

 18f4b592a8 ▾

...

mTower / tee / lib / libutee / tee\_api\_objects.c



tdrozovsky Fixed warnings

 History

 1 contributor

799 lines (644 sloc) | 18.6 KB

...

```

1 // SPDX-License-Identifier: BSD-2-Clause
2 /*
3  * Copyright (c) 2014, STMicroelectronics International N.V.
4  * All rights reserved.
5  *
6  * Redistribution and use in source and binary forms, with or without
7  * modification, are permitted provided that the following conditions are met:
8  *
9  * 1. Redistributions of source code must retain the above copyright notice,
10 * this list of conditions and the following disclaimer.
11 *
12 * 2. Redistributions in binary form must reproduce the above copyright notice,
13 * this list of conditions and the following disclaimer in the documentation
14 * and/or other materials provided with the distribution.
15 *
16 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
17 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
18 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
19 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
20 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
21 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
22 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
23 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
24 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
25 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
26 * POSSIBILITY OF SUCH DAMAGE.
27 */
28 #include <stdlib.h>
29 #include <string.h>

```

```

30
31 #include <tee_api.h>
32 #include <utee_syscalls.h>
33 // #include "tee_api_private.h"
34 #include "utee_types.h"
35
36 #define TEE_USAGE_DEFAULT    0xffffffff
37
38 #define TEE_ATTR_BIT_VALUE          (1 << 29)
39 #define TEE_ATTR_BIT_PROTECTED      (1 << 28)
40
41 void __utee_from_attr(struct utee_attribute *ua, const TEE_Attribute *attrs,
42                     uint32_t attr_count)
43 {
44     size_t n;
45
46     for (n = 0; n < attr_count; n++) {
47         ua[n].attribute_id = attrs[n].attributeID;
48         if (attrs[n].attributeID & TEE_ATTR_BIT_VALUE) {
49             ua[n].a = attrs[n].content.value.a;
50             ua[n].b = attrs[n].content.value.b;
51         } else {
52             ua[n].a = (uintptr_t)attrs[n].content.ref.buffer;
53             ua[n].b = attrs[n].content.ref.length;
54         }
55     }
56 }
57
58 /* Data and Key Storage API - Generic Object Functions */
59 /*
60  * Use of this function is deprecated
61  * new code SHOULD use the TEE_GetObjectInfo1 function instead
62  * These functions will be removed at some future major revision of
63  * this specification
64  */
65 void TEE_GetObjectInfo(TEE_ObjectHandle object, TEE_ObjectInfo *objectInfo)
66 {
67     TEE_Result res;
68
69     res = utee_cryp_obj_get_info((unsigned long)object, objectInfo);
70
71     if (res != TEE_SUCCESS)
72         TEE_Panic(res);
73
74     if (objectInfo->objectType == TEE_TYPE_CORRUPTED_OBJECT) {
75         objectInfo->keySize = 0;
76         objectInfo->maxKeySize = 0;
77         objectInfo->objectUsage = 0;
78         objectInfo->dataSize = 0;

```

```

79         objectInfo->dataPosition = 0;
80         objectInfo->handleFlags = 0;
81     }
82 }
83
84 TEE_Result TEE_GetObjectInfo1(TEE_ObjectHandle object, TEE_ObjectInfo *objectInfo)
85 {
86     TEE_Result res;
87
88     res = utee_cryp_obj_get_info((unsigned long)object, objectInfo);
89
90     if (res != TEE_SUCCESS &&
91         res != TEE_ERROR_CORRUPT_OBJECT &&
92         res != TEE_ERROR_STORAGE_NOT_AVAILABLE)
93         TEE_Panic(res);
94
95     return res;
96 }
97
98 /*
99  * Use of this function is deprecated
100  * new code SHOULD use the TEE_RestrictObjectUsage1 function instead
101  * These functions will be removed at some future major revision of
102  * this specification
103  */
104 void TEE_RestrictObjectUsage(TEE_ObjectHandle object, uint32_t objectUsage)
105 {
106     TEE_Result res;
107     TEE_ObjectInfo objectInfo;
108
109     res = utee_cryp_obj_get_info((unsigned long)object, &objectInfo);
110     if (objectInfo.objectType == TEE_TYPE_CORRUPTED_OBJECT)
111         return;
112
113     res = TEE_RestrictObjectUsage1(object, objectUsage);
114
115     if (res != TEE_SUCCESS)
116         TEE_Panic(res);
117 }
118
119 TEE_Result TEE_RestrictObjectUsage1(TEE_ObjectHandle object, uint32_t objectUsage)
120 {
121     TEE_Result res;
122
123     res = utee_cryp_obj_restrict_usage((unsigned long)object, objectUsage);
124
125     if (res != TEE_SUCCESS &&
126         res != TEE_ERROR_CORRUPT_OBJECT &&
127         res != TEE_ERROR_STORAGE_NOT_AVAILABLE)

```

```

128         TEE_Panic(res);
129
130     return res;
131 }
132
133 TEE_Result TEE_GetObjectBufferAttribute(TEE_ObjectHandle object,
134                                         uint32_t attributeID, void *buffer,
135                                         uint32_t *size)
136 {
137     TEE_Result res;
138     TEE_ObjectInfo info;
139     uint64_t sz;
140
141     res = uttee_cryp_obj_get_info((unsigned long)object, &info);
142     if (res != TEE_SUCCESS)
143         goto exit;
144
145     /* This function only supports reference attributes */
146     if ((attributeID & TEE_ATTR_BIT_VALUE)) {
147         res = TEE_ERROR_BAD_PARAMETERS;
148         goto exit;
149     }
150
151     sz = *size;
152     res = uttee_cryp_obj_get_attr((unsigned long)object, attributeID,
153                                   buffer, &sz);
154     *size = sz;
155
156 exit:
157     if (res != TEE_SUCCESS &&
158         res != TEE_ERROR_ITEM_NOT_FOUND &&
159         res != TEE_ERROR_SHORT_BUFFER &&
160         res != TEE_ERROR_CORRUPT_OBJECT &&
161         res != TEE_ERROR_STORAGE_NOT_AVAILABLE)
162         TEE_Panic(res);
163
164     return res;
165 }
166
167 TEE_Result TEE_GetObjectValueAttribute(TEE_ObjectHandle object,
168                                         uint32_t attributeID, uint32_t *a,
169                                         uint32_t *b)
170 {
171     TEE_Result res;
172     TEE_ObjectInfo info;
173     uint32_t buf[2];
174     uint64_t size = sizeof(buf);
175
176     res = uttee_cryp_obj_get_info((unsigned long)object, &info);

```

[illegible]

```

226 {
227     TEE_Result res;
228     uint32_t obj;
229
230     res = utee_cryp_obj_alloc(objectType, maxKeySize, &obj);
231
232     if (res != TEE_SUCCESS &&
233         res != TEE_ERROR_OUT_OF_MEMORY &&
234         res != TEE_ERROR_NOT_SUPPORTED)
235         TEE_Panic(res);
236
237     if (res == TEE_SUCCESS)
238         *object = (TEE_ObjectHandle)(uintptr_t)obj;
239
240     return res;
241 }
242
243 void TEE_FreeTransientObject(TEE_ObjectHandle object)
244 {
245     TEE_Result res;
246     TEE_ObjectInfo info;
247
248     if (object == TEE_HANDLE_NULL)
249         return;
250
251     res = utee_cryp_obj_get_info((unsigned long)object, &info);
252     if (res != TEE_SUCCESS)
253         TEE_Panic(res);
254
255     if ((info.handleFlags & TEE_HANDLE_FLAG_PERSISTENT) != 0)
256         TEE_Panic(0);
257
258     res = utee_cryp_obj_close((unsigned long)object);
259     if (res != TEE_SUCCESS)
260         TEE_Panic(res);
261 }
262
263 void TEE_ResetTransientObject(TEE_ObjectHandle object)
264 {
265     TEE_Result res;
266     TEE_ObjectInfo info;
267
268     if (object == TEE_HANDLE_NULL)
269         return;
270
271     res = utee_cryp_obj_get_info((unsigned long)object, &info);
272     if (res != TEE_SUCCESS)
273         TEE_Panic(res);
274

```

```

275     if ((info.handleFlags & TEE_HANDLE_FLAG_PERSISTENT) != 0)
276         TEE_Panic(0);
277
278     res = utee_cryp_obj_reset((unsigned long)object);
279     if (res != TEE_SUCCESS)
280         TEE_Panic(res);
281 }
282
283 TEE_Result TEE_PopulateTransientObject(TEE_ObjectHandle object,
284                                       const TEE_Attribute *attrs,
285                                       uint32_t attrCount)
286 {
287     TEE_Result res;
288     TEE_ObjectInfo info;
289     struct utee_attribute ua[attrCount];
290
291     res = utee_cryp_obj_get_info((unsigned long)object, &info);
292     if (res != TEE_SUCCESS)
293         TEE_Panic(res);
294
295     /* Must be a transient object */
296     if ((info.handleFlags & TEE_HANDLE_FLAG_PERSISTENT) != 0)
297         TEE_Panic(0);
298
299     /* Must not be initialized already */
300     if ((info.handleFlags & TEE_HANDLE_FLAG_INITIALIZED) != 0)
301         TEE_Panic(0);
302
303     __utee_from_attr(ua, attrs, attrCount);
304     res = utee_cryp_obj_populate((unsigned long)object, ua, attrCount);
305     if (res != TEE_SUCCESS && res != TEE_ERROR_BAD_PARAMETERS)
306         TEE_Panic(res);
307     return res;
308 }
309
310 void TEE_InitRefAttribute(TEE_Attribute *attr, uint32_t attributeID,
311                          const void *buffer, uint32_t length)
312 {
313     if (attr == NULL)
314         TEE_Panic(0);
315     if ((attributeID & TEE_ATTR_BIT_VALUE) != 0)
316         TEE_Panic(0);
317     attr->attributeID = attributeID;
318     attr->content.ref.buffer = (void *)buffer;
319     attr->content.ref.length = length;
320 }
321
322 void TEE_InitValueAttribute(TEE_Attribute *attr, uint32_t attributeID,
323                             uint32_t a, uint32_t b)

```

```

324 {
325     if (attr == NULL)
326         TEE_Panic(0);
327     if ((attributeID & TEE_ATTR_BIT_VALUE) == 0)
328         TEE_Panic(0);
329     attr->attributeID = attributeID;
330     attr->content.value.a = a;
331     attr->content.value.b = b;
332 }
333
334 /*
335  * Use of this function is deprecated
336  * new code SHOULD use the TEE_CopyObjectAttributes1 function instead
337  * These functions will be removed at some future major revision of
338  * this specification
339  */
340 void TEE_CopyObjectAttributes(TEE_ObjectHandle destObject,
341                               TEE_ObjectHandle srcObject)
342 {
343     TEE_Result res;
344     TEE_ObjectInfo src_info;
345
346     res = utee_cryp_obj_get_info((unsigned long)srcObject, &src_info);
347     if (src_info.objectType == TEE_TYPE_CORRUPTED_OBJECT)
348         return;
349
350     res = TEE_CopyObjectAttributes1(destObject, srcObject);
351     if (res != TEE_SUCCESS)
352         TEE_Panic(res);
353 }
354
355 TEE_Result TEE_CopyObjectAttributes1(TEE_ObjectHandle destObject,
356                                     TEE_ObjectHandle srcObject)
357 {
358     TEE_Result res;
359     TEE_ObjectInfo dst_info;
360     TEE_ObjectInfo src_info;
361
362     res = utee_cryp_obj_get_info((unsigned long)destObject, &dst_info);
363     if (res != TEE_SUCCESS)
364         goto exit;
365
366     res = utee_cryp_obj_get_info((unsigned long)srcObject, &src_info);
367     if (res != TEE_SUCCESS)
368         goto exit;
369
370     if (!(src_info.handleFlags & TEE_HANDLE_FLAG_INITIALIZED))
371         TEE_Panic(0);
372

```



```

373     if ((dst_info.handleFlags & TEE_HANDLE_FLAG_PERSISTENT))
374         TEE_Panic(0);
375
376     if ((dst_info.handleFlags & TEE_HANDLE_FLAG_INITIALIZED))
377         TEE_Panic(0);
378
379     res = utee_cryp_obj_copy((unsigned long)destObject,
380                             (unsigned long)srcObject);
381
382 exit:
383     if (res != TEE_SUCCESS &&
384         res != TEE_ERROR_CORRUPT_OBJECT &&
385         res != TEE_ERROR_STORAGE_NOT_AVAILABLE)
386         TEE_Panic(res);
387
388     return res;
389 }
390
391 TEE_Result TEE_GenerateKey(TEE_ObjectHandle object, uint32_t keySize,
392                           const TEE_Attribute *params, uint32_t paramCount)
393 {
394     TEE_Result res;
395     struct utee_attribute ua[paramCount];
396
397     __utee_from_attr(ua, params, paramCount);
398     res = utee_cryp_obj_generate_key((unsigned long)object, keySize,
399                                     ua, paramCount);
400
401     if (res != TEE_SUCCESS && res != TEE_ERROR_BAD_PARAMETERS)
402         TEE_Panic(res);
403
404     return res;
405 }
406
407 /* Data and Key Storage API - Persistent Object Functions */
408
409 TEE_Result TEE_OpenPersistentObject(uint32_t storageID, const void *objectID,
410                                     uint32_t objectIDLen, uint32_t flags,
411                                     TEE_ObjectHandle *object)
412 {
413     TEE_Result res;
414     uint32_t obj;
415
416     if (!objectID) {
417         res = TEE_ERROR_ITEM_NOT_FOUND;
418         goto out;
419     }
420
421     if (objectIDLen > TEE_OBJECT_ID_MAX_LEN) {

```

[illegible]

```

471     if (res == TEE_SUCCESS) {
472         if (object)
473             *object = (TEE_ObjectHandle)(uintptr_t)obj;
474         else
475             res = utee_cryp_obj_close(obj);
476         if (res == TEE_SUCCESS)
477             goto out;
478     }
479 err:
480     if (object)
481         *object = TEE_HANDLE_NULL;
482     if (res == TEE_ERROR_ITEM_NOT_FOUND ||
483         res == TEE_ERROR_ACCESS_CONFLICT ||
484         res == TEE_ERROR_OUT_OF_MEMORY ||
485         res == TEE_ERROR_STORAGE_NO_SPACE ||
486         res == TEE_ERROR_CORRUPT_OBJECT ||
487         res == TEE_ERROR_STORAGE_NOT_AVAILABLE)
488         return res;
489     TEE_Panic(res);
490 out:
491     return TEE_SUCCESS;
492 }
493
494 /*
495  * Use of this function is deprecated
496  * new code SHOULD use the TEE_CloseAndDeletePersistentObject1 function instead
497  * These functions will be removed at some future major revision of
498  * this specification
499  */
500 void TEE_CloseAndDeletePersistentObject(TEE_ObjectHandle object)
501 {
502     TEE_Result res;
503
504     if (object == TEE_HANDLE_NULL)
505         return;
506
507     res = TEE_CloseAndDeletePersistentObject1(object);
508
509     if (res != TEE_SUCCESS)
510         TEE_Panic(0);
511 }
512
513 TEE_Result TEE_CloseAndDeletePersistentObject1(TEE_ObjectHandle object)
514 {
515     TEE_Result res;
516
517     if (object == TEE_HANDLE_NULL)
518         return TEE_ERROR_STORAGE_NOT_AVAILABLE;
519

```

```

520     res = utee_storage_obj_del((unsigned long)object);
521
522     if (res != TEE_SUCCESS && res != TEE_ERROR_STORAGE_NOT_AVAILABLE)
523         TEE_Panic(res);
524
525     return res;
526 }
527
528
529 TEE_Result TEE_RenamePersistentObject(TEE_ObjectHandle object,
530                                     const void *newObjectID,
531                                     uint32_t newObjectIDLen)
532 {
533     TEE_Result res;
534
535     if (object == TEE_HANDLE_NULL) {
536         res = TEE_ERROR_ITEM_NOT_FOUND;
537         goto out;
538     }
539
540     if (!newObjectID) {
541         res = TEE_ERROR_BAD_PARAMETERS;
542         goto out;
543     }
544
545     if (newObjectIDLen > TEE_OBJECT_ID_MAX_LEN) {
546         res = TEE_ERROR_BAD_PARAMETERS;
547         goto out;
548     }
549
550     res = utee_storage_obj_rename((unsigned long)object, newObjectID,
551                                 newObjectIDLen);
552
553 out:
554     if (res != TEE_SUCCESS &&
555         res != TEE_ERROR_ACCESS_CONFLICT &&
556         res != TEE_ERROR_CORRUPT_OBJECT &&
557         res != TEE_ERROR_STORAGE_NOT_AVAILABLE)
558         TEE_Panic(res);
559
560     return res;
561 }
562
563 TEE_Result TEE_AllocatePersistentObjectEnumerator(TEE_ObjectEnumHandle *
564                                                  objectEnumerator)
565 {
566     TEE_Result res;
567     uint32_t oe;
568

```

```

569     if (!objectEnumerator)
570         return TEE_ERROR_BAD_PARAMETERS;
571
572     res = utee_storage_alloc_enum(&oe);
573
574     if (res != TEE_SUCCESS)
575         oe = TEE_HANDLE_NULL;
576
577     *objectEnumerator = (TEE_ObjectEnumHandle)(uintptr_t)oe;
578
579     if (res != TEE_SUCCESS &&
580         res != TEE_ERROR_ACCESS_CONFLICT)
581         TEE_Panic(res);
582
583     return res;
584 }
585
586 void TEE_FreePersistentObjectEnumerator(TEE_ObjectEnumHandle objectEnumerator)
587 {
588     TEE_Result res;
589
590     if (objectEnumerator == TEE_HANDLE_NULL)
591         return;
592
593     res = utee_storage_free_enum((unsigned long)objectEnumerator);
594
595     if (res != TEE_SUCCESS)
596         TEE_Panic(res);
597 }
598
599 void TEE_ResetPersistentObjectEnumerator(TEE_ObjectEnumHandle objectEnumerator)
600 {
601     TEE_Result res;
602
603     if (objectEnumerator == TEE_HANDLE_NULL)
604         return;
605
606     res = utee_storage_reset_enum((unsigned long)objectEnumerator);
607
608     if (res != TEE_SUCCESS)
609         TEE_Panic(res);
610 }
611
612 TEE_Result TEE_StartPersistentObjectEnumerator(TEE_ObjectEnumHandle
613                                               objectEnumerator,
614                                               uint32_t storageID)
615 {
616     TEE_Result res;
617

```

```

618     res = utee_storage_start_enum((unsigned long)objectEnumerator,
619                                   storageID);
620
621     if (res != TEE_SUCCESS &&
622         res != TEE_ERROR_ITEM_NOT_FOUND &&
623         res != TEE_ERROR_CORRUPT_OBJECT &&
624         res != TEE_ERROR_STORAGE_NOT_AVAILABLE)
625         TEE_Panic(res);
626
627     return res;
628 }
629
630 TEE_Result TEE_GetNextPersistentObject(TEE_ObjectEnumHandle objectEnumerator,
631                                       TEE_ObjectInfo *objectInfo,
632                                       void *objectID, uint32_t *objectIDLen)
633 {
634     TEE_Result res;
635     uint64_t len;
636     TEE_ObjectInfo local_info;
637     TEE_ObjectInfo *pt_info;
638
639     if (!objectID) {
640         res = TEE_ERROR_BAD_PARAMETERS;
641         goto out;
642     }
643
644     if (!objectIDLen) {
645         res = TEE_ERROR_BAD_PARAMETERS;
646         goto out;
647     }
648
649     if (objectInfo)
650         pt_info = objectInfo;
651     else
652         pt_info = &local_info;
653     len = *objectIDLen;
654     res = utee_storage_next_enum((unsigned long)objectEnumerator,
655                                 pt_info, objectID, &len);
656     *objectIDLen = len;
657
658 out:
659     if (res != TEE_SUCCESS &&
660         res != TEE_ERROR_ITEM_NOT_FOUND &&
661         res != TEE_ERROR_CORRUPT_OBJECT &&
662         res != TEE_ERROR_STORAGE_NOT_AVAILABLE)
663         TEE_Panic(res);
664
665     return res;
666 }

```

```

667
668 /* Data and Key Storage API - Data Stream Access Functions */
669
670 TEE_Result TEE_ReadObjectData(TEE_ObjectHandle object, void *buffer,
671                               uint32_t size, uint32_t *count)
672 {
673     TEE_Result res;
674     uint64_t cnt64;
675
676     if (object == TEE_HANDLE_NULL) {
677         res = TEE_ERROR_BAD_PARAMETERS;
678         goto out;
679     }
680
681     cnt64 = *count;
682     res = utee_storage_obj_read((unsigned long)object, buffer, size,
683                                &cnt64);
684     *count = cnt64;
685
686 out:
687     if (res != TEE_SUCCESS &&
688         res != TEE_ERROR_CORRUPT_OBJECT &&
689         res != TEE_ERROR_STORAGE_NOT_AVAILABLE)
690         TEE_Panic(res);
691
692     return res;
693 }
694
695 TEE_Result TEE_WriteObjectData(TEE_ObjectHandle object, const void *buffer,
696                                uint32_t size)
697 {
698     TEE_Result res;
699
700     if (object == TEE_HANDLE_NULL) {
701         res = TEE_ERROR_BAD_PARAMETERS;
702         goto out;
703     }
704
705     if (size > TEE_DATA_MAX_POSITION) {
706         res = TEE_ERROR_OVERFLOW;
707         goto out;
708     }
709
710     res = utee_storage_obj_write((unsigned long)object, buffer, size);
711
712 out:
713     if (res != TEE_SUCCESS &&
714         res != TEE_ERROR_STORAGE_NO_SPACE &&
715         res != TEE_ERROR_OVERFLOW &&

```

```

716         res != TEE_ERROR_CORRUPT_OBJECT &&
717         res != TEE_ERROR_STORAGE_NOT_AVAILABLE)
718             TEE_Panic(res);
719
720     return res;
721 }
722
723 TEE_Result TEE_TruncateObjectData(TEE_ObjectHandle object, uint32_t size)
724 {
725     TEE_Result res;
726
727     if (object == TEE_HANDLE_NULL) {
728         res = TEE_ERROR_BAD_PARAMETERS;
729         goto out;
730     }
731
732     res = utee_storage_obj_trunc((unsigned long)object, size);
733
734 out:
735     if (res != TEE_SUCCESS &&
736         res != TEE_ERROR_STORAGE_NO_SPACE &&
737         res != TEE_ERROR_CORRUPT_OBJECT &&
738         res != TEE_ERROR_STORAGE_NOT_AVAILABLE)
739         TEE_Panic(res);
740
741     return res;
742 }
743
744 TEE_Result TEE_SeekObjectData(TEE_ObjectHandle object, int32_t offset,
745                               TEE_Whence whence)
746 {
747     TEE_Result res;
748     TEE_ObjectInfo info;
749
750     if (object == TEE_HANDLE_NULL) {
751         res = TEE_ERROR_BAD_PARAMETERS;
752         goto out;
753     }
754
755     res = utee_cryp_obj_get_info((unsigned long)object, &info);
756     if (res != TEE_SUCCESS)
757         goto out;
758
759     switch (whence) {
760     case TEE_DATA_SEEK_SET:
761         if (offset > 0 && (uint32_t)offset > TEE_DATA_MAX_POSITION) {
762             res = TEE_ERROR_OVERFLOW;
763             goto out;
764         }

```



```

765         break;
766     case TEE_DATA_SEEK_CUR:
767         if (offset > 0 &&
768             ((uint32_t)offset + info.dataPosition >
769              TEE_DATA_MAX_POSITION ||
770              (uint32_t)offset + info.dataPosition <
771              info.dataPosition)) {
772             res = TEE_ERROR_OVERFLOW;
773             goto out;
774         }
775         break;
776     case TEE_DATA_SEEK_END:
777         if (offset > 0 &&
778             ((uint32_t)offset + info.dataSize > TEE_DATA_MAX_POSITION ||
779              (uint32_t)offset + info.dataSize < info.dataSize)) {
780             res = TEE_ERROR_OVERFLOW;
781             goto out;
782         }
783         break;
784     default:
785         res = TEE_ERROR_ITEM_NOT_FOUND;
786         goto out;
787     }
788
789     res = utee_storage_obj_seek((unsigned long)object, offset, whence);
790
791 out:
792     if (res != TEE_SUCCESS &&
793         res != TEE_ERROR_OVERFLOW &&
794         res != TEE_ERROR_CORRUPT_OBJECT &&
795         res != TEE_ERROR_STORAGE_NOT_AVAILABLE)
796         TEE_Panic(res);
797
798     return res;
799 }

```