Secure Storage Updates in OP-TEE: What’s new since SFO15 and what’s left to do?

Jerome Forissier, Huawei Technologies/HiSilicon
Agenda

- What is secure storage?
- OP-TEE timeline
- What’s new?
  - Framework for multiple storage backends
  - RPMB (Replay-Protected Memory Block)
  - SQLite
  - Encryption
  - xtest updates
- Next steps
What is Secure Storage?

- Persistent data store for crypto keys or other application-specific data
- Accessible to Trusted Applications only
  - Each TA has its own storage (TA isolation)
- Isolated from the non-secure world
  - Secure Storage data can be read, modified or deleted by user applications or the OS kernel
- OP-TEE implements the GlobalPlatform™ TEE Internal Core API v1.1
  - Chapter 5: Trusted Storage API for Data and Keys; Persistent Object [Enumeration] Functions and Data Stream Access Functions
    
    TEE_OpenPersistentObject()
    TEE_CreatePersistentObject()
    TEE_CloseAndDeletePersistentObject1()
    TEE_RenamePersistentObject()
    TEE_ReadObjectData()
    TEE_WriteObjectData()
    TEE_TruncateObjectData()
    TEE_SeekObjectData()
    TEE_AllocatePersistentObjectEnumerator()
    TEE_FreePersistentObjectEnumerator()
    TEE_ResetPersistentObjectEnumerator()
    TEE_StartPersistentObjectEnumerator()
    TEE_GetNextPersistentObject()
OP-TEE Timeline

LCU14  HKG15  SFO15  BKK16  LAS16

0.1.0  0.2.0  0.3.0  1.0.0  1.1.0  2.0.0  2.1.0  (next)

<table>
<thead>
<tr>
<th>Secure Storage Improvements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encryption</td>
</tr>
<tr>
<td>Atomicity</td>
</tr>
<tr>
<td>xtest 20000 (corruption tests)</td>
</tr>
<tr>
<td>Multiple storage support</td>
</tr>
<tr>
<td>Better TA isolation</td>
</tr>
<tr>
<td>SQLite storage</td>
</tr>
<tr>
<td>RPMB</td>
</tr>
<tr>
<td>struct tee_file_operations</td>
</tr>
</tbody>
</table>

Linaro Connect Las Vegas 2016
Multiple storage support

- **OP-TEE < 2.0.0**
  - Secure storage is implemented on top of the Rich OS filesystem. We call it the “REE FS”
  - Trusted Application may only use TEE_STORAGE_PRIVATE as the storageID parameter

- **OP-TEE = 2.0.0**
  - Replay Protected Memory Block (RPMB) storage is introduced. It is a compile-time option to replace the REE FS (CFG_RPMB_FS=y).

- **OP-TEE = 2.1.0**
  - Code is refactored to allow for compile time and runtime selection of the storage backend
  - Compile time options (both may be enabled simultaneously):
    ```
    CFG_REE_FS  ?= y; CFG_RPMB_FS  ?= n
    ```
  - Runtime flags used by Trusted Applications:
    ```
    #define TEE_STORAGE_PRIVATE        0x00000001
    #define TEE_STORAGE_PRIVATE_REE    0x80000000
    #define TEE_STORAGE_PRIVATE_RPMB   0x80000100
    ```

- **OP-TEE > 2.1.0 (next)**
  - SQLite storage is introduced: CFG_SQL_FS  ?= n
    ```
    #define TEE_STORAGE_PRIVATE_SQL    0x80000200
    ```
Replay Protected Memory Block support (1)

- Added in 2.0.0
- OP-TEE had some legacy RPMB code which was re-worked recently
- Main source files:
  - core/tee/tee_svc_storage.c
  - core/tee/tee_rpmb_fs.c
  - core/tee/tee_fs_key_manager.c
- FS layout is as follows:
  - [ Partition data (512 bytes) | FAT entries -> ... <- File blocks ]
- File Allocation Table entries are dynamically allocated from the bottom while file blocks are allocated from the top
- The allocator is `tee_mm_alloc()` / `tee_mm_alloc2()`
Replay Protected Memory Block support (2)

- eMMC device ID is selected at compile time
  - CFG_RPMB_FS_DEV_ID ?= 0 in mk/config.mk
  - Will use /dev/mmcblk0rpmb
- Device is accessed through Normal World (tee-suppliant)
  - There is no MMC driver inside OP-TEE
- tee-suppliant contains emulation code so it’s easy to test without a real device
  - RPMB_EMU := 1 in tee-suppliant/Makefile
  - Comment out to access the real device
- RPMB key is programmed on first use
  - SHA256(HUK) or a predefined test key if CFG_RPMB_TESTKEY = y
  - Possible attack: replace the eMMC with a new one (or simulate the same) and the key will leak out. We’ll fix that.
Replay Protected Memory Block support (3)

- Atomicity: read/write/rename etc. can’t leave objects in a partially modified state
  - FAT block is updated only after data blocks have been written successfully
  - RPMB spec ensures atomic write of rel_wr_blkcnt blocks or less (at least 1), this is enough for our need
  - Data writes larger than rel_wr_blkcnt cannot be updated in-place; allocate/read/update/write/commit to FAT instead
- HiKey can access the onboard eMMC since kernel 4.8-rc1
  - Commit [af63762](“mmc: dw_mmc: k3: add MMC_CAP_CMD23”)
  - An external eMMC module with a microSD adapter can also be used
Replay Protected Memory Block support (4)
SQLite (1)

- A SQLite backend is proposed as a simpler and more robust alternative to the REE FS
- Based on SQLite3 (public domain) and libsqlfs (LGPLv2 or later)
- Moves the complexity of handling atomic updates from secure world to normal world (tee-suppliant). Indeed, libsqlfs provides:
  
  ```c
  sqlfs_begin_transaction(fs);
  sqlfs_complete_transaction(fs, rollback);
  ```
- TA data are stored in a single file in the Normal World filesystem:
  
  `/data/tee/sstore.db`
Encryption (1)

- Data Encryption is turned on with CFG_ENC_FS=y (default y).
  - Useful even for RPMB, since the device is accessed through the Normal World
- Authenticated block encryption, 1 key per file (File Encryption Key).
- FEK is AES-encrypted using a 256-bit Trusted app Storage Key (TSK) then stored in the file’s metadata
- The TSK is derived from the SSK and the TA UUID using HMAC_SHA256
  - Prevents TA2 from accessing data owned by TA1 when REE FS files are moved from storage area of TA1 to TA2
- The SSK is derived from a Hardware Unique Key (HUK) and a constant string using HMAC_SHA256
Encryption (2)

- **REE FS, SQL FS:**
  - Metadata are always authenticated and encrypted
  - Data may or may not be authenticated and encrypted depending on CFG_ENC_FS (default: y)
  - Algorithm is AES-GCM

- **RPMB FS:**
  - Metadata are never encrypted
  - Data may or may not be encrypted depending on CFG_ENC_FS.
  - Data and metadata are always authenticated (RPMB protocol)
  - Data encryption algorithm is AES-CBC with ESSIV (no need for GCM as above because authentication is handled by RPMB)
xtest updates

- Test all supported backends
  - Client application automatically includes conf.h generated during OP-TEE build, so all CFG_* settings may be used
  - xtest 6xxx loops on all supported storage IDs

- Test TA storage isolation
  - xtest 6015
  - Checks that two TAs won’t share data
  - Checks that REE FS can’t be fooled into opening data storage belonging to another TA

- Concurrency test
  - xtest 6016
  - 4 threads (pthread) run a loop, invoking the storage TA to perform create/write/verify/delete on distinct objects. TA has TA_FLAG_MULTI_SESSION.
What’s next?

- RPMB: don’t program key unless some debug/testing CFG_ is set
- Improve derivation of SSK from HUK
  - Should be done by the hardware crypto module. HUK should never be read by software.
  - Unfortunately, we have no such driver upstream :( 
- Anti-rollback protection and controlled rollback for storage owned by non-secure OS: how?
- Prevent TA impersonation
  - Per-TA signing keys and use root key to sign key pairs only (not the TA itself)
  - Not only restricted to storage
- Further code simplifications?
- Is SQL FS useful? Should it replace REE FS? When?
Thank You

#LAS16
For further information: www.linaro.org
LAS16 keynotes and videos on: connect.linaro.org