

Integration of a Cryptographic API with Configurable Hardware and Software Backends in RIOT

Lena Boeckmann

September 8, 2021

Crypto Support in RIOT

- RIOT provides built-in software implementations and third party libraries
- Only pure software implementations
- Run on supported platforms, but are quite inefficient

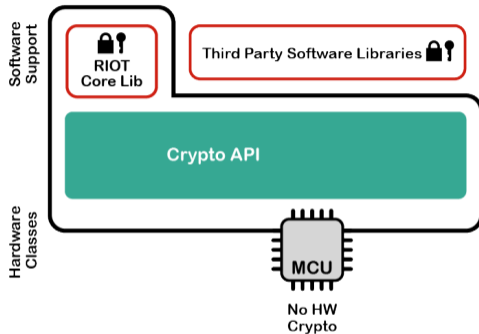
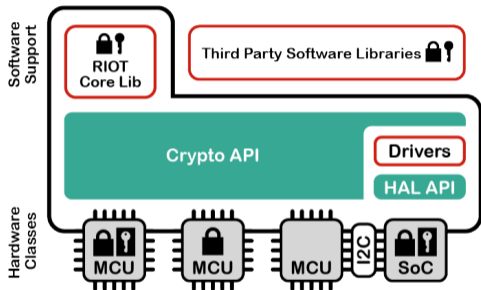


Figure: Currently supported crypto backends in RIOT

Cryptographic Backends



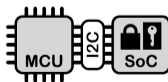
Crypto performed by software libraries



Crypto performed by hardware without key storage

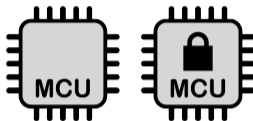


Crypto performed by hardware with key storage



Crypto performed by external hardware with key storage

Transparent Drivers for Unprotected Keys

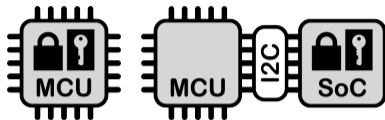


- Keys are stored in RAM or ROM
- Can be operated on by any implementation that accepts plain text key material
- User can freely choose between hardware or software implementations

Definition

Transparent drivers are software implementations and drivers that can be invoked without being dependent on the actual location of a key.

Opaque Drivers for Protected Keys



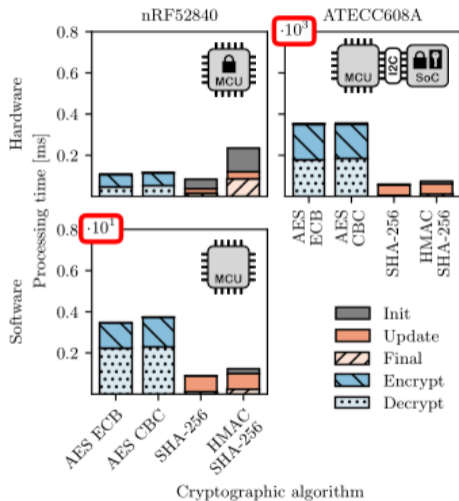
- Dedicated key storage areas (often slots)
- Only crypto processor can access key material
- Caller provides key identifier or slot number
- Only driver assigned to processor the key is located in can be invoked

Definition

Opaque drivers operate on protected keys. They are bound to a key location and can not be chosen freely.

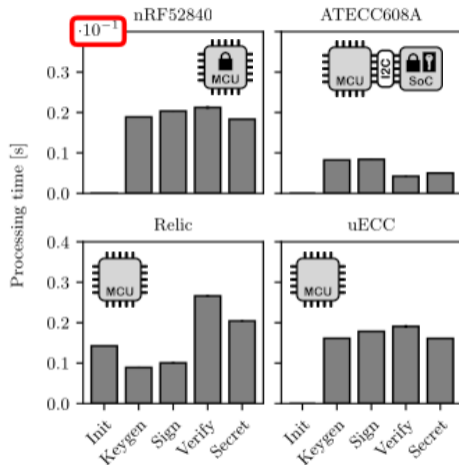
Benefits of Hardware Crypto in Symmetric Operations

- Accelerator on nRF52840 is faster than RIOT software implementation
- Secure Element ATECC608A is less efficient, comes with benefit of protected storage



Benefits of Hardware Crypto in Asymmetric Operations

- Secure Element ATECC608A is faster than software
- Accelerator on nRF52840 outperforms software and secure element



Cryptographic primitives

Challenges and Solution

- Platforms with varying hardware crypto capabilities need to be supported
- Both protected and unprotected key storage need to be supported
- Hardware and software backends should be exchanged transparently beneath a unified API

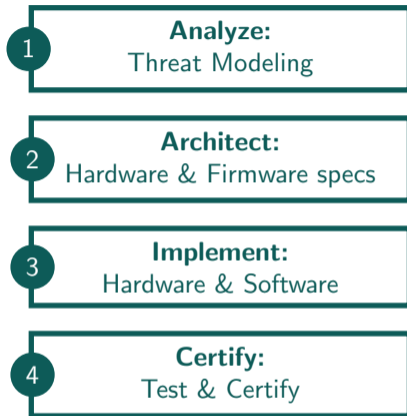
Our Solution:

Implementing the ARM Platform Security Architecture Crypto API

- 1 Introduction
- 2 What is PSA Crypto?**
- 3 Why should we use PSA Crypto?
- 4 Reference Implementations
- 5 Integration in RIOT
- 6 Implementation Status and Outlook

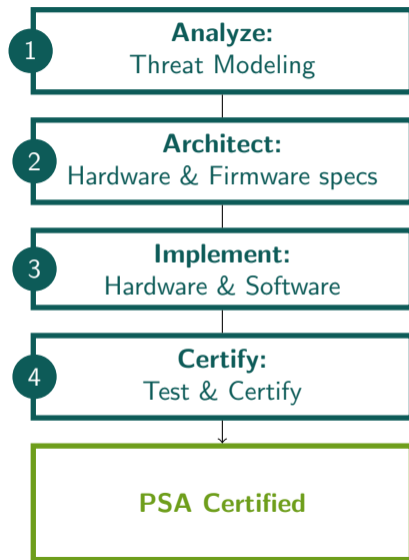
What is PSA?

- ARM Platform Security Architecture
- Framework for development of secure IoT systems



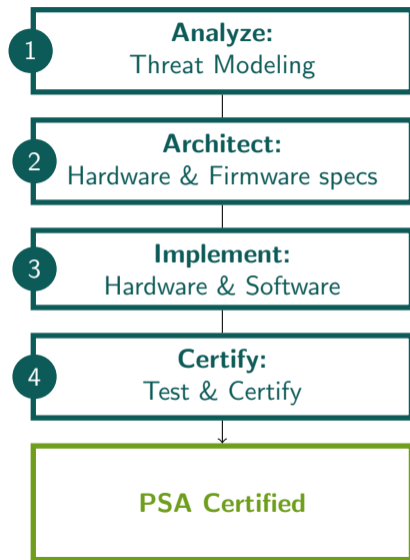
What is PSA?

- ARM Platform Security Architecture
- Framework for development of secure IoT systems
- Threat models, specifications and reference implementations
- Implementations may be certified through PSA Certified scheme



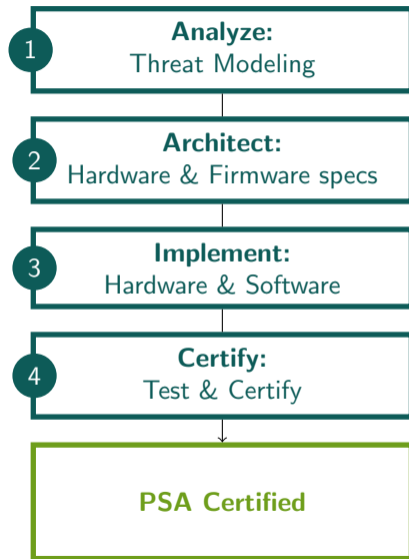
What is PSA?

- ARM Platform Security Architecture
- Framework for development of secure IoT systems
- Threat models, specifications and reference implementations
- Implementations may be certified through PSA Certified scheme
- Open source test suite for implementation verification



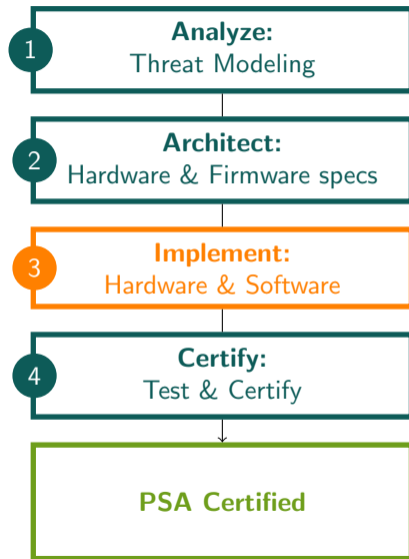
What is PSA?

- Where can we find PSA Crypto?

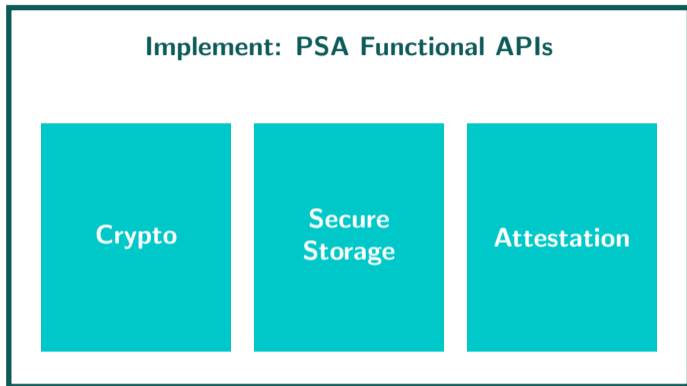


What is PSA?

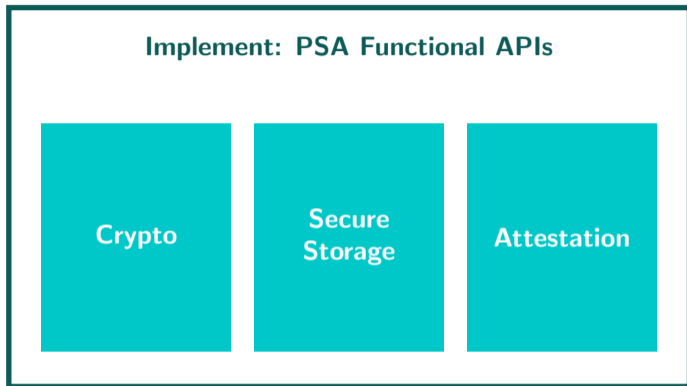
- Where can we find PSA Crypto?



What is the PSA Crypto API?

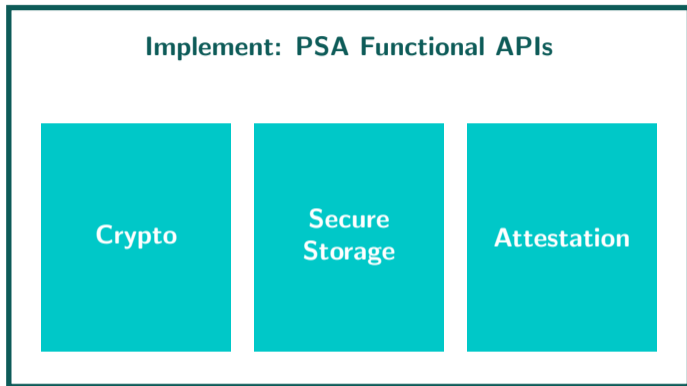


What is the PSA Crypto API?



- Platform independent, suitable for IoT devices

What is the PSA Crypto API?



- Platform independent, suitable for IoT devices
- Designed with usability and portability in mind

- 1 Introduction
- 2 What is PSA Crypto?
- 3 Why should we use PSA Crypto?**
- 4 Reference Implementations
- 5 Integration in RIOT
- 6 Implementation Status and Outlook

What do we get?



PSA Crypto

Complete API design specification

What do we get?

Arch Tests

Test Suite for verification

PSA Crypto

Complete API design specification

What do we get?

Arch Tests

Test Suite for verification

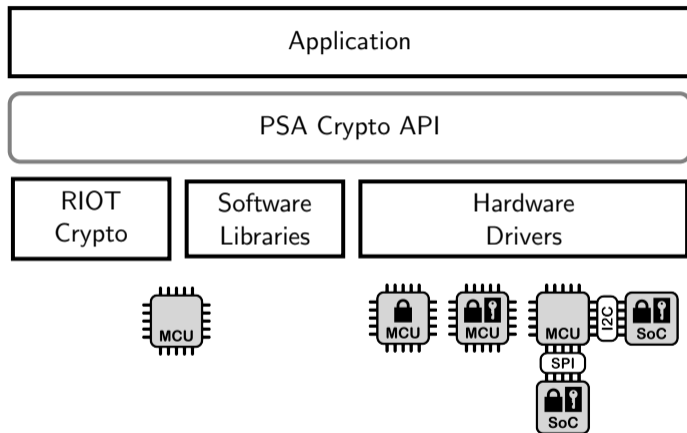
PSA Crypto

Complete API design specification

Implementation
Freedom

Total freedom under the hood

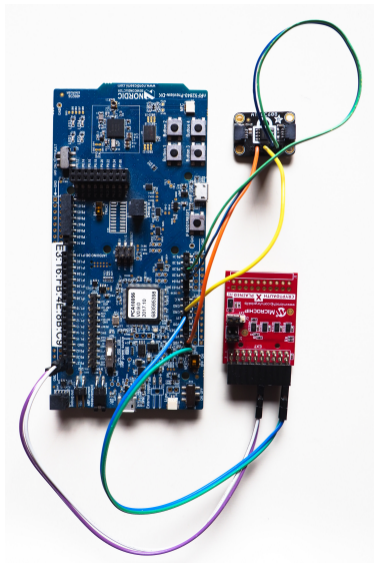
Backend Flexibility



- Hardware agnostic application development possible
- Backends can be exchanged and combined as needed
- Use of multiple secure elements possible

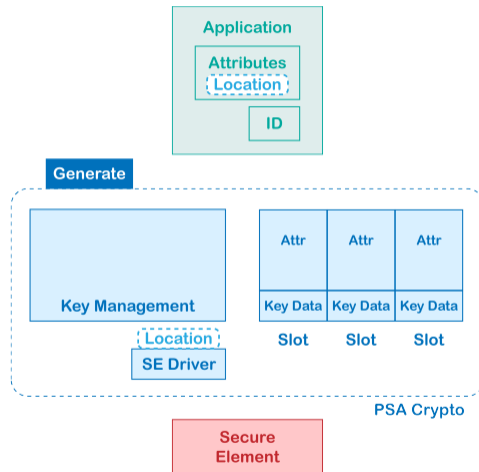
Secure Element Handling

- Multiple SE's can be managed by a driver registry
- Calls are dispatched to appropriate driver depending on key location
- PSA Crypto reference implementation provides an API for SEs



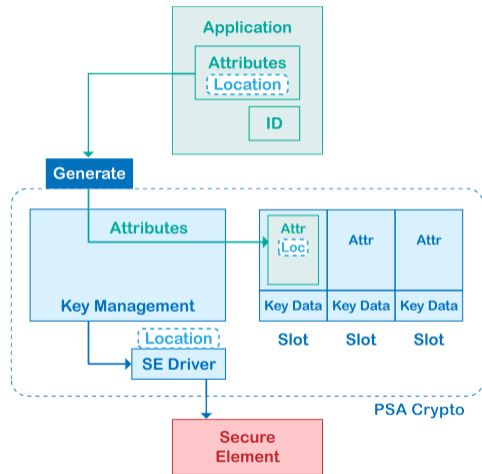
Handling Keys in Protected Storage

- 1 Application provides key attributes with key location



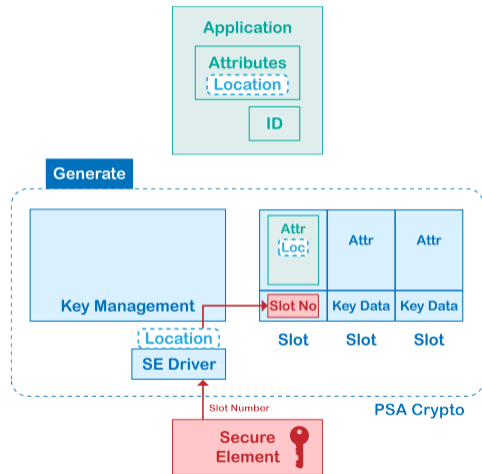
Handling Keys in Protected Storage

- 1 Application provides key attributes with key location
- 2 Attributes are stored and opaque SE driver is invoked



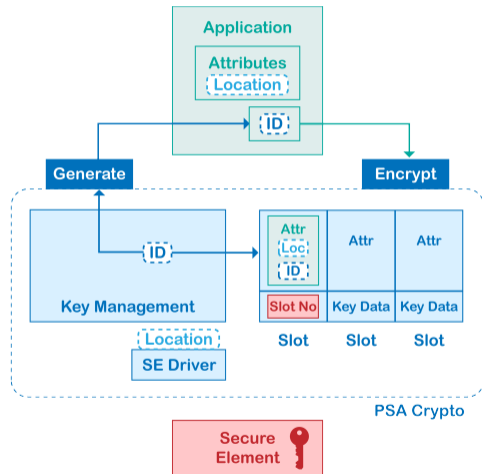
Handling Keys in Protected Storage

- 1 Application provides key attributes with key location
- 2 Attributes are stored and opaque SE driver is invoked
- 3 SE generates and stores key in free slot, returns slot number



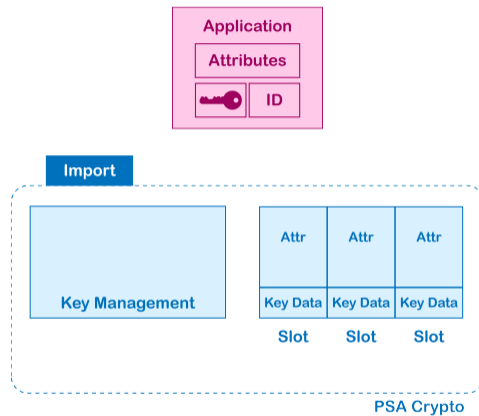
Handling Keys in Protected Storage

- 1 Application provides key attributes with key location
- 2 Attributes are stored and opaque SE driver is invoked
- 3 SE generates and stores key in free slot, returns slot number
- 4 Key manager returns key ID to application for later use



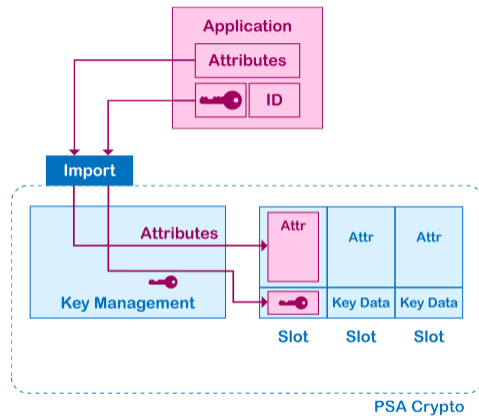
Handling Externally Provided Keys

- 1 Application provides attributes and key material



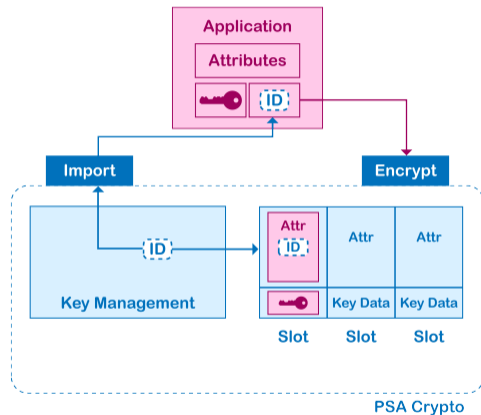
Handling Externally Provided Keys

- 1 Application provides attributes and key material
- 2 Both are stored in local memory



Handling Externally Provided Keys

- 1 Application provides attributes and key material
- 2 Both are stored in local memory
- 3 Key manager returns key ID for later use



- 1 Introduction
- 2 What is PSA Crypto?
- 3 Why should we use PSA Crypto?
- 4 Reference Implementations**
- 5 Integration in RIOT
- 6 Implementation Status and Outlook

What is mbedTLS doing?

- Working on reference implementations for PSA Functional APIs
- We're following their progress and have been in contact
- Still under construction

Driver Wrapper Generation

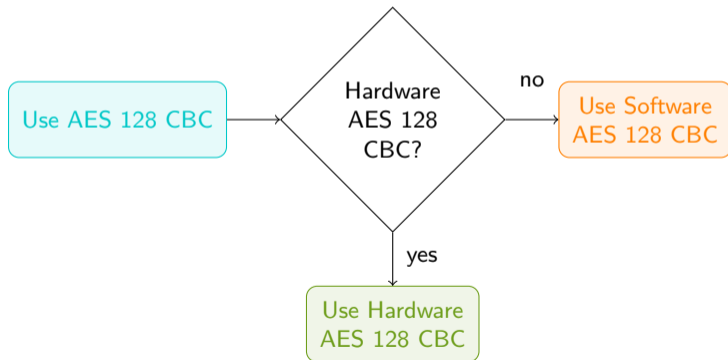
Support of different backends in mbedTLS

- Provide information about available drivers at compile time
- Driver description files in JSON format
- Code generator will parse descriptions and generate wrapper
- Wrapper contains calls to available drivers and software fallback
- Code generator does not exist, yet

- 1 Introduction
- 2 What is PSA Crypto?
- 3 Why should we use PSA Crypto?
- 4 Reference Implementations
- 5 Integration in RIOT**
- 6 Implementation Status and Outlook

Configuring Crypto Backends with Kconfig

- If CPU defines symbol for hardware crypto (e.g. HAS_HW_AES_128_CBC), hardware is default backend
- User may change default configuration if desired



Menuconfig Walkthrough

```
make BOARD=nrf52840dk menuconfig

(Top)
-----
RIOT Configuration
[*] Cortex-M Floating Point Unit (FPU) support
[*] RIOT Core --->
    Drivers --->
    System --->
    Packages --->
    External Modules ----
    *** RIOT is in a migration phase. ***
    *** Some configuration options may not be here. Use CFLAGS instead. ***

-----
[Space/Enter] Toggle/enter  [ESC] Leave menu      [S] Save
[O] Load          [?] Symbol info      [/] Jump to symbol
[F] Toggle show-help mode  [C] Toggle show-name mode  [A] Toggle show-all mode
[Q] Quit (prompts for save) [D] Save minimal config (advanced)
```

Menuconfig Walkthrough

```
make BOARD=nrf52840dk menuconfig
```

(Top) → System

RIOT Configuration

- [] iolist scatter / gather IO
- [] ISR Pipe ----
- [] Locally Unique ID Generator
- [] Math statistics
- [] Dynamic allocation in static memory arrays
- [] Mineplex font
- Networking --->
- [*] NewLib --->
 - Standard Input/Output (STDIO) --->
- [] OD Hex Dump
- Posix --->
- [] One way malloc
- [] Phdat
- [] Platform-independent Power Management
- [] A simple CLI progress bar
- [] UNIX like ps command
- [*] PSA Crypto --->
 - [] Pseudo-Random Number Generation ----
 - [] Semaphore
 - [] Serial number arithmetic (RFC 1982)
 - [] Shell interpreter ----
 - Test utilities --->

+++++

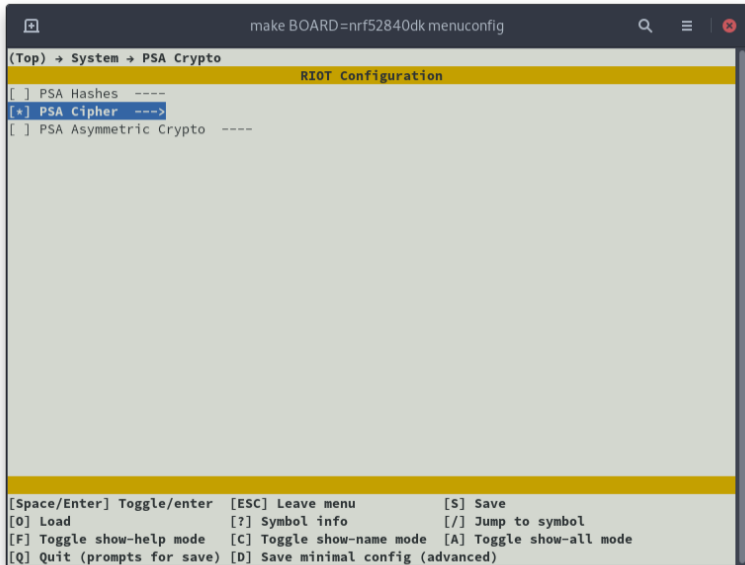
[Space/Enter] Toggle/enter [ESC] Leave menu [S] Save

[O] Load [?] Symbol info [/] Jump to symbol

[F] Toggle show-help mode [C] Toggle show-name mode [A] Toggle show-all mode

[Q] Quit (prompts for save) [D] Save minimal config (advanced)

Menuconfig Walkthrough



```
make BOARD=nrf52840dk menuconfig
(Top) → System → PSA Crypto
RIOT Configuration
[ ] PSA Hashes ----
[*] PSA Cipher --->
[ ] PSA Asymmetric Crypto ----

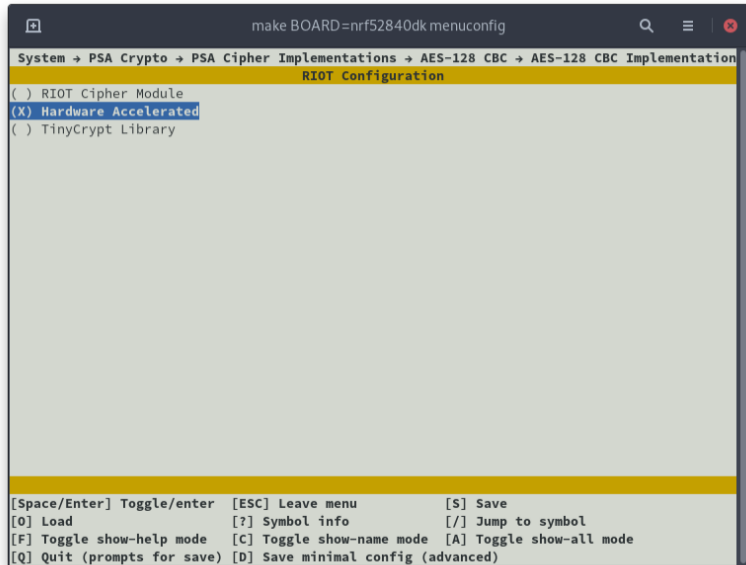
[Space/Enter] Toggle/enter  [ESC] Leave menu          [S] Save
[O] Load                  [?] Symbol info          [/] Jump to symbol
[F] Toggle show-help mode  [C] Toggle show-name mode [A] Toggle show-all mode
[Q] Quit (prompts for save) [D] Save minimal config (advanced)
```


Menuconfig Walkthrough

```
make BOARD=nrf52840dk menuconfig
(Top) → System → PSA Crypto → PSA Cipher Implementations
RIOT Configuration
[ ] AES-128 ECB ----
[*] AES-128 CBC --->
[ ] AES-192 CBC ----
[ ] AES-256 CBC ----

[Space/Enter] Toggle/enter  [ESC] Leave menu          [S] Save
[O] Load                [?] Symbol info          [/] Jump to symbol
[F] Toggle show-help mode [C] Toggle show-name mode [A] Toggle show-all mode
[Q] Quit (prompts for save) [D] Save minimal config (advanced)
```

Menuconfig Walkthrough



The screenshot shows a terminal window titled "make BOARD=nrf52840dk menuconfig". The breadcrumb path is "System → PSA Crypto → PSA Cipher Implementations → AES-128 CBC → AES-128 CBC Implementation". The current menu is "RIOT Configuration", which is highlighted in yellow. It contains three options: "RIOT Cipher Module" (disabled), "Hardware Accelerated" (checked and highlighted in blue), and "TinyCrypt Library" (disabled). At the bottom, a yellow bar contains a list of keyboard shortcuts: [Space/Enter] Toggle/enter, [ESC] Leave menu, [S] Save, [O] Load, [?] Symbol info, [/] Jump to symbol, [F] Toggle show-help mode, [C] Toggle show-name mode, [A] Toggle show-all mode, and [Q] Quit (prompts for save), [D] Save minimal config (advanced).

```
make BOARD=nrf52840dk menuconfig
System → PSA Crypto → PSA Cipher Implementations → AES-128 CBC → AES-128 CBC Implementation
RIOT Configuration
( ) RIOT Cipher Module
(X) Hardware Accelerated
( ) TinyCrypt Library

[Space/Enter] Toggle/enter  [ESC] Leave menu          [S] Save
[O] Load                    [?] Symbol info          [/] Jump to symbol
[F] Toggle show-help mode  [C] Toggle show-name mode [A] Toggle show-all mode
[Q] Quit (prompts for save) [D] Save minimal config (advanced)
```

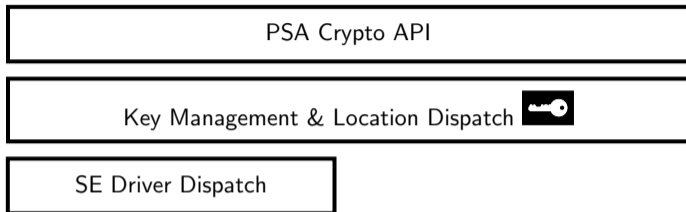
PSA Crypto Implementation Structure



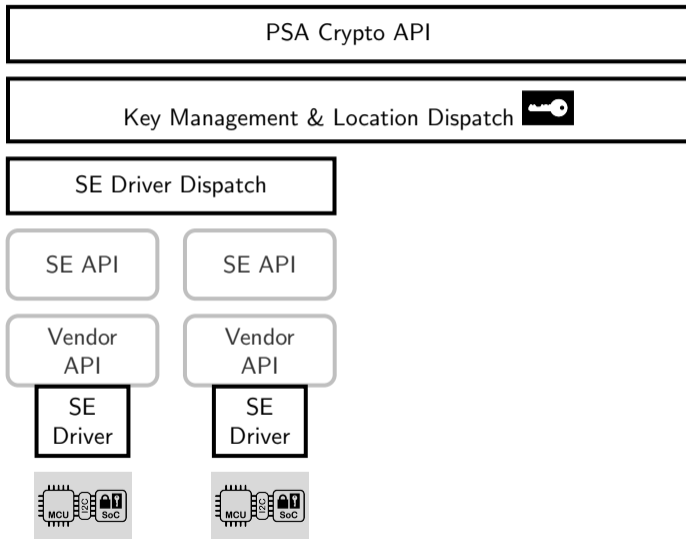
PSA Crypto Implementation Structure



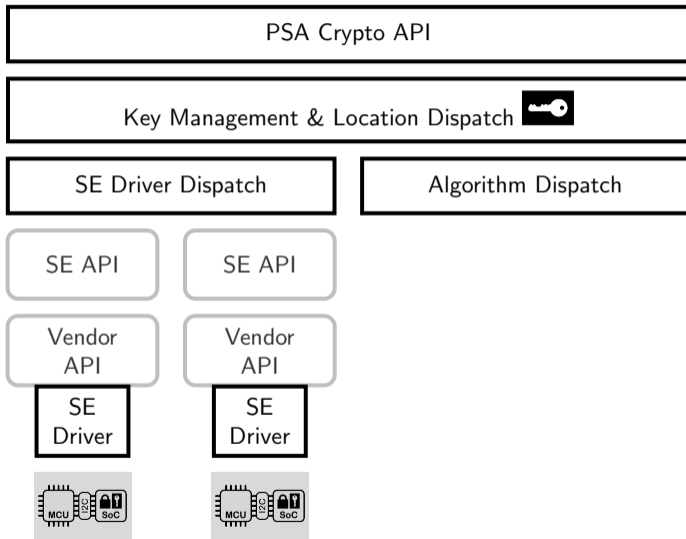
PSA Crypto Implementation Structure



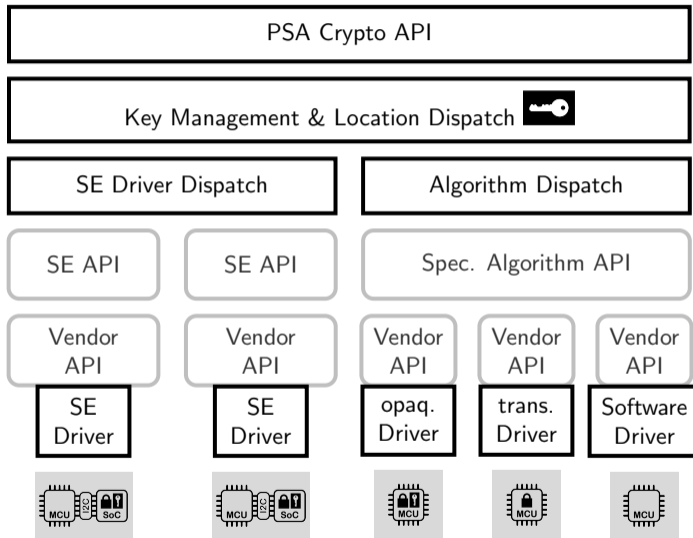
PSA Crypto Implementation Structure



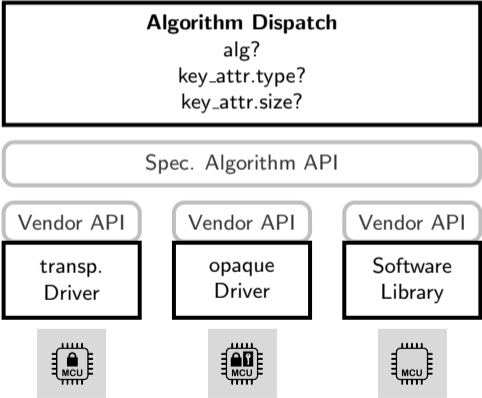
PSA Crypto Implementation Structure



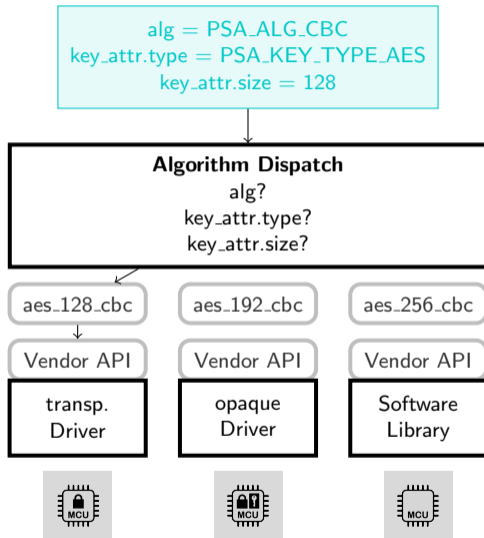
PSA Crypto Implementation Structure



Algorithm Dispatch



Algorithm Dispatch



Application Example of AES Encryption

```
1 uint8_t key[] = { ... };
2 uint8_t key_length = AES_128_KEY_LENGTH;
3
4 psa_key_id_t key_id;
5 psa_key_attributes_t attr = psa_key_attributes_init();
6
7 // Set key attributes
8 psa_key_lifetime_t lifetime = PSA_KEY_LOCATION_LOCAL_STORAGE |
9                               PSA_KEY_PERSISTENCE_VOLATILE;
10
11 psa_set_key_lifetime(&attr, lifetime);
12 psa_set_key_algorithm(&attr, PSA_ALG_CBC_NO_PADDING);
13 psa_set_key_usage_flags(&attr, PSA_KEY_USAGE_ENCRYPT);
14 psa_set_key_type(&attr, PSA_KEY_TYPE_AES);
15 psa_set_key_bits(&attr, 128);
16
17 // Import key and store it as specified in lifetime
18 psa_import_key(&attr, key, key_length, &key_id);
19
20 // Encrypt some plaintext with key belonging to key_id
21 psa_cipher_encrypt(key_id, PSA_ALG_CBC_NO_PADDING, plaintext,
22                   plaintext_length, output, output_size, &output_length);
```

Implementation Status

Component	Status	Description	Next Steps
PSA Arch Testsuite as package	✓	Hash tests work	Enable other algorithm tests
Volatile Key Management	✓	Volatile keys in local memory and SEs can be handled	Add support for persistent keys
Cryptographic Operations	✓	Some hashes and ciphers work	Extend support for cryptographic operations
Secure Element Handling	✓	Multiple devices can be handled	Add support for other devices

Outlook: Support of operations in trusted execution environments