be at the callers discretion as to how best to proceed, e.g. the user could close the object and re-open it. A new return code could be defined, for example, TEE_GetObjectInfo function (#6).
5	96	5.5.1 § 2	T	Ambiguous definition. The *KeySize* variable needs a more precise definition. If object size is terminated by object attributes, the size can be greater than *maxKeySize* variable. For example RSA-key consists of multiple components and lets assume that RSA modulo is equal to *maxKeySize*.	Guessing. *KeySize* variable is representing current key strength and is determined for example by RSA modulo length? Add clarification: *keySize*: Representing current key strength.

#	Page#	Chapter / Paragraph (*)	Type(**)	Comment (Problem & Reason)	Proposed Resolution
				Which leads inevitably to *keySize* > *maxKeySize*.	
6	96	5.5.1 § 3	E	Deprecated naming. Variable *maxKeySize* name is not self explaining.	Add further clarification for example: (bullet) *maxKeySize*: Representing object cryptographic key maximal strength.
7	99	5.5.3 § 1	T	Uniform and useful functionality. TEE_GetObjectValueAttribute allows for parameters *a* and *b to* be null, TEE_GetObjectBufferAttribute *buffer* and *size* should also be allowed to be null.	It is a security risk, if TEE_GetObjectBufferAttribute *size* parameter is null and *buffer* parameter is not (possibility of buffer overflow). If the caller wishes to check if the object contains an attribute both buffer and size could be null. If *buffer* parameter is null and the size is not, the *size* could contain the required length of the buffer on return, to allow the caller to allocate the required space for the buffer (if attributes are found). Also useful for just checking the attribute length.
8	99	5.5.3	T	Ambiguous definition. If the object is not initialized, the TA panics. This panic reason overlaps the return code TEE_ERROR_ITEM_NOT_FOUND, because if the object is not initialized then nothing is not be found either.	If object is not initialized, function could return code TEE_ERROR_ITEM_NOT_FOUND. Thus improving usability.
9	100	5.5.4	T	Ambiguous definition. Apply same conclusion as point #8.	Apply same proposition as point #8.
10	102	5.6.1	T	General functionality. Apply same conclusion as point #12.	Apply same proposition as point #12.
11	101	5.5.5	T	Unrealistic requirements. TEE_CloseObject is a void function, it is unreasonable to assume that the close operation can always complete without incident. This leads to the persistent object always being in “closed” state. Most close operations also flush any remaining buffers but this function as defined would only allow for the freeing of the object handle from the TA's memory.	Function should have return values: -TEE_SUCCESS: In case of success. -TEE_ERROR_CORRUPT_OBJECT: If the persistent object is corrupt.
12	102	5.6	T	General functionality. Requirement for allocating all resources in TEE_AllocateTransientObject function is setting up unfeasible and resource wasting transient objects. - Infeasible: Apparently you should allocate all your resources here and in effect this means that it should also,	Transient object allocation should be divided. TEE_AllocateTransientObject function should allocate a new object handler and meta structures for the expected attributes. TEE_PopulateTransientObject and

#	Page#	Chapter / Paragraph (*)	Type(**)	Comment (Problem & Reason)	Proposed Resolution
				for example, allocate all temporary buffers which would be needed by other functions. - Infeasible: You cannot use third party solution. Because this solution could use, for example, temporary buffers. - Resource wasting: If an object is not initialized, it is only wasting resources. - Resource wasting: If object is initialized with weaker key and this generally means less space for key, but object is allocated according to it *maxKeySize* → Resource waste - Resource wasting: Attribute size can not be predicted exactly (can not be assumed it is used in "standardized" way). For example if one is calculating RSA modulo from different sized prime numbers. One of the used primes could be smaller than that other but both are allocated according to *maxKeySize*. - Generally: The benefits gained by have a single allocation point and the complexity of implementation are questionable. There is no guarantee that the operation will not fail due to lack of resources. - Generally: This approach is inflexible going forward. The allocating function will become large and overly complex, and it still requires special functionality in other functions	TEE_GenerateKey functions should allocate reference attribute buffers and populate them according to the received parameters or generated key. TEE_PopulateTransientObject and TEE_GenerateKey functions need to have a new return code TEE_ERROR_OUT_OF_MEMORY added to support this added functionality. Also it should be noted that TEE_CopyObjectAttributes will also need to be able to return TEE_ERROR_OUT_OF_MEMORY if the situation arises. Note: This is only an overview of the problem. If you required, we can draft a more detailed version that is in line with the overall specification.
13	102	5.6	T	Useful functionality. See first #14. This point is only valid if TEE_PopulateTransientObject attributes are copied.	Proposing new object handle flag: TEE_HANDLE_FLAG_REFERENCE. Flag can only be set to transient object and only prior initialization. If flag is set, the object attributes are not copied during a call to TEE_PopulateTransientObject, instead the ownership of the attributres is transferred to the transient object. This prevents a double allocation for the same object. Of course attribute length should be consistent with object maxKeysize.
14	107	5.6.4	T	Undefined behavior. TEE_PopulateTransientObject does not define how parameters should be handled. Should parameters in *attrs*-arrays be deep copied (including the reference attribute buffer) or should a shallow reference be created? If taking only a reference, should the ownership be	In the function specification precise language e.g. "deep copy" / "transfer ownership" should be used.

#	Page#	Chapter / Paragraph (*)	Type(**)	Comment (Problem & Reason)	Proposed Resolution
				transferred?	
15	107	6.6.4	T	Unnecessary functionality. If parameter *attr* contains an unexpected parameter, this will force a panic.	Remove panic reason and allow extra attributes in the attribute array. Additional benefit of proposition is that the caller could use the same attribute array in different populate/generate calls. TEE_PopulateTransientObject could then only copy necessary parameters thus aiding usability and efficiency.
16	111	5.6.6	T	General functionality. Apply same conclusion as point #12.	If destination object is consistent with source object, destination object attributes are copied (reference attribute buffers are allocated prior to copy). The function will require the possibility to return TEE_ERROR_OUT_OF_MEMORY.
17	111	5.6.6	E	Clarification. The specification should point out the curiosity that the destination object key strength should only be equal to or less than source object. Never greater.	Note or warning that copied object is weaker.
18	113	5.6.7	T	General functionality. Apply same conclusion as point #12.	Apply same proposition as point #12.
19	118	5.7.2	E (T)	Unnecessary complexity. See also #2. What could be practical use case for 0-byte ID length?	Apply same proposition as point #2.
20	124	5.7.5	E (T)	Unnecessary complexity. Apply same conclusion as point #19.	Apply same proposition as point #19.
21	125	5.8	T	Useful error. It is not known beforehand what is the size requirement when allocating an enumerator. Because enumeration is not started during the allocation. It would be more appropriate for a TEE implementation if you can allocate memory in TEE_StartPersistentObjectEnumeration. This could allow optimization of the TEE implementation.	Should allow TEE_ERROR_OUT_OF_MEMORY in TEE_StartPersistentObjectEnumerationfunction. Function specification should point out that extra resource allocation is not recommended.
22	125	5.8	T	New functionality. Removing a persistent object from storage is wasting resources. To remove a persistent object user must allocate/open persistent object and then remove persistent object. Removing object from storage in a more efficiency way should be available, if for example TA is cleaning up its storage.	New function: - TEE_DeletePersistentObject(void *objectID, uint32_t objectIDLen); Function to remove persistent object from storage. Prior to deletion this function MUST check object access rights and MUST be “granted” meta

#	Page#	Chapter / Paragraph (*)	Type(**)	Comment (Problem & Reason)	Proposed Resolution
					access right. - Return TEE_SUCCESS if meta access right was granted and object deleted from storage. - Return TEE_ERROR_ACCESS_CONFLICT if access right conflict was detected. - TEE_ERROR_ITEM_NOT_FOUND if persistent object not found.
23	132	5.9.2	T	Useful requirement. Function specifies that a write operation should be atomic and that is a justifiable requirement. But it is hard to leverage in a TEE implementation if arbitrary block writing is not required to be atomic.	Define block size that must be atomically written to storage. For example TEE must guarantee that up to 4k block is written atomically and blocks over 4k is written non atomically. Non-atomically written data must be monitored to detect if the write operation failed and return code TEE_ERROR_OBJECT_CORRUPTED.
24	137	6.1	T	Useful constant. It could be useful if there would be a constant for operation state. Defining constant would make specification more readable. TEE implementation does not have to be aware, is 1 or 0 active state.	Make a subsection 6.2 Constant to chapter 6.1 and define TEE_OPERATION_STATE_ACTIVE = 1, TEE_OPERATION_STATE_INITIAL = 0 Change explanation at page 146, chapter 6.2.4 bulletin 7: operationState: Filled with operation state
25	138	6.1.3	T	Possible error. The size of *KeyInformation*-array is hard coded to one, but if an algorithm requires multiple keys, for example AES-XTS the array would not be large enough to support this requirement.	Hard code array size according to its biggest size. The biggest size is determined by which algorithm requires most keys. Currently this is the two key requirement of AES-XTS.
26	139	6.2	T	Unfeasible requirements. Apply same conclusion as point #12.	Operation allocation should be divided. TEE_AllocateOperation should allocate a new operation handler and meta structures for expected attributes. TEE_SetOperationKey(2) should allocate reference attribute buffers and copy attributes according to received parameters. The TEE_SetOperationKey(2) functions would require the ability to return TEE_ERROR_OUT_OF_MEMORY if such a situation is encountered.. It should also be noted that TEE_CopyOperation will also need the ability to return

#	Page#	Chapter / Paragraph (*)	Type(**)	Comment (Problem & Reason)	Proposed Resolution
					TEE_ERROR_OUT_OF_MEMORY.
27	146	6.2.4	E(T)	Undefined behavior. Function specification does not specify how operation size should be calculated (uint32_t *operationSize as function parameter). Should operation size be determined by Attributes? Attributes + operation handle? Else?	Specify exactly what is the required behavior.
28	144-147	6.2.3– 6.2.4	T	Redundancy. TEE_GetOperationInfo is subset (almost) from TEE_GetOperationInfoMultiple function. There is no point of two functions that fulfill the same task. In addition to this useful variables are scattered between both functions. Operation state could be useful at TEE_GetOperationInfo.	As a naming convention TEE_GetOperationInfo is more descriptive name than TEE_GetOperationInfoMultiple. Therefore TEE_GetOperationInfoMultiple functionality should be merged into TEE_GetOperationInfo. TEE_OperationInfo struct should be supplement with missing fields that existin the current TEE_OperationMultiple struct.
29	148	6.2.5	E	Helpful specification. If operation is only meaningful in a multistage operation, it should list operations. It would make it more clear for the TEE implementation and of course for user also, when there are exact definitions.	List operation types DIGEST, CIPHER and MAC → other types will be ignored and cause a panic.
30	148	6.2.5	T	Unnecessary panic reason. If key is not set for operation, response is panic. It seems like an excessive response? It would be more usable if TA would not panic. It does not reveal sensitive information about operation.	Remove panic, if key is not set and function returns to doing nothing.
31	149	6.2.6	T	Unexpected functionality. It should not be possible to set a new key without clearing the operation key first. This will make the interface more dynamic, but it would expose uncontrolled behavior to the user. The user should keep track of the operation state. This additional functionality adds complexity to the function and benefit is outweighed but the potential undesired behavior.	A new key can not be set for and operation if a key is already set. This should cause panic reason.
32	149	6.2.6	T	Hidden functionality. The clearing of a function key should be separated out into its own function as it is a distinctly separate operation.	Introduce new function TEE_ClearOperationKeys, which clear functions key. Function will set operation handle to state immediately after operation handle allocation. If TEE_SetOperationKey function is called when key is set → panic reason

#	Page#	Chapter / Paragraph (*)	Type(**)	Comment (Problem & Reason)	Proposed Resolution
33	151	6.2.7	T	Unexpected functionality. Apply same conclusion as point #39 and #40	Apply same proposition as point #39 and 40.
34	153	6.3	T	Uniform definition. It would allow more flexibility in the TEE implementation if there would be an initializion function for digest initialization.	Introduce TEE_DigestInit function that will initialize a digest operation.
35	154	6.3.2	T	Clarification. Apply same conclusion as point #38.	Apply same proposition as point #38.
36	155	6.4.1	T	Unexpected functionality. It should not be possible to initialize an operation if it is in active or initialized states. This will make the interface more dynamic, but it could raise uncontrolled behavior for the user. User should keep track of the operation state. Additional functionality adds complexity to function and the gained benefits are questionable.	If operation is active or initialized → panic. User should use TEE_ResetOperation function.
37	156	6.4.2	T	Clarification or unnecessary functionality. What is the meaning of no output is generated unless sufficient input is not supplied? For example should no pad algorithm source data be buffered if it is not block sized? If that is the case, this is unnecessary functionality. It is user responsibility to supply a correct size source buffer. At the end, user still must provide correct source buffer, before operation can be finalized. This adds complexity, which might be left “unsaid”.	Source buffer should be correct size. If not, it is a panic reason. Cryptographic failure.
38	157	6.4.3	T	Clarification. Function definition should instruct exactly what should happen in operation handler if operation is completed successfully. Loose definition leaves too much room for interpretation.	A good practice would be to instruct call TEE_ResetOperation function. Change to specification: “The operation handle can be reused or re-initialized” should be replaced with “If operation return code is TEE_SUCCESS, TEE_ResetOperation function is called.” This would make the statement “and is set to initial state afterward” obsolete and it could be removed.
39	158	6.5.1	T	Unexpected functionality. Apply same conclusion as point #36.	Apply same proposition as point #36.
40	158	6.5.2	T	Clarification or unnecessary functionality. Apply same conclusion as point #37.	Apply same proposition as point #37.

#	Page#	Chapter / Paragraph (*)	Type(**)	Comment (Problem & Reason)	Proposed Resolution
41	162	6.6	T	Security flaw. Authenticated encryption function is leaking plain text at decryption phase. A user may call TEE_AEUpdate multiple times in TEE_MODE_DECRYPT and this might pass back plain text prior to the MAC being verified. The flaw is pointed out in paper Authenticated Encryption Primitives for Size-Constrained Trusted Computing by Jan-Erik Ekberg, Alexandra Afanasyeva, and N. Asokan (http://dx.doi.org/10.1007/978-3-642-30921-2_1)	*Proposal 1*: Remove TEE_AEUpdate function and enforce use only secure TEE_AEDecryptFinal function. Function is secure, because you can only decrypt one piece cipher text (function accepting src data) and MAC can be checked prior to it's passing back. Notice: This force also at encryption phase use only TEE_AEEncrpytFinal. *Proposal 2*: Decryption is done in two stages (use decrypt-encrypt-decrypt strategy). TEE_AEUpdate function is decrypting source data, but does not pass plain text back to caller. It will encrypt plain text. When user calls TEE_DecryptFinal function, MAC will be verified. If MAC is okay, TEE_DecryptFinal function will reveal temporary key as one of the out parameters. With this key user can decrypt data to plain. Proposal 2 does not need big changes for API. It would be TEE implementation specification, how temporary key is generated and what algorithm is used to encrypt-decrypt, because this would be completely opaque to user. Changes to API: - Add return code TEE_AEDecryptFinal TEE_ERROR_OUT_OF_MEMORY, - Add out parameter to TEE_AEDecryptFinal function TEE_OperationHandle *decryptOperation. Operation handle is allocated and initialized according to encrypt operation. With this operation handler, user is able to decrypt data that he received at TEE_AEUpdate function. - Add explanation 6.6 chapter about decrypt-encrypt-decrypt strategy. Note: This is only a grandiose example. If you are interested about my idea, I can draft a more

#	Page#	Chapter / Paragraph (*)	Type(**)	Comment (Problem & Reason)	Proposed Resolution
					detailed version. My draft would follow this specification style.
42	162	6.6.1	T	Error. Subsequent calls are not possible, if operation state stays initial after TEE_AEInit function.	Afterward operation state should move to active state.
43	173	6.8.1	T	Undefined behavior. Object type is defined, but attribute type is not. If derived secret is used with symmetric operations, it can be guessed to TEE_ATTR_SECRET_VALUE. By placing the calculated value in TEE_ATTR_SECRET_VALUE it restricts the solution because the attribute is not extractable. This is constraining if you do not use the value in cryptographic operation.	Replace returned object handler with void pointer and 32-bit unsigned integer which is representing buffer length. Buffer is allocated by function and therefore function will be needing return code TEE_ERROR_OUT_OF_MEMORY.
44	173	6.8.1	T	Unnecessary complexity. Returning derived value in object handler is clumsy. See #43	See #43
45	188	7.2.4	T	Obsolete return codes. TA can have only one persistent time, it is feasible to suggest that the time variable is initialized when TA is loaded.	Remove all return codes and change function return value to void.
46	190	8	T	General. Big integer operation might require inner state. For example big integer negation. Opaque handler allow for more flexibility in the TEE implementation. The granularity gained by allowing the user to managing memory directly is small, because it still requires the use of TEE_bigInt with defined function.	TEE_BigInt should handled in opaque way. API should define: typedef struct __TEE_BigInt *TEE_BigInt.
47	190	8	T	New subsection and functionality. TEE_BigInt will be needing allocation, initialization and free functions, if point 61 is agreed.	Add new subsection 8.5 Generic functions. - TEE_Result TEE_AllocateBigInt(TEE_BigInit *BigInt, uint32_t size): Allocated resources for TEE_BigInt and initializes meta structures. - Void TEE_BigInitInit(TEE_BigInt, bigInt, Void *bigInit, uint32_t length): Populate big init. -Void TEE_FreeBigInit(TEE_BigInit bigInt): Free resources -Return codes and panics: Same style as for example object handler or operation handler.
48	190	8	T	General. The inability for almost all of these functions to return a valid error is far too restrictive. There are numerous cases where an error could occur and this information should	Functions, which return type is void, should be replaced with TEE_Result. In case of success the return code is TEE_SUCCESS and in case

#	Page#	Chapter / Paragraph (*)	Type(**)	Comment (Problem & Reason)	Proposed Resolution
				be conveyed to the caller of the function. An operation can fail due to a number of reasons including, out of memory, if temporary buffer is needed.	of any error the return code is TEE_ERROR_GENERIC or a set of error codes defined that match the possible fault conditions.
49	201	8.7	T	Useful function. No bit set function.	Introduce bit of setting functionality: - Void TEE_BigIntSetBit(TEE_BigInt *op, uint32_t bitPos, uint32_t setBit);
50	-	-	T	New concept. If object allocation could be made more dynamic it would be possible to review the *maxKeySize*-concept. From security point of view it is arguable if you handle max key strength rather than min key strength. In min strength enforcing you always use “secure” key.	Change key *maxKeysize* to *minKeySize* and definitions at functions where value is used.
51	-	-	T	Good practice. TEE_FreeTransientObject, TEE_ObjectClose, TEE_CloseAndDeletePersistentObject, TEE_FreePersistentObjectEnumerator and TEE_FreeOperation should always set operation/object/enumeration to NULL.	Add “The value pointed to by object/operation/enumeration is set to TEE_HANDLE_NULL” to the function specification.
52	-	6.1	E	Reordering document. If new subsection (6.2 Constant) is defined, it could also contain chapter 5.4 operation related constants. By this operation constant finds its own chapter -> more readable.	Move table 5-6 to new subsection 6.2

(*)(e.g. 5.3 §2 – line 6)
(**)T=Technical, E=Editorial