PSA APIs

An overview

Attestation API

- 1.0.1 – we are in the process of dropping a new release which allows signing using symmetric attestation keys (using COSE Mac0)
- Fully documented in ARM IHI 0085
- TF-M master is slightly behind the spec (1.0.0) – 1.0.1 is currently sitting in a dev branch
- Consists of one very simple interface:

```
psa_status_t psa_initial_attest_get_token(const uint8_t *auth_challenge,
                                          size_t challenge_size,
                                          uint8_t *token_buf,
                                          size_t token_buf_size,
                                          size_t *token_size);
```

Attestation API

- If the call returns `PSA_SUCCESS`, then `token_buf` contains the Initial Attestation Token (IAT) as a COSE Sign1 (or maybe Mac0) blob of `*token_size` bytes.
- Its payload is -- like EAT -- a collection of claims
- All signed by a special private key (IAK), which is part of the RoT stashed securely in a protected location (e.g., eFuse)
- The claims can be split into 4/5 distinct categories:
  - Caller related, target identification, target state, SW inventory and verifier.

Attestation Claims (caller)

Claim	Mandatory	Description
Auth Challenge	Yes	Input object from the caller. For example, this can be a cryptographic nonce or a hash of locally attested data. The length must be 32, 48, or 64 bytes.
Client ID	Yes	Represents the Partition ID of the caller. It is a signed integer whereby negative values represent callers from the NSPE and where positive IDs represent callers from the SPE. The full definition of the partition ID is defined in the PSA Firmware Framework (PSA-FF). It is essential that this claim is checked in the verification process to ensure that a security domain cannot spoof a report from another security domain.

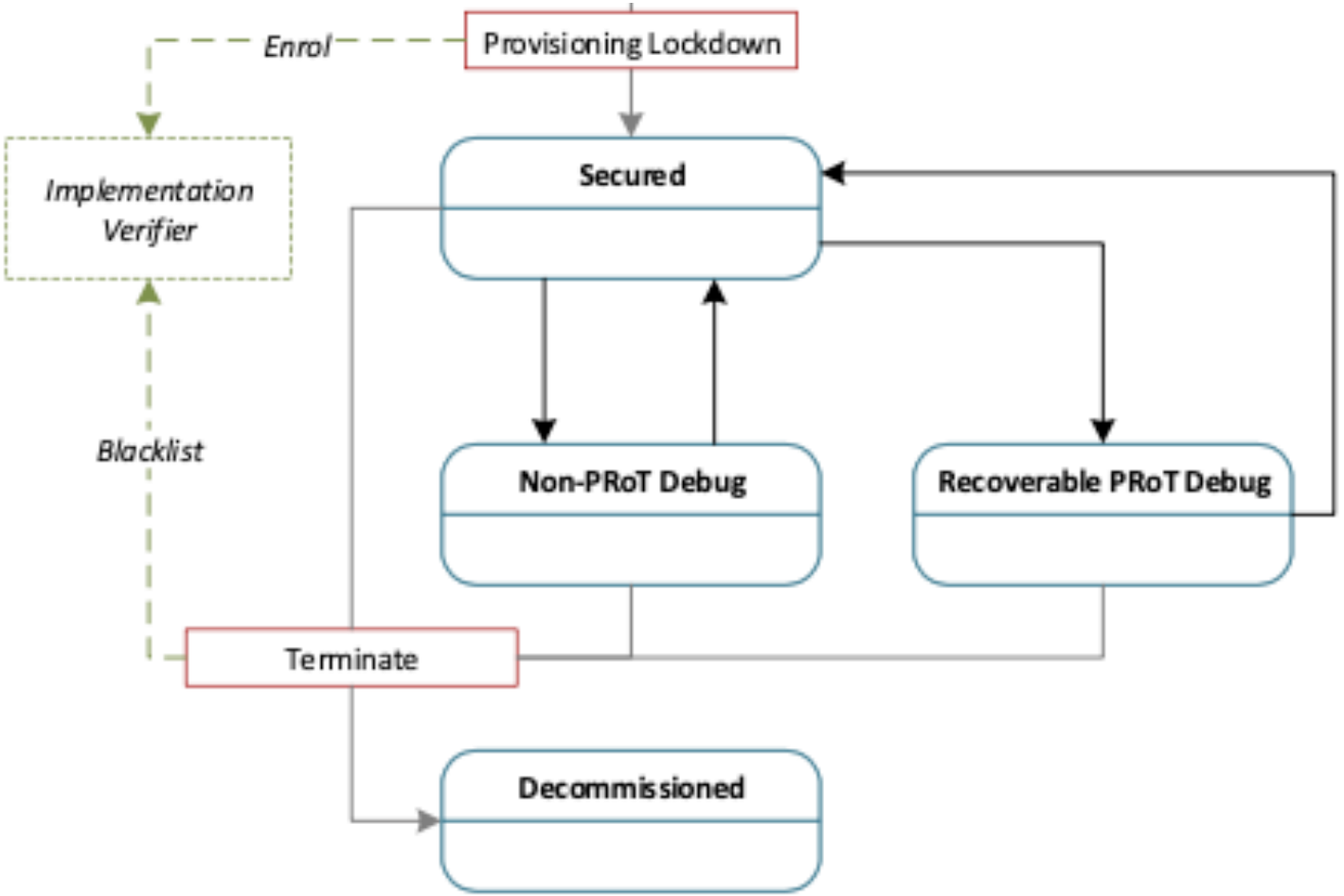
Attestation Claims (target identification)

Instance ID	Yes	Represents the unique identifier of the instance. It is a hash of the public key corresponding to the Initial Attestation Key (IAK). If the IAK is a symmetric key then the Instance ID is a hash of the IAK. The full definition is in the PSA-SM.
Implementation ID	Yes	Uniquely identifies the underlying immutable PSA RoT. A verification service can use this claim to locate the details of the verification process. Such details include the implementation's origin and associated certification state. The full definition is in the PSA-SM.
Hardware version	No	Provides metadata linking the token to the GDSII that went to fabrication for this instance. It can be used to link the class of chip and PSA RoT to the data on a certification website. It must be represented as a thirteen-digit EAN-13
Boot seed	Yes	Represents a random value created at system boot time that can allow differentiation of reports from different boot sessions.

Attestation Claims (target state)

Security Lifecycle	Yes	Represents the current lifecycle state of the PSA RoT. The state is represented by an integer that is divided to convey a major state and a minor state. A major state is mandatory and defined by PSA-SM. A minor state is optional and IMPLEMENTATION DEFINED. The PSA security lifecycle state and implementation state are encoded as follows: • version[15:8] – PSA security lifecycle state • version[7:0] – IMPLEMENTATION DEFINED state. ...

Attestation Claims (target state)



Attestation Claims (target SW inventory)

Software components	Yes (unless the No Software Measurements claim is specified)	A list of software components that represent all the software loaded by the PSA Root of Trust. This claim is needed for the rules outlined in the PSA-SM. Each entry has the following fields: 1. Measurement type 2. Measurement value 3. Version 4. Signer ID 5. Measurement description The full definition of the software component is described in Software Components This claim is required to be compliant with the PSA-SM.

Attestation Claims (verifier)

Verification service indicator	No	A hint used by a relying party to locate a validation service for the token. The value is a text string that can be used to locate the service or a URL specifying the address of the service. A verifier may choose to ignore this claim in favor of other information.

...

Attestation API (usage examples)

- Device enrollment
  - CA agent:
    - Send nonce;
  - Device:
    - Create key-pair;
    - Create proof-of-possession (e.g., CSR);
    - Hash CSR and nonce and use it as the challenge to create an IAT;
    - Send CSR and IAT to the CA agent;
  - CA agent:
    - Verify IAT, CSR and their binding;
    - If everything is OK, create certificate and send it back to Device.
- Dynamic / derived claims (e.g., certification)
- Integrity reporting / efficient upgrade campaign
- … add your own!

Crypto API

- 1.0.0 beta 3 -- never been closer!
- Fully documented in ARM IHI 0086
- Scalable implementation via profiling
- Mem constrained: streaming interface (multipart processing)
- Key material is opaque to the caller (it’s always used through handles)
  - Typically -- unless policy explicitly allows -- cannot be extracted
- Interface to crypto accelerators not currently spec’d (draft)

Crypto API

- No strong policy stance: not an opinionated API
- Granular interface: callers must precisely identify their algorithm choice, key sizes, parameters
  - Risk of shooting yourself in the foot
  - Trade-off w/ support for legacy protocols
  - Responsibility shifted to the API implementor
- Allow multiple architectural realisations (from linked library a la mbedTLS, to frontend-backend with crypto accelerator / SE, to PARSEC-like architectures, i.e. 1 backend and multiple frontends)

Crypto API (use cases)

- TLS
- Secure storage
- Network credentials
- Device pairing
- Secure boot (FW integrity and authenticity)
- Factory provisioning

Storage API

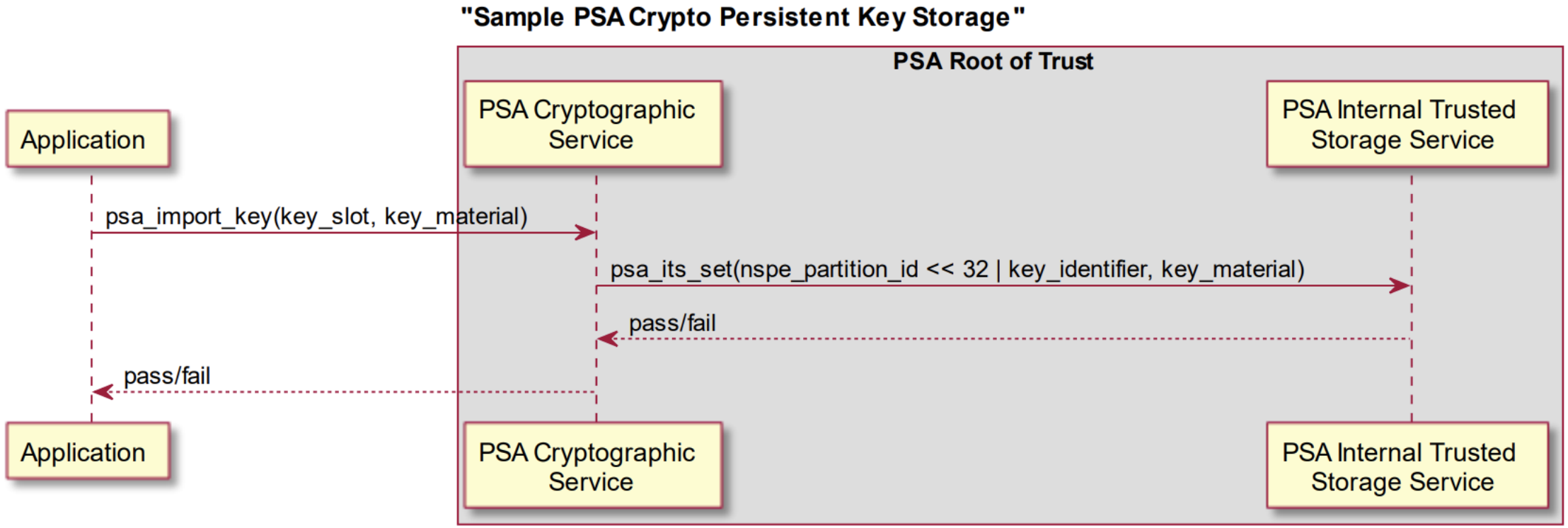
- 1.0.0
- Fully documented in ARM IHI 0087
- K-V storage interface to be used with device-protected storage
- Each unit of storage is name spaced, which means one partition can never access data of another partition
- Simple get/set/remove API
- Reference implementation in TF-M

Storage API

Two different interfaces are described:

- Internal trusted storage, implemented by the PSA RoT
  - Small(-ish) objects, on-chip flash
  - Secure storage for device's "intimate" data, e.g.:
    - Identity keys (privacy protection)
    - FW verification keys (integrity protection)
    - FW rollback counters (integrity protection)
- Protected storage, implemented by the ARoT or even NSPE if there are no ARoT services
  - Larger objects, hosted on externally to the MCU package, typically on external flash
  - Protection of data at rest

Storage API (example)



PSA API recap

- They are public, they are well documented (we think), they are implemented (although at different stages of maturity, but we are quickly converging)
- Please use them and tell us what you think!