Platform Security Architecture Application Guide:

How to secure a smart door lock using
PSA principles and Arm security technologies

Revision 2.0

2019

In collaboration with Cypress Semiconductor

Abstract

With the explosive growth in the Internet of Things (IoT), and the number of devices soaring, security has become a huge challenge globally. Designing consistent security across connected devices can be a minefield to navigate and implementation can be costly. The Platform Security Architecture aims to address this challenge. It is a common framework that enables the IoT ecosystem to move forward with stronger, scalable security and greater confidence. PSA reduces the cost and risk associated with deploying robust security solutions, whilst also speeding up the time-to-market. To learn more about PSA and the benefits, you can check out this whitepaper.

This application guide describes how to design security for a smart door lock device using PSA guidelines and principles. It examines how to develop a threat model for a smart door lock and shows how security counter-measures are derived effectively for this application. It considers the hardware architectures and security IP necessary to maintain asset security; from trusted boot to secure hardware partitioning. Additionally, using the PSA Firmware Framework, smart door lock functionalities are isolated and partitioned into Secure and Non-secure processing environments.

Finally, this guide shows how open source Trusted Firmware-M APIs are used as a mechanism to isolate smart door lock security critical functionalities from Non-secure code.

In this whitepaper, we will look at two implementations of TF-M; firstly with the Armv8-M architecture, and secondly with a dual Armv7-M architecture with additional security features for hardware-based isolation.

By applying PSA principles and guidelines; analyze, architect, implement and certify, this application guide walks through some of the main steps necessary to achieve a holistic robust security solution for a smart door lock device. This guide demonstrates that applications like a smart door lock can adopt PSA principles easily and reap the benefits of consistent, predictable security.

The application guide is for system architecture developers and security personnel, either from chip vendors, RTOS vendors, or OEMs. It assumes that the reader is familiar with basic security concepts in networking, operating systems, and data protection, as well as standard techniques in cryptography, such as authentication, hashing, encryption, and digital certificates. A basic knowledge of the PSA specifications will also make it easier to follow this application guide.

Glossary

Term	Meaning
AEAD	Authenticated Encryption with Associated Data
API	Application Program Interface
ARoT	Application Root of Trust
eMMC	Embedded Multimedia Card
	(Low-cost flash memory with a built-in controller)
FF	Firmware Framework
GP	GlobalPlatform
IETF	The Internet Engineering Task Force
IoT	Internet of Things
IPC	Inter-process Communications
IRQ	Interrupt Request
MAC	Message Authentication Code
MPU	Memory Protection Unit
MPU	Micro Processor Units
NB-IoT	Narrow Band Internet of Things
NFC	Near Field Communication
NSPE	Non-secure Processing Environment
NVM	Non-volatile memory
OEM	Original Equipment Manufacturer
OS	Operating System
OTP	One Time Programmable
	(A characteristic of some types of NVM)
PIN	Personal Identification Number
PKCS	The Public-Key Cryptography Standards
PSA-RoT	PSA Root of Trust
PSA	Platform Security Architecture
ROM	Read-only Memory
ROTPK	Root of Trust Public Key
	(for firmware verification)
RPMB	Replay-protected Memory Block
RSA	Rivest, Shamir and Adleman
	(An algorithm for public-key cryptography)
RTOS	Real Time OS
SCA	Side-channel Attack
SoC	System on Chip
SPE	Secure Processing Environment
	(Contains the PSA-RoT and the ARoT)
SPM	Secure Partition Manager
TBFU	Trusted Boot Firmware Update

Glossary

Term	Meaning
TBSA-M	Trusted Base System Architecture for Armv6-M, Armv7-M and Armv8-M
TBSA-M ACK	Trusted Base System Architecture for M Profile Architecture Compliance Kit
TF-M	Trusted Firmware for M-class processors
TMSA	Threat Model and Security Analysis
TOE	Target of Evaluation
TRNG	True Random Number Generator
TSF	TOE Security Functionality
T.xxxx	Threat.xxxx (The threat classification representative. T. = Threat)
UI	User Interface
YAML	Yet Another Markup Language

Contents

1. PSA Introduction

Arm technologies are found in billions of Internet of Things (IoT) devices. The success of IoT is heavily dependent upon the trust and security built into billions of different connected devices. As security attacks and threats continue to evolve, and the attack surface grows, so must the counter-measures to combat them.

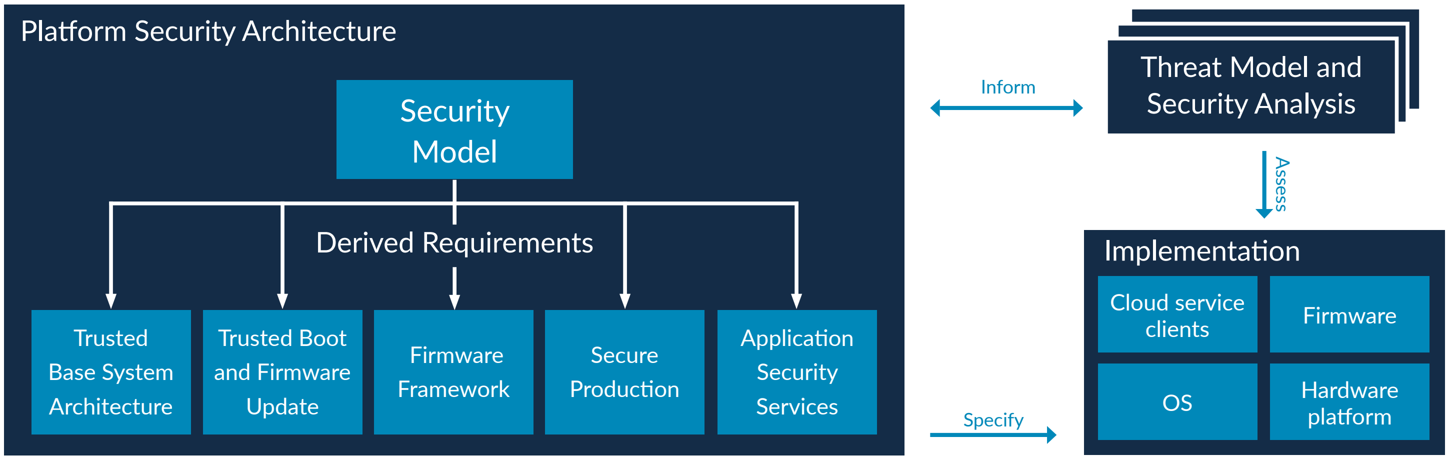
However, security events that have happened in past years have revealed that many IoT products are not designed with adequate protection against malicious attacks. As a result, the industry still faces many challenges:

- Security can be expensive to implement throughout a device's lifecycle.
- IoT device security is difficult to manage at scale.
- Security specialists are expensive and in short supply, particularly for smaller businesses and start-ups.
- The security landscape is ever-evolving, with new attack vulnerabilities continuously emerging.
- A lack of confidence in the data being passed to, and from, sensors and actuators.

These challenges mean that the industry is currently unable to fully realize the economic benefits of IoT. With these trends and security issues in mind, Arm announced the Platform Security Architecture (PSA) in 2017. The PSA is a robust system architecture covering both hardware and firmware, codifying these common security principles into a set of system requirements and interfaces. It is aimed at different entities throughout the supply chain, from chip designers and device developers to cloud providers and network infrastructure providers and software vendors. As open-source solutions to the PSA, we have Trusted Firmware-M (TF-M) which launched in early 2018 and Arm Mbed OS to address security in device hardware, software, and communication throughout the device lifecycle. TF-M supports Armv8-M or dual Armv7-M-based Cortex-M processors with flexible and configurable design which allows adaptation to fit different customer needs. In addition, the cloud-based Arm Pelion IoT Platform offers a flexible, secure, and efficient foundation spanning connectivity, device, and data management which enables protection during the entire lifetime of the IoT infrastructure and intelligent devices.

PSA draws and builds upon best practice from across the industry, which enables Arm partners and the wider ecosystem to consistently design-in the right level of security for all connected devices.

Figure 1: Platform Security Architecture


Platform Security Architecture
Security Model
Derived Requirements
Trusted Base System Architecture
Trusted Boot and Firmware Update
Firmware Framework
Secure Production
Application Security Services
Inform
Threat Model and Security Analysis
Assess
Implementation
Cloud service clients
Firmware
OS
Hardware platform
Specify

As illustrated in Figure 1, PSA provides a recipe for building secure systems and it is defined in four stages:

- Stage one: Analyze (a process where you analyze your product, assets and security risks, in a process called Threat Models and Security Analyses - TMSA).
- Stage two: Architect (Firmware and hardware architecture specifications).
- Stage three: Implement (hardware IP and open-source reference code).
- Stage four: Certify (multi-level assurance scheme, PSA Certified™).

This application guide demonstrates how these four phases of the PSA can be applied to secure an IoT device.

It outlines the design and implementation of a smart door lock device, by leveraging PSA, security IP and TF-M. The smart door lock is a popular topic in the IoT sphere as it relies on a solid security implementation and maintenance, since a security breach might lead to severe damage, such as a loss of goods or physical danger.

A basic set of PSA guidelines are considered from the perspective of an IoT device architecture or developer. It is not a comprehensive smart door lock solution covering all the security considerations, it also does not cover physical mechanisms. As a result, the intention is to reduce, rather than eliminate, the requirement for security analysis during system design.

2. Analyze - PSA TMSA

2.1 PSA TMSA overview

Security needs to be considered at the outset when designing a device or a system-on-chip (SoC). With the inherent diversity of IoT there is a greater need for device manufacturers to have a reference Threat Model and Security Analysis (TMSA) for their products. Arm has created a series of TMSA examples for different IoT products to show how this might be done, in a way that is understandable for those who are not experts in security.

The PSA TMSAs also contain useful appendices that show how Arm TrustZone and Arm CryptoIsland technology can be used to meet some of the Security Functional Requirements (SFRs) identified during the threat model and security analysis process. These documents are useful as a starting point and a developer can derive a bespoke TMSA for a particular target device [1].

The TMSA helps to understand security threats and requirements, and it addresses the following aspects:

- The assets to protect in the device.
- The likely threats.
- The scope and severity of potential attacks.
- The types of potential attacker and their methods.
- The mitigations that are needed.

In this analyze stage, we propose a security model for a smart door lock and present an outline of the TMSA, including the assets to protect, an adversary model, threats and the corresponding security objectives.

2.2 Smart door lock - how much security is needed?

IoT devices have varying security requirements – a wireless sensor in a field that measures sunlight and the water content of the soil will probably not be subject to the same level of malware or hardware attacks as a smart door lock that runs a comprehensive app that interacts with the external world.

Security is always a balance between the cost and effort that the system designer is prepared to invest to protect assets, and the cost and effort an attacker is likely to spend on an attack, as shown in Figure 2.

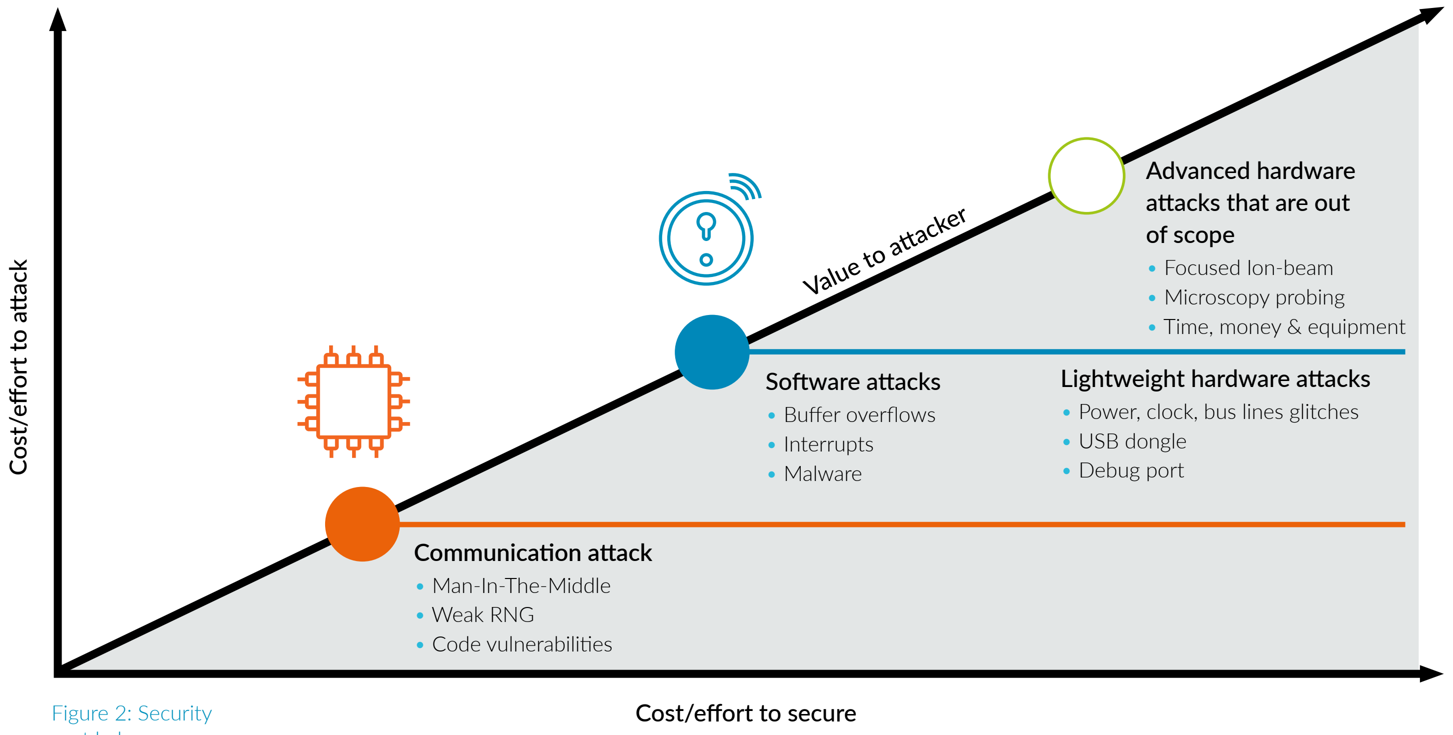


Figure 2: Security cost balance

In this application guide, we assume that protection from software attack and lightweight electrical attacks, such as probing and JTAG attacks, is a minimum requirement. If this level of protection is chosen, then the design must not have shared private keys (a class key used among groups, i.e. a device model) to minimize class break vulnerabilities via a side-channel attack (SCA) or a perturbation attack.

The designer can choose to have more advanced threats in scope and to add additional protection in silicon, for example counter-measures to SCAs and perturbation attacks.

2.3 Smart door lock overview

A smart door lock is an electromechanical lock which performs locking and unlocking operations on a door when it receives instructions from an authorized device using a wireless protocol and a cryptographic key to execute the authorization process [2]. The smart door lock also monitors access and sends alerts for the different monitored or critical device events.

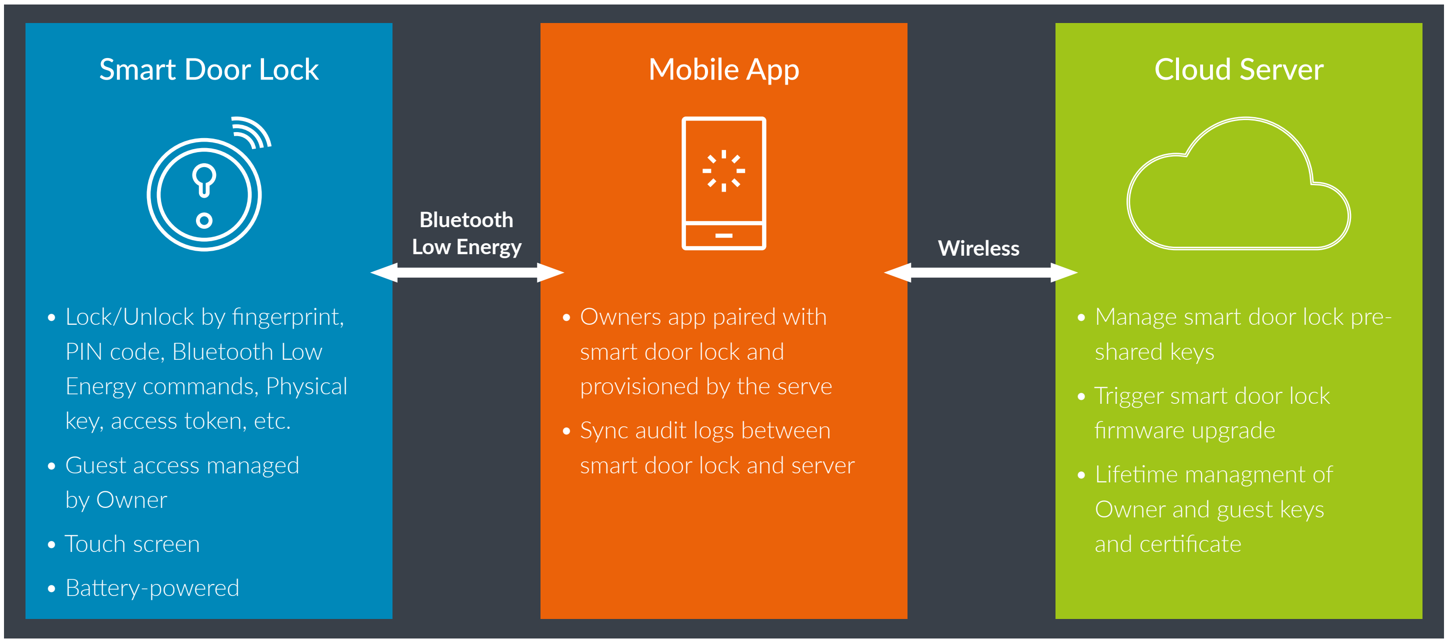
Many smart door locks offer a mobile application which allows the user to lock and unlock doors remotely. Connectivity is the most important function that is supported by the many available technologies.

The popular connectivity candidates include Bluetooth Low Energy, Wi-Fi, ZigBee, 2G/3G/4G cellular, NB-IoT and NFC, amongst others.

Although each connectivity option has unique capabilities and advantages, all smart door locks share some common requirements, such as low-power and security. Long battery life is a key requirement because most smart door locks are battery-powered and replacing or recharging the batteries is inconvenient. Bluetooth Low Energy is a dominant technology on the market today because its low energy requirements result in a battery life of up to five years [3]. Wi-Fi adds native internet connectivity, but Wi-Fi significantly increases the power consumption and shortens the battery life from around five years to one year.

In this application guide, we look at the most popular network architecture and features of the smart door lock device in relation to PSA analysis, architecture and implementation. Figure 3 shows a typical smart door lock system in a simple conceptual diagram:

Figure 3: Smart door lock system


Smart Door Lock
Lock/Unlock by fingerprint, PIN code, Bluetooth Low Energy commands, Physical key, access token, etc.
Guest access managed by Owner
Touch screen
Battery-powered
Bluetooth Low Energy
Mobile App
Owners app paired with smart door lock and provisioned by the serve
Sync audit logs between smart door lock and server
Wireless
Cloud Server
Manage smart door lock pre-shared keys
Trigger smart door lock firmware upgrade
Lifetime managment of Owner and guest keys and certificate

2.4 Smart door lock assets

The Target of Evaluation (TOE, also known as the use case) assets for a smart door lock are categorized as:

- Smart door lock device ID.
- Firmware and its certificates.
- Owner/guest credentials, including biometric data.
- Audit logs.
- Configuration and user data.
- Network connectivity.
- Biometric sensor, Bluetooth Low Energy and other hardware resources that are out of the TOE.

2.5 Adversary model

An attacker is a threat agent (a person or a process acting on behalf of an agent) that tries to undermine the TOE security policy and the TOE Security Functionality (TSF). The attacker tries to change the properties of the assets defined in section 2.4.

The following list details the threat adversary models that smart door locks might want to protect against:

+ Remote attackers:
  - Software attackers are the most prevalent type of attacker.
  - Network attackers, such as man-in-the-middle.
+ Local/physical attackers:
  - Local software attackers.
  - Network connectivity.
  - Simple hardware attackers with limited resources, knowledge or equipment. Examples include, a USB dongle, debug port, voltage/current measurement or port scanner.
+ Smart door lock-specific attackers:
  - Relay attackers who try to set up a communication channel, such as via Bluetooth Low Energy.
  - Revoked attackers with temporary legitimate access permissions. The permissions would be revoked.
  - Thief attackers who steal the smart door lock owner's authorized device, such as a smartphone or Bluetooth Low Energy token [4].
+ Other attackers (beyond the scope of this application guide):
  - Insider attackers who may be from the OEM, ODM, silicon vendor or another third party (developers, server operator).
  - Advanced hardware attackers who are capable of mill down, focused ion-beam lithography, microscopy probing, etc.

2.6 Security threats and attack patterns

The smart door lock's network architecture, features, assets to protect, as well as potential attackers have been identified. Taking all these aspects into account, the STRIDE threats and corresponding attack scenarios listed below are addressed in this TMSA:

+ T.Spoofing ("T." represents "Threat")
    - An attacker impersonates a legitimate admin, owner or guest of the smart door lock and performs malicious actions.
    - An attacker forces remote entities to recognize a rogue device under its control as a valid smart door lock by modifying or cloning device ID, or replacing a remote entities certificate, to gain illegal access to the configured system network.Local/ physical attackers:
+ T.Tampering
    - An attacker succeeds in loading and executing rogue code onto the smart door lock.
    - An attacker abuses the firmware update mechanism to do version rollback and exploit a version with a legacy bug.
    - An attacker abuses the digital port, such as debug features or physical access to modify local assets.
    - An attacker intercepts and modifies network communication data.
    - An attacker uses a simple side-channel analysis attack to interfere with the circuit and to try to reset the smart door lock into a default state (which can be "unlocked").
+ T.Repudiation
    - An attacker modifies event logs that are stored locally or in transit to suppress critical alerts/events or to erase security events.
    - An attacker changes the system time to make events that are logged with the wrong time stamp.
+ T.Information-Disclosure
    - An attacker carries out reverse engineering by extracting firmware stored in local memory or by intercepting the firmware OTA package.
    - An attacker extracts sensitive information, such as confidential data or user privacy data. This extraction of sensitive data can be done by exploiting weak cryptography to steal assets from local storage or network packages, or by eavesdropping buses and other side-channel analysis methods.
+ T.Denial-of-Service
    - An attacker injects rogue code or exploits a firmware flaw to make the smart door lock permanently non-functional.
    - An attacker tampers with the critical configuration to make the smart door lock disconnect from the network.
+ T.Elevation-of-Privilege
    - An attacker exploits the debug port to inject rogue code and take control of the smart door lock in order to abuse other resources such as the biometric sensor, cylinder or to drain the battery.
    - An attacker exploits the control flow for device usurpation or to cause abnormal behavior.

2.7 Security objectives

As counter-measure mechanisms to the security threats identified in section 2.5, the major security objectives are set to:

- Device Identification and Attestation – Uniquely identify the client device and support attestation and data binding based on the unique ID.
- Software Isolation – Isolating the execution environment between the NSPE and SPE, also between the PSA-RoT Services and other code of the SPE.T.
- State-of-the-art cryptography algorithms and key sizes – Certificate-based authentication and communication, plus cryptographic services including True Random Number Generator (TRNG), symmetric encryption, and nonce counter, etc.
- Trusted Boot and Secure Firmware Update – Trusted boot and secure update, including anti-rollback and security epoch feature.
- Secure Storage – For TOE assets, protection against data extraction, data rollback (via security epoch version), data sharing or device clone.
- Secure State – Maintain the runtime secure state which is measurable and attestable, including protection from simple physical tamper and SCA.
- Access Control – Authenticate admin before granting access to the smart door lock configuration and logs and before performing a firmware update.
- Audit Log – Maintain a log of all significant events and allow access and analysis of these logs to authorized admin only.
- Secure Debug – Restrict access to debug features by a deactivation or access control mechanism.

	Threats					
Security Objectives	T. Spoofing	T. Tampering	T. Repudiation	T. Information-Disclosure	T. Denial-of-Service	T. Evaluation-of-Privilege
Device Identification and Attestation	✓					
State-of-the-art Cryptography	✓	✓	✓	✓	✓	
Software Isolation	✓	✓	✓	✓	✓	✓
Trustred Boot and Secure Firmware Update		✓			✓	✓
Secure Storage	✓	✓	✓	✓		
Secure State		✓			✓	✓
Access Control	✓	✓				✓
Audit Log			✓			
Secure Debug		✓		✓		✓

Table 1: Security objectives rationale

The mapping matrix in Table 1 shows the suitability of each of these security objectives to manage the identified security threats. Please refer to the "Security Objectives Rationale" section of PSA TMSA specifications [1] for examples and more details on the rationale of the mapping matrix.

2.8 Security policy

Security policy, which is beyond the scope of this application guide, is also critical to the overall smart door lock system security. Security policy controls the key management for cryptographic keys, as well as their credentials and certificates. These cryptographic keys are securely managed during the lifecycle of the smart door lock when it is used without the smart door lock device.

2.9 TMSA threats summary table

Table 3 in the appendix details assets, threats, counter-measures, security requirements, and the corresponding Arm security IP for the smart door lock. Both the PSA specifications and threats summary table provide useful implementation guidance. In the following sections, this application guide details the design and implementation of the smart door

3. Architect - PSA Specifications

3.1 PSA TBSA-M overview

The foundation of PSA separates the system into a Secure Processing Environment (SPE), which is for the sensitive assets and the code that manages them, and a Non-secure Processing Environment (NSPE), which is where the main application and communication firmware executes. The SPE is isolated from the NSPE. PSA TBSA-compliant devices are required to implement hardware to support the PSA isolation model.

The following hardware mechanisms can be used to implement the PSA isolation:

- Memory Protection Unit (MPU) based isolation.
- TrustZone-based isolation.
- Dual Micro Processor Units (MPUs) or Multiple CPUs.
- Trusted Subsystem (integrated/off-chip).
- Other isolation solutions, such as Custom Logic.

One way to achieve software isolation, is with Arm TrustZone. This provides a solution with optimum robustness, performance at low cost, with wider applicability. To address the security of all embedded and IoT markets, especially those that require efficient security or digital signal control, Arm Cortex-M23 and Cortex-M33 were introduced in 2016. Cortex-M23 targets the most area-constrained and energy-constrained applications and the Cortex-M33 targets the more capable systems. Armv8-M with the TrustZone security extension allows multiple security domains to exist within a single processor system, and this extension provides a significant security enhancement over what was possible with earlier architectures.

The Cortex-M23, Cortex-M33 and Cortex-M35P processors achieve an optimal blend between real-time determinism, energy efficiency, software productivity, and system security. This blend makes many new applications and opportunities across diverse markets possible.

In addition to the above, Armv7-M based dual-core architectures can also be used to achieve the same security goals by implementing hardware mechanisms that meet PSA isolation requirements.

To meet the higher security requirements for robustness or efficiency, an assisted architecture can be adopted. An assisted architecture has one or more trusted subsystems. A typical example of an assisted architecture is adding a dedicated hardware to accelerate and offload some of the cryptographic operations from the SPE software and to provide increased protection to high-value assets, such as Root of Trust (RoT) keys.

Such assisted architectures can implement a hardware Key Store that allows use of the keys by cryptographic accelerators at the same time as preventing the keys from being read by the NSPE and SPE software. Assisted architectures can contain hardware for governing lifecycle state transitions and enforcing lifecycle state policies. In addition, assisted architectures can also provide hardware counter-measures for:

- Invasive attacks, such as probing.
- Side-channel attacks (SCAs), for example power and electromagnetic emission analysis.
- Perturbation attacks, for example, clock or voltage manipulation.
- Detected tamper events by a hardware-initiated response.

The Arm CryptoCell Family is a comprehensive security subsystem solution that can be used to implement an assisted architecture. CryptoCell provides security services, additional trust anchors, and security mechanisms to ensure that the execution state is safe. These security mechanisms include:

- Persistent storage of secrets.
- Rollback prevention.
- Validation of loaded software.
- Validation of software updates.
- Cryptography.
- True Random Number Generation (TRNG).

In the Arm CryptoCell family, CryptoCell-312 is specifically for low-power, low-area designs.

The Arm CryptoIsland product family is a security enclave that enables on-die robust platform security services. CryptoIsland-300 is based on the Armv6-M CPU and CryptoCell-312. Such a security enclave is fully isolated from other execution environments, which, given its ease of use, makes it suitable for stringent certifications.

Physical tampering and simple SCAs are also possible threats to a smart door lock device. CryptoIsland-300P is an advanced novel solution that mitigates the threat of SCA at the source of the problem, by drastically reducing the leakage of sensitive information through power consumption and electromagnetic emanations. Cortex-M35P (Armv8-M with TrustZone) further extends the anti-tampering features to add physical resilience and system safety functions, such as lockstep, configurable parity, and observability, at the same time as maintaining performance.

In addition to isolation, the PSA TBSA-M also defines a set of hardware requirements. For example:

- Infrastructure of access policy on transactions, interrupt, secure RAM.
- Fuse function and confidentiality.
- Cryptographic key usage and size.
- Trusted Boot.
- Trusted timers.
- Version counter.
- Entropy source.
- Debug protection module.
- External Interface Peripherals.

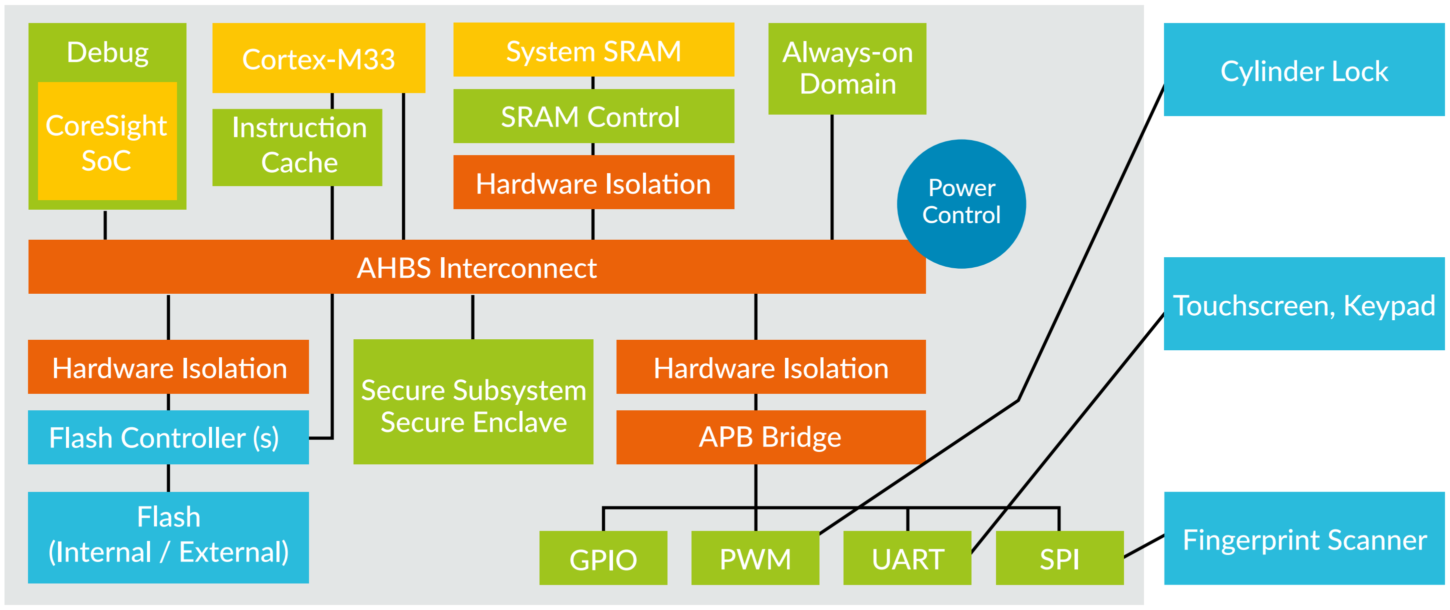


Figure 4:
Example smart door lock SoC

*1: ***Hardware Isolation*** *can be Arm TrustZone or Custom Logic (hardware isolation IP).*

*2: ***Secure Subsystem / Secure Enclave*** *can be TrustZone Filters + CryptoCell, CryptoIsland or a separate secure core.*

The example SoC in Figure 4 fulfils the TBSA-M requirements such as base system isolation, cryptography trusted boot, and debug protection, etc.

Most security assets and settings that need to be stored on-chip require OTP non-volatile storage, i.e. Fuse or a secure element or other trusted embedded NVM, in order to ensure that values cannot be changed. The following table lists several typical fields and sizes:

Name	On-chip data size	Off-chip data size	Notes
Hardware Unique Root Key (HUK)	128 bits	0 bits	Access by trusted code or by trusted hardware only
Root of Trust Public Key (ROTPK)	RSA 3072 bits (key)	0 bits	Three options of ROTPK
	128 bits (digest)	RSA 3072 bits (key)	Digest option is to save OTP size and cost
	ECC 256 bits	0 bits	ROTPK should be accessible only during boot
Version Counter	64 values	0 bits	For SPE version control

Table 2: Root keys & Version counter

As shown in Table 2, TBSA-M mandates two embedded root keys for different security purposes:

+ **HUK** - Instance unique hardware key (Hardware Unique Key) that provides Root of Trust for confidentiality or authenticity. HUK is the root seed for deriving other Root of Trust secrets, i.e. a device binding key.
+ **ROTPK** - The public key half of an asymmetric key pair. ROTPK is responsible for securely authenticating the first stage of the mutable code and provides Root of Trust for authenticity.

To reduce fuse cost, Key Derivation Functions (KDFs) can be used to generate several symmetric keys or asymmetric key pairs for different purposes. However, KDFs can also reduce the security impact when key disclosure happens. A key derivation operation must use a cryptographic one-way function that preserves the entropy of the source key. Common derivation constructions use a keyed Hash Message Authentication Code (HMAC) or a Cipher-based Message Authentication Code (CMAC). Please refer to the NIST's recommendations [5] for more details on the HMAC and CMAC. The following Figure 5 shows a possible KDF usage scenario. In the example, the KDF generates asymmetric ECC key pairs, a temporary AES session key and a secure storage data binding AES key.

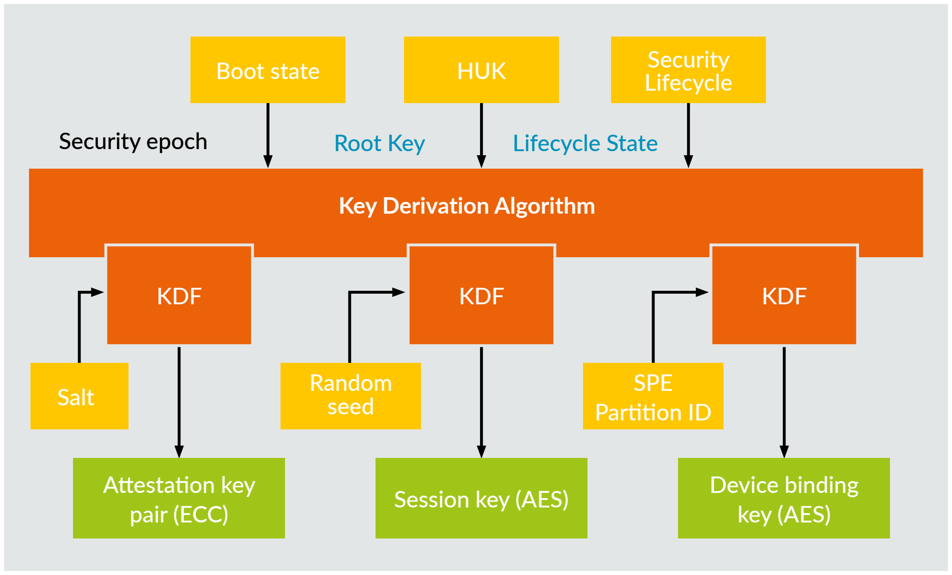


Figure 5: Key Derivation Functions (KDFs)

The TBSA-M requires that the HUK is not exposed to the NSPE. Therefore, this device root key must be moved to execute inside the SPE, or through the key management feature in assisted architecture, i.e. CryptoCell or CryptoIsland. The TBSA-M also requires that power management is controlled from the SPE.

This application guide does not aim to cover all these details of the TBSA-M requirements. Audiences who are working on SoC design and who are aiming to be PSA-compliant should read the PSA TBSA-M specification and look at the TBSA-M test kits which are designed so that chip vendors can accelerate development cycles. Test Scenario and Test Validation Methodology documents in the source tree (under /docs folder) give more details on test cases and how to use them to check the hardware implementation against the PSA TBSA-M specification.

3.2 PSA TBFU

3.2.1 PSA TBFU Overview and Keynotes

To ensure that only authenticated code runs on a device, trusted boot and chain of trust must be established. The PSA Trusted Boot and Firmware Update (TBFU) specification describes the technical requirements that are needed to ensure that, from the point of a SoC reset, only the correct and intended firmware, operating system, and Root of Trust Services, will be authenticated, loaded and executed. Trusted Boot refers specifically to the concept of validating that an image is authorized before it is booted. Firmware Update refers to the verification of the update before it is stored to the NVM. These two concepts (Trusted Boot and Firmware Update) are complementary to each other.

A. Cryptographic algorithms

The PSA TBFU requires firmware validation to use public key cryptography, such as RSA or ECC. Cryptographic algorithms can be implemented with hardware-dedicated engines or in software. Both implementations can be protected against non-invasive side-channel attacks (SCAs). Secure Hash Algorithm 2 (SHA-256 and SHA-384) should be used for all cryptographic hashes. Any use of RSA must follow the PKCS#1 standard.

B. Storage

Root of Trust starts with trusted software stored in the internal, immutable memory inside the SoC. Internal or external flash, that can be easily reprogrammed or erased, is considered mistrusted storage. As a result, the device relies on authentication to protect it from tampering. By providing data at rest protection, data in motion protection, and data that is bound to the SoC identity - to prevent cloning and substitution attacks - we have mutable storages that are trusted.

C. Chain of trust

The device must include at least one immutable firmware verification public key, known as a Root of Trust Public Key (ROTPK). Any use of a MAC to authenticate a firmware image manifest must be done in the SPE, use an on-chip key, and be in the form of an HMAC or a CMAC signature. Secrets used by a trusted component must be scrubbed from volatile memory before ownership of the memory is transferred to a less trusted component.

D. Image verification

Each loaded image must be verified before execution and the boot process must be uninterruptible during signature verification to prevent race conditions. The update process must be an atomic operation. Authentication data used to verify images must be in on-chip memory before use.

E. Firmware upgrade and manifest

Firmware upgrade consists of a manifest and an image. A manifest contains metadata about the firmware image and the metadata is protected against modification. The manifest should contain at least the following fields:

- Format version.
- Image signing key ID.
- Image hash algorithm and value.
- Image size and type.
- Product class.
- Software version.
- Security epoch.
- Manifest signature.
- The Internet Engineering Task Force (IETF).
- SUIT manifest format (recommended for a constrained device).
- X.509v3 content certificate format (recommended for a capable device).

Firmware images are authenticated to check the provenance and integrity, and to check the authorizing update against a device security policy. The security policy might include:

- The public keys required to verify the image.
- Whether the update is targeted for that specific device, based on the device identity.
- Whether the device has enough power to perform all the required steps.

Image updates that include security enhancements or vulnerability fixes must increase the counter value when signing the manifest. Extremely severe security vulnerabilities which require permanent revocation of older images increase the security epoch value.

F. Anti-rollback

Firmware must use non-volatile (NV) version counters to protect against rollback. On-chip secure storage, One Time Programmable (OTP), secure elements or eMMC RPMB are possible solutions. This monotonic rollback counter is in one-way growing, never overflows, and has at least the 64 values that are required for the SPE. Only images of a higher version or the same version can be installed, and the rollback counter is increased to match the successfully installed higher version.

G. Key management and signing

The security of the trusted boot process is primarily dependent on the secrecy of the private portion of the ROTPK. If the firmware image signer loses control of the private signing key, it must be revoked if the private key of the ROTPK is not compromised. This revocation is achieved by signing a new key certificate with an incremented key version value.

Furthermore, the PSA TBFU specification provides guidance information on security features such as measured boot and attestation, security epoch, and best practices for image swapping in a firmware upgrade, etc.

3.2.2 Smart Door Lock - Trusted Boot Design Consideration

If possible, split the bootloader into two stages –immutable and mutable. The first code of the trusted boot process is an immutable bootloader placed in a boot ROM or a locked eFlash sector.

The immutable bootloader must only contain the flash and cryptographic primitives that are necessary to read and validate the next stage. Additional functionality should be an upgradable part of the SPE software, either in a secondary bootloader or a Secure Partition Manager (SPM). In practical terms, a mutable secondary bootloader can be part of the SPE to simplify the booting chain.

The immutable bootloader must enable any available watchdog timers as soon as possible before the next stage to reduce the risk of tampered memory from a physical attack. It is also recommended that the physical security and storage of keys, controlled access to those keys, and auditing of access must be assessed. This assessment is particularly relevant for the creation of certificates for use in production and development of devices. In addition, appropriate processes for the handling and management of signing keys should also be assessed. Figure 6 illustrates a multiple stage trusted boot:

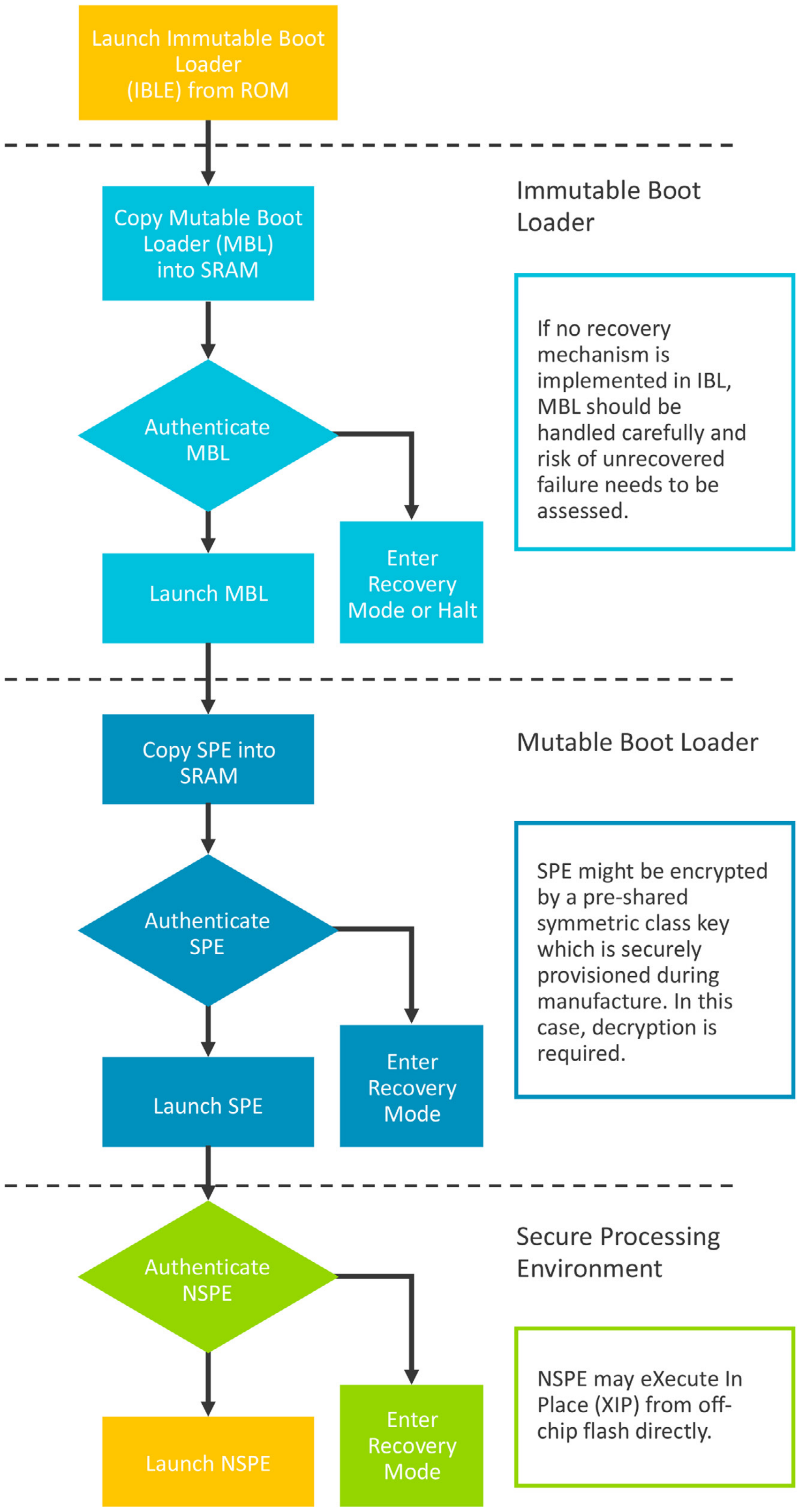

Launch Immutable Boot Loader (IBLE) from ROM
Copy Mutable Boot Loader (MBL) into SRAM
Authenticate MBL
Launch MBL
Enter Recovery Mode or Halt
Immutable Boot Loader
If no recovery mechanism is implemented in IBL, MBL should be handled carefully and risk of unrecovered failure needs to be assessed.
Copy SPE into SRAM
Authenticate SPE
Launch SPE
Enter Recovery Mode
Mutable Boot Loader
SPE might be encrypted by a pre-shared symmetric class key which is securely provisioned during manufacture. In this case, decryption is required.
Authenticate NSPE
Launch NSPE
Enter Recovery Mode
Secure Processing Environment
NSPE may eXecute In Place (XIP) from off-chip flash directly.

Figure 6:
The two phases of trusted boot

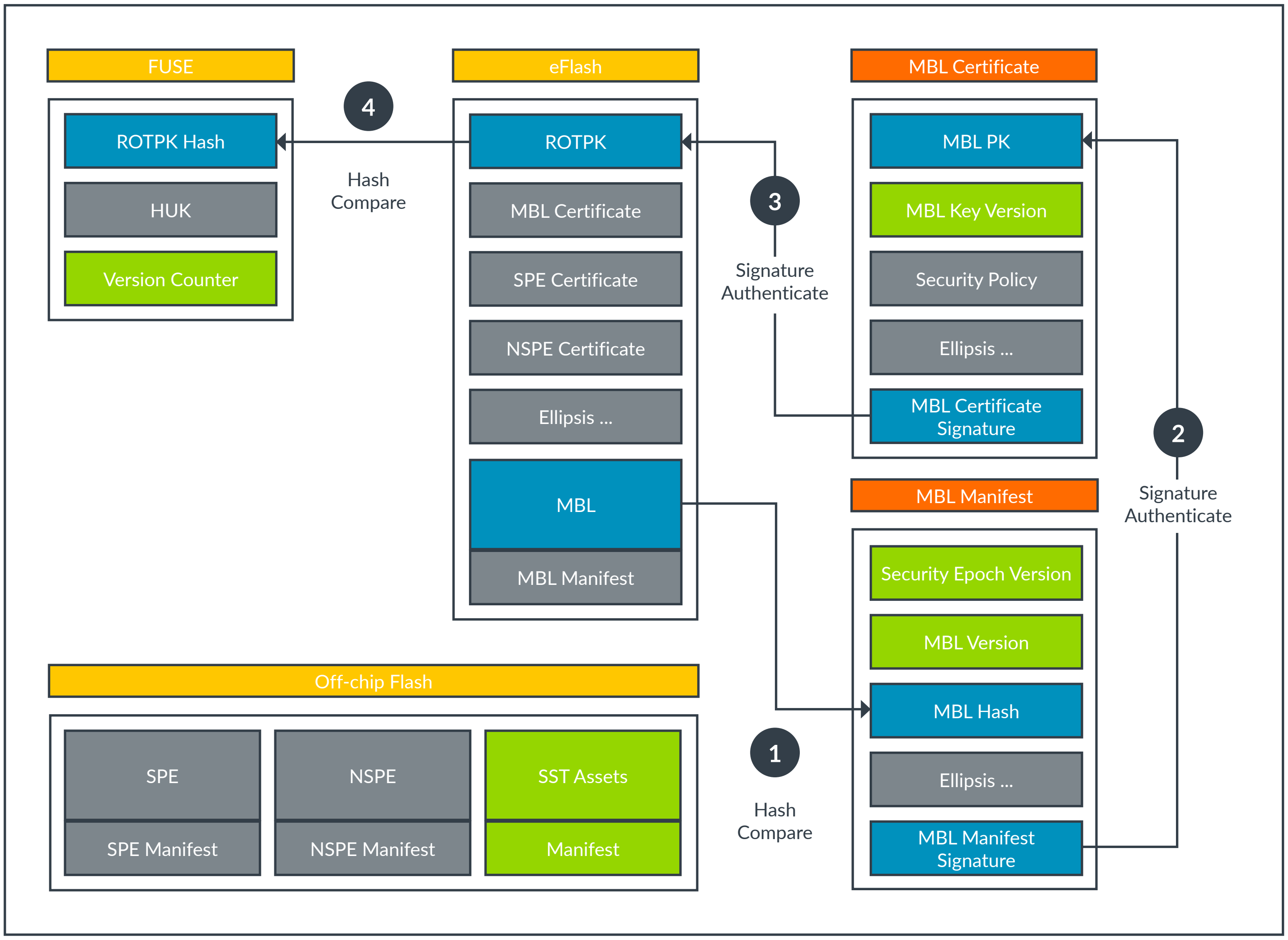


Figure 7: Signature authentication chain

The Mutable Boot Loader (MBL), SPE code, and NSPE code are authenticated in a similar way. The following diagram further shows signatures and certificates of MBL, SPE and NSPE with some high-level data structure that is defined in the TBFU specification.

Figure 7 illustrates one possible signing schema with the Mutable Boot Loader as an example. In this design:

- The Mutable Boot Loader (MBL) hash is calculated in **step 1**, then the MBL hash is compared with the hash embedded in its manifest.
- The manifest is signed by the MBL private key offline and it can be validated in **step 2**, by the corresponding MBL Public Key contained in the MBL certificate provisioned on the device.
- The MBL certificate is upgradable and needs to be further validated by the ROTPK from the key pair used to sign the MBL certificate. This validation is done in **step 3**.
- The ROTPK is not stored in the OTP to reduce the cost. In **step 4**, the hash of the ROTPK is calculated and compared with the value stored in OTP for tamper detection.

In addition, the green blocks in Figure 7 show blocks that contain versioning information for further version control during validation. The firmware anti-rollback and Secure Storage legacy data invalidation can be easily implemented by adding proper version comparation.

3.2.3 Secure Boot - Where To Start

At the time of writing, there is no off-the-shelf open source secure boot solution that is fully PSA TBFU-compliant. For more details on designing TBFU from the very beginning, please refer to MCUboot [6] which is currently maintained by JUUL Labs.

MCUboot has been integrated into the TF-M for secure boot authentication. In the TF-M, SHA-256 and RSA-2048 are used to validate the signed image containing both the SPE and NSPE code. The swapping operation of the firmware upgrade is configurable by using the MCUBOOT_NO_SWAP compile time switch.

To launch the TF-M code with MCUboot, AN521 and AN519 are currently supported platforms on the Fixed Virtual Platform (FVP) and MPS3 or MPS2 FPGA prototyping boards. AN521 and AN519 respectively are used for Cortex-M33 and Cortex-M23 simulation.

For Armv7-M based dual-core systems, launching TF-M code with MCUboot is also supported with vendor ported versions of the library.

3.3 Firmware architecture

3.3.1 PSA-FF isolation level

One core security aspect in the PSA is isolation on different levels, including hardware, software, and different trust levels. Isolation and segmentation of the critical software modules help to ensure that a compromise of one model is not directly exploitable on other models.

Increased isolation improves the security and robustness of the system by reducing its vulnerability to software defects. However, this increased isolation comes at the expense of additional hardware, memory, performance or energy. To support implementations that provide different security, performance and cost trade-offs, the PSA Security Model (SM) specifies three levels of isolation:

- SPE isolation (Level 1).
- PSA Root of Trust isolation (Level 2).
- Maximum firmware isolation (Level 3).

Figure 8 illustrates the high-level architecture of a PSA Firmware Framework. In this figure, Secure Partition represents the minimal isolation unit, from a PSA isolation perspective.

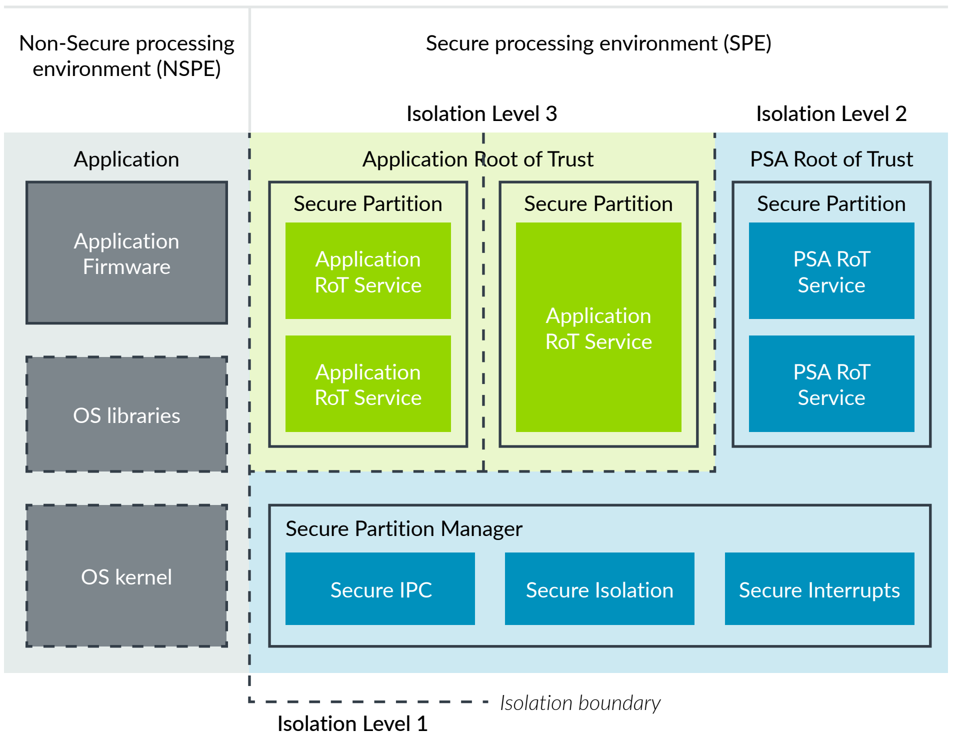


Figure 8:
Advanced isolation

3.3.2 PSA-FF interface

The SPE RoT Services are accessed from other partitions via the PSA Secure IPC framework that is implemented in the Secure Partition Manager (SPM). The IPC framework is a connection-based client/server model that provides Remote Procedure Call (RPC) behavior for calling clients.

As shown in Figure 9, clients can be either in the NSPE or a Secure Partition in the SPE, and a server implements a RoT Service within a Secure Partition.

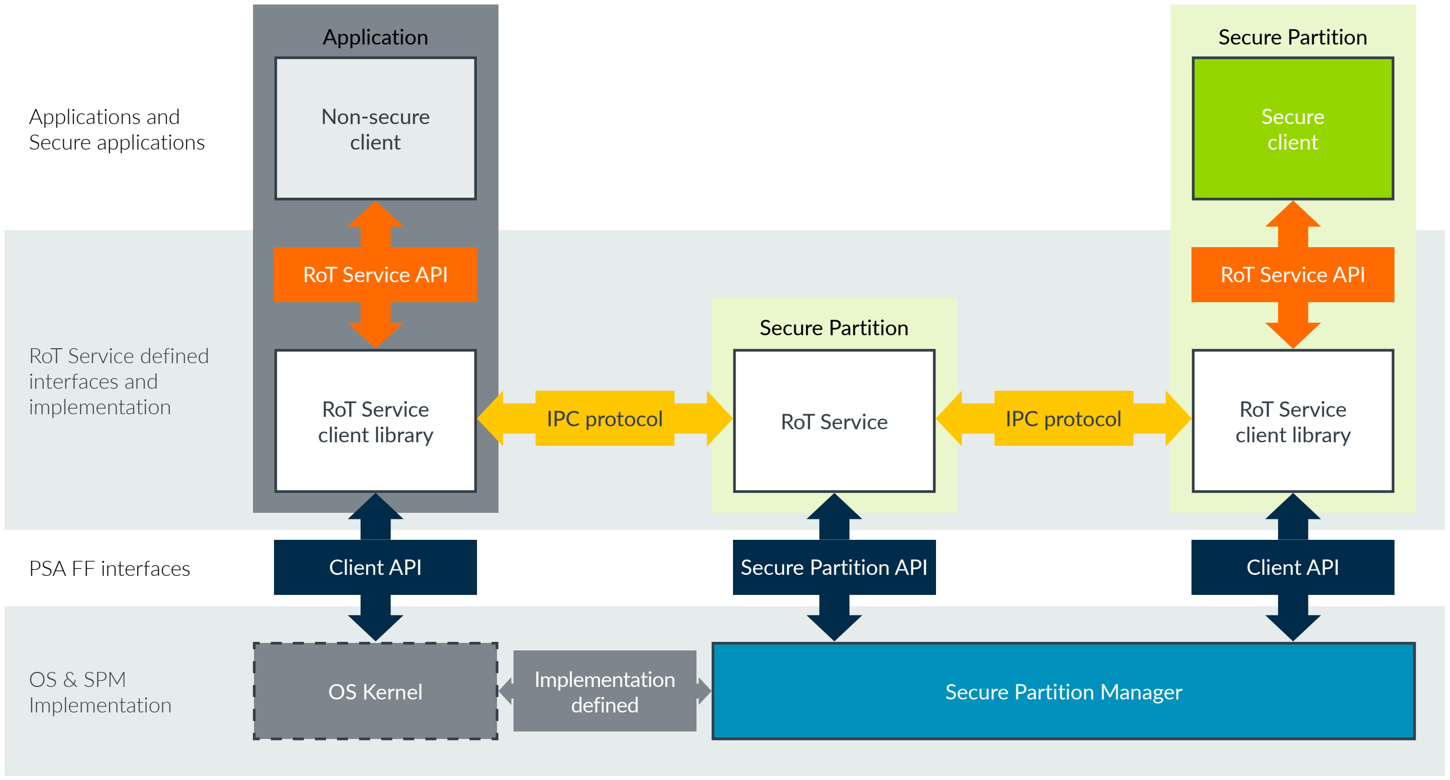


Figure 9: Example RoT Service architecture

To mask underlying hardware differences, APIs are required to provide a consistent developer experience across different chips and platforms. Arm has created two sets of APIs that are aimed at different developer communities. Together, these sets of APIs enable efficient development of software, security functions, and hardware.

- **PSA Developer APIs** are the top-level APIs that are used by application developers and RTOS vendors. These APIs have been designed to be used by software developers who wish to use the hardware security features, but who are not necessarily security experts themselves.
- **PSA Firmware Framework APIs** are designed for developers of secure functions, also known as Application Root of Trust Service (ARoT). Security experts who wish to add their own security functionality can develop an ARoT service that can be used on different chips which use these standard APIs.

3.3.3 Smart door lock firmware

A smart door lock generally includes the following security features:

- Authentication methods – including the mechanical key, PIN codes, security token, and biometrics, such as fingerprint, hand geometry, eye scan or voice ID.
- Secure communication.
- Audit log of security events.
- Trusted boot and secure software update.
- Secure storage to protect the private data of the end-user.
- Secure debug feature.
- Protection from physical tampering and simple SCA.

A typical smart door lock software stack may contain the following functionality modules:

- Normal function – User interface (UI), power management, wireless connectivity, and sensors, etc.
- Security function – Data binding, audit logs, secure storage, secure firmware update, cylinder motor subsystem, secure debug, etc.
- Drivers – LCD, keypad, buzzer, motor, sensors, Bluetooth Low Energy, Cellular, UART, I2C, SPI, LED, etc.
- OS – RTOS in the NSPE.

3.3.3.1 Firmware design considerations

In addition to the isolation described in section 3.3.1, another security principle is one of least privilege, that requires that any software component can access only the necessary information and resources.

Isolation and SPE minimization

To use the SPE correctly, there are two things to consider:

- The APIs between the NSPE and the SPE should be minimized and well-structured.
- The codes in the SPE environment should be robust and minimized.

The more APIs that are exposed to the NSPE from the SPE, the more attack possibilities are created. The more code that is in the SPE, the more potential bugs and vulnerabilities are created.

For example, the complete RTOS or huge driver stack must not be placed into the SPE directly. Instead, weighting, assessing and re-architecting of the SPE code is needed in order to isolate sensitive activities and assets.

External inputs handling

External unauthenticated input is inherently mistrusted. This type of external input needs to be handled carefully as it could be used to exploit vulnerabilities. The basic rule is: code which either contains privileged operations or has access rights to confidential assets always avoids processing such risky, unauthenticated input. For example, if a process is responsible for system configuration and the granting of some privileges, then the parse of external input data will not be done within this process.

The best practice for fulfilling these security principles is to build a system from the beginning, adding only the required functionality to each Secure Partition. By removing unnecessary functionality and having less exposure to code, there are fewer security risks and potential failures.

As a summary, these common rules should be adopted to reduce the potential attack surfaces:

- Implement only necessary features to minimize the system exposure (i.e. minimize library function for the application) and do not leave in code that is not used.
- Keep the SPE code as simple as possible.
- Give the SPE APIs as little exposure as possible to the NSPE clients.
- Grant least privilege and the accessible resources to each Secure Partition.

3.3.3.2 Smart door lock assets partitioning

To apply the security principles mentioned in section 3.3.3.1 into a smart door lock design, and to separate the functionalities into the SPE and the NSPE, in this section we will focus on the code and data assets that are sensitive and critical to both the device and system security. Here are a few typical examples of assets on a smart door lock:

1. **Confidential code:** A biometric authentication code, such as a fingerprint authentication algorithm, is a common and core smart door lock feature. The code of the authentication algorithm is highly sensitive and should be strongly protected against reverse engineering. This code usually includes converting a fingerprint image into a mathematical representation which is then followed by verification of the fingerprint image against the template created during setup.

***Note**: The robustness of the authentication algorithm itself is beyond the scope of this document. Multifactor authentication, i.e. PIN code plus fingerprint authentication, can be adopted to improve the overall security level. However, this level of authentication is a design decision and a trade-off between usability and security.*

2. **Critical process:** Controlling the security-related functions such as fingerprint enrollment and authentication, lock and unlock door commands handling, and cylinder motor control is critical to guarantee the security of the smart door lock. The unencrypted data and processing of the data are fully controlled in a secure execution environment that is implemented via the ARoT services.
3. **Secret data:** User credentials or other security-related data such as the enrolled PIN code and the standard fingerprint template are critical to authentication and need to be protected against tampering. The fingerprint data or PIN input also requires protection during transit and processing.
4. **Secure peripheral:** The fingerprint scanner and possible pulse and heat sensors for live finger detection are set as a trusted peripheral to be accessible from the SPE only. If all the fingerprint processing activities are implemented in the SPE as expected, there is no need to share this secure peripheral with the NSPE.
5. **Shared peripheral:** The PIN code input via the touch panel also acts as a user interface to display messages from both the NSPE and SPE. In such cases, the PIN code input should be configured as a shared resource. However, the peripheral switches to the SPE control mode when the PIN code is inputted to reduce the risk of tampering or spying by a potential malicious code that is injected into the NSPE by an attacker.
6. **NSPE code:** Drivers such as network stack/protocols, Bluetooth, and Bluetooth Low Energy are usually huge and complex. The size and complexity of the drivers makes them prone to fatal bugs and exposes them to many attack vectors. Furthermore, these standard drivers are shared by many OEMs and there is no confidentiality, thus these drivers should be placed into the NSPE to simplify the SPE and reduce risks.

3.3.3.3 Smart door lock software architecture

Applying the design considerations mentioned in the previous section, 3.3.3.2, the major part of the firmware could be architected as in the following figure.

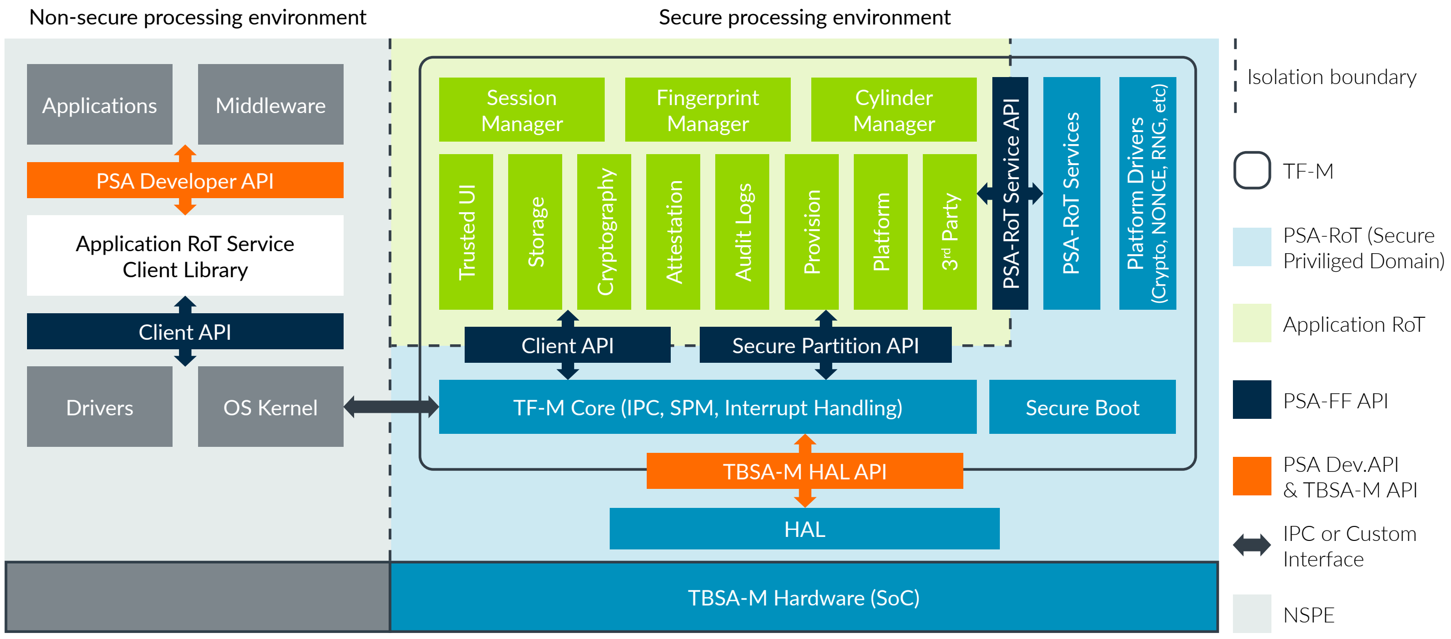


Figure 10: Smart door lock software architecture

Figure 10 shows that this firmware architecture is designed by leveraging the TF-M framework. It takes advantage of existing ARoT services, i.e. Storage, Cryptography, Attestation, and Audit Logs. Several ARoT services for the smart door lock are illustrated in Figure 10 and are described the next sections, for example, the Session Manager, Fingerprint Manager and the Cylinder Manager.

3.3.4 Fingerprint enrollment flow

The following Figure 11 illustrates the fingerprint enrollment procedure both in terms of how the NSPE and the SPE communicate and how the ARoT services interact:

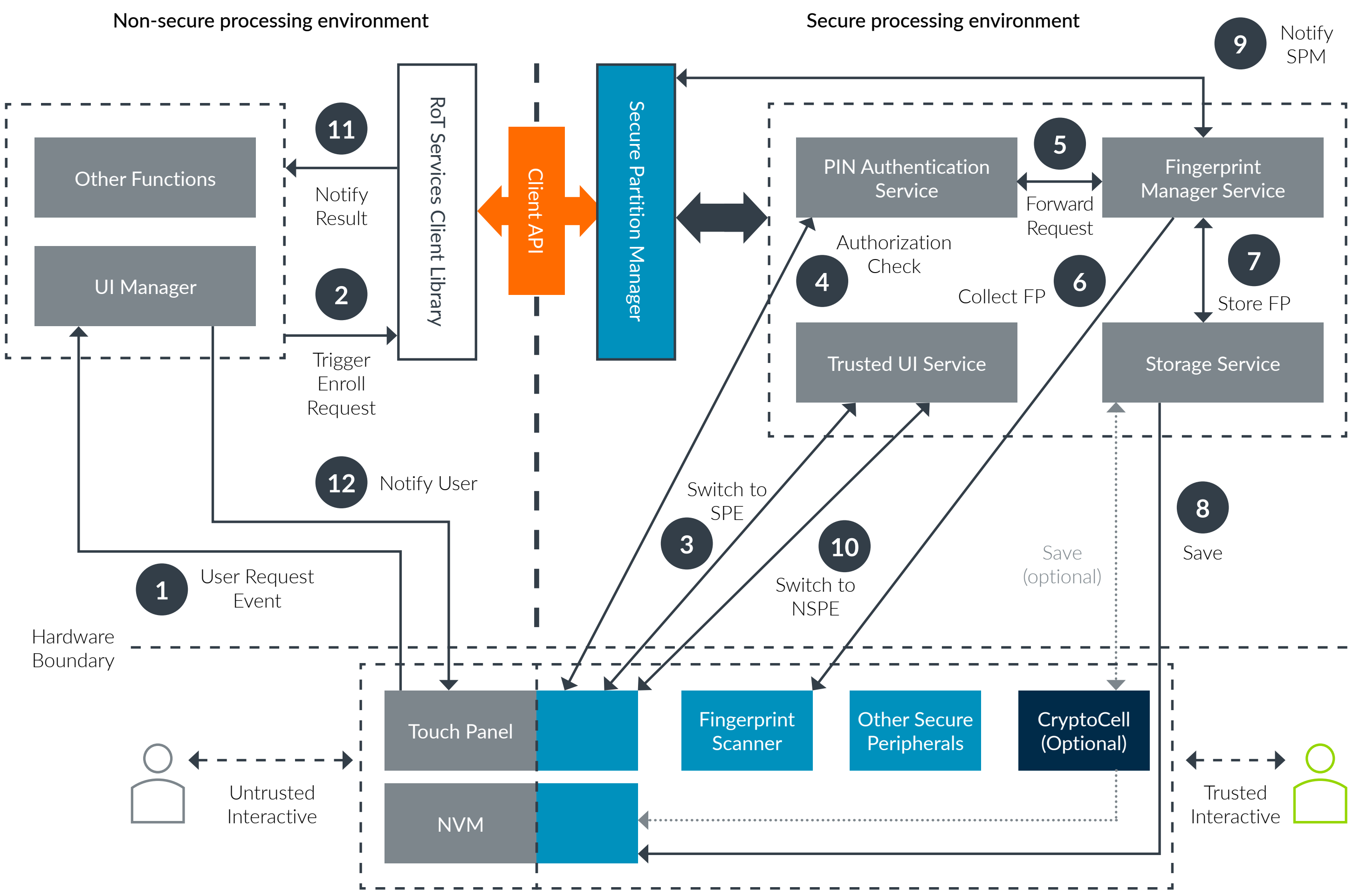


Figure 11: Fingerprint enrollment procedure

The following steps correspond to the procedure numbers marked in above Figure 11:

1. The smart door lock owner raises an enrollment request through the User Interface (UI) Manager in the NSPE.
2. The UI Manager redirects the request to the SPE, via the RoT Services Client Library and through the Client API (PSA-IPC interfaces) to the Secure Partition Manager (SPM). The RoT Services Client Library may serve multiple function modules inside the dotted rectangle.
3. The SPM dispatches this request to the Trusted UI Service in a Secure Partition. The Trusted UI Service then takes control of the Touch Panel to guarantee that there is no spy or tamper from the NSPE when the Touch Panel is being used.

4. Once the Touch Panel is under the SPE control, the user can input the owner PIN code for the authority check. The check is done by the PIN Authentication Service.
5. The PIN Authentication Service forwards the fingerprint enrollment request to another security partition, the Fingerprint (FP) Manager Service.
6. The FP Manager Service controls the fingerprint scanner which is fully under the SPE control as a trusted peripheral to collect the user's fingerprint.
7. The FP Manager Service processes the scanned image, converts it into target template format and sends the data to the Secure Storage Service.
8. The template data is stored into the secure storage, for example as an encrypted data partition on internal or external flash, or other persistent storage. Integrating CryptoCell-312 could be an enhancement option -as shown in gray dotted path - for the cryptography key management and acceleration, or even if adopting CryptoCell-312P to leverage the advanced tamper prevention feature. Adding CryptoCell can reduce the key exposure at both the NSPE and SPE for many different use cases.
9. The FP Manager Service sends the processing result back as a notification to the SPM.
10. The SPM first sends the command to the Trusted UI Service, to release the Touch Panel and gives the control back to the NSPE. The NSPE then displays the generic system notification, which is considered as non-sensitive in this design example.
11. The SPM then sends the notification to the NSPE UI Manager, via the RoT Services Client Library
12. The UI Manager creates a pop-up notification message on the Touch Panel, and the fingerprint enrollment procedure is complete.

3.4 Other PSA specifications

This section gives a brief introduction to the PSA specifications, which are beyond the scope of this application guide. An understanding of the PSA security rationale is useful for IoT system architects.

3.4.1 Security Model

The PSA Security Model (SM) defines the overall security architecture for designing and deploying trusted PSA-compliant devices within ecosystems. It is the top-level document for all the other PSA specifications and defines common language, high-level robustness rules, and models for the robustness rules.

3.4.2 Secure Production

There is an informative document, not yet published, that identifies and discusses the general need for infrastructure and common frameworks to facilitate these factory processes, as well as their dependencies on the Root of Trust that are established in the device security architecture.

Deployment of the actual factory provisioning and device management infrastructure should be done by industry stakeholders themselves or using services such as the Arm Pelion Device Management (PDM) platform, which allows device manufacturers to configure millions of devices with both unique cryptography identities and the PDM connection parameters, before the devices leave the factory.

4. Implement - PSA on Armv8-M

4.1 TF-M overview

An important part of the PSA is open-source firmware. There is a reference PSA firmware implementation, available from Open Governance trustedfirmware.org [7], hosted by Linaro Community Division, in the form of Trusted Firmware for Cortex-M processors (TF-M).

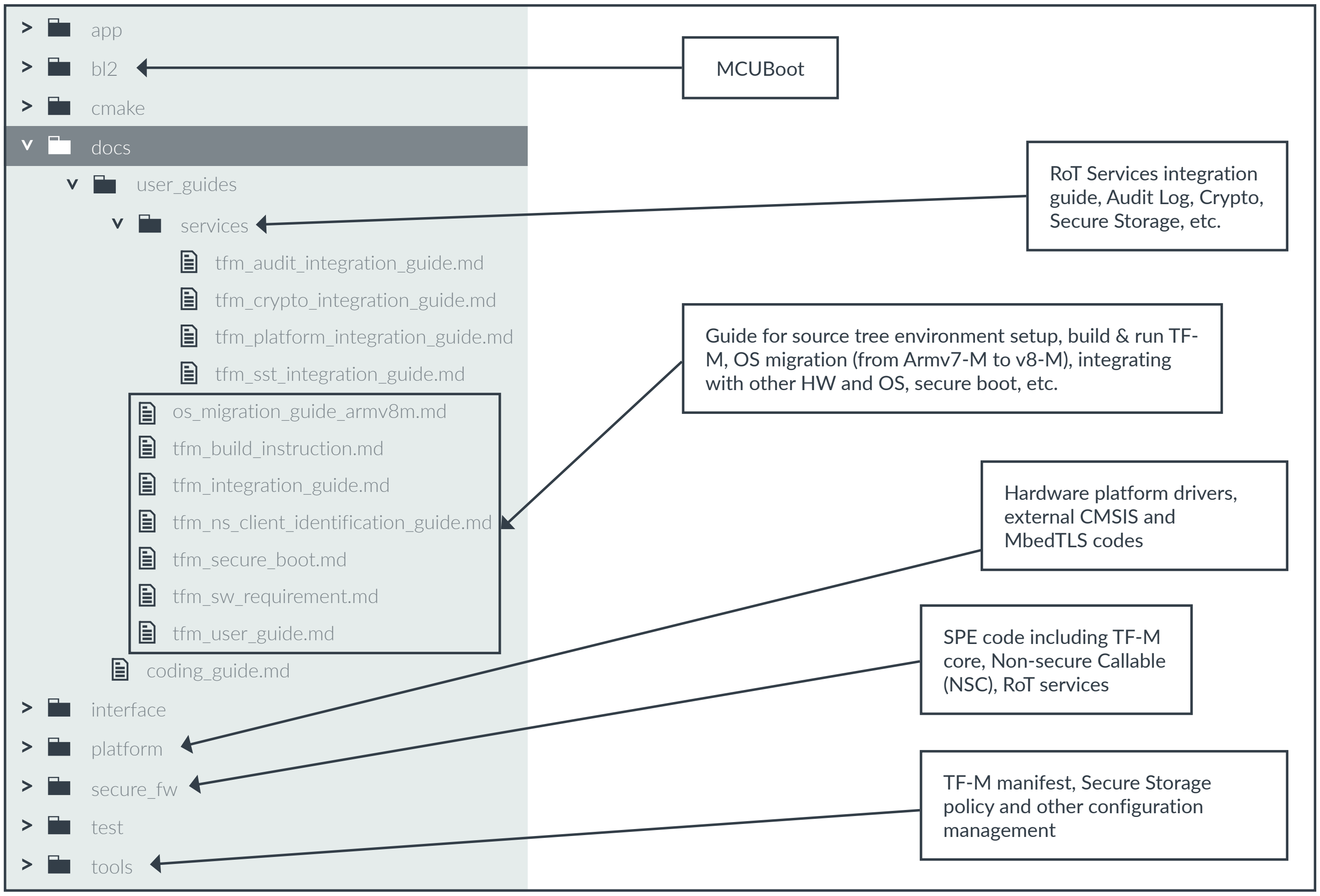
TF-M leverages experience from TF-A, the successful open-source project for the Arm Cortex-A platform. TF-M runs in the SPE and contains the Secure Partition Manager and RoT Services. TF-M includes the following security foundation features:

- Isolated SPE and NSPE execution environments.
- Trusted device initialization and trusted boot.
- Secure services invoked from NSPE applications.

TF-M is designed to be highly modular, so that by disabling features, it can be reduced in size to fit to highly constrained devices, at the same time as still providing basic secure computing support. The TF-M infrastructure allows any other secure service to be added easily.

4.1.1 TF-M codebase

TF-M is available and cloneable from https://git.trustedfirmware.org/trusted-firmware-m, the source tree was structured in the same way as in Figure 12 when this document was written. All the guidance documents are in the /docs sub folder.



app
bl2
cmake
docs
user_guides
services
tfm_audit_integration_guide.md
tfm_crypto_integration_guide.md
tfm_platform_integration_guide.md
tfm_sst_integration_guide.md
os_migration_guide_armv8m.md
tfm_build_instruction.md
tfm_integration_guide.md
tfm_ns_client_identification_guide.md
tfm_secure_boot.md
tfm_sw_requirement.md
tfm_user_guide.md
coding_guide.md
interface
platform
secure_fw
test
tools
MCUBoot
RoT Services integration guide, Audit Log, Crypto, Secure Storage, etc.
Guide for source tree environment setup, build & run TF-M, OS migration (from Armv7-M to v8-M), integrating with other HW and OS, secure boot, etc.
Hardware platform drivers, external CMSIS and MbedTLS codes
SPE code including TF-M core, Non-secure Callable (NSC), RoT services
TF-M manifest, Secure Storage policy and other configuration management

Figure 12:
TF-M source tree

As already mentioned, the tfm_user_guide.md and the tfm_integration_guide.md are the entry points of these documents for compiling and running TF-M on Fixed Virtual Platform models, or for further integration of the TF-M with other target hardware platforms or RTOS.

4.1.2 TF-M boot flow

The secure bootloader, MCUboot, starts the system and launches the SPE. The SPE then initializes the TF-M core and Secure Partitions. The TF-M core is in charge of the SPE and all the Secure Partitions.

The SPE finally jumps to the NSPE main() to launch the demo application or regression tests, depending on different build options.

4.1.3 Available application RoT services

The built-in ARoT services APIs contain the following three modules at the time of writing:

1. **Crypto Service**

The TF-M Crypto Service allows the application to use cryptography primitives such as symmetric and asymmetric ciphers, hash, Message Authentication Codes (MACs) and Authenticated Encryption with Associated Data (AEAD). Additionally, there are sets of APIs for key policy and key management, and random number generation etc.

2. **Secure Storage Service**

The Secure Storage Service is a fundamental feature for asset protection. The TF-M Secure Storage (SST) Service allows the storage of various types of data which have security implications. It is meant to store the platform credentials (keys, certificates, and hashes, etc.) that require strict access controls.

3. **Audit Logging Service**

The TF-M Audit Logging Service allows secure services in the system to log critical system events and information that have security implications. This is required to analyze the system behavior, system events and triage system issues offline. This Audit Logging Service offers a mitigation against the repudiation threat.

Attestation is also an important ARoT service. It covers the generation of a report by a device or subsystem that can be verified by third-party software. Attestation forms part of building a trust model for including a device in a secure system.

The Entity Attestation Token (EAT) PSA-RoT Service is currently being developed by TF-M. The EAT acts as a report card for IoT devices and contains claims that can be verified cryptographically. The EAT is being adopted by the PSA, GlobalPlatform (GP) and standardized in the IETF and GP. In the PSA, EAT can be used to bind any application-specific attestation protocol that is implemented in the corresponding ARoT service.

In addition to these ARoT services, a Provisioning Service and other ARoT services will be available soon.

4.1.4 TF-M integration

To work with the current TF-M, a target OS needs to support the Armv8-M architecture. Depending upon the system configuration, this might require configuring drivers to use appropriate address ranges.

Please refer to the tfm_integration_guide.md for more details on how to integrate other hardware platform or RTOS. For example, the following hardware platforms currently support TF-M directly:

- AN521: Soft Macro Model (SMM) Cortex-M33 SSE-200 subsystem for MPS2+.
- AN519: Cortex-M23 IoT Kit subsystem for MPS2+.
- Musca-A1 (Cortex-M33 SSE-200 subsystem) and Musca-B1 test chip board.

4.2 Add an Application RoT service

This section describes how to add an Application RoT service into the SPE. For example, the Cylinder Manager Service for locking and unlocking the smart door lock cylinder. The lock or unlock commands come via different channels, for example, fingerprint detection, PIN password, or smartphone commands.

4.2.1 Secure Partition manifest

Each RoT service belongs to one Secure Partition, which is the minimal unit that the SPE manages from a security perspective. To assemble and allocate resources within the SPE, each Secure Partition must have resource requirements declared in a manifest file.

A manifest file defines the values and identifiers that are used by the Secure Partition source code, as well as the access control between Secure Partitions. Access control between Secure Partitions is done by listing the dependencies to other RoT services (and referenced by the Service ID). A manifest file may contain Secure Partition ID, entry point symbol, minimum stack and heap size, reserved IRQ lines, etc. Please refer to the TF-M document and source code for a full value list.

Manifest files are analyzed at the compile time by the SPE build tools to validate dependencies and produce header files that satisfy the required isolation level.

4.2.2 ARoT folder structure

All application RoT services are located inside the /secure_fw/services folder. As a result, a new folder (i.e. cylinder_manager) and its manifest should be created accordingly for this Cylinder Manager Service. There is a manifest YAML file and python script that facilitate the addition or modification of header files. Please refer to the user guide documents and the existing ARoT services for more detail.

4.2.3 Smart door lock unlock operation with TF-M APIs

Figure 13 shows an example procedure for processing the unlock command issued from the owner of the smart door lock via their smartphone app.

The requirements are that the person's smartphone has already been paired with the smart door lock and that the smartphone app has been provisioned securely by the cloud server to have a pre-shared key for payload encryption during communication. The Bluetooth Low Energy driver is placed in the NSPE to simplify the SPE code and reduce the attack surfaces. The communication data processing which involves cryptography operations and the cylinder operation are done in the SPE for security reasons.

As illustrated in Figure 13, the NSPE code mainly acts as a router to unpack and repack the communication data. The NSPE code connects the app on the smartphone to the SPE. The NSPE code also controls some UI-related operations to update the smart door lock status and notify the owner or guest. There are three steps involved in unlocking the door:

1. Set up a secure session between the smartphone app and the smart door lock. The blue arrows in the diagram represent the tasks for this step.

2. Using the secure session, the smartphone app issues an unlock command to unlock the door. The smart door lock unlocks the cylinder and reports the status back to the smartphone app. The yellow arrows in the diagram represent the tasks for this step.

3. Close the secure session. The orange arrows in the diagram represent the tasks for this step.

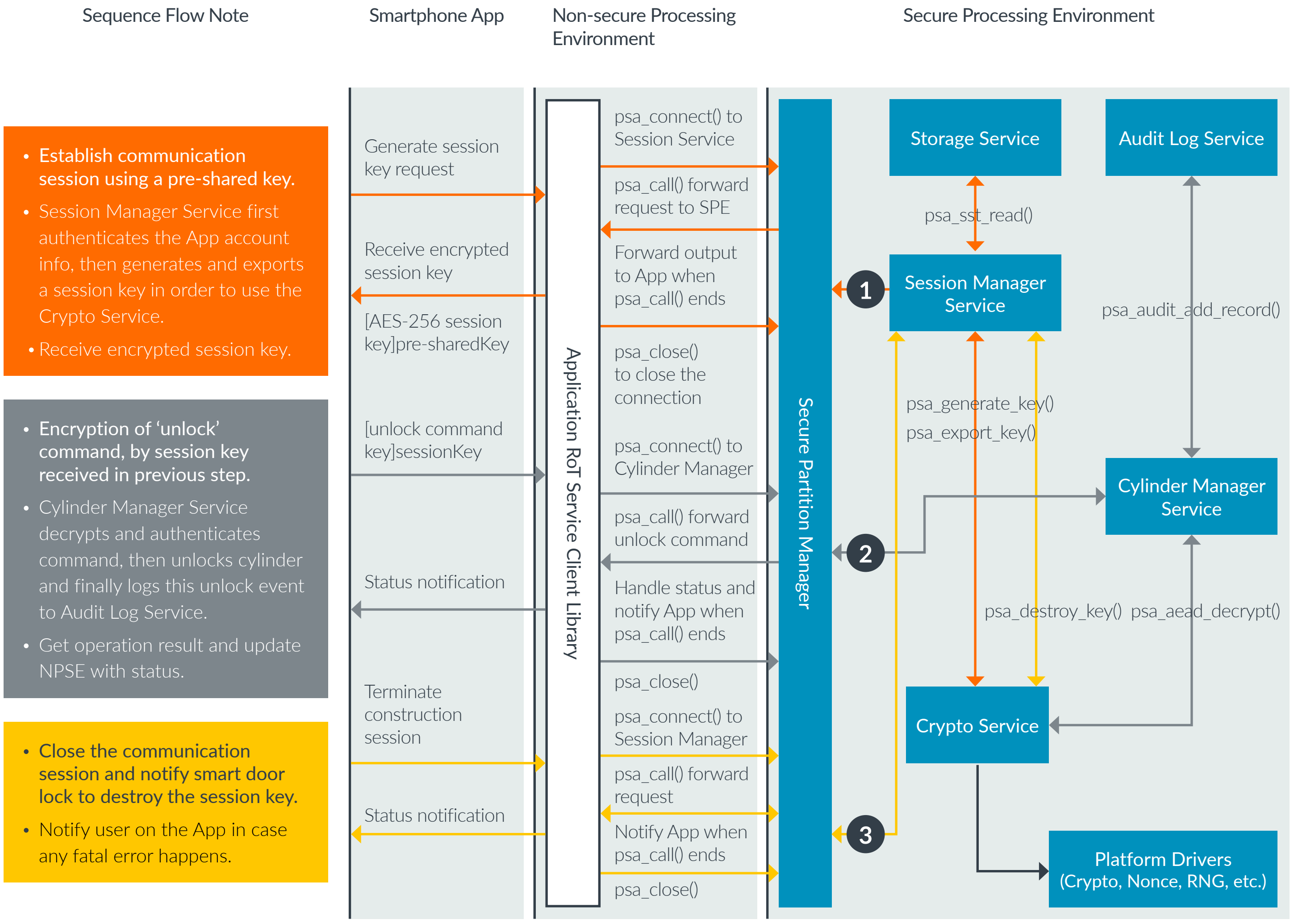

Sequence Flow Note
Smartphone App
Non-secure Processing Environment
Secure Processing Environment
Establish communication session using a pre-shared key.
Session Manager Service first authenticates the App account info, then generates and exports a session key in order to use the Crypto Service.
Receive encrypted session key.
Encryption of 'unlock' command, by session key received in previous step.
Cylinder Manager Service decrypts and authenticates command, then unlocks cylinder and finally logs this unlock event to Audit Log Service.
Get operation result and update NPSE with status.
Close the communication session and notify smart door lock to destroy the session key.
Notify user on the App in case any fatal error happens.
Generate session key request
Receive encrypted session key
[AES-256 session key]pre-sharedKey
[unlock command key]sessionKey
Status notification
Terminate construction session
Status notification
Application RoT Service Client Library
psa_connect() to Session Service
psa_call() forward request to SPE
Forward output to App when psa_call() ends
psa_close() to close the connection
psa_connect() to Cylinder Manager
psa_call() forward unlock command
Handle status and notify App when psa_call() ends
psa_close()
psa_connect() to Session Manager
psa_call() forward request
Notify App when psa_call() ends
psa_close()
Secure Partition Manager
Storage Service
Audit Log Service
psa_sst_read()
1
Session Manager Service
psa_audit_add_record()
psa_generate_key()
psa_export_key()
Cylinder Manager Service
2
psa_destroy_key()
psa_aead_decrypt()
Crypto Service
3
Platform Drivers (Crypto, Nonce, RNG, etc.)

Figure 13: Smart door lock unlock operation with TF-M APIs

Please be aware that the sequence flow of the ARoT Service APIs usage in Figure 13 is conceptual and, for simplification, does not cover all the details of cryptography keys usage and management.

4.4 TF-M evaluation

Either the Musca development boards, or the Corstone FVP simulation platforms, or the pb, are available for building and running the TF-M sample code. The Musca development board is targeted for use as a secure foundation platform for software development and to prototype TF-M code. You can register and apply to request your loaner board from the Arm Developer website.

5. Implement - PSA on Armv7-M

5.1 TF-M Support for Armv7-M

TF-M is also supported for Armv7-M architectures where dual core systems and can provide an isolation mechanism between Secure and Non-secure worlds. TF-M for Armv7-M is supported by the Cypress PSoC64 line of Secure MCUs and this can be used as the platform to implement the smart door lock defined in this paper.

TF-M runs in the SPE (Cortex M0+) and includes the following security foundation features:

- Isolated SPE and NSPE execution environments.
- Secure Boot and Trusted device initialization
- Secure services invoked from the NSPE applications

5.1.1 TF-M Armv7-M codebase

The code base used for PSoC64 PSA targets are cloneable from https://git.trustedfirmware.org/trusted-firmware-m.git under the branch 'feature-twincpu'. The source tree will be identical to the one shown in Figure 12.

5.1.2 TF-M boot flow

TF-M for PSoC64 includes the Cypress Secure Bootloader, which is a PSoC64 optimized port of the MCUboot, launches the SPE environment in the Secure Core (M0+). The SPE then initializes Secure Partitions. The M0+ core is in charge of the SPE and all the Secure Partitions.

The SPE finally launches the Application core (M4) and jumps to the NSPE main() to launch the demo application or regression tests, depending on different build options.

Each one of these boot images are checked both for validity and authenticity by checking signatures using public keys. This check is done in a sequential manner i.e., one stage verifies and launches the next stage to get secure chain images.

5.1.3 Available application RoT services

The built-in ARoT services APIs contain the following three modules at the time of writing:

1. Crypto Service

The RoT Crypto service provides Arm Mbed, PSA-compliant, APIs which allow the application to use cryptography primitives such as symmetric and asymmetric ciphers, hashes, Message Authentication Codes (MACs) and Authenticated Encryption with Associated Data (AEAD). Additionally, there are sets of APIs for key policy and key management, random number generation etc.

2. Secure Storage Service

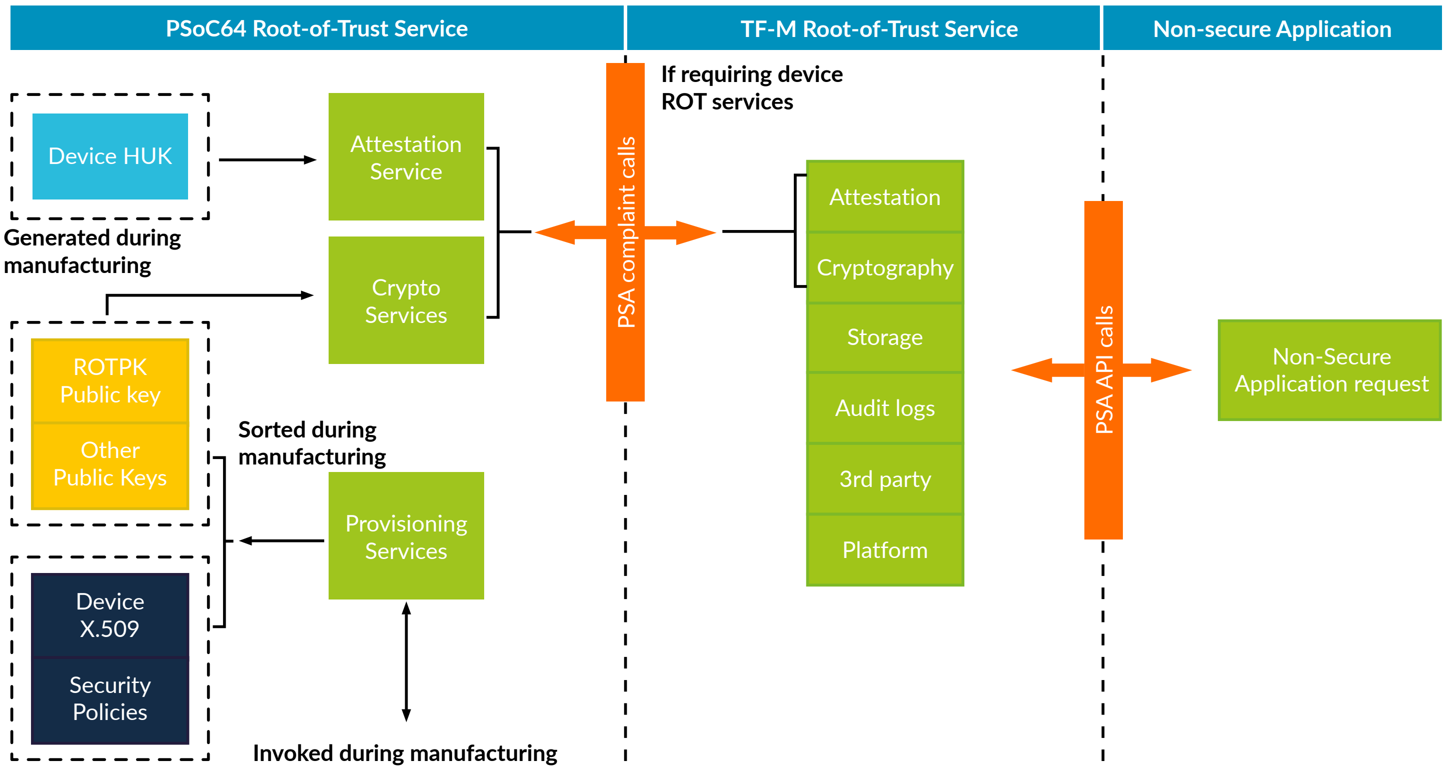
The Secure Storage Service is a fundamental feature for asset protection. The TF-M has internal Trusted storage which allows the storage of various types of data which have security implications. It is meant to store the platform credentials (keys, certificates, and hashes, etc.) that require strict access controls.

3. Audit Logging Service

The TF-M Audit Logging Service allows secure services in the system to log critical system events and information that have security implications. This is required to analyze the system behavior, system events and triage system issues offline. This Audit Logging Service offers a mitigation against the repudiation threat.

5.1.4 Other PSoC64 RoT services

The PSoC64 has also has built-in, PSA-compliant services which immutably reside in protection context than Application RoT SPE calls. These services provide Crypto and secure storage services, similar to the description in 5.1.3 and also provide the below additional services.



1. Attestation

Attestation is an important ARoT service. It covers the generation of a report by a device or subsystem that can be verified by third-party software. Attestation forms part of building a trust model for including a device in a secure system.

Attestation is performed using the immutable Root of Trust service provided by the PSoC64. The Attestation RoT service can be used to check the validity of any arbitrary memory region in the chip by hashing the region, adding a random number to avoid replay, and signing it with the unique device private key, ensuring authenticity of the return. The service requestor will be able to compare this against the expected value to ensure that the chip has not been compromised.

2. Provisioning

The PSoC64 line of Secure MCU's provides several provisioning services outside of the TF-M scope which can be used to securely inject user credentials and transfer the Root of Trust during product manufacturing. This is beyond the scope of this application note, but on a high level the series of steps are;

1) Securely transfer the ROTPK into the PSoC64 device using cryptographically signed tokens.
2) Securely transfer user assets like public keys, certificates and security policies to the device and protect them.

5.2 Add an application RoT service

This section describes how to add an Application RoT service into the SPE. For example, the Cylinder Manager Service for locking and unlocking the smart door lock cylinder. The lock or unlock commands come via different channels, for example, fingerprint detection, PIN password, or smartphone commands.

5.2.1 Secure Partition manifest

Each RoT service belongs to one Secure Partition, which is the minimal unit that the SPE manages from a security perspective. To assemble and allocate resources within the SPE, each Secure Partition must have resource requirements declared in a manifest file.

A manifest file defines the values and identifiers that are used by the Secure Partition source code, as well as the access control between Secure Partitions. Access control between Secure Partitions is done by listing the dependencies to other RoT services and referenced by the Service ID. A manifest file may contain Secure Partition ID, Entry point symbol, Minimum stack and heap size, Reserved IRQ lines, etc. Please refer to the TF-M Secure Partition Manager (SPM) documentation for a full value list.

Manifest files are analyzed, at the compile time, by the SPE build tools to validate dependencies and produce header files that satisfy the required isolation level.

5.2.3 Smart door lock - provisioning services

The addition of the provisioning service adds the ability to form a unique, trusted device identity which can be used as a trust anchor. Provisioning also ties into the trustworthiness of attestation as the verification of it is done by a unique, generated device key in the part.

Using provisioning services, the smart door lock designer will be able to uniquely form an Instance Unique hardware key (HUK) for every device in the secure manufacturing environment. This is a one-time operation and can never be re-used as the information exists in OTP non-volatile storage.

As a part of this step, the device also generates a unique device root key and exports the associated public key. This device root public key can be attested by the trusted manufacturing by forming a unique X.509 certificate to be placed into the device, as well as sent to the smart door lock designer.

After deployment, if the server chooses to make an attestation call service on a device, the device will send a signed digest using the device root private key. The validity of the returned packet can be verified using the unique device public key in X.509 ertificate; protecting against the use case where malicious firmware tried to spoof an attestation return.

5.2.2 Smart door lock unlock operation with TF-M APIs

The actual PSA API calls to achieve the functionality is identical to Section 4.2.3, Figure 13.

5.3 TF-M evaluation

Evaluation of the Armv7-M PSA architecture and sample code can be done on the CY8CPROTO-064-SB prototyping kit in conjunction with Mbed OS release.

6. PSA Certification

PSA Certified™ is an independent security testing program devised by several companies. It enables IoT chipsets and devices to be tested in laboratory conditions to evaluate their level of security and help developers and customers trust that they can achieve the level of security they need. Working with leading test labs, PSA certified provides multi-level assurance for a device, depending on the security requirements established through analysis of threats for a specific use case.

There are two types of certification, functional certification and a multi-level security certification.

Functional API certification checks the implementation of PSA Developer APIs. The APIs provide top-level security functions of the PSA Root of Trust (PSA-RoT) which reduce fragmentation and development time costs.

The multi-level security certification scheme uses independent test labs to review the security requirements of the generic parts of IoT platforms and SoCs. There are three progressive levels of security certification, with increasing depth of evaluation. The tests result in a digital certificate with a unique PSA reference number, indicating which level of testing has been awarded, creating industry trust and confidence in devices.

The PSoC64 line of Secure MCU's is both PSA Certified Level 1 and PSA Functional API Certification, with PSA Certified Level 2 making good progress. PSA Certified Level 1 sets up the foundation by certifying hardware mechanisms for isolation and PSA Certified Level 2 expands the scope to the PSA-RoT and threat resistance with software and hardware attacks.

7. Conclusion

PSA is a robust system architecture covering both hardware and firmware, codifying these common security principles into a set of system requirements and interfaces. PSA itself is architecture-agnostic and it is expected that Cortex-A and other architecture platform will be covered and supported officially in the future.

Explore more details of PSA:

- pages.arm.com/PSA-Building-a-secure-IoT.html

The PSA specifications were publicly released in October 2018. To support the PSA, Arm offers IP with embedded security features, for example, Armv8-M processors including Cortex-M23, Cortex-M33 and Cortex-M35P, CrytpoCell and CryptoIsland. Arm also offers TF-M as an open-source software for a PSA-compliant implementation. Both the Musca development board and the FVP simulation platform are available free of charge, and upon request, as an execution platform. For anyone who wants to quickly access and study the PSA specifications, then download, compile and launch the TF-M on a loaner Musca board, we have the following information:

Explore more details on TF-M and Musca boards:

- git.trustedfirmware.org/trusted-firmware-m.git/about/
- www.arm.com/products/development-tools/development-boards/musca-a1-iot

Explore more details of TF-M and PSoC64 SB board:

- www.cypress.com/CY8CPROTO-064S1-SB

As a PSA use case on security aspects of the smart door lock design and implementation, this application guide covers all the three parts of the PSA:

Analyze - Threat Models and Security Analyses

Explore more details of the Asset Tracker, Smart Meter and Network Camera TMSA, by downloading the following resources:

- Asset Tracker TMSA
- Water Meter TMSA
- Network Camera TMSA

Architect - firmware architecture and hardware specifications

Explore more details of the security model, TBSA-M, TBFU and PSA-FF specifications, by entering your details to download the following resources:

- Security Model
- TBSA-M
- TBFU
- Firmware Framework

Implement – Arm hardware IP and open-source TF-M

Explore more details on Trusted Firmware-M:

- www.trustedfirmware.org/about/

Taking some hardware and software design considerations of a typical smart door lock into account, the application guide describes:

- How to do threat modelling and security analysis for a target device. The advantage of the Armv8-M architecture and some available Arm IP that supports PSA TBSA-M compliance

Explore more details of the Arm security IP from the Arm website:

- www.arm.com/products/silicon-ip-security

- The key points of TBFU, Trusted Boot flowchart and typical firmware image signature manifest
- PSA Firmware Framework and security considerations when isolating the SPE from the NSPE
- The flowchart of fingerprint enrollment process, and the PSA secure IPC and RoT Services APIs usage for unlock operation

This application guide addresses the security of a smart door lock by leveraging the PSA. The PSA provides hardware-backed, scalable security that can be applied across a variety of devices, which will allow the right level of security to be designed for all kinds of IoT devices. Similar security analysis and design approaches could be applied to many other IoT devices such as smart meters, smart speakers, and others.

8. References

[1] PSA TMSA examples

[2] Smart lock wiki

[3] Texas Instruments, Application Report, Smart Door Lock With SimpleLink™ Platform, 2018

[4] Smart Locks: Lessons for Securing Commodity Internet of Things Devices, Grant Ho, Derek Leung, Pratyush Mishra, Ashkan Hosseini, Dawn Song, David Wagner, University of California, Berkeley, 2016

[5] NIST SP 800-108

[6] MCUboot home page

[7] Trustedfirmware.org home page

9. Appendix - Threat Summary Table

Asset	Threats (STRIDE)	Attack Scenario	Impact of Vulnerability	Mitigation/ Security Requirement	Arm's Technology
Firmware	Tampering	Rogue code injection	Execute malicious code	Support secure boot flows and firmware authentication	[CI] Root of Trust [CI] Loaded SW validation [PSA] Trusted Boot features [PSA] Secure debug (SDC-600)
		Debug port abuse or physical tampering local storage	Physically install malware	Prevent voltage/current glitches and debug protection	
		Strong magnetic ring attack (simple side-channel attack)	Interfere the circuit and try to reset the smart door lock into default state (might be unlock)	Maintain secure state Anti-tampering protection at SoC level	CryptoIsland-300P, Cortex-M35P, CryptoCell-312P
	Information disclosure	Extract local memory or intercepting firmware OTA package	Reverse engineering to explore firmware vulnerability	Execute-only RAM Firmware encryption	Memory Protection Unit [CI] Secure cryptography and RNG support [PSA] Cryptographic operations trusted functions
	Denial of service	Rogue code injection or firmware flaw exploiting	Permanent bricking of device	Stack protection Disaster detection and recovery	Arm Pointer Authentication [CI] Secure cryptography and RNG support [PSA] Cryptographic operations trusted functions
	Elevation of privilege	Abuse firmware update mechanism	Firmware version rollback to older buggy version	Support secure firmware update and anti-rollback	[CI] SW update validation [CI] Rollback protection [PSA] Firmware update features
Firmware certificate	Spoofing	Steal private key of asymmetric key pair	Impersonate system admin to generate fake certificate	Organizational policy for key management	Pelion Device Management service
	Tampering	Digital port abuse or physical tampering local storage	Replace certificates to install malware	Secure storage accessing to SPE only (hardware enforced isolation)	[PSA] Use HW based isolation (i.e. Trustzone and Trustzone filters) to enforce access controls [CI] Data protection functionalities, in particular support for asset use policy [CI] Persistent trusted storage [PSA] Secure storage trusted functions
	Information disclosure	Exploiting weak cryptography	Steal secrets	Use state-of-the-art cryptographic algorithms and key sizes	[CI] Secure cryptography and RNG support [PSA] Cryptographic operations trusted functions
		Advanced side-channel analysis		Anti-tampering protection at SoC level	CryptoIsland-300P, Cortex-M35P, CryptoCell-312P
	Denial of service	Remotely tampering local storage	Permanent bricking of device	Enforce access control and least privilege Disaster detection and recovery	[PSA] Isolation and SPE to manage credentials [CI] Secure cryptography and RNG support [PSA] Cryptographic operations trusted functions

Device ID	Spoofing	Modify or clone Device Unique ID	Impersonate another legit device	Secure provisioning and OTP	Pelion Device Management service
	Tampering	Debug port abuse or physical tampering local storage	Tamper unique Device ID	Do not expose Device ID, use alias identity keys or anonymous device on-boarding and attestation	[CI] Secure cryptography and RNG support [PSA] Cryptographic operations trust-ed functions
	Information disclosure	Exploiting weak cryptography	Extract secrets and clone device	Use state-of-the-art cryptographic algorithms and key sizes	
Credentials including biometric data	Spoofing	Replace credential	Impersonate device owner or admin	Enforce access control of credentials and principle of least privilege	[CI] Data protection functionalities, in particular support for asset use policy [PSA] Isolation and SPE to manage credentials
	Information disclosure	Exploiting weak cryptography	Steal credentials	Use state-of-the-art cryptographic algorithms and key sizes	[CI] Secure cryptography and RNG support [PSA] Cryptographic operations trusted functions
		Advanced Side Channel Analysis		Anti-tampering protection at SoC level	CryptoIsland-300P, Cortex-M35P, CryptoCell-312P
	Elevation of privilege	Debug port abuse	Device usurpation Modify firmware and install malware (cf. Firmware asset)	Hardware enforced secure storage accessing to SPE only Secure storage in SPE Debug protection	[CI] Persistent trusted storage functionality [PSA] Secure storage trusted functions [PSA] Secure debug (SDC-600)
Configuration	Spoofing	Device impersonation	Illegal access to configured network		
	Information disclosure	Extract confidential configuration	Illegal access to confidential information		[PSA] Use HW based isolation (i.e. Trustzone and Trustzone filters) to enforce access controls
	Denial of service	Tampering local storage or configuration command over network	Device unavailability, i.e. disconnect from network	Secure storage accessing to SPE only (hardware enforced isolation)	[CI] Data protection functionalities, in particular support for asset use policy [CI] Persistent trusted storage [PSA] Secure storage trusted functions
	Elevation of privilege	Tampering local storage or configuration command over network	Device abnormal behavior	Use TLS, IPSec or HTTPS protocol for communication	[PSA] Cryptographic operations and RNG trusted functions MbedTLS

Event logs	Tampering	Tampering event logs stored locally or in transit	Add fake security event to the logs	Enforce access control of logs and principle of least privilege Event log protection by encryption and authentication Use TLS, IPSec or HTTPS protocol for communication	[CI] Secure cryptography and RNG support [PSA] Cryptographic operations and RNG trusted functions [CI] Data protection functionalities, in particular support for asset use policy [PSA] Audit logs trusted functions MbedTLS
	Repudiation		Suppress critical alerts and events Erase security events logs		
		Tampering system time	Wrong events timeline	Secure timestamp	[PSA] Secure system time
	Information disclosure	Illegally access by exploiting firmware flaws	Unauthorized access to event logs	Secure storage in SPE	[CI] Data protection functionalities, in particular support for asset use policy [PSA] Audit logs trusted functions
User data	Tampering	Tampering local storage	Modify local user data	Secure storage accessing to SPE only (hardware enforced isolation)	[PSA] Use HW based isolation (i.e. Trustzone and Trustzone filters) to enforce access controls [CI] Data protection functionalities, in particular support for asset use policy [CI] Persistent trusted storage [PSA] Secure storage trusted functions
	Information disclosure	Extract user privacy data	Illegal access to user privacy data		
Network connectivity	Tampering	Intercept and tamper network communication data	Modify assets in transit	Data encryption and use TLS, IPSec or HTTPS protocol for communication	[CI] Secure cryptography and RNG support [PSA] Cryptographic operations and RNG trusted functions [PSA] SPE to manage server authentication and communication encryption [PSA] Cryptographic operations and RNG trusted functions MbedTLS
	Information disclosure	Intercept and extract network data	Illegal access to confidential information in transit, i.e. via Man-In-The-Middle		
Device resources	Information disclosure	Simple side-channel analysis	Eavesdropping buses and extract sensitive information	Local data and communication encryption	[CI] Secure cryptography and RNG support [PSA] Cryptographic operations and RNG trusted functions
	Elevation of privilege	Rogue code injection or firmware flaw exploiting	Abuse resources, biometric sensor, cylinder, etc., i.e. cause the device to catch fire – thermal control or drain battery	Execution environment isolation Enforce access control to critical resources	[PSA] Use HW based isolation (i.e. Trustzone and Trustzone filters) to enforce access controls [CI] Isolated secure enclave and secure subsystem [PSA] SPE to manage sensitive operations

Table 3:
Smart door lock threats summary table