Object and Chunk Storage Analysis in OP-TEE (Raspberry Pi 3)

---

1. Objective

The objective of this report is to analyze how secure objects and data chunks are managed and stored by the OP-TEE (Open Portable Trusted Execution Environment) on a Raspberry Pi 3 platform. It documents the observed configuration parameters, memory layout, and explains how the secure storage system handles large data through chunked streaming within the Trusted Execution Environment (TEE).

2. Overview of OP-TEE Secure Storage

2.1. Secure Object

A **secure object** in OP-TEE represents a file-like structure stored in encrypted and integrity-protected form. Objects are created and accessed by Trusted Applications (TAs) through APIs such as:

```
TEE_CreatePersistentObject();
TEE_WriteObjectData();
TEE_ReadObjectData();
```

Objects remain persistent across reboots and are stored either in:

- RPMB (Replay-Protected Memory Block) – hardware-secured storage
- REE FS (Rich Execution Environment File System) – software-encrypted backend on the Linux filesystem

Your system uses **REE FS** as confirmed by build configuration flags.

2.2. Chunk

A **chunk** is the unit of data transferred per write/read operation between the normal world (REE) and the secure world (TEE). When an application writes a file, data is divided into chunks before being processed and encrypted by TEE.

```
TEE_WriteObjectData(object, buffer, chunk_size);
```

Each chunk is encrypted and sent to the REE filesystem sequentially. This prevents the TEE from needing to load the entire object into secure memory at once.

3. Storage Backend and Filesystem

Table 1: Storage Backend Configuration

Parameter	Description / Status
CFG_REE_FS	Enabled (=1): REE File System backend active
CFG_REE_FS_TA	Enabled (=1): Trusted Applications loaded from REE FS
CFG_REE_FS_TA_BUFFERED	Not set: Dynamic (unbuffered) streaming mode
CFG_REE_FS_INTEGRITY_RPMB	Not set: No hardware integrity binding
CFG_REE_FS_ALLOW_RESET	Not set: Filesystem reset disabled

Result: OP-TEE uses **unbuffered REE FS mode**, meaning chunk sizes are determined dynamically at runtime and data is streamed directly between TEE and the Linux filesystem.

4. Memory Configuration (Confirmed from Build Files)

Table 2: Memory Configuration Summary

Parameter	Value	Description
CFG_CORE_HEAP_SIZE	65536 bytes (64 KB)	Total dynamic heap memory available to TEE core
CFG_TEE_RAM_VA_SIZE	0x00700000 (7 MB)	Virtual address space reserved for TEE core
CFG_SHMEM_SIZE	0x00400000 (4 MB)	Shared memory between REE and TEE
CFG_REE_FS_TA_BUFFER_SIZE	Not defined	Dynamic chunking (no fixed buffer)

Parameter	Value	Description
Backend	REE FS	Software-encrypted filesystem under `/data/tee/`

5. Heap and Chunk Relationship

Table 3: Heap and Chunk Relationship

Aspect	Explanation
Heap (CFG_CORE_HEAP_SIZE)	Total memory pool available for temporary allocations (buffers, RPCs, crypto, etc.)
Chunk	Portion of data written or read in one operation. Each chunk is temporarily stored in the heap before encryption or decryption.
Relationship	Chunk size cannot exceed available heap memory. Since heap = 64 KB, the effective per-write chunk $\leq$ 64 KB.

Thus, while the heap limits the amount of data processed per call, the total object size can still be very large because OP-TEE streams chunks sequentially to disk.

6. Storage Operation Flow

1. **Create object:** TEE initializes an encrypted file structure.
2. **Write data:** Application divides the input file into small chunks (heap size). Each chunk is encrypted and written to REE FS.
3. **Close object:** Metadata and integrity data are finalized.

```
size_t chunk = 64 * 1024; // 64 KB
for (size_t i = 0; i < total_size; i += chunk)
    TEE_WriteObjectData(object, data + i, chunk);
TEE_CloseObject(object);
```

7. Practical Limits

Table 4: Practical Limits in Secure Storage

Type	Limiting Factor	Typical Value / Behavior
Per-write limit	Heap memory (64 KB)	Maximum chunk size per write

Type	Limiting Factor	Typical Value / Behavior
Per-object limit	REE FS filesystem capacity	Practically limited by SD card size
Shared memory limit	RPC buffer (4 MB)	Controls REE–TEE data transfer
Overall secure RAM	7 MB	Includes code, stack, and heap

8. Findings Summary

Table 5: Summary of Findings

Parameter	Value / Status	Explanation
Storage backend	REE FS	Software-encrypted secure storage
Chunk size	Dynamic $\leq$ 64 KB	Determined by heap and shared memory
Heap size (CFG_CORE_HEAP_SIZE)	64 KB	Fixed at compile time
TEE virtual memory	7 MB	Secure OS core mapping
Shared memory (CFG_SHMEM_SIZE)	4 MB	Normal–secure world data exchange
Integrity protection	Software hashing	No RPMB binding
Max object size	Up to REE filesystem capacity	Limited only by SD card
File location	`/data/tee/`	Encrypted files stored on REE side

9. Conclusion

The analyzed configuration and logs confirm that:

- The system uses **REE FS** as its secure storage backend.
- There is **no fixed chunk-buffer macro**; OP-TEE allocates transfer buffers dynamically.
- The heap (`CFG_CORE_HEAP_SIZE = 64 KB`) defines the maximum per-operation chunk size.
- Chunk-based streaming enables OP-TEE to securely store large objects despite limited memory.
- Actual data size is limited only by available REE filesystem capacity.

Hence, the Raspberry Pi 3 OP-TEE setup performs secure object storage using **dynamic chunk streaming within a 64 KB heap and 4 MB shared memory window**, providing confidentiality and persistence while maintaining low memory usage.