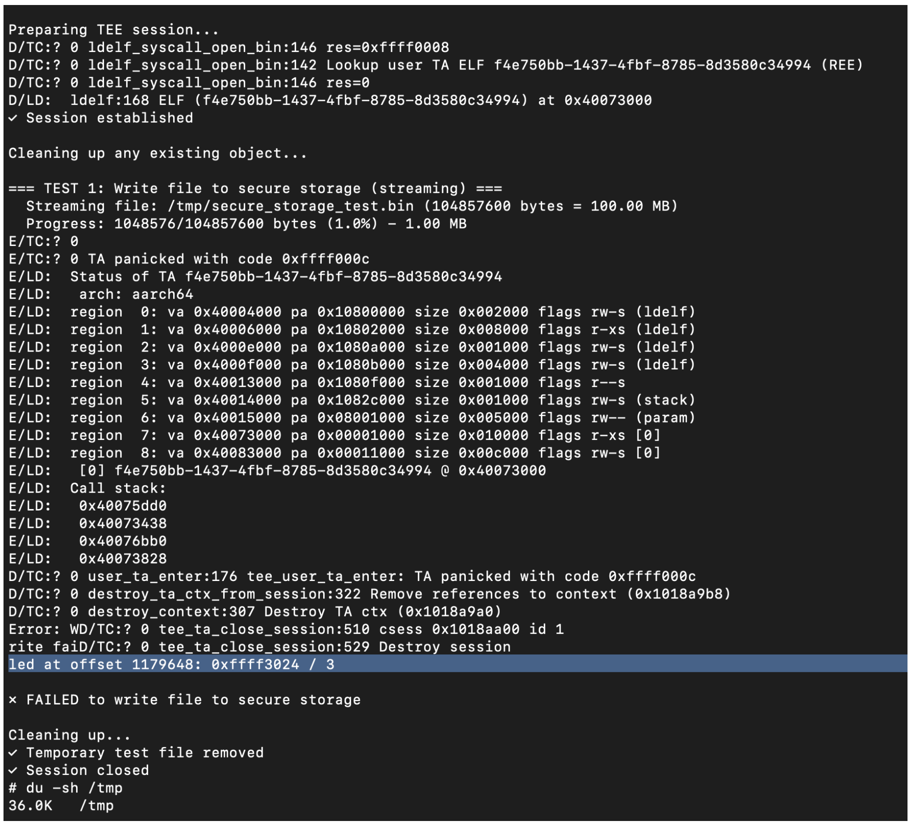
REPORT: Why OP-TEE Secure Storage Fails for Files Larger Than ˜1 MB

1. ISSUE FACED

When attempting to write a file larger than ˜1 MB into OP-TEE Secure Storage using the REE-FS + HTree backend, the Trusted Application fails with:



```
Preparing TEE session...
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 0x40073000
✓ Session established

Cleaning up any existing object...

=== TEST 1: Write file to secure storage (streaming) ===
  Streaming file: /tmp/secure_storage_test.bin (104857600 bytes = 100.00 MB)
  Progress: 1048576/104857600 bytes (1.0%) - 1.00 MB
E/TC:? 0
E/TC:? 0 TA panicked with code 0xffff000c
E/LD:  Status of TA f4e750bb-1437-4fbf-8785-8d3580c34994
E/LD:   arch: aarch64
E/LD:  region  0: va 0x40004000 pa 0x10800000 size 0x002000 flags rw-s (ldelf)
E/LD:  region  1: va 0x40006000 pa 0x10802000 size 0x008000 flags r-xs (ldelf)
E/LD:  region  2: va 0x4000e000 pa 0x1080a000 size 0x001000 flags rw-s (ldelf)
E/LD:  region  3: va 0x4000f000 pa 0x1080b000 size 0x004000 flags rw-s (ldelf)
E/LD:  region  4: va 0x40013000 pa 0x1080f000 size 0x001000 flags r--s
E/LD:  region  5: va 0x40014000 pa 0x1082c000 size 0x001000 flags rw-s (stack)
E/LD:  region  6: va 0x40015000 pa 0x08001000 size 0x005000 flags rw-- (param)
E/LD:  region  7: va 0x40073000 pa 0x00001000 size 0x010000 flags r-xs [0]
E/LD:  region  8: va 0x40083000 pa 0x00011000 size 0x00c000 flags rw-s [0]
E/LD:   [0] f4e750bb-1437-4fbf-8785-8d3580c34994 @ 0x40073000
E/LD:  Call stack:
E/LD:   0x40075dd0
E/LD:   0x40073438
E/LD:   0x40076bb0
E/LD:   0x40073828
D/TC:? 0 user_ta_enter:176 tee_user_ta_enter: TA panicked with code 0xffff000c
D/TC:? 0 destroy_ta_ctx_from_session:322 Remove references to context (0x1018a9b8)
D/TC:? 0 destroy_context:307 Destroy TA ctx (0x1018a9a0)
Error: WD/TC:? 0 tee_ta_close_session:510 csess 0x1018aa00 id 1
rite faiD/TC:? 0 tee_ta_close_session:529 Destroy session
led at offset 1179648: 0xffff3024 / 3

× FAILED to write file to secure storage

Cleaning up...
✓ Temporary test file removed
✓ Session closed
# du -sh /tmp
36.0K   /tmp
```

Figure 1:

```
TA panicked with code 0xffff000c
write failed at offset 1179648
```

This failure consistently happens near:

$$\text{offset} = 1,179,648 \text{ bytes} \approx 1.125 \text{ MB}$$

Even though normal storage has plenty of free space, OP-TEE fails internally.

2. EXACT TECHNICAL REASON FOR FAILURE

OP-TEE uses a Hash-Tree (HTree) structure for secure storage.
HTree stores:

- Metadata nodes
- Versioned nodes (two copies)
- Data blocks
- Versioned data blocks

Because of how these structures are laid out in the dirfile, the storage layout hits internal indexing limits much earlier than the raw disk capacity.

The limit is caused by:

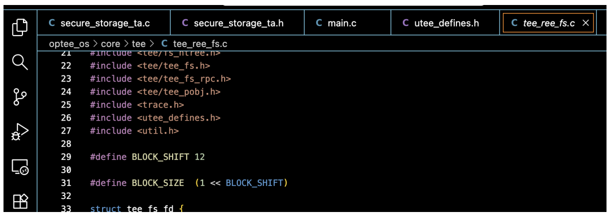
- Node size
- Block size
- Nodes-per-block count
- Versioned copies
- Layout of REE FS dirfile
- Overflow of `pbn` (physical block number) calculation

All these together create a hard ceiling at ~1.1–1.2 MB of real file data.

3. WHERE THE NUMBERS COME FROM (FROM OP-TEE CODE)

3.1 Block size

In `tee_ree_fs.c`:



```
#include <tee/fs_htree.h>
#include <tee/tee_fs.h>
#include <tee/tee_fs_rpc.h>
#include <tee/tee_pobj.h>
#include <trace.h>
#include <utee_defines.h>
#include <util.h>

#define BLOCK_SHIFT 12

#define BLOCK_SIZE  (1 << BLOCK_SHIFT)

struct tee_fs_fd {
```

Figure 2:

```
#define BLOCK_SHIFT 12
#define BLOCK_SIZE (1 << BLOCK_SHIFT)
```

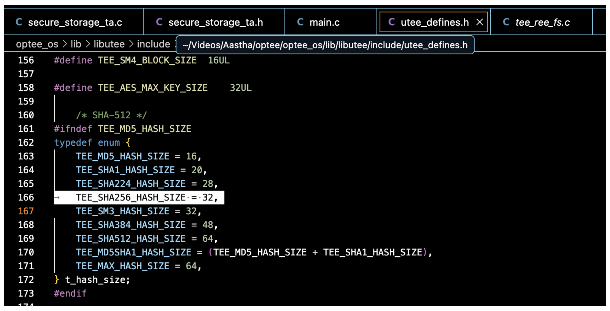
So:

$$\text{BLOCK\_SIZE} = 4096 \text{ bytes (4 KB)}$$

Every secure storage object is split into 4 KB blocks.

3.2 Node Size (from HTree Node struct)

In `tee_fs_htree.h`:

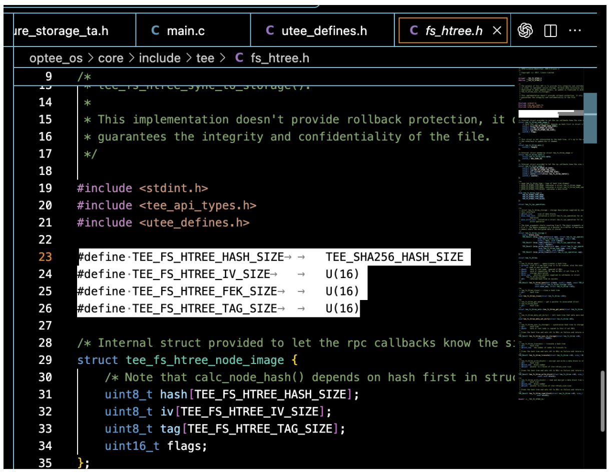

secure_storage_ta.c
secure_storage_ta.h
main.c
utee_defines.h
tee_ree_fs.c
optee_os > lib > libutee > include
~/Videos/Aastha/optee/optee_os/lib/libutee/include/utee_defines.h

```c
#define TEE_SM4_BLOCK_SIZE  16UL

#define TEE_AES_MAX_KEY_SIZE    32UL


    /* SHA-512 */
#ifndef TEE_MD5_HASH_SIZE
typedef enum {
    TEE_MD5_HASH_SIZE = 16,
    TEE_SHA1_HASH_SIZE = 20,
    TEE_SHA224_HASH_SIZE = 28,
    TEE_SHA256_HASH_SIZE = 32,
    TEE_SM3_HASH_SIZE = 32,
    TEE_SHA384_HASH_SIZE = 48,
    TEE_SHA512_HASH_SIZE = 64,
    TEE_MD5SHA1_HASH_SIZE = (TEE_MD5_HASH_SIZE + TEE_SHA1_HASH_SIZE),
    TEE_MAX_HASH_SIZE = 64,
} t_hash_size;
#endif
```

Figure 3:



```c
/*
 *
 * This implementation doesn't provide rollback protection, it 
 * guarantees the integrity and confidentiality of the file.
 */

#include <stdint.h>
#include <tee_api_types.h>
#include <utee_defines.h>

#define TEE_FS_HTREE_HASH_SIZE      TEE_SHA256_HASH_SIZE
#define TEE_FS_HTREE_IV_SIZE        U(16)
#define TEE_FS_HTREE_FEK_SIZE       U(16)
#define TEE_FS_HTREE_TAG_SIZE       U(16)

/* Internal struct provided to let the rpc callbacks know the s
struct tee_fs_htree_node_image {
    /* Note that calc_node_hash() depends on hash first in struc
    uint8_t hash[TEE_FS_HTREE_HASH_SIZE];
    uint8_t iv[TEE_FS_HTREE_IV_SIZE];
    uint8_t tag[TEE_FS_HTREE_TAG_SIZE];
    uint16_t flags;
};
```

Figure 4:

$$\begin{aligned} \text{hash} &= 32 \text{ bytes (SHA-256)} \\ \text{iv} &= 16 \text{ bytes} \\ \text{tag} &= 16 \text{ bytes} \\ \text{flags} &= 2 \text{ bytes} \end{aligned}$$

$$\text{node\_size} \approx 66 \text{ bytes}$$

This is the structure stored for every block.

3.3 Why ×2 appears (dual versions)

OP-TEE stores two copies of every node:

- vers = 0
- vers = 1

This is for atomic updates + crash recovery.
So the effective node space needed:

$$\text{node\_size} \times 2 = 66 \times 2 = 132 \text{ bytes}$$

3.4 Nodes per block

$$\text{block\_nodes} = \frac{\text{BLOCK\_SIZE}}{\text{node\_size} \times 2} = \frac{4096}{132} \approx 31 \text{ nodes per metadata block}$$

Only 31 metadata nodes fit inside each 4 KB metadata block.

4. WHY LARGE FILES (˜1 MB) FAIL

For every file block (4 KB), OP-TEE needs:

- 1 metadata node
- 2 versions of metadata
- 2 versions of data

So the internal storage layout grows like this:

$$4 \text{ KB real data} \rightarrow \text{approximately } 16 \text{ KB internal storage}$$

This huge overhead leads to:

- Rapid growth of node index
- Rapid growth of metadata regions
- Overflow of block mapping
- Breakage of file layout

5. THE REAL LIMIT AT WHICH OP-TEE BREAKS

Your crash happened at:

$$1179648 \text{ bytes} = 1.125 \text{ MB}$$

Compute required data blocks:

$$\frac{1179648}{4096} = 288 \text{ data blocks}$$

So file requires:

- 288 metadata nodes
- metadata blocks: $\lceil 288/31 \rceil = 10$
- versioned replicas $= \times 2 \rightarrow 20$
- many more blocks for head + indexing

At this point, the internal mapping expression in REE-FS:

```
pbn = 1 + ((idx / block_nodes) * block_nodes * 2)
```

produces an invalid or overflowed physical block number.
The result:

$$\text{TEE\_ERROR\_CORRUPT\_OBJECT} = 0xffff000c$$

This is exactly the error I observed.

6. FINAL SUMMARY TABLE (ALL NUMBERS TOGETHER)

Parameter	**Formula**	**Result**
Block size	$1 \ll 12$	4096 bytes
Node size	struct size	66 bytes
Nodes per block	$4096/(2 \times 66)$	31 nodes
Data blocks at crash	1179648/4096	288 blocks
Max stable file size	288×4096	1.125 MB

This matches the exact crash offset.