What is it for?

- Keys
- Hashes
- Certificates
- Audit logs
- Sensitive user data
- ...anything requiring confidentiality, authenticity or rollback protection
PSA Storage

PSA Internal Trusted Storage (ITS)

- PSA Root of Trust Service
- Internal storage only (e.g. eFlash)
- Storage is inherently trusted: no encryption, authentication or rollback protection required in service itself
- Small datasets (e.g. keys)
- Implemented by TF-M ITS service

PSA Protected Storage (PS)

- Application Root of Trust Service
- Can use external (untrusted) storage
- Storage may be accessible to attacker: option for encryption, authentication and rollback protection in service
- Large datasets
- Implemented by TF-M Secure Storage (SST) service
How do I use it?

- Straightforward developer-facing APIs
  - Accessible to both Non-Secure and Secure callers
- uid/value semantics
  - Set data to a uid
  - Get data associated with a uid
  - Get info about a uid
  - Remove uid
- Access control: each partition can access only its own assets
- Separate APIs for ITS and PS
  - Follow same pattern

```c
psa_status_t psa_its_set(psa_storage_uid_t uid,
                        size_t data_length,
                        const void *p_data,
                        psa_storage_create_flags_t create_flags)

psa_status_t psa_its_get(psa_storage_uid_t uid,
                         size_t data_offset,
                         size_t data_size,
                         void *p_data,
                         size_t *p_data_length)

psa_status_t psa_its_get_info(psa_storage_uid_t uid,
                               struct psa_storage_info_t *p_info)

psa_status_t psa_its_remove(psa_storage_uid_t uid)
```

...and equivalent for Protected Storage
TF-M Secure Storage

- SST and ITS services each provided by their own partition in TF-M
  - ITS is PSA RoT, SST is Application RoT
  - SST depends on Crypto, which depends on ITS
- ITS is smallest possible wrapper around filesystem
  - Main addition is access control based on client IDs
- SST also adds protection for data-at-rest
  - Encryption, authentication, rollback protection
    - Controlled by build flags, depending on required level of protection
  - Authentication & encryption: AEAD (AES-128-GCM) using HUK, via Crypto service
  - Rollback protection: collect MACs in table, keep version in NV counter
TF-M Secure Storage cont.

- Both services use same lightweight flash filesystem as backend
  - Non-hierarchical
  - Integer file IDs
  - Create/write/delete APIs
  - Reliability in case of power failure
    - Can use 2 or >=4 flash blocks
  - No fragmentation
  - Flash layer can use internal or external flash device
PS uses Crypto service for encryption

PS

<init>

psa_open_key(TFM_CRYPTO_KEY_ID_HUK, huk)

psa_key_derivation(&sst_key_generator, huk)

psa_generator_import_key(storage_key, &sst_key_generator)

psa_ps_set(uid, data)

psa_aead_encrypt(storage_key, data)

Crypto Service

HUK

Storage key
Crypto service uses ITS for storing persistent keys

Crypto

- `psa_create_key(key)`

ITS

- `psa_its_set(uid)`
- `psa_its_get(uid)`

- `psa_cipher_encrypt(key, data)`
Upcoming features

• Sharing common filesystem code between ITS and PS
  • SST calls ITS APIs as its backend ‘filesystem’
  • SST partition essentially becomes an encryption, authentication and rollback protection layer on top of ITS
  • Shrinks the stack size of SST, at cost of concurrent requests to ITS/PS APIs requests having to wait
    – (alternative is put FS code in shared code region, would reduce overhead a bit)

• Protected Storage 1.0

• Integration with HW keys, via crypto service

• Scalable internal buffers
  • Support for different profiles

• Key diversification
  • One key per client, or per asset

• NAND flash support
The Arm trademarks featured in this presentation are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. All other marks featured may be trademarks of their respective owners.

www.arm.com/company/policies/trademarks