TF-M platform layer doesn't support real-world usecases well

- Large amounts of hardcoded crypto keys / attestation values
- Dummy implementation of a lot of key functionality require reimplementing
  - Some platforms have done this, notably for the NV counters which has an almost-working implementation
- Key derivation function returns hardcoded keys
- We don't currently offer any support / guidance on how to implement this as a platform
  - Therefore, no platforms (at least in upstream code) provide a production-ready implementation
  - Our best idea currently is to load keys / attestation values from ITS
    - But we don't have any way to provision ITS, or to create ITS filesystems that can be flashed to devices
CryptoCell provisioning has some problems

- Platforms with a CryptoCell-312 accelerator do support provisioning
- There are some issues with the current system however
  - They are provisioned with random IAKs, which causes problems with testing
  - It only provisions the IAK, HUK, and BL2 ROTPKs
    - And only supports 2 BL2 images, which causes issues with certain platforms (notably diphda)
- The provisioning system isn't easily extensible to add new data
- The platforms are still using dummy implementations of the NV counters etc
Proposed solution: add OTP HAL

- Add HAL to allow the platform layer to retrieve data that has been stored in OTP
- Use OTP to store all data that needs to vary on a per-device basis
- Add a mechanism to provision this data on first boot
  - This would then be common across all TF-M platforms, as it uses the OTP HAL to be generic
  - This would be tied to lifecycle-state, which would then also have a real implementation
- By default, use secure on-chip flash instead of real OTP on platforms that lack it
  - This is acceptable to be used in production, provided the flash is locked to secure-privileged access only. Real OTP is still recommended if available however.
  - This makes the OTP HAL compatible with all platforms that support ITS.
  - We can use a tweak in the driver to make this semantically compatible with real OTP (where it's not possible to change a 1 bit to a 0 bit).
Incidental related upgrades

• Since the OTP HAL allows platforms to store secrets in a production-ready way, there is only a small amount non-production-ready code left in the platform layer. Therefore, it makes sense to also:
  • Upgrade the NV counters implementation to one that can handle asynchronous power loss, which makes it acceptable for production
  • Tweak the NV seed implementation be used to provide entropy by default this change went into 1.4, we just need to load the initial seed value from OTP instead of hardcoding it
  • Change the key-derivation operation to perform actual key-derivation instead of returning a hardcoded key
• This leaves us with a platform layer that is by default able to be used in production
  • Provided the platform has internal flash
Design decisions

- API design
- NV counters
- Internal flash allocation
- Provisioning
API design

- API needs to be generic to support different OTP implementations
- We can't make any assumptions on data layout
  - Some implementations may store certain fields in hardware (e.g. CC312 with LCS)
  - Some might have non-contiguous OTP address spaces
- Proposal: Generic getter / setter API
  - Use IDs to indicate which OTP data element to get/set
- Design decision: Don't define how large OTP data elements are
  - Provide an API to get the size of the data element from a given implementation
  - Pro: More flexible – CC312 has limited space so can use more limited NV counters
  - Con: Makes the API harder to use, particularly with regard to memory allocation
  - Alternative: Define sizes in the API
API design (cont.)

```c
enum tfm_otp_element_id_t {
    PLAT_OTP_ID_HUK = 0,
    PLAT_OTP_ID_IAK,
    PLAT_OTP_ID_IAK_LEN,
    PLAT_OTP_ID_IAK_TYPE,
    PLAT_OTP_ID_IAK_ID,
    PLAT_OTP_ID_BOOT_SEED,
    PLAT_OTP_ID_LCS,
    PLAT_OTP_ID_IMPLEMENTATION_ID,
    PLAT_OTP_ID_HW_VERSION,
    PLAT_OTP_ID_VERIFICATION_SERVICE_URL,
    PLAT_OTP_ID_PROFILE_DEFINITION,
    <. . .>}

enum tfm_plat_err_t tfm_plat_otp_init(void);

enum tfm_plat_err_t tfm_plat_otp_read(enum tfm_otp_element_id_t id,
    size_t out_len, uint8_t *out);

enum tfm_plat_err_t tfm_plat_otp_write(enum tfm_otp_element_id_t id,
    size_t in_len,
    const uint8_t *in);

enum tfm_plat_err_t tfm_plat_otp_get_size(enum tfm_otp_element_id_t id,
    size_t *size);
```
NV counters

• Ideally, we'd use the OTP api for all NV counters
  • However, due to OTP semantics only unary NV counters can be stored in OTP
    – 0x3 is encoded as 0b111, not as 0b011, since we can't unset bits
  • Because of Unary representation OTP NV counters are limited on size
    – 4 bytes gets a counter that goes up to 32
• OTP NV counters are suitable for BL2 images
  • Since they need to count to ~512
• OTP NV counters are not suitable for PS
  • Since we need to be able to do arbitrary amounts of writes, which would require too much OTP space
• Proposal: Use the current NV counter implementation for PS and use the OTP api for BL2
  • Con: We need two distinct code-paths
  • Pro: BL2 counters are more secure (when the platform has real OTP)
  • Alternative: Use the current NV counter implementation for both
Internal flash allocation

- Many platforms have very limited internal flash
- Internal flash backing for OTP HAL requires at least two sectors
  - (not needed if the platform has real OTP)
  - To allow for a backup sector to counter asynchronous power loss during writes
- Proposal: Share space with the NV counters
  - Pro: These also require a backup sector
  - Pro: Can share implementation for restoring from backup
Provisioning data is input to the device
- Currently using a debugger, or combining a binary into the image to flash
- This needs consultation to align best with what market require

- The provisioning service interprets a predefined data structure, containing all the required assets
- The provisioning service calls the OTP HAL to insert the data into either real OTP or OTP emulated in embedded flash.
Provisioning (cont.)

- Provisioning is done in two stages, corresponding to the two provisioning lifecycle states
  - Currently HUK provisioning is done in the first stage, all other in the second stage
    - This may be changed based on feedback
  - Provisioning writes to the LCS OTP ID to perform a state change
    - How this is handled may depend on the hardware (Notably CC312)

- Provisioning is done via the OTP HAL
  - This means it is the same on all platforms

- By default, a dummy provisioning bundle will be loaded which contains the current dummy TF-M keys / data

- Where possible, provisioning will continue to run TF-M afterwards
  - Not on CC312, as it requires reboots between lifecycle states
  - This allows a first-run experience that is the same as the current state
Provisioning (cont.)

• Design decision: How to inject the keys/provisioning data
  • Currently there is an empty struct, the provisioner can fill the struct with keys via a debugger
    – The struct contains a magic value that must be set in order for it to be used for provisioning
  • Possibility: A HAL function to fill the struct
    – Allows platforms to have flexibility
  • Possibility: Provide default implementation of a serial-port protocol
    – Read data from the UART?
• It's unclear how well these fit into actual provisioning flows
• Feedback would be welcomed
  – Does this flow fit with intermediate environments (IDEs etc)?
  – Does this flow fit with factory provisioning flows?
Dummy keys

• Dummy keys are still used during development
• There is now a runtime warning if dummy keys are provisioned to the device
• There is also a runtime warning if the keys on the device are dummy ones
  • In case the code is reflashed to disable dummy provisioning, but the OTP isn't / can't be reset
  • This is done by comparing with the first 32 bytes of the dummy HUK
Future improvements

• Support for encrypted provisioning data
  • Unclear which keys would be used for provisioning

• Support for temporary provisioning LCSes with secure debug disabled
  • To prevent leaking decrypted keys

• On-device key generation where entropy is available
Any Questions?

• Patches at [https://review.trustedfirmware.org/c/TF-M/trusted-firmware-m/+/10595/2](https://review.trustedfirmware.org/c/TF-M/trusted-firmware-m/+/10595/2)
  • Subject to changes from feedback
Thank You
Danke
Gracias
谢谢
ありがとうございます
Asante
Merci
감사합니다
धन्यवाद
شكرًا
ধন্যবাদ
תודה