Report: Secure Object Lifecycle and Time Measurement in OP-TEE

1. Introduction

The OP-TEE (Open Portable Trusted Execution Environment) framework provides a secure execution environment isolated from the normal operating system. One of its key features is **secure storage**, which allows the Trusted Applications (TAs) to securely store and retrieve data in encrypted form.

This report explains the internal process of **object creation**, **reading**, and **deletion** in secure storage, as implemented through a Trusted Application (TA). It also elaborates on the **method used to measure read latency**, i.e., the time taken to fetch data from secure storage.

2. Secure Storage Overview

Secure storage in OP-TEE allows TAs to persist data securely in the device's non-volatile memory. All data stored here is automatically encrypted, integrity-protected, and isolated per TA.

There are two kinds of storage objects in OP-TEE:

- **Transient objects** – exist only in RAM during a TA's lifetime.
- **Persistent objects** – stored permanently in secure flash and survive reboots.

The code discussed in this report works with **persistent objects**, using the `TEE_CreatePersistentObject()` and `TEE_OpenPersistentObject()` APIs.

3. Object Lifecycle Operations

The lifecycle of a secure storage object consists of three primary operations:

1. **Creation (Write)**
2. **Reading**
3. **Deletion**

Each is explained below.

3.1. Object Creation

The creation process is initiated when the host application (running in the normal world) invokes the command `TA_SECURE_STORAGE_CMD_WRITE_RAW` through the `TEEC_InvokeCommand()` API.

Within the Trusted Application (`secure_storage_ta.c`), the corresponding handler function `create_raw_object()` executes the following key steps:

1. **Parameter Extraction**
   The TA receives two memory references:
   - Parameter 0 → Object identifier (`obj_id`)
   - Parameter 1 → Data to be written

Object Creation

```
TEE_CreatePersistentObject(TEE_STORAGE_PRIVATE,
                   obj_id, obj_id_sz,
                   obj_data_flag,
                   TEE_HANDLE_NULL,
                   NULL, 0,
                   &object);
```

2. This API creates an encrypted and integrity-protected object within OP-TEE's **private storage area**.

Data Writing

```
TEE_WriteObjectData(object, data, data_sz);
```

3. The input data is securely written into the persistent object.

Closing the Object

```
TEE_CloseObject(object);
```

4. The object handle is closed, completing the write operation.

🟢 **Outcome:**
An encrypted and authenticated file is stored inside secure storage, typically under `/data/tee/`, isolated per TA UUID.

3.2. Object Reading

When the host requests data retrieval (`TA_SECURE_STORAGE_CMD_READ_RAW`), the TA's function `read_raw_object()` is executed.

The steps involved are:

Opening the Object

```
TEE_OpenPersistentObject(TEE_STORAGE_PRIVATE, obj_id, obj_id_sz,
                         TEE_DATA_FLAG_ACCESS_READ, &object);
```

1. The object is located in secure storage and opened for reading.

Fetching Object Metadata

```
TEE_GetObjectInfo1(object, &object_info);
```

2. Retrieves size and attributes of the object.

Reading Object Data

```
TEE_ReadObjectData(object, data, object_info.dataSize, &read_bytes);
```

3. Reads the decrypted data into the TA's secure memory buffer.

Returning Data to Normal World
The buffer is copied into the shared memory area for the normal world application:

```
TEE_MemMove(params[1].memref.buffer, data, read_bytes);
```

4.

🟢 **Outcome:**
Data stored in secure storage is securely decrypted and transferred to the requesting application via a trusted memory channel.

3.3. Object Deletion

For the deletion command (`TA_SECURE_STORAGE_CMD_DELETE`), the function `delete_object()` is executed.

The process includes:

Opening the Object

```
TEE_OpenPersistentObject(TEE_STORAGE_PRIVATE, obj_id, obj_id_sz,
                         TEE_DATA_FLAG_ACCESS_WRITE_META, &object);
```

1. This ensures metadata modification permissions.

Deleting the Object

```
TEE_CloseAndDeletePersistentObject1(object);
```

2. The object is permanently deleted from secure storage.

🟢 **Outcome:**
The object is securely removed from flash storage; its data cannot be recovered.

---

4. Time Measurement During Read Operation

4.1. Purpose

To evaluate the performance of secure storage, the **time taken to read data** from secure storage can be measured. This provides insight into the latency introduced by the secure storage layer (including decryption and integrity verification).

4.2. API Used

The TA uses the OP-TEE internal API:

```
TEE_GetSystemTime(struct TEE_Time *time);
```

This function returns the current system time inside the TEE in seconds and milliseconds.

4.3. Implementation

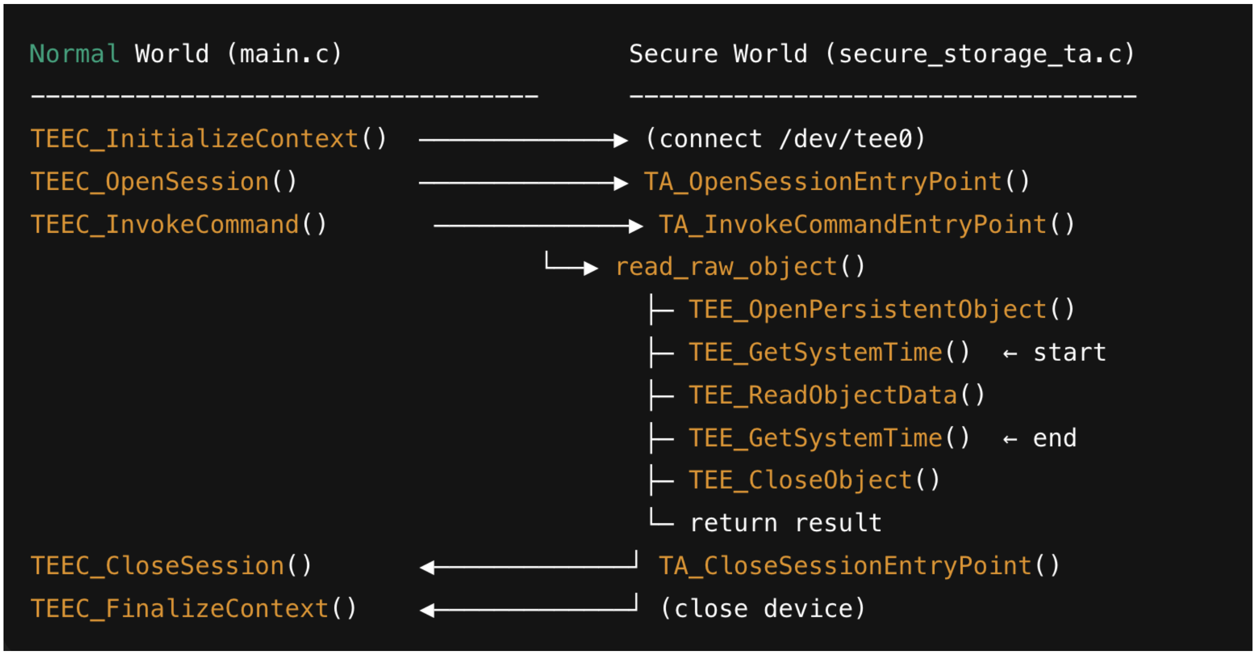
Within the `read_raw_object()` function, timing calls are placed before and after the `TEE_ReadObjectData()` function:

```
TEE_Time start_time, end_time;
uint32_t elapsed_ms;

TEE_GetSystemTime(&start_time);
res = TEE_ReadObjectData(object, data, object_info.dataSize, &read_bytes);
TEE_GetSystemTime(&end_time);

elapsed_ms = (end_time.seconds - start_time.seconds) * 1000 +
        (end_time.millis - start_time.millis);

IMSG("Time taken to read object: %u ms", elapsed_ms);
```



```
Normal World (main.c)                    Secure World (secure_storage_ta.c)
----------------------------------       ----------------------------------
TEEC_InitializeContext()   ──────────────▶ (connect /dev/tee0)
TEEC_OpenSession()         ──────────────▶ TA_OpenSessionEntryPoint()
TEEC_InvokeCommand()        ──────────────▶ TA_InvokeCommandEntryPoint()
                                   └──▶ read_raw_object()
                                          ├─ TEE_OpenPersistentObject()
                                          ├─ TEE_GetSystemTime()  ← start
                                          ├─ TEE_ReadObjectData()
                                          ├─ TEE_GetSystemTime()  ← end
                                          ├─ TEE_CloseObject()
                                          └─ return result
TEEC_CloseSession()        ◀──────────────┘ TA_CloseSessionEntryPoint()
TEEC_FinalizeContext()     ◀──────────────┘ (close device)
```

4.4. Measured Interval

The calculated time represents:

> “The total duration taken by the secure storage subsystem to read, decrypt, and copy the object’s data into the TA’s secure memory buffer.”

In other words, the time covers the **read operation only**, starting **immediately before the data read begins** and ending **as soon as the read operation completes**.

If the timer is moved before `TEE_OpenPersistentObject()` and after `TEE_CloseObject()`, the total **open + read + close** time can also be measured.

4.5. Output Example

When the TA executes, the secure world console displays:

```
I/TA: Opening object: object#1
I/TA: Time taken to read object: 3 ms
I/TA: Successfully read 7000 bytes
```

This indicates that reading the encrypted data and transferring it into memory took approximately 3 milliseconds.

5. Storage Characteristics

- Each Trusted Application has its **own private storage directory**, identified by its UUID.
- Objects are stored as encrypted files; names and contents are obfuscated.
- Normal world processes **cannot access or interpret** these files.
- Data integrity and confidentiality are ensured through hardware-backed cryptography.

Example path on Raspberry Pi 3:

/data/tee/<TA_UUID>/xxxxxxxx.enc

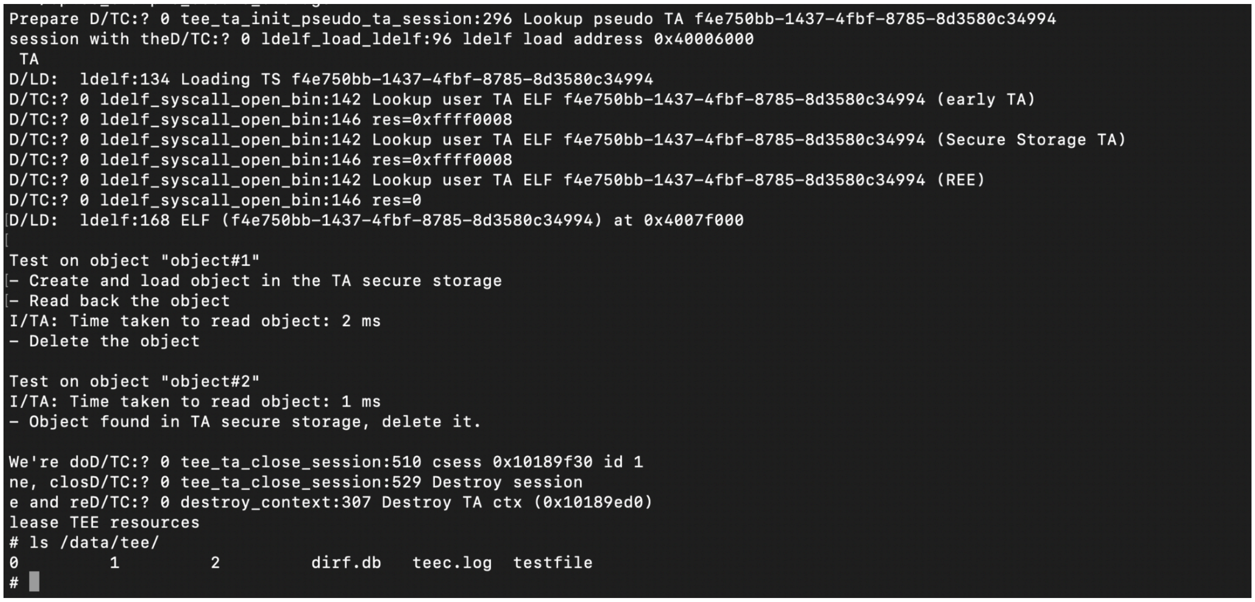
6. Conclusion

The OP-TEE secure storage mechanism provides a robust means to store sensitive data persistently within the TEE. Each object's lifecycle—from creation to deletion—is securely managed and protected from the normal world.

The inclusion of timing measurement using `TEE_GetSystemTime()` enables developers to evaluate the performance overhead associated with secure storage operations.

In this implementation:

- **Creation** encrypts and writes the object.
- **Reading** decrypts and retrieves it.
- **Deletion** securely removes it.
- **Timing measurement** quantifies the latency of secure read operations.



```
Prepare D/TC:? 0 tee_ta_init_pseudo_ta_session:296 Lookup pseudo TA f4e750bb-1437-4fbf-8785-8d3580c34994
session with theD/TC:? 0 ldelf_load_ldelf:96 ldelf load address 0x40006000
 TA
D/LD:  ldelf:134 Loading TS f4e750bb-1437-4fbf-8785-8d3580c34994
D/TC:? 0 ldelf_syscall_open_bin:142 Lookup user TA ELF f4e750bb-1437-4fbf-8785-8d3580c34994 (early TA)
D/TC:? 0 ldelf_syscall_open_bin:146 res=0xffff0008
D/TC:? 0 ldelf_syscall_open_bin:142 Lookup user TA ELF f4e750bb-1437-4fbf-8785-8d3580c34994 (Secure Storage TA)
D/TC:? 0 ldelf_syscall_open_bin:146 res=0xffff0008
D/TC:? 0 ldelf_syscall_open_bin:142 Lookup user TA ELF f4e750bb-1437-4fbf-8785-8d3580c34994 (REE)
D/TC:? 0 ldelf_syscall_open_bin:146 res=0
D/LD:  ldelf:168 ELF (f4e750bb-1437-4fbf-8785-8d3580c34994) at 0x4007f000

Test on object "object#1"
- Create and load object in the TA secure storage
- Read back the object
I/TA: Time taken to read object: 2 ms
- Delete the object

Test on object "object#2"
I/TA: Time taken to read object: 1 ms
- Object found in TA secure storage, delete it.

We're doD/TC:? 0 tee_ta_close_session:510 csess 0x10189f30 id 1
ne, closD/TC:? 0 tee_ta_close_session:529 Destroy session
e and reD/TC:? 0 destroy_context:307 Destroy TA ctx (0x10189ed0)
lease TEE resources
# ls /data/tee/
0         1         2         dirf.db   teec.log  testfile
# 
```