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
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

1. **Analyze:** Threat Modeling
2. **Architect:** Hardware & Firmware specs
3. **Implement:** Hardware & Software
4. **Certify:** Test & Certify
What is PSA?

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

1. **Analyze:** Threat Modeling
2. **Architect:** Hardware & Firmware specs
3. **Implement:** Hardware & Software
4. **Certify:** Test & Certify
What is PSA?

- ARM Platform Security Architecture
- Framework for development of secure IoT systems
- Threat models, specifications and reference implementations

1. Analyze: Threat Modeling
2. Architect: Hardware & Firmware specs
3. Implement: Hardware & Software
4. Certify: Test & Certify
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

Analyze: Threat Modeling

Architect: Hardware & Firmware specs

Implement: Hardware & Software

Certify: Test & Certify

PSA Certified
What is PSA?

- Where can we find PSA Crypto?
What is PSA?

- Where can we find PSA Crypto?
What is the PSA Crypto API?

Implement: PSA Functional APIs

- Crypto
- Secure Storage
- Attestation

• Platform independent, suitable for IoT devices
• Designed with usability and portability in mind
What is the PSA Crypto API?

Implement: PSA Functional APIs

- Crypto
- Secure Storage
- Attestation

- 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
Introduction

What is PSA Crypto?

Why should we use PSA Crypto?

Reference Implementations

Integration in RIOT

Implementation Status and Outlook
What do we get?

- PSA Crypto
- Complete API design specification
- Arch Tests
- Test Suite for verification
- Implementation
- Freedom
- Total freedom under the hood
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

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

Application

Attributes

ID

Location

Generate

Key Management

SE Driver

Slot No

Key Data

Key Data

Slot

Slot

Slot

PSA Crypto

Secure Element

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

Diagram:
- Application
  - Attributes
  - Key Material
- Key Manager
  - Returns Key ID for later use
- Key Management
  - Import
  - Attr
  - Key Data
  - Slot
- PSA Crypto
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
Introduction

What is PSA Crypto?

Why should we use PSA Crypto?

Reference Implementations

Integration in RIOT

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

![Flowchart]

- Use AES 128 CBC
- Hardware AES 128 CBC?
  - yes: Use Hardware AES 128 CBC
  - no: Use Software AES 128 CBC
Menuconfig Walkthrough

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  [$] Save
[0] Load  [?] Symbol info  [/] Jump to symbol
[Q] Quit (prompts for save)  [D] Save minimal config (advanced)
Menuconfig Walkthrough
Menuconfig Walkthrough

(Top) → System → PSA Crypto

RIOT Configuration

[ ] PSA Hashes
[at] PSA Cipher
[ ] PSA Asymmetric Crypto

[Space/Enter] Toggle/enter  [ESC] Leave menu  [$] Save
[0] Load  [?] Symbol info  [\] Jump to symbol
[Q] Quit (prompts for save)  [D] Save minimal config (advanced)
Menuconfig Walkthrough
Menuconfig Walkthrough

make BOARD=nrf52840dk menuconfig

System → PSA Crypto → PSA Cipher Implementations → AES-128 CBC → AES-128 CBC Implementation

RIOT Configuration

(X) Hardware Accelerated

([Space/Enter] Toggle/enter  [ESC] Leave menu  [$] Save
[0] Load  [?] Symbol info  [/] Jump to symbol
[Q] Quit (prompts for save)  [D] Save minimal config (advanced)
PSA Crypto Implementation Structure

PSA Crypto API
PSA Crypto Implementation Structure

PSA Crypto API

Key Management & Location Dispatch

Vendor API

Vendor API

Vendor API

Vendor API

Vendor API

Vendor API

Vendor API

Vendor API
PSA Crypto Implementation Structure

- PSA Crypto API
- Key Management & Location Dispatch
- SE Driver Dispatch
PSA Crypto Implementation Structure

PSA Crypto API

Key Management & Location Dispatch

SE Driver Dispatch

SE API

SE API

Vendor API

Vendor API

SE Driver

SE Driver
PSA Crypto Implementation Structure

PSA Crypto API

Key Management & Location Dispatch

SE Driver Dispatch

Algorithm Dispatch

SE API

Vendor API

SE Driver

SE Driver

Vendor API

Vendor API
PSA Crypto Implementation Structure

PSA Crypto API

Key Management & Location Dispatch

SE Driver Dispatch

Algorithm Dispatch

SE API

SE API

Vendor API

Vendor API

Spec. Algorithm API

Vendor API

Vendor API

Vendor API

Vendor API

SE Driver

SE Driver

SE Driver

Software Driver

Vendor API

Vendor API

Software Driver

Vendor API

Vendor API

Vendor API

Vendor API
Algorithm Dispatch

Algorithm Dispatch
alg?
key_attr.type?
key_attr.size?

Spec. Algorithm API

Vendor API
transp. Driver

Vendor API
opaque Driver

Vendor API
Software Library
Algorithm Dispatch

alg = PSA_ALG_CBC
key_attr.type = PSA_KEY_TYPE_AES
key_attr.size = 128

Algorithm Dispatch

alg?
key_attr.type?
key_attr.size?

aes_128_cbc
Vendor API
transp. Driver

aes_192_cbc
Vendor API
opaque Driver

aes_256_cbc
Vendor API
Software Library
Application Example of AES Encryption

```c
uint8_t key[] = { ... };
uint8_t key_length = AES_128_KEY_LENGTH;

psa_key_id_t key_id;
psa_key_attributes_t attr = psa_key_attributes_init();

// Set key attributes
psa_key_lifetime_t lifetime = PSA_KEY_LOCATION_LOCAL_STORAGE |
    PSA_KEY_PERSISTENCE_VOLATILE;

psa_set_key_lifetime(&attr, lifetime);
psa_set_key_algorithm(&attr, PSA_ALG_CBC_NO_PADDING);
psa_set_key_usage_flags(&attr, PSA_KEY_USAGE_ENCRYPT);
psa_set_key_type(&attr, PSA_KEY_TYPE_AES);
psa_set_key_bits(&attr, 128);

// Import key and store it as specified in lifetime
psa_import_key(&attr, key, key_length, &key_id);

// Encrypt some plaintext with key belonging to key_id
psa_cipher_encrypt(key_id, PSA_ALG_CBC_NO_PADDING, plaintext,
    plaintext_length, output, output_size, &output_length);
```
# Implementation Status

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

**Outlook:** Support of operations in trusted execution environments