TF-PSA-Crypto-Drivers

Suggesting a different way to distribute HW and SW supported crypto across different vendors

Frank Audun Kvamtrø

May-June 2024

Prerequisite knowledge: TF-PSA-Crypto

Mbed TLS today include **TLS/DTLS and X.509 support**, **crypto toolbox support** and **platform abstraction layers**

TF-M only requires **crypto toolbox support** and **platform abstraction layers**

It has been decided to ***split these two features out*** *and place it in a self-contained repository called [TF-PSA-Crypto] in the timeline of developing* ***Mbed TLS 4.X.Y*** *releases. The repo is established already but this is only a read-only version of the relevant components*

Introduction

The intent of this presentation is to showcase a suggestion to host *HW and SW based PSA crypto drivers and configurations* in a supplementary repository to TF-PSA-Crypto, with an established boundary between them for version control and improved continuous integration

This presentation is written to showcase how companies currently host driver resources and how they add complexities related to:

- Overall architecture
- Multiple build systems
- Handling configurations
  - Sometimes in multiple images/builds

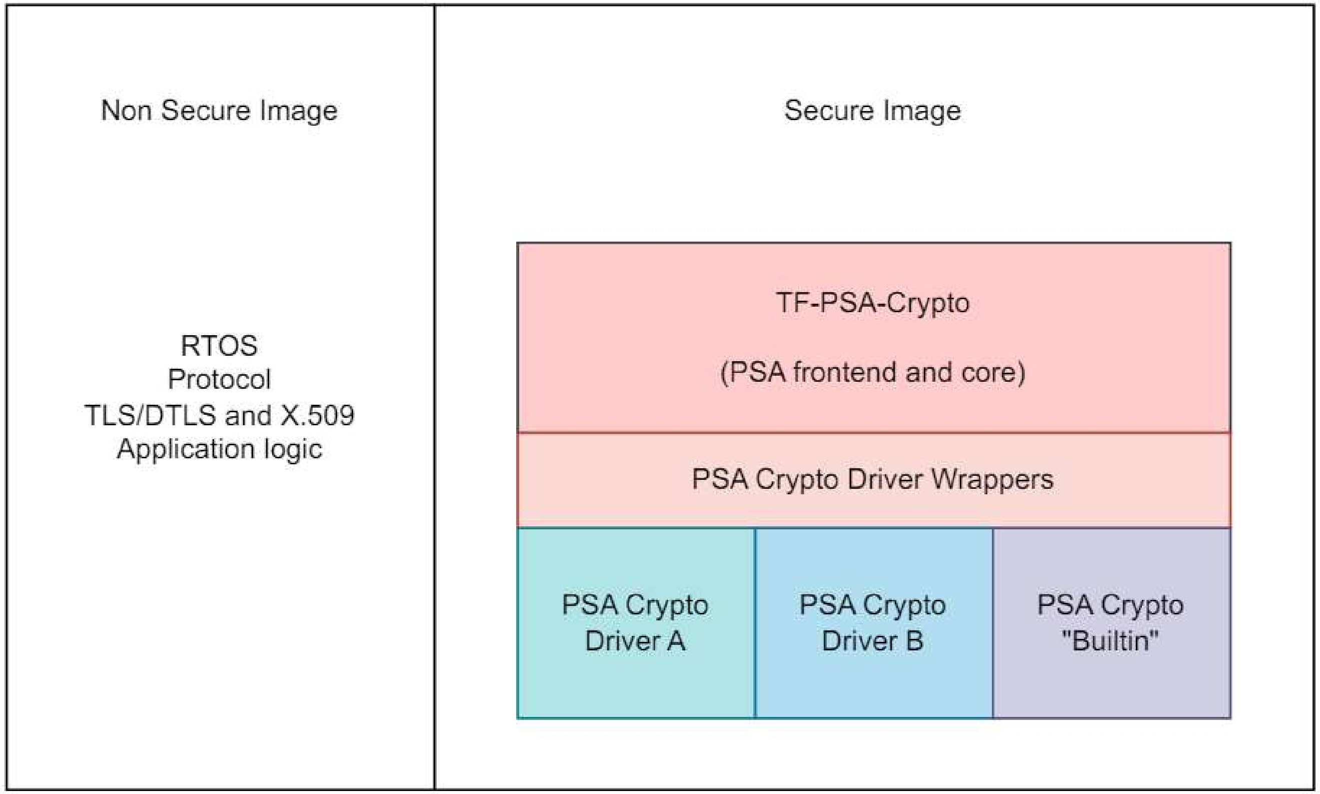
Note that this presentation is written with Trusted Firmware projects and deliverables in mind, but the topic applies to devices that can't/won't use TF-M or TrustZone seperation

PSA crypto drivers – an architectural view

Architectural shortcomings:

- **PSA Crypto Driver Wrappers** seem misplaced living in TF-PSA-Crypto (not extendable)
- TF-PSA-crypto **builtin** support is the only thing that seems "accurately placed"
- There exist and **ext** addition for the **p256-m PSA crypto driver**, but this is a "one off"

=> People create replacements and scatter PSA crypto drivers and the **PSA Crypto Driver Wrappers** and **PSA crypto drivers** around in forks



Where to place PSA crypto drivers?

Rationale for placement (by examples):

- Placing drivers, build-logic and configuration in an RTOS/SDK (e.g. Zephyr) is a *valid option*
  - But the build-system may be different than TF-M/Mbed TLS (Zephyr: Kconfig and CMake based)
  - And TF-PSA-Crypto would need to source Zephyr/SDK located CMakeLists.txt
  - And TF-PSA-Crypto would needs to be aware of what configurations are enabled
  - *And the driver is a dependency for TF-PSA-Crypto, not the RTOS/SDK...*
- Placing drivers, build-logic and configuration in TF-M is a *valid option*
  - But the CMake logic can’t make use of any RTOS build system or features
  - And TF-PSA-Crypto needs to be aware of what configurations are enabled
  - *And the driver is a dependency for TF-PSA-Crypto, not TF-M...*

Although both options are completely valid, they still have some ramifications that makes integration difficult!

TF-PSA-Crypto-Drivers - Configurations

Our claim: *“Configurations should live with the drivers”*

TF-PSA-Crypto doesn’t know what the drivers are as it has no vendor-like construct beyond PSA crypto driver support for builtin features and ext/p256-m and the ability to auto-generate PSA Crypto Driver Wrappers for supporting any acme_ prefixed PSA crypto driver

Our Claim: *“Mbed TLS and/or TF-PSA-Crypto is not the scopes that decides what features that are available and can be enabled”*

TF-PSA-Crypto doesn’t know how crypto should be used in the system. It doesn’t’ know what drivers you support, and it doesn’t know what the features TF-M or by extension the application is requiring to be enabled…

Explanation: acme_ PSA crypto drivers

The term **acme_** is used to Mbed TLS to document and describe a custom PSA crypto driver for **any type** of device that can enact PSA crypto driver entry points and tie it into **actual usage**

Some traits:

- **acme_** is supported in the scripts that can generate boilerplate code for drivers and integration towards a generated version of **PSA Crypto Driver Wrappers**
  - With expecency that generated source-files for drivers will continue using this prefix in naming conventions and configurations
- The prefix **acme_** is prefixed on all driver entry points (e.g. hash driver APIs)
- Driver configurations will follow the pattern PSA_CRYPTO_DRIVER**_XXXX_ACME**

PSA crypto driver are not self sustained

Our claim: *"Isolating PSA crypto drivers as a dependency for TF-PSA-Crypto is better than placement in TF-M and/or custom RTOS and/or SDK deliverable"*

PSA crypto drivers can't be used **by themselves** if we standardize on PSA crypto APIs in application scope, for the highest level of portability.

They require:

- A configured build of PSA core
- A configured and/or generated build of **PSA Crypto Driver Wrappers**
  - With the appropriate configurations to route calls to the **enabled** PSA crypto driver

*Side note: Some systems don't even need to use TF-M, as the execution providing isolation can happen off-chip and could be used via an **opaque PSA crypto driver** (e.g. a Secure Element or Trusted Enclave)*

TF-PSA-Crypto-Drivers – A solution

We propose:

- Establish TF-PSA-Crypto-Drivers to host vendor specific PSA crypto drivers
  - Supplementary to TF-PSA-Crypto (e.g. git submodule)
  - With established rules for naming conventions and driver configurations
- Move PSA crypto drivers from TF-M, RTOS and SDKs into this scope
  - Separated with vendor folders
- Allow both generated or checked in version of PSA Crypto Driver Wrappers
  - Hostable e.g. from vendor folder

Side effect: This allow for devices that can't/won't use TF-M or TrustZone, but are able to provide isolation by other means can place their code and binaries in a more appropriate location

Grand requirement: Stabilized PSA crypto driver APIs

A lot of time and effort is expected on vendors to be able to follow the development of Mbed TLS crypto toolbox and driver integration (soon moved to TF-PSA-Crypto)

Currently the PSA crypto driver APIs is an **internal construct** in Mbed TLS together with the tooling for generation of **acme_ prefixed PSA crypto drivers** and tie-in to **builtin drivers** and **ext/p256-m**

A lack of standardization of these APIs complicates **continous integration** and delays adoption of PSA crypto both with and without TF-M

Essentially, companies risk being stuck in a «infinite catching-up mode», only seeing and resolving issues when things break as there is ***no firm contract***

Further complications: Compatibility layer

PSA crypto drivers and may require certain compatibilities beyond the PSA crypto driver API:

- PSA dependencies (shared headers)
  - Structure sizes for operations
  - Algorithm IDs, types, defines and error codes
- Consistency on platform layer abstractions
  - Heap and/or other memory management?
  - C-library features for printf, snprintf etc.
  - Trace, logs and UART support
  - OS constructs for mutex and other types of locks, timing, threading etc.
- “Borderline” platform APIs:
  - ASN.1/OID/etc.
  - Other types of format constructs reused in TF-M, TLS/DTLS, and other use-cases: (E.g. COSE/CBOR)

If TF-PSA-Crypto-Drivers is to be considered…

.. then It is natural to start thinking about meaningful **requirements** that can be shared in the community.

And the focus of this should be done to **enable cooperation between companies**, to **improve user experience** and to ensure that Trusted Firmware organization can provide **high quality deliverables** with **good support!**

We will give some examples on some generic requirements in slides to follow

Requirements:

- **PSA crypto driver APIs** must be **standardized** and **continually supported**
  - It can no longer be an internal API in Mbed TLS (soon TF-PSA-Crypto) deliverables
  - This API should hopefully be stable, but we understand that Mbed TLS 4.0.0 may lead to API changes
- The role of PSA core vs PSA crypto driver **responsibilities** must be cleared up
  - And we argue that the safest bet is to allow the PSA crypto driver make more decisions
  - … which hopefully leads to a potentially smaller PSA core, when optimizing!
- TF-PSA-Crypto-Driver must allow **device-specific hosting**:
  - Vendor decided copyright and licenses
  - Support distribution of sources and binaries
  - Vendor specific CMake that a TF-PSA-Crypto build can pick up
  - Vendor specific documentation related to the deliverables
  - Vendor specific handling of configurations (e.g. Kconfig based)

Not required

This are examples of things we will not set as requirements:

“Versioning of TF-PSA-Crypto-Drivers coupled to every TF-PSA-Crypto release”

Although it may be practical to consider a strategy for tagging releases that add/change the API

- E.g. for new algorithm/driver API support
- E.g. when a new PSA crypto API spec version is adopted in TF-PSA-Crypto and TF-M
- E.g. when there are additions to the baseline PSA crypto driver wrappers for security fixes

“Trusted Firmware making claims of support beyond TF-M and Mbed TLS deliverables”

This is fully vendor-owned. TF-PSA-Crypto-Drivers may include relevant security fixes from a vendor's perspective, but this should not influence the process of deliverables except for vendor specific commits (in their own scope), updated documentation (release notes etc.)