Usable Security for an IoT OS: Integrating the Zoo of Embedded Crypto Components Below a Common API

Lena Boeckmann, Peter Kietzmann, Leandro Lanzieri, Thomas Schmidt, Matthias Wählisch

lena.boeckmann@haw-hamburg.de

International Conference on Embedded Wireless Systems and Networks 2022

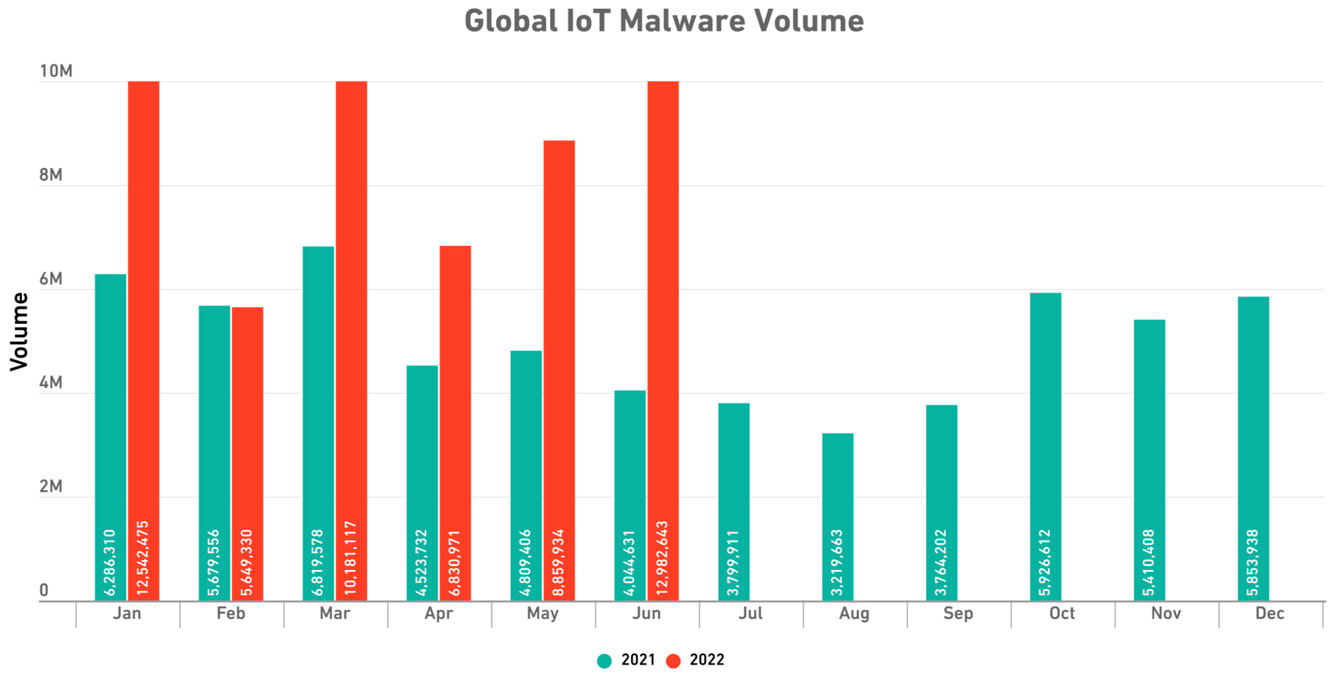
HAW HAMBURG

iNET

Freie Universität Berlin

Security in the IoT

- 77% increase in IoT malware worldwide in first half of 2022[1]
- Growing threat potential and increasing number of attacks

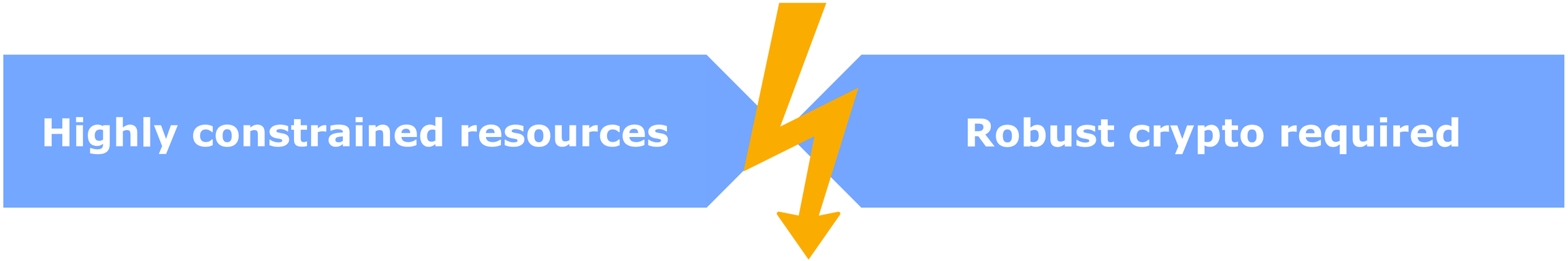


[1]Mid-Year Update: 2022 SonicWall Cyber Threat Report
https://www.sonicwall.com/medialibrary/en/white-paper/mid-year-2022-cyber-threat-report.pdf

Security in the IoT

Cryptography is the basis for securing IoT systems

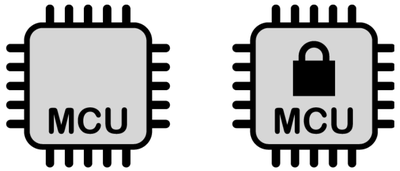
Challenges for Cryptography in the IoT



- Solutions:
  - Hardware: accelerators and protected key storage
  - Software: embrace heterogeneous backends and capabilities
- A versatile IoT OS should support many solutions

Backend Classification

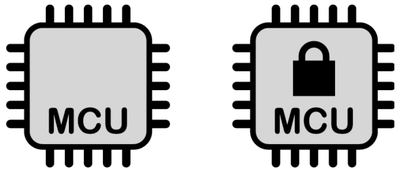
Backend Classification



- Key storage on-chip in memory
- **Keys** are passed to driver/library API

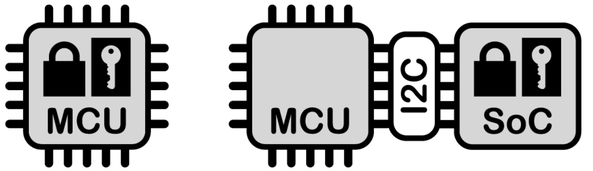
Transparent

Backend Classification



- Key storage on-chip in memory
- **Keys** are passed to driver/library API

Transparent



- Key storage in protected memory
- **Key references** are passed to driver API

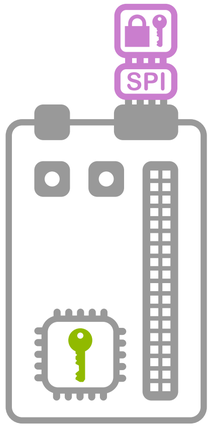
Opaque

On-Chip **External**

The Zoo of Crypto Backends in the IoT

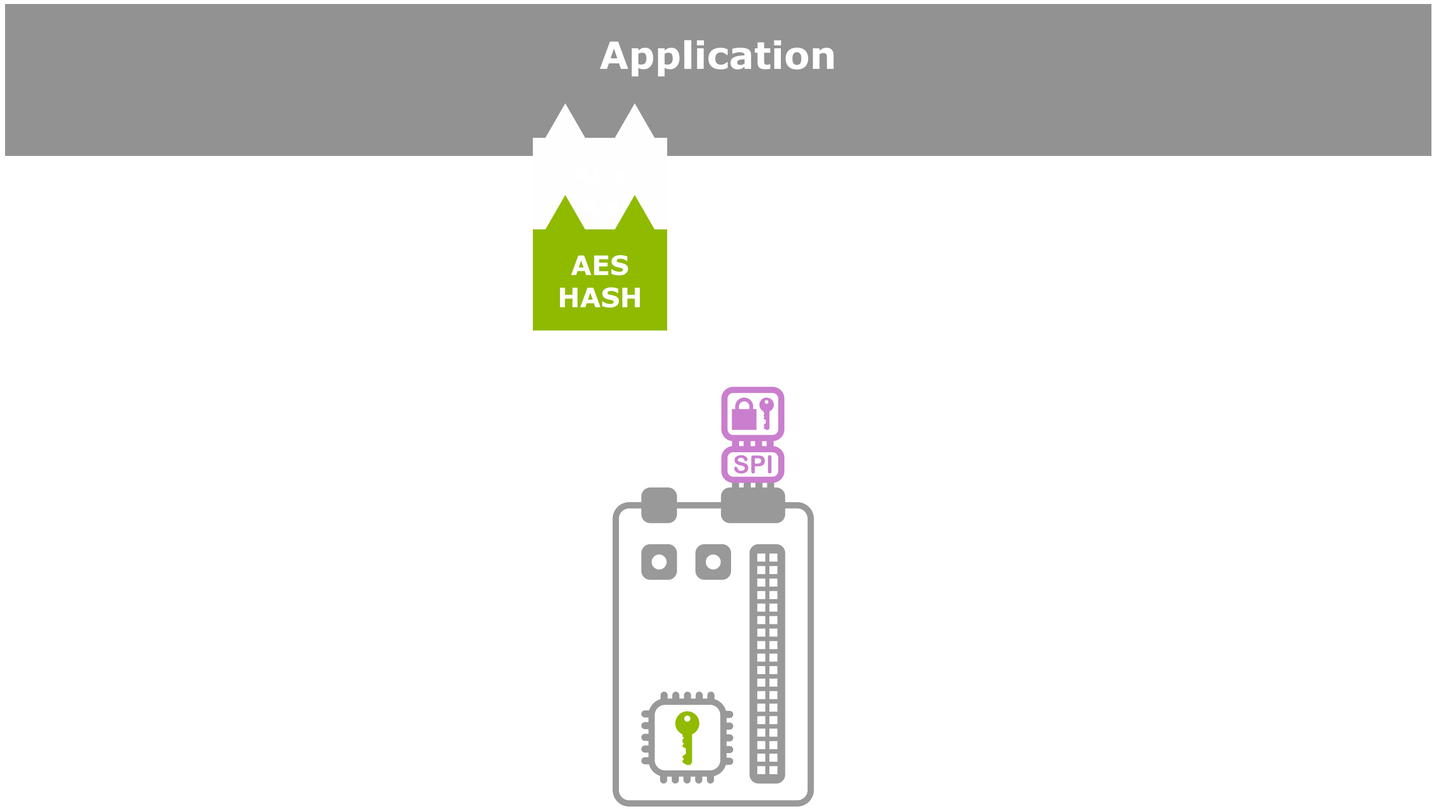
Application

Driver APIs:



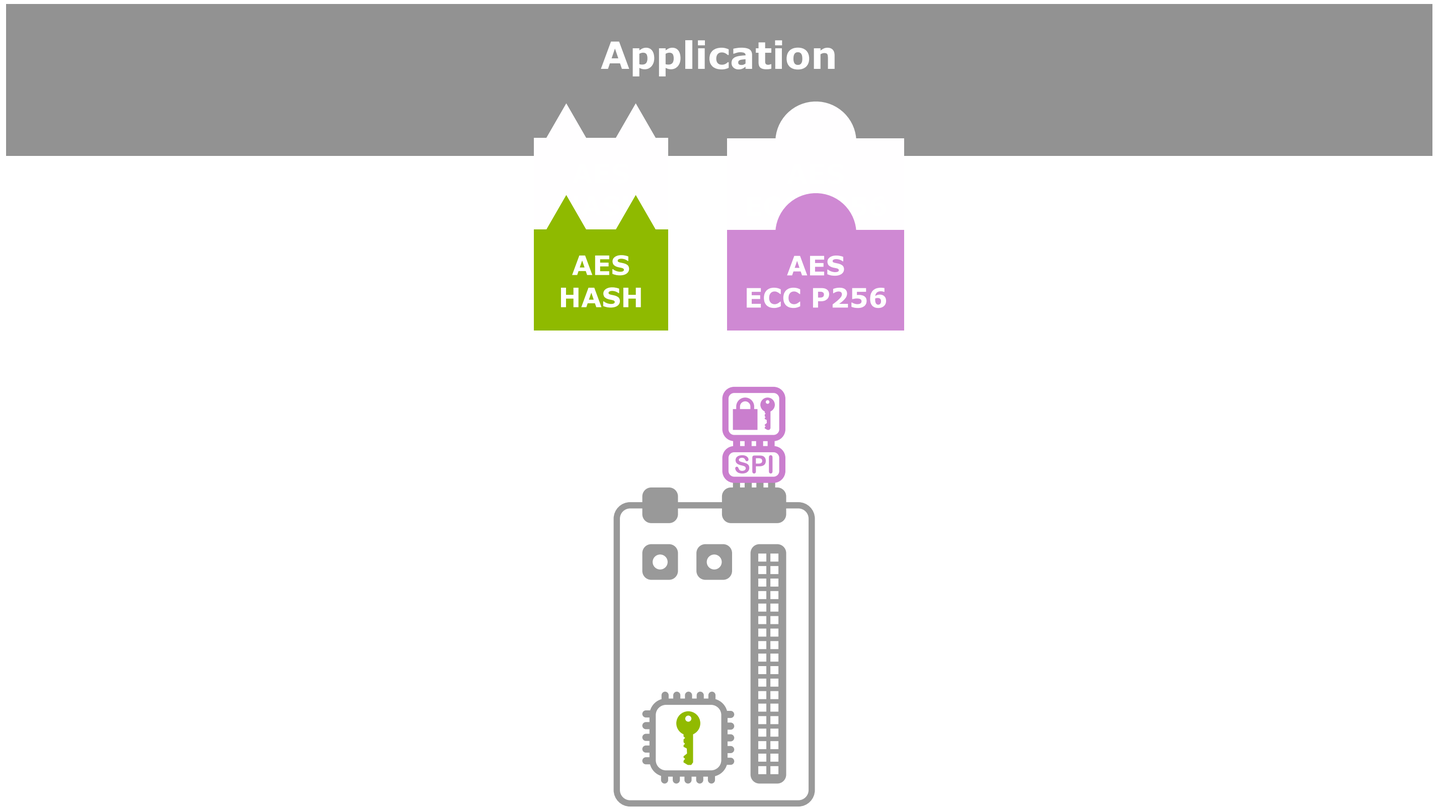
The Zoo of Crypto Backends in the IoT

Driver APIs:



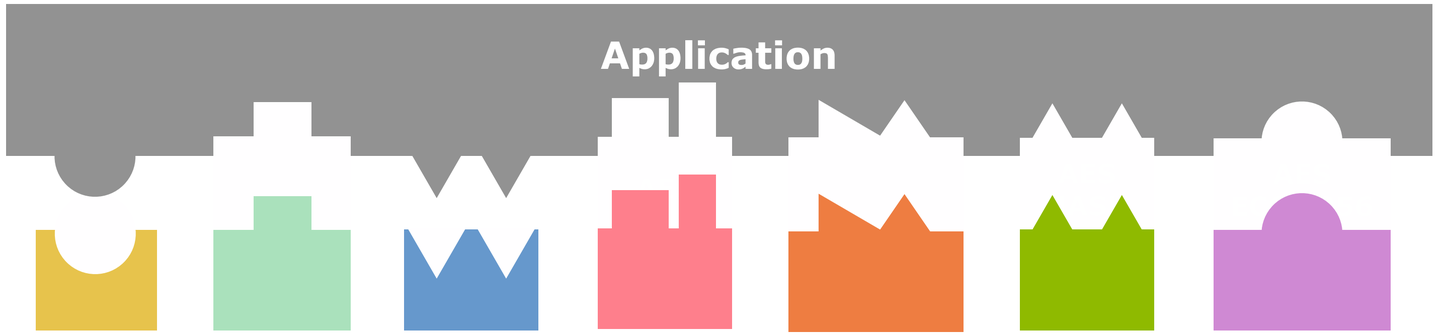
The Zoo of Crypto Backends in the IoT

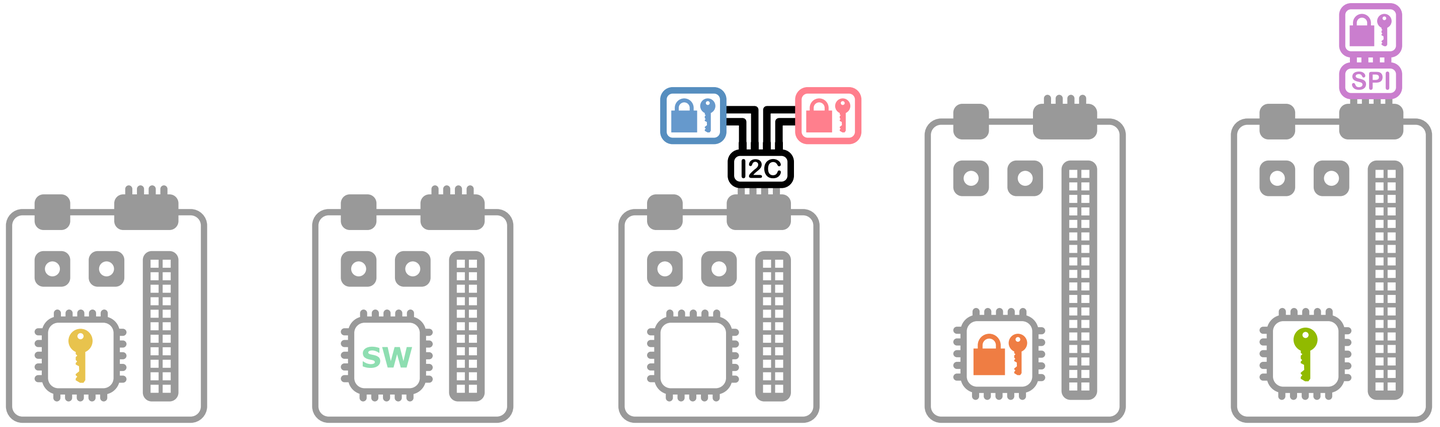
Driver APIs:



The Zoo of Crypto Backends in the IoT

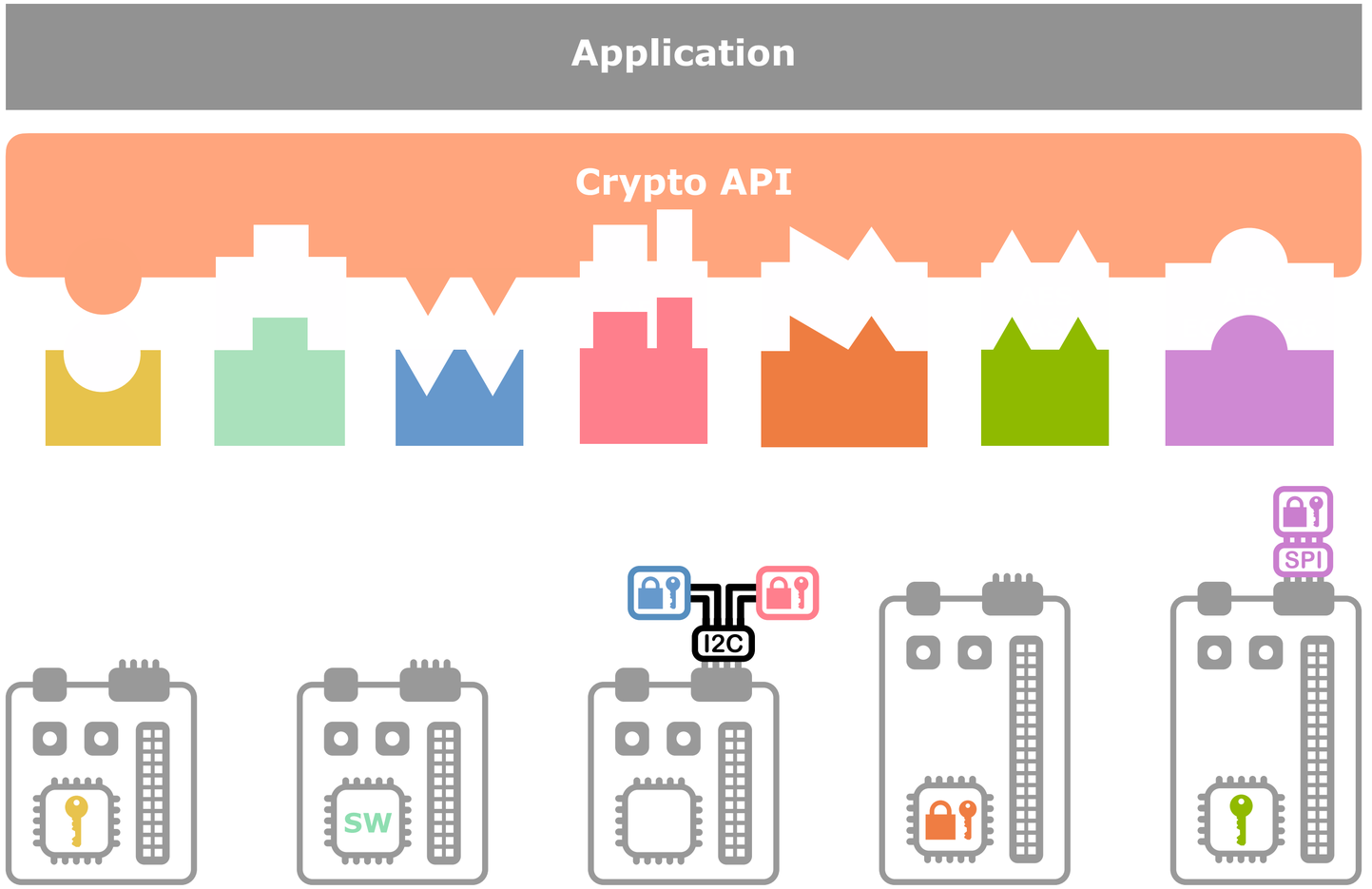
Driver APIs:



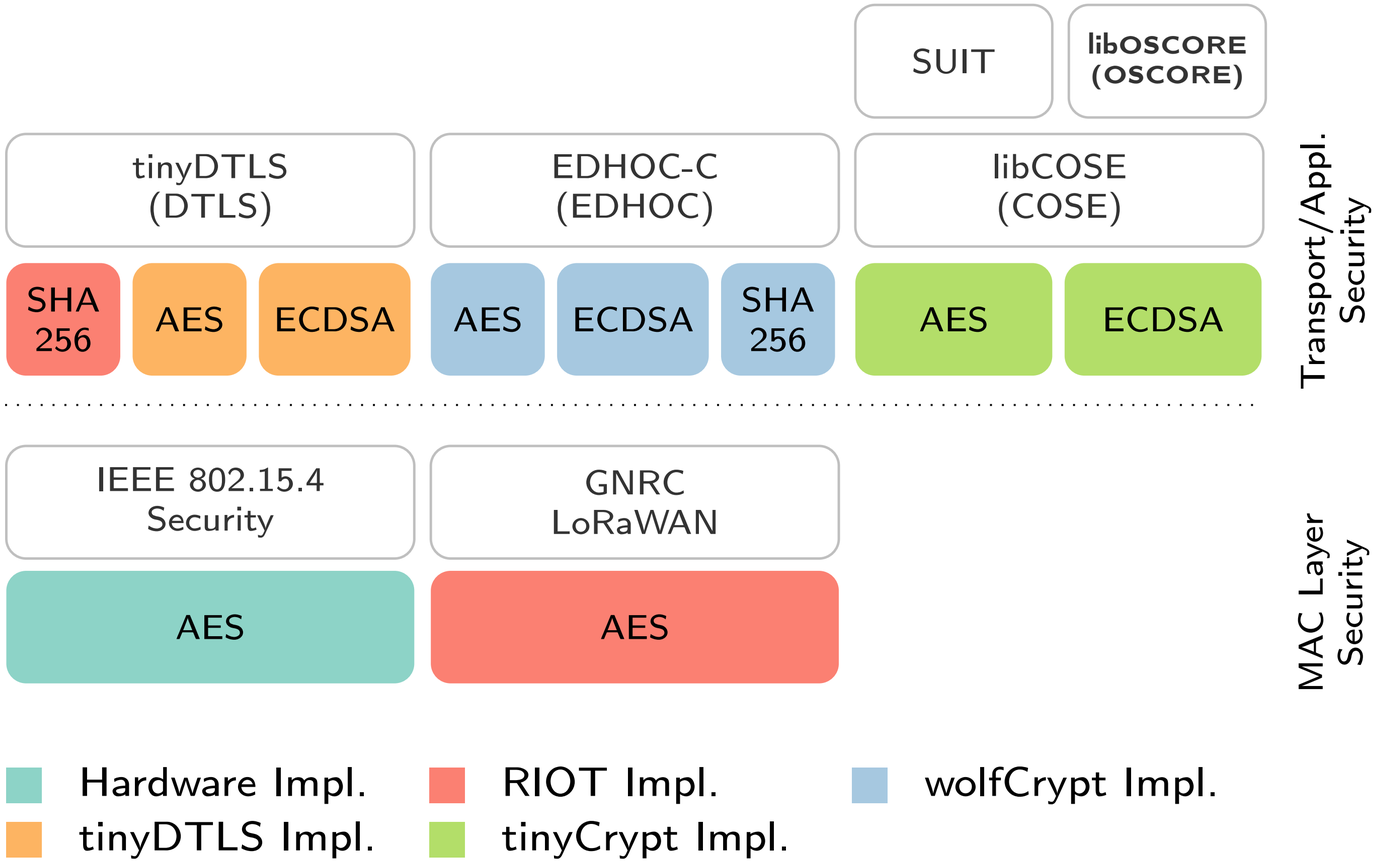


The Zoo of Crypto Backends in the IoT

Driver APIs:

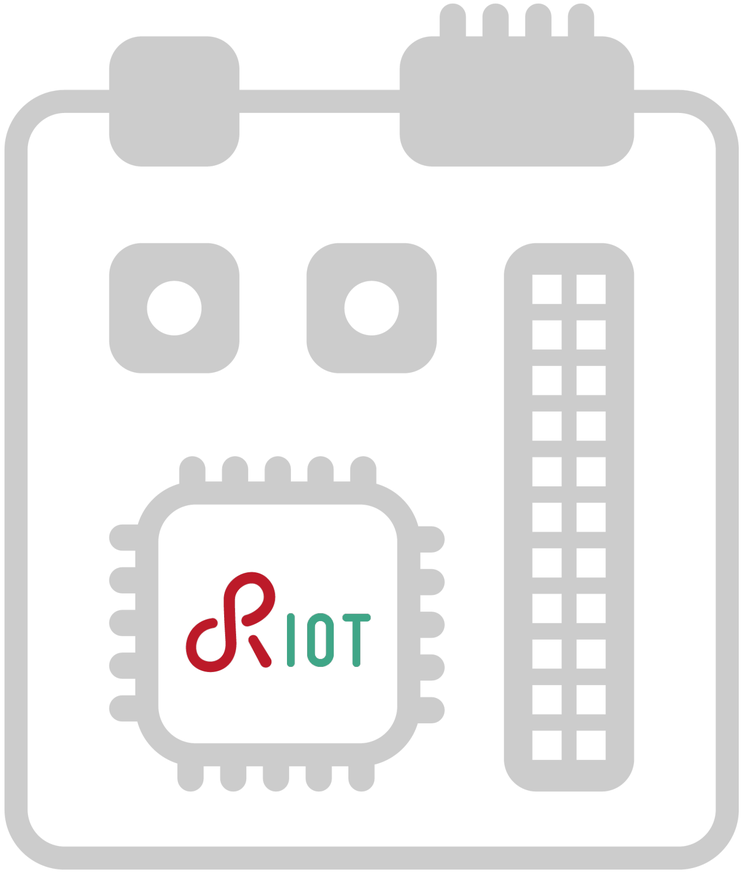


Secure Protocol Stack in an IoT OS



RIOT OS

- Open Source IoT operating system
- Supports > 240 different platforms, 68 MCU's with varying crypto capabilities
- Integrates several crypto software libraries and external crypto devices (Secure Elements)



https://www.riot-os.org/

Outline

1. Requirements for a Crypto API
2. ARM PSA Crypto API
3. Integration in RIOT
4. Evaluation
5. Conclusion

Requirements for a Crypto API

Portability

- Support transparent backend exchange
- Exchange RIOT implementation with other implementations

Usability

- Simplify development of secure applications
- Good documentation and state-of-the-art examples
- Handle keys by reference

Configurability

- Support all relevant algorithms
- Support hardware and software backends
- Allow for any combination of drivers and libraries

ARM PSA Crypto API

Where does PSA Crypto come from?

- ARM **P**latform **S**ecurity **A**rchitecure
- Framework for securing IoT systems
- Standardized resources for hardware, firmware and software development
- Certification process
- PSA Crypto API is part of PSA specifications

Why is PSA the Right Choice?

Portability

- Indirect, ID-based key access
- Backends with and without protected storage
- Used by other OSes and libraries (e.g. MbedTLS, FreeRTOS)

Usability

- Internal key handling without exposure to users
- Restricts key usage with policies
- Comes with extensive documentation

Configurability

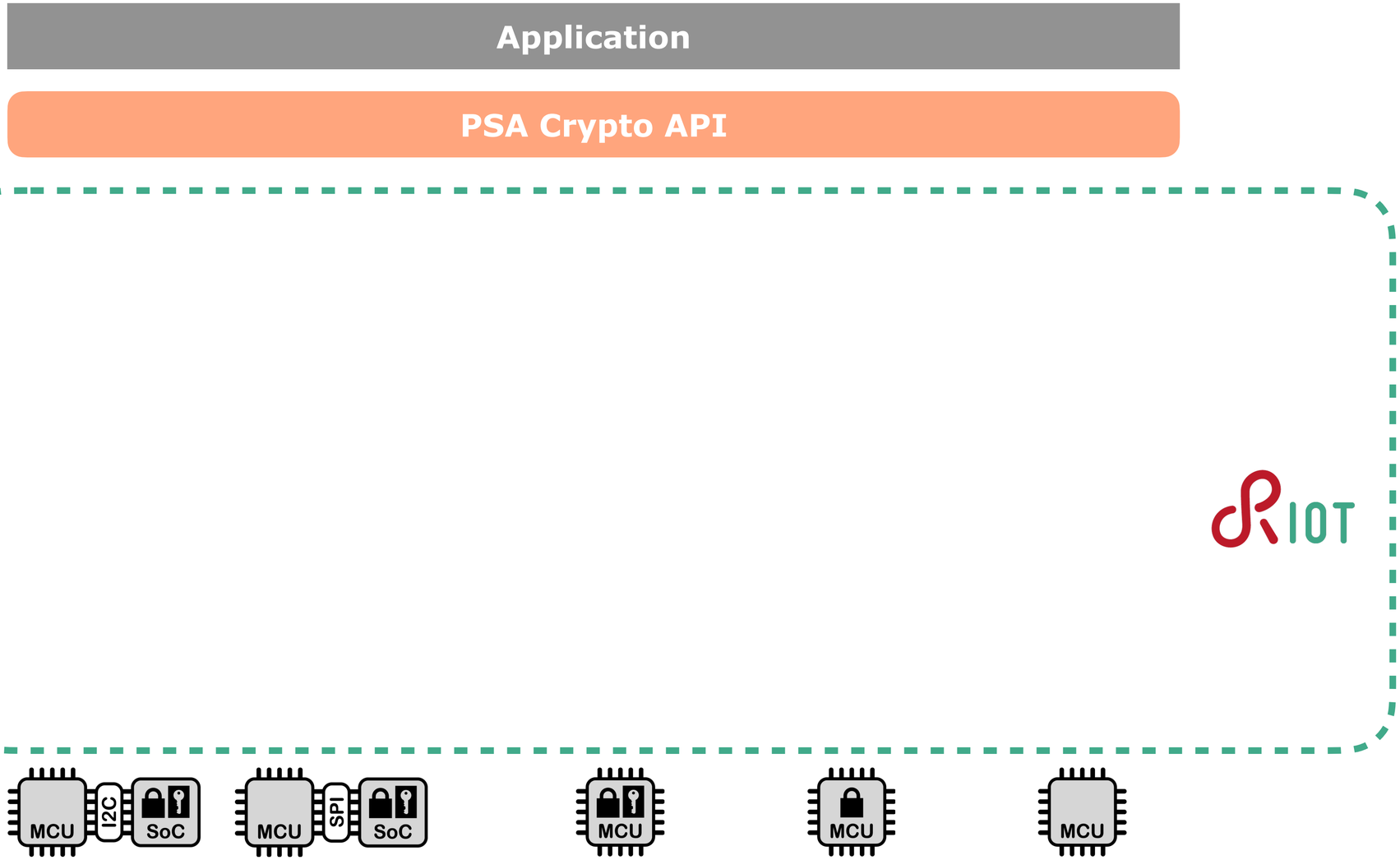
- Generic, backend agnostic interface
- Supports multiple secure elements

Bonus

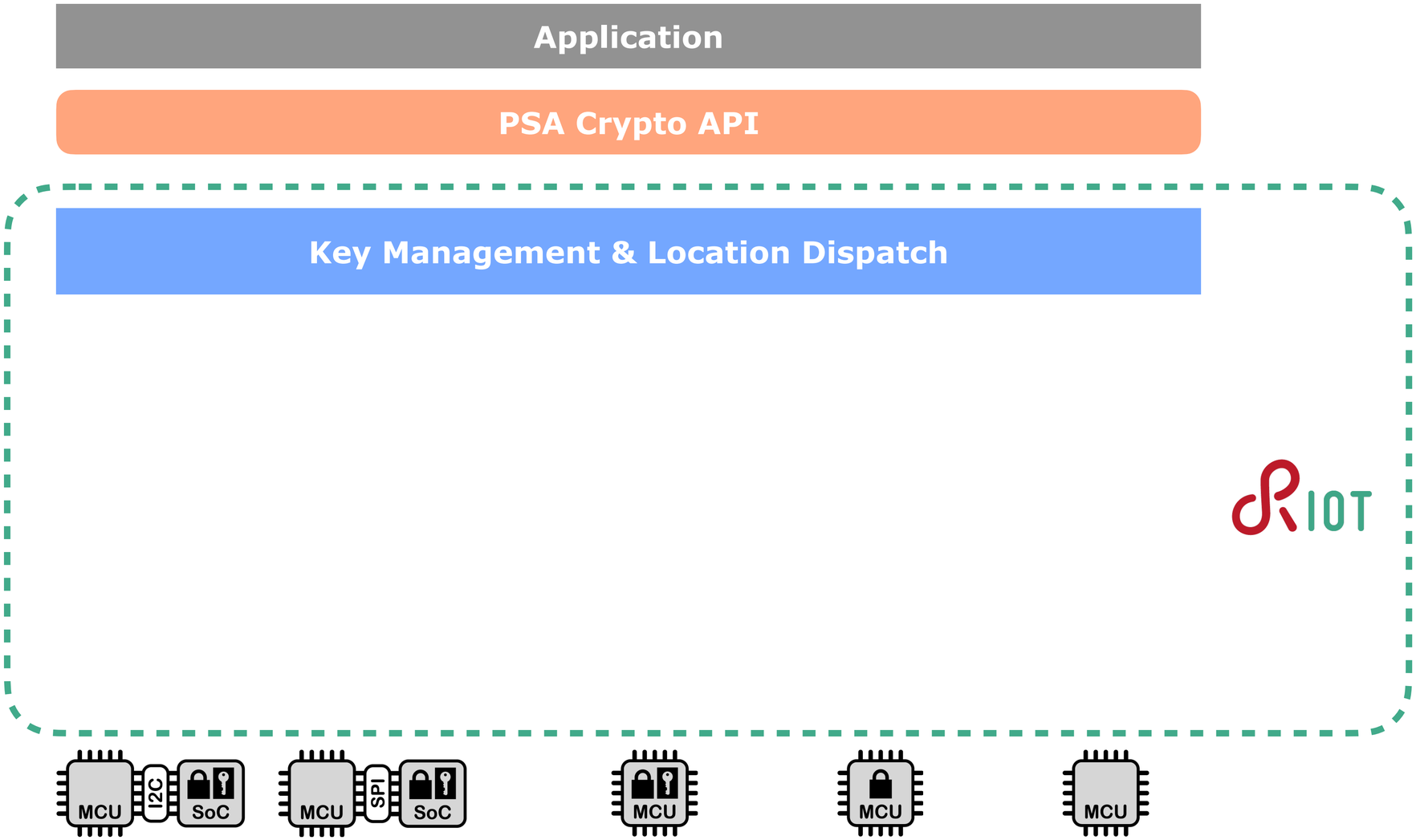
Open Source Test Suite

Integration in RIOT

Integration in RIOT



Integration in RIOT



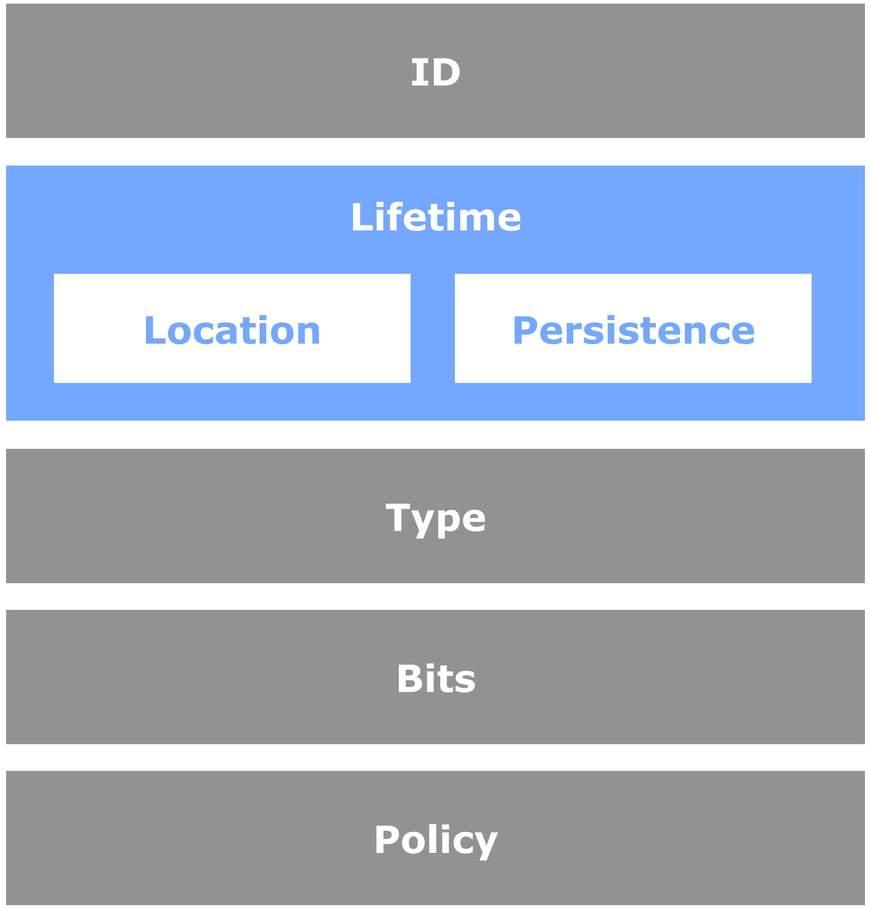
Key Management

- Keys need to be created before they can be used
- Key attributes hold metadata (location, policies, etc.)
- Cannot be changed without destroying key

psa_key_attributes_t

Key Attributes
Lifetime

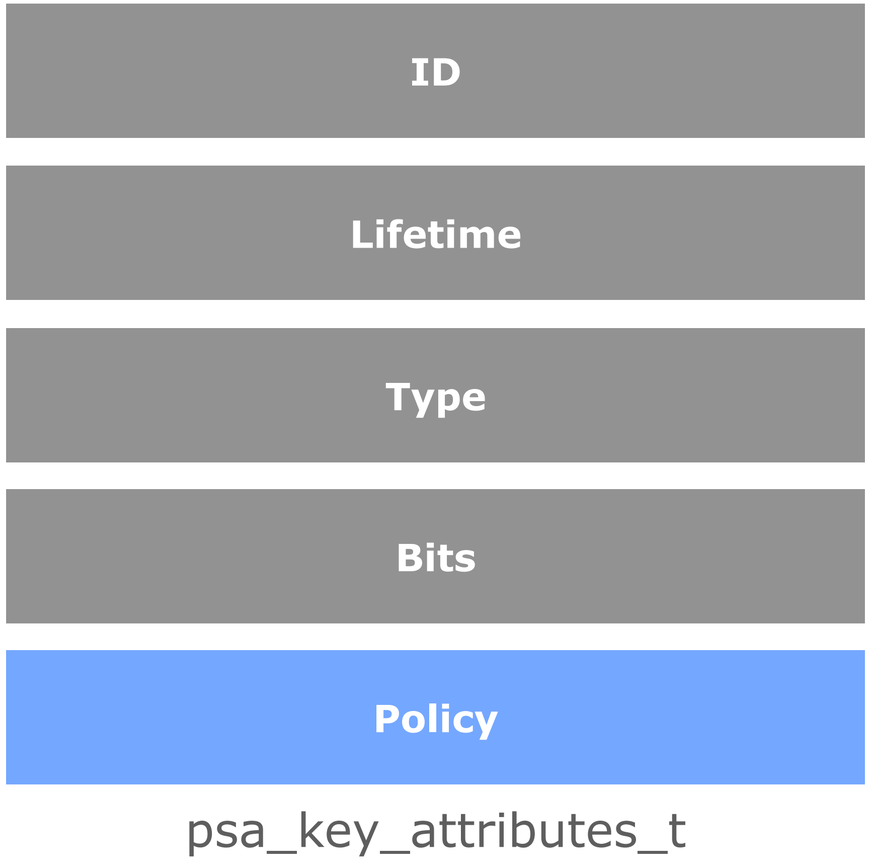
- Location:
  - Key storage location
  - Local volatile, persistent memory or protected hardware storage
- Persistence:
  - Volatile, persistent, read-only



psa_key_attributes_t

Key Attributes
Policy

- Permitted algorithms
- Usage Flags:
  - Encrypt, Decrypt
  - Sign, Verify
  - Export, Copy, Cache
  - Derivation



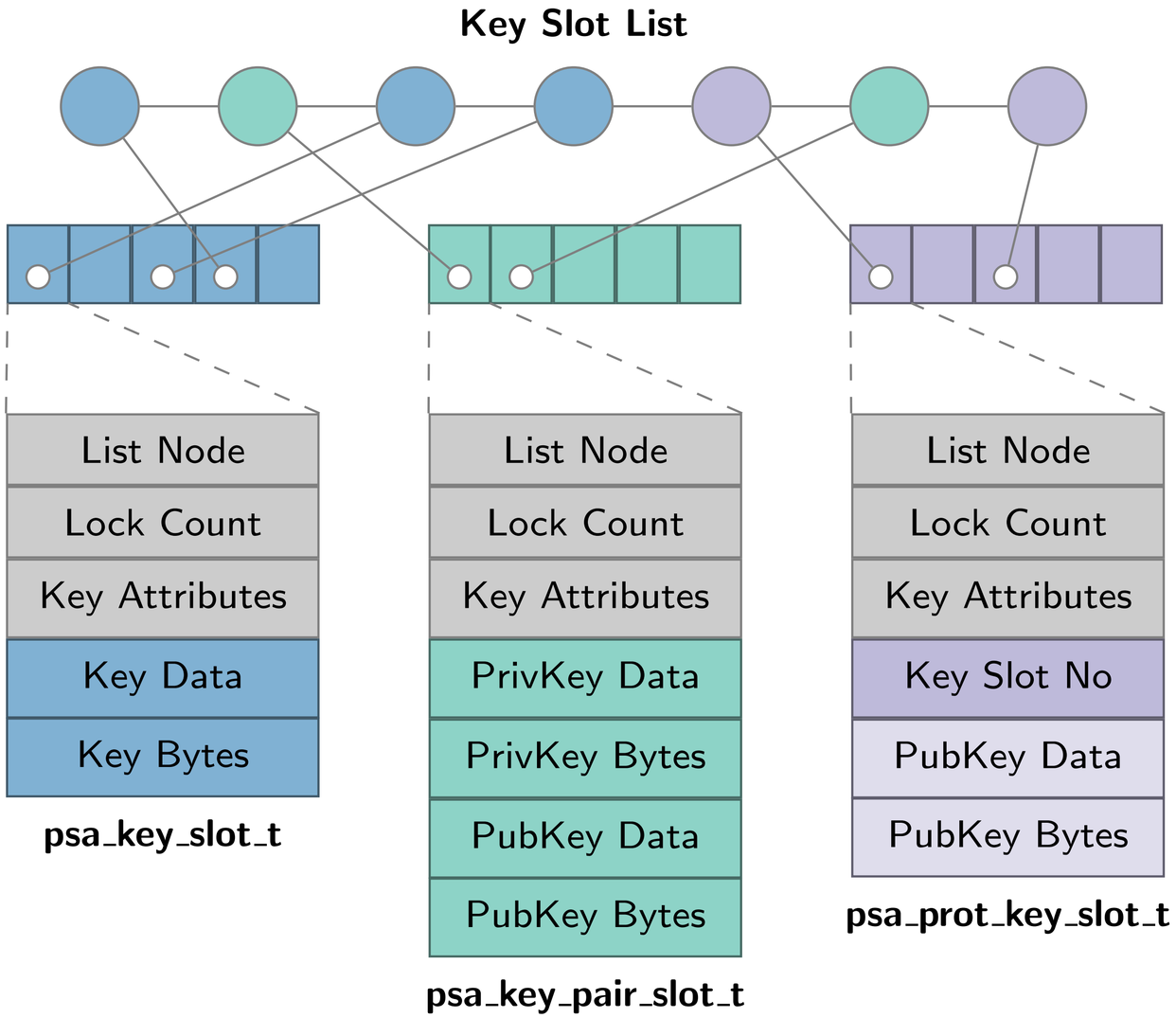
psa_key_attributes_t

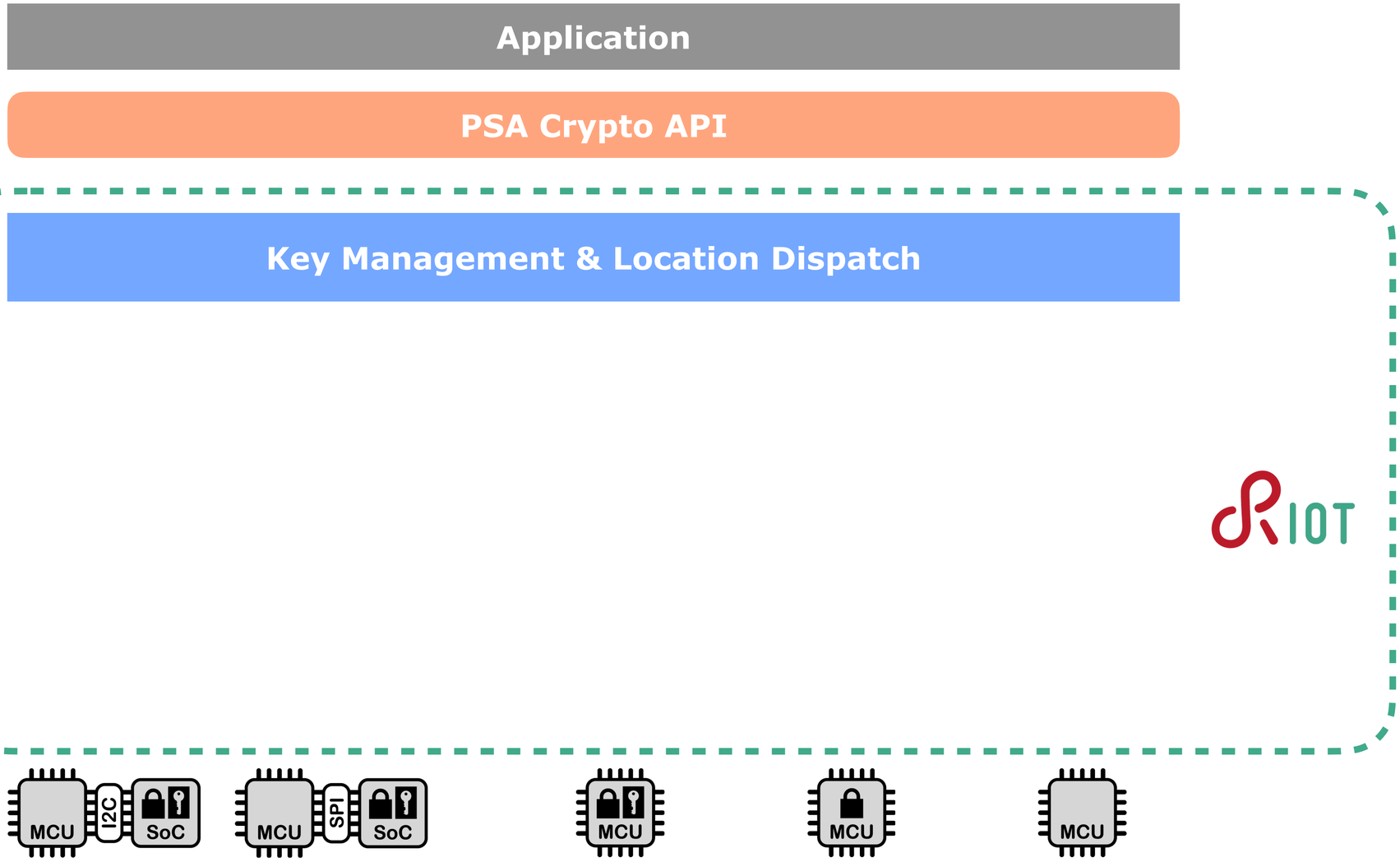
Key Slot Management

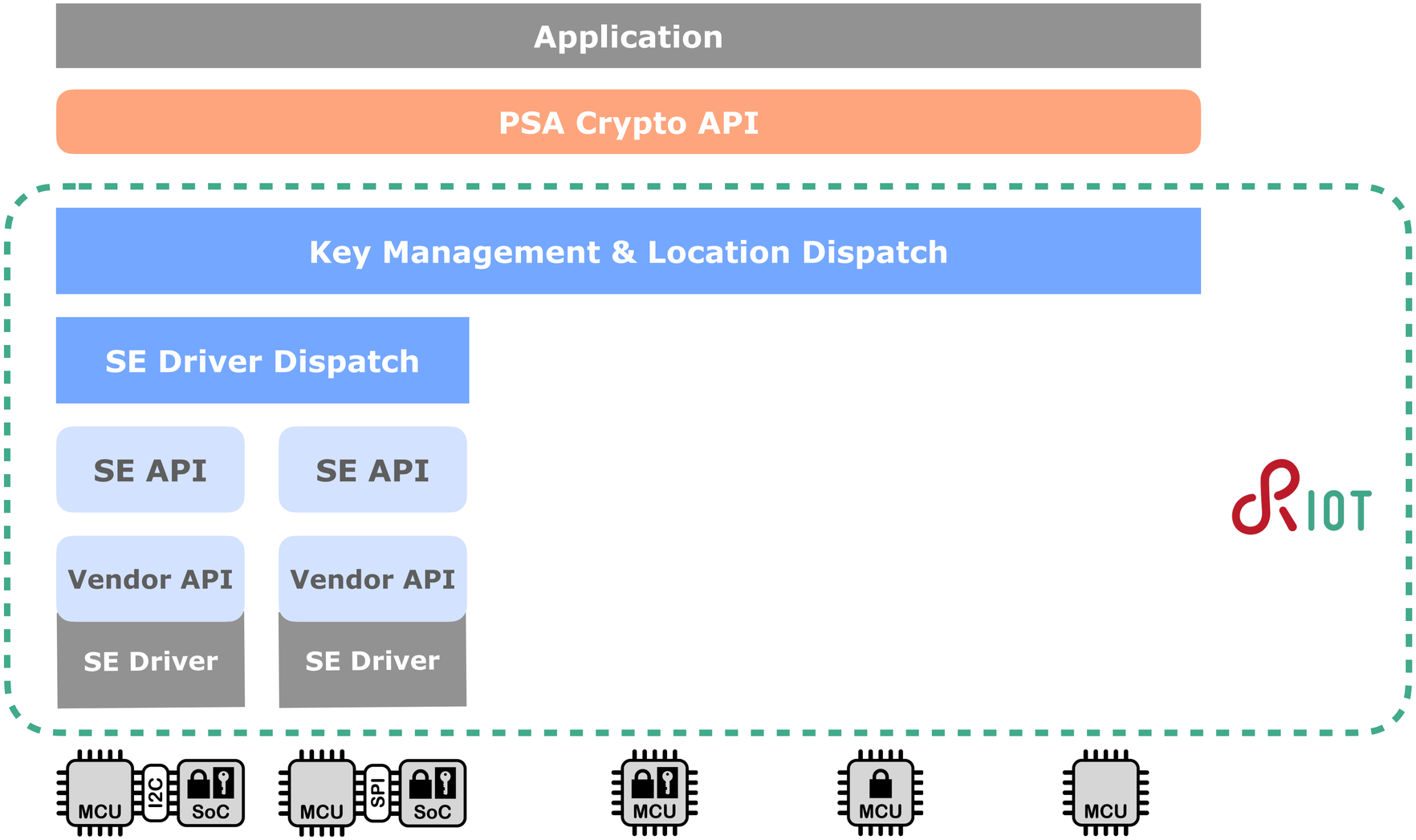
- Keys and key references are stored in virtual key slots
- Key sizes largely vary (16 bytes for AES-128, several hundred bytes for RSA)
- Flexible slot sizes needed

Key Slots

- Three types:
  - Single keys and unstructured data
  - Asymmetric key pairs
  - Reference to protected key

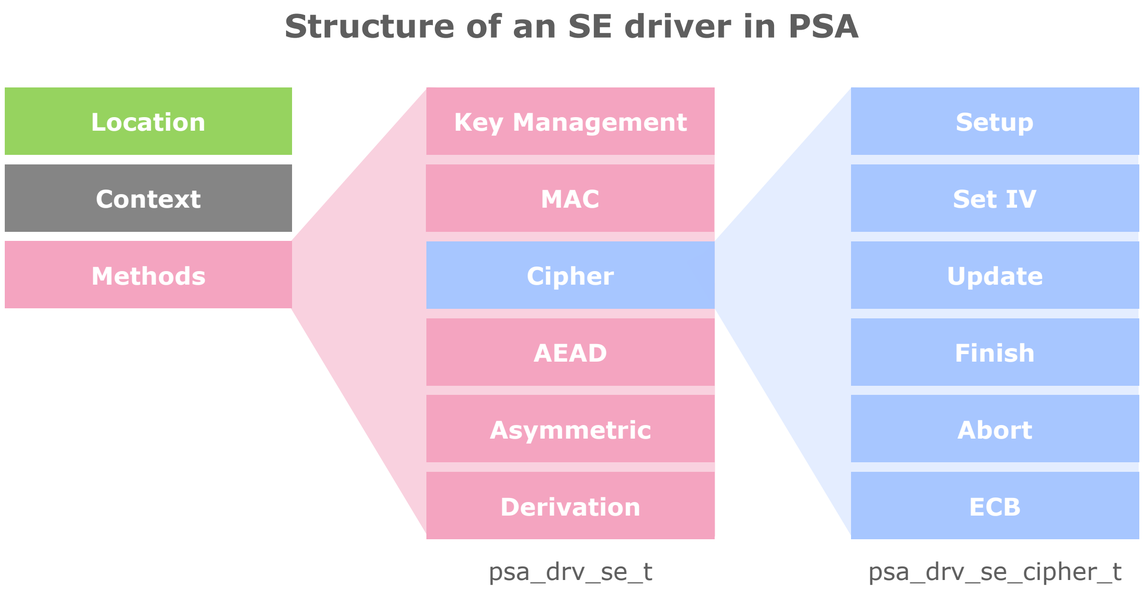


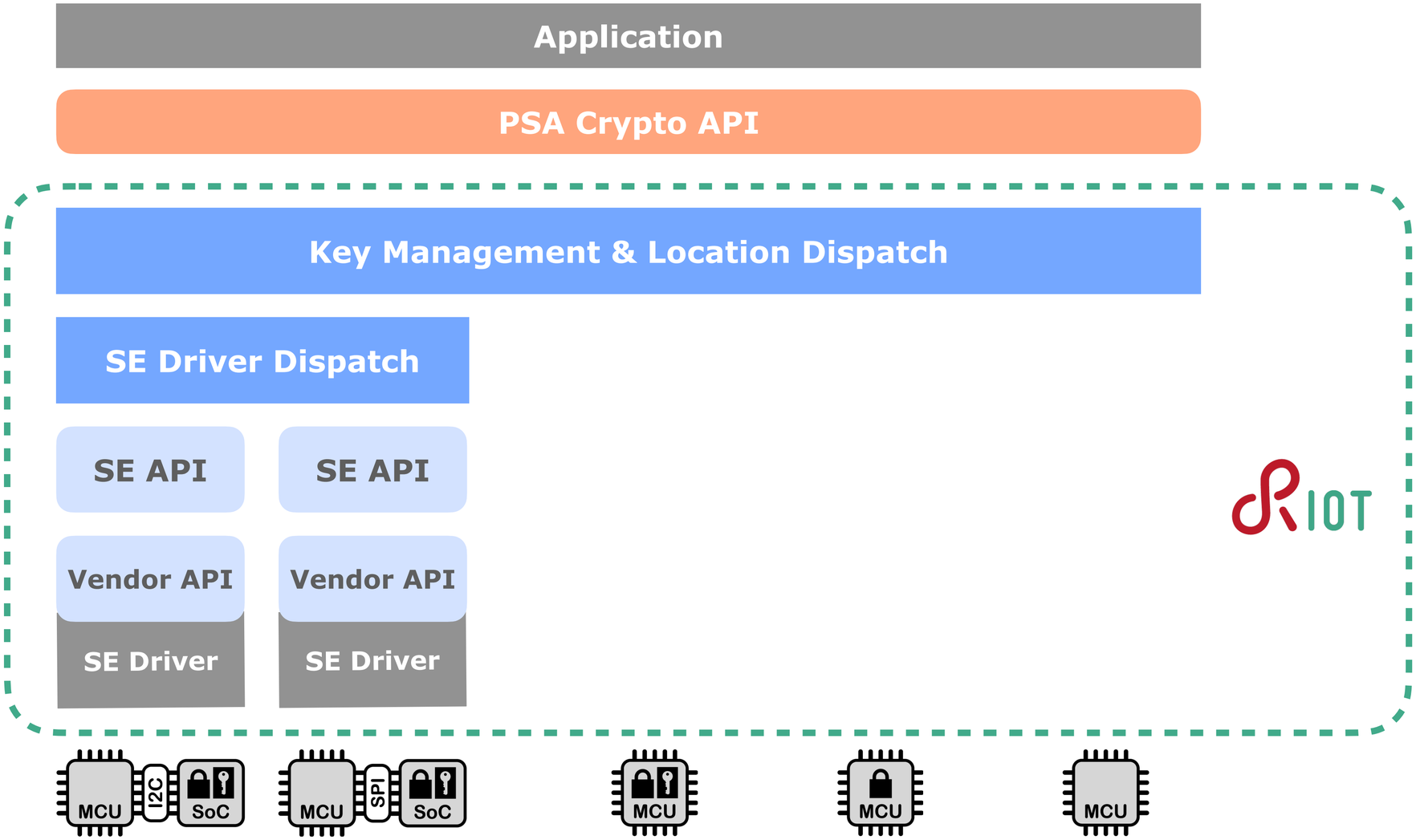

Application
PSA Crypto API
Key Management & Location Dispatch
RIOT
MCU
I2C
SoC
MCU
SPI
SoC
MCU
MCU
MCU

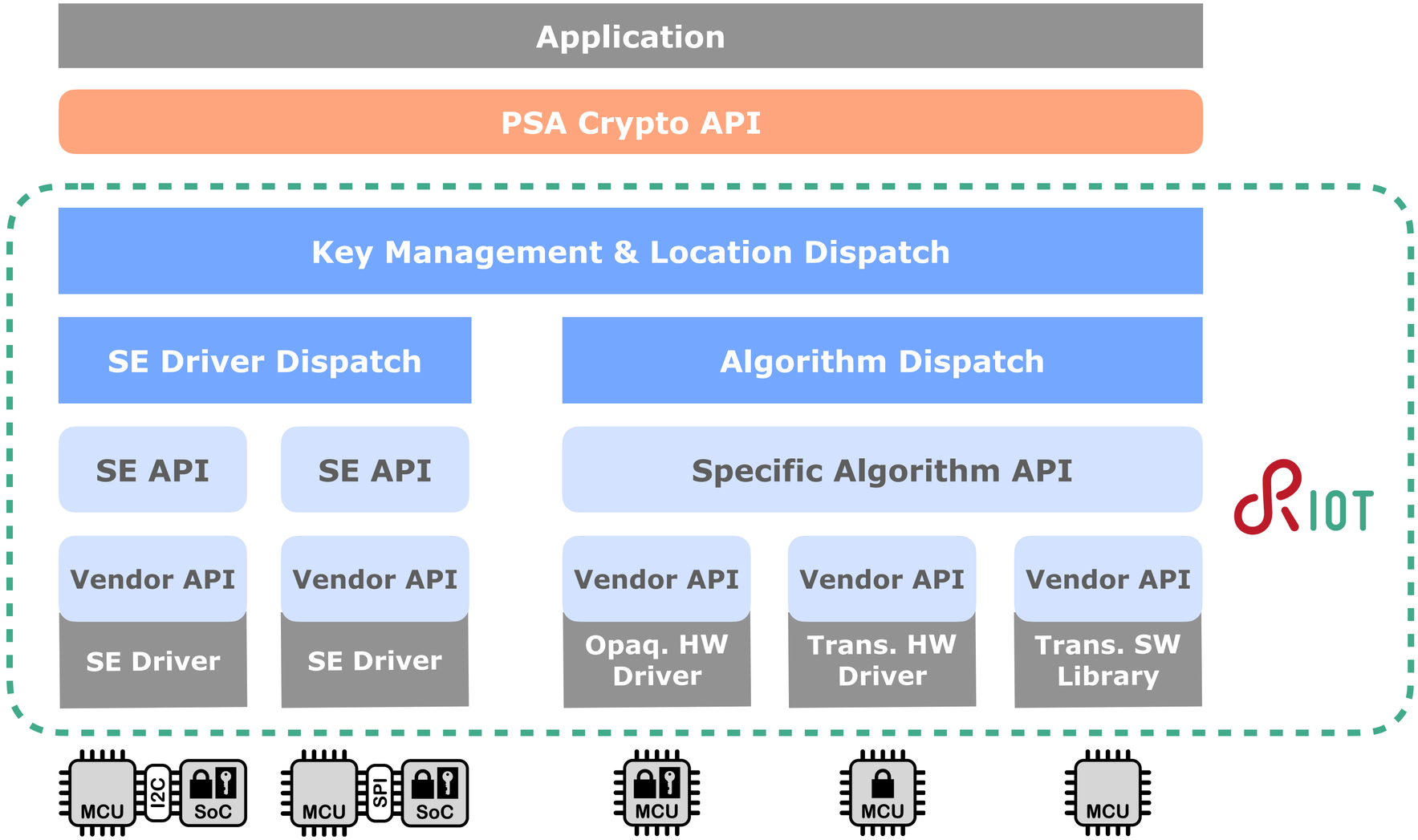

Application
PSA Crypto API
Key Management & Location Dispatch
SE Driver Dispatch
SE API
SE API
Vendor API
Vendor API
SE Driver
SE Driver
RIOT
MCU
I2C
SoC
MCU
SPI
SoC
MCU
MCU
MCU

Secure Element Handling

- Multiple SEs for runtime selection
- SE devices with persistent `location` value
- Device drivers must implement generic SE interface
- At startup:
  - OS function **`auto_init`** registers devices with SE management module
  - SE module stores `method pointers`, `location` and context in global driver list
- At runtime: drivers are dynamically retrieved from list

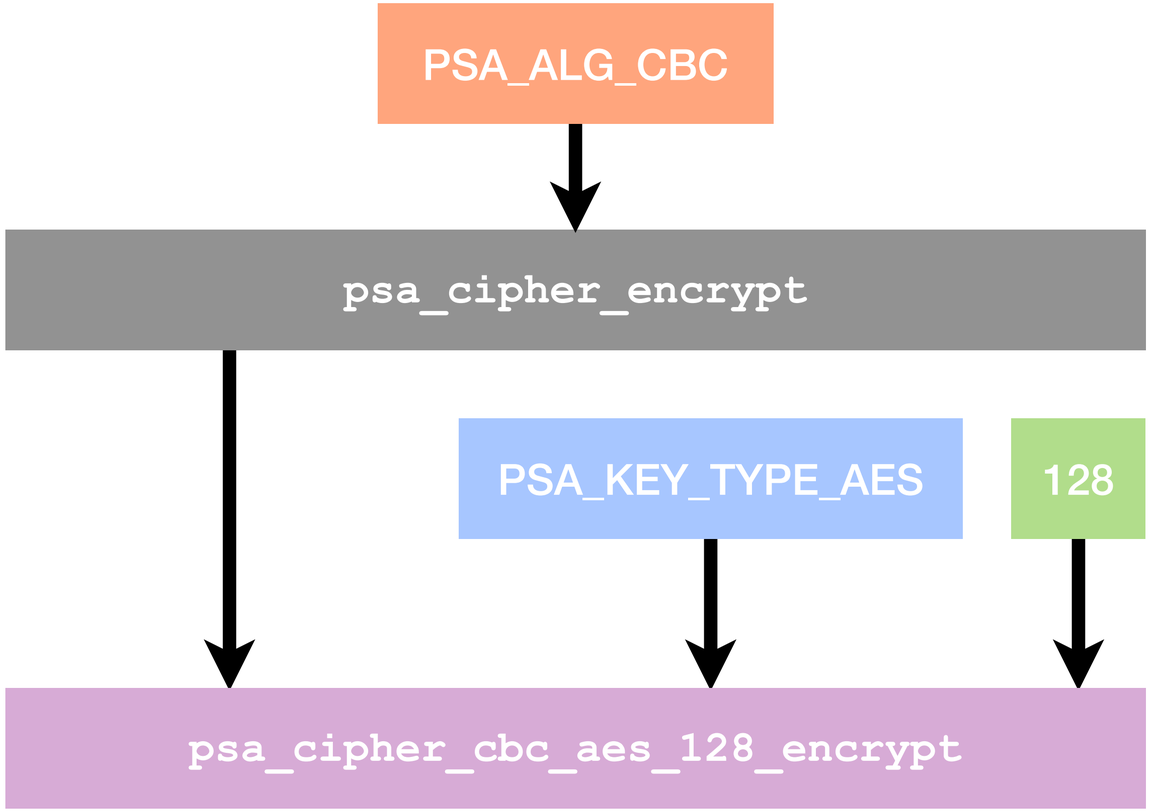



Application
PSA Crypto API
Key Management & Location Dispatch
SE Driver Dispatch
SE API
SE API
Vendor API
Vendor API
SE Driver
SE Driver
RIOT
MCU
I2C
SoC
MCU
SPI
SoC
MCU
MCU
MCU


Application
PSA Crypto API
Key Management & Location Dispatch
SE Driver Dispatch
Algorithm Dispatch
SE API
SE API
Specific Algorithm API
Vendor API
Vendor API
Vendor API
Vendor API
Vendor API
SE Driver
SE Driver
Opaq. HW Driver
Trans. HW Driver
Trans. SW Library
RIOT
MCU
I2C
SoC
MCU
SPI
SoC
MCU
MCU
MCU

Algorithm Dispatcher

- Allows for fine grained backend combinations
- Maps `algorithm`, `key type` and `key size` to `specific algorithm API`
- Drivers and libraries implement algorithm specific API



Kconfig for Backend Configuration

What is Kconfig?

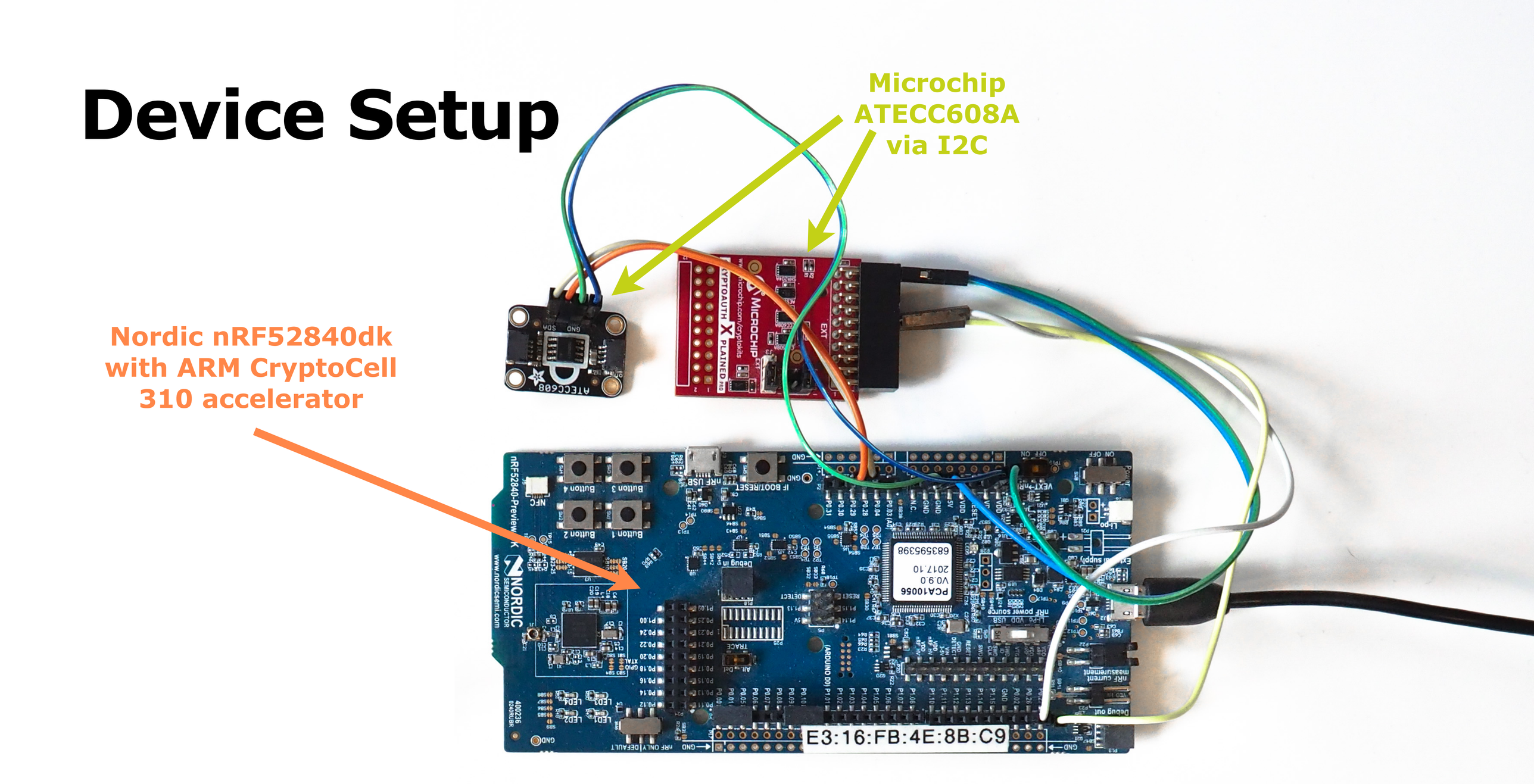
- Selection based system configuration
- Select build-time options to enable/disable features
- Specified default selections or auto-selection in case of predefined conditions

Modeling Crypto Hardware Features

- PSA module defines configuration options (e.g. **PSA_CIPHER_AES_128_CBC, PSA_KEY_SLOT_COUNT**)
- CPUs define hardware capabilities (e.g. **HAS_PERIPH_CIPHER_AES_128_CBC**)
- Libraries and drivers define available options (e.g. **RIOT_CIPHER_AES_128_CBC**)
- User selects crypto operations to use and number of needed keys
- Build system automatically selects hardware implementation if available - otherwise fallback to software

Evaluation

Device Setup



Measured Resources

Operations: HMAC SHA-256, AES-128 CBC, ECDSA (P-256)

Backends: RIOT Software Modules, CryptoCell 310, ATECC608A

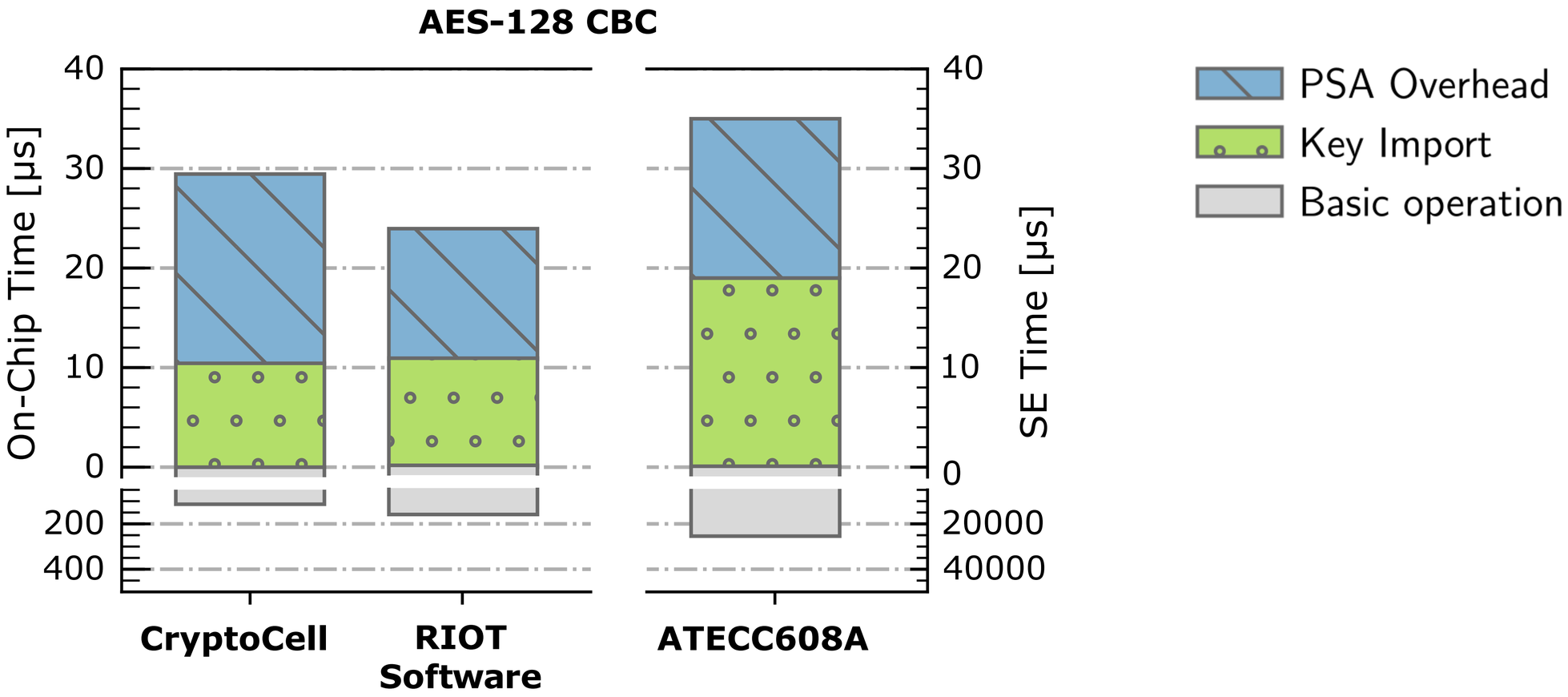
Processing Time

- Complete processing of API functions and internal driver calls

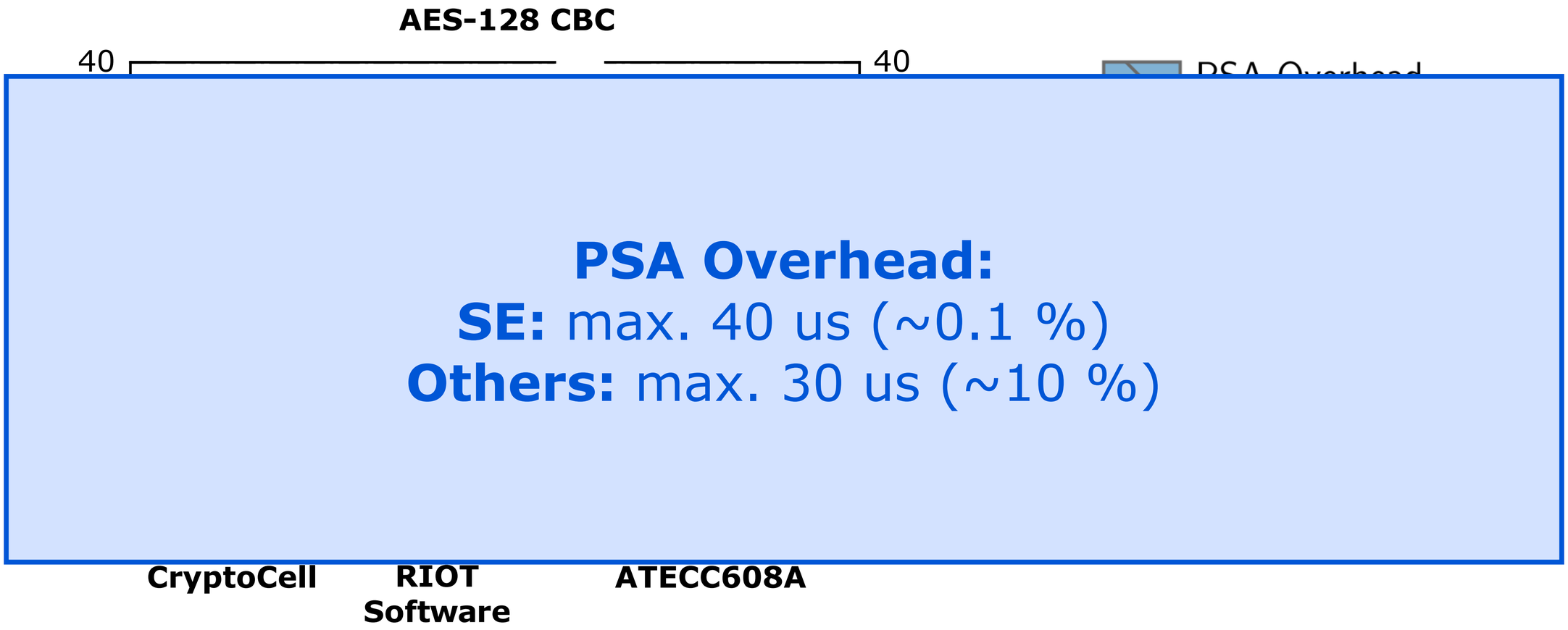
Memory Overhead

- Accumulation of crypto related objects in ELF file

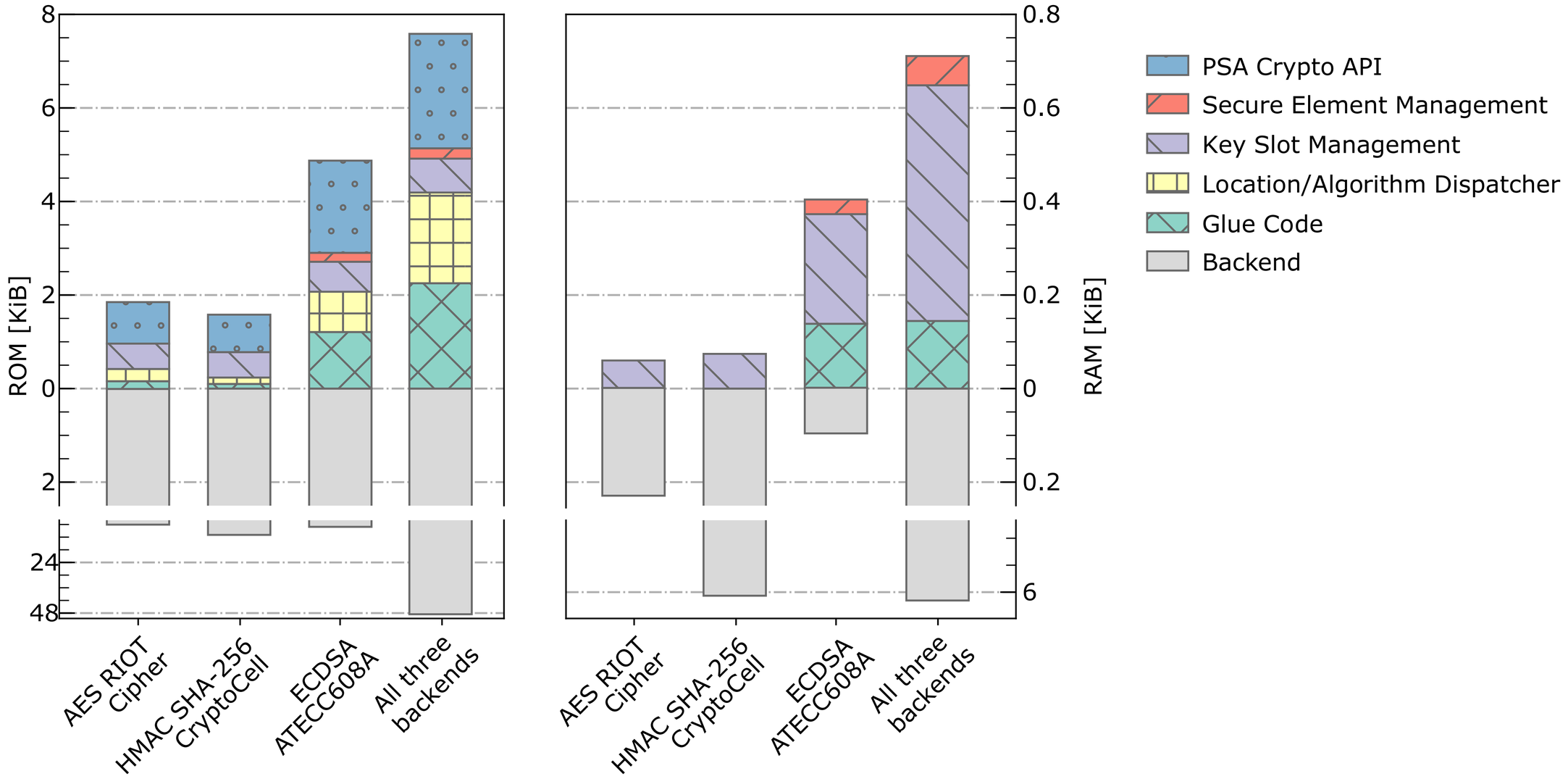
Processing Time



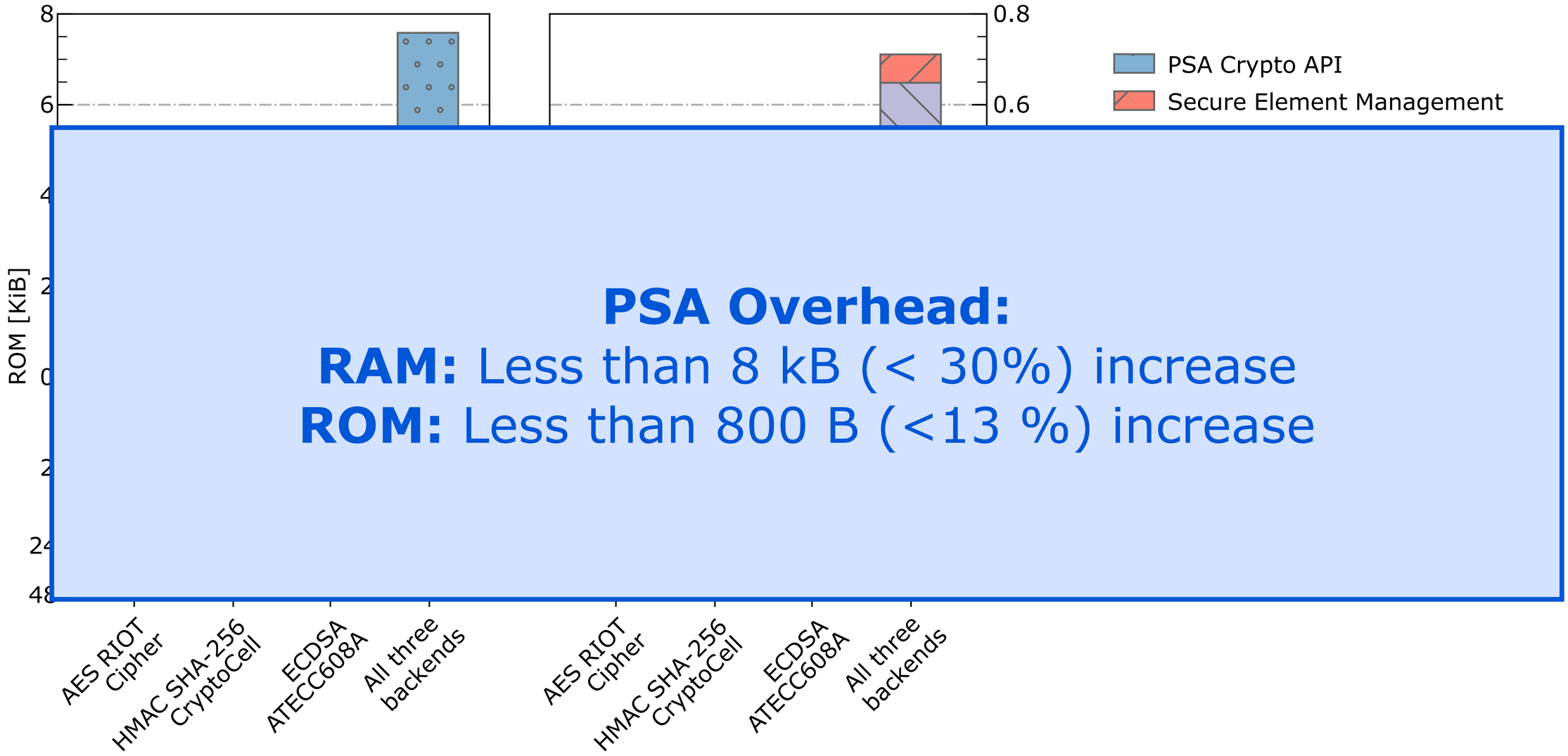
Processing Time



Memory Overhead



Memory Overhead

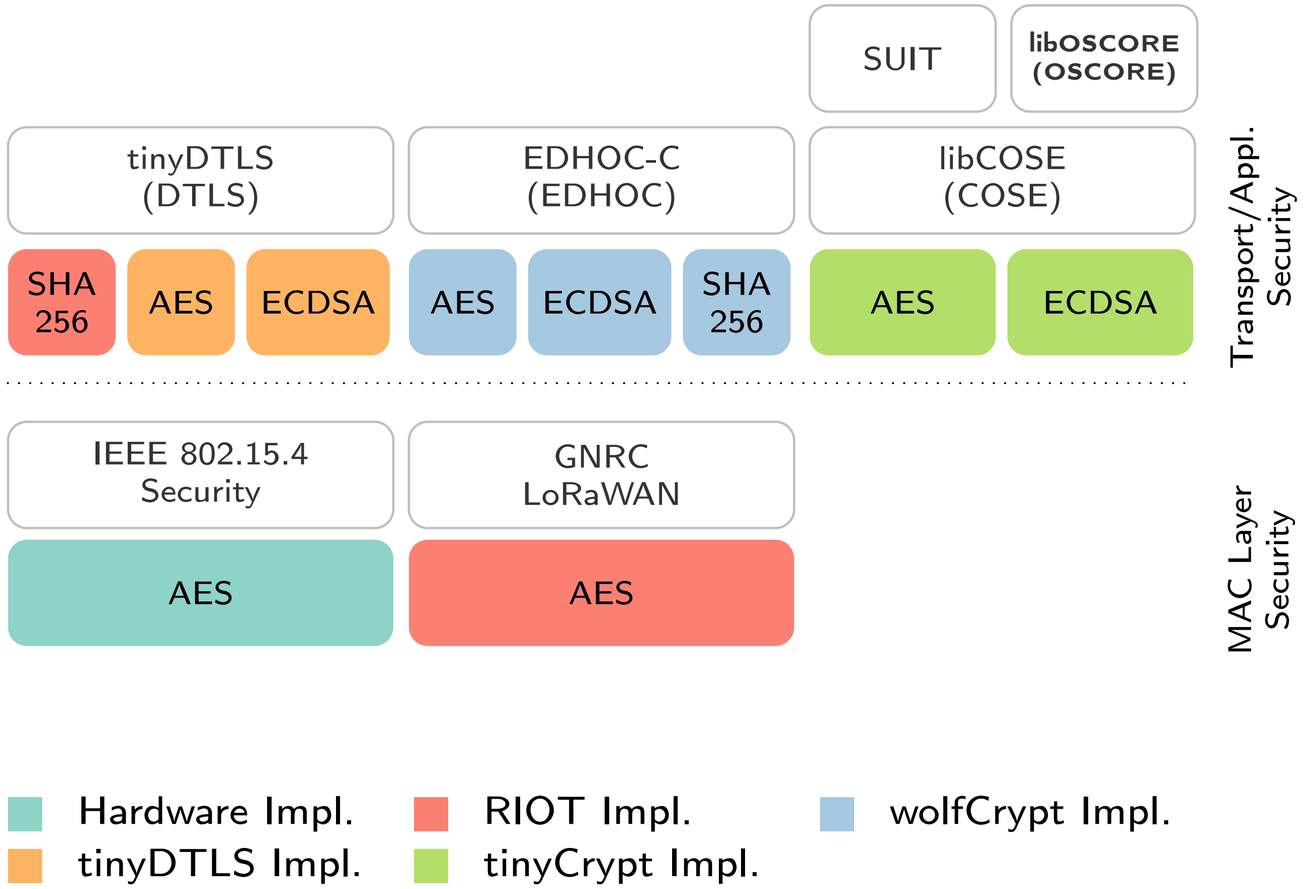


PSA Overhead:

RAM: Less than 8 kB (< 30%) increase

ROM: Less than 800 B (<13 %) increase

Code Deduplication
Secure Protocol Stack in RIOT

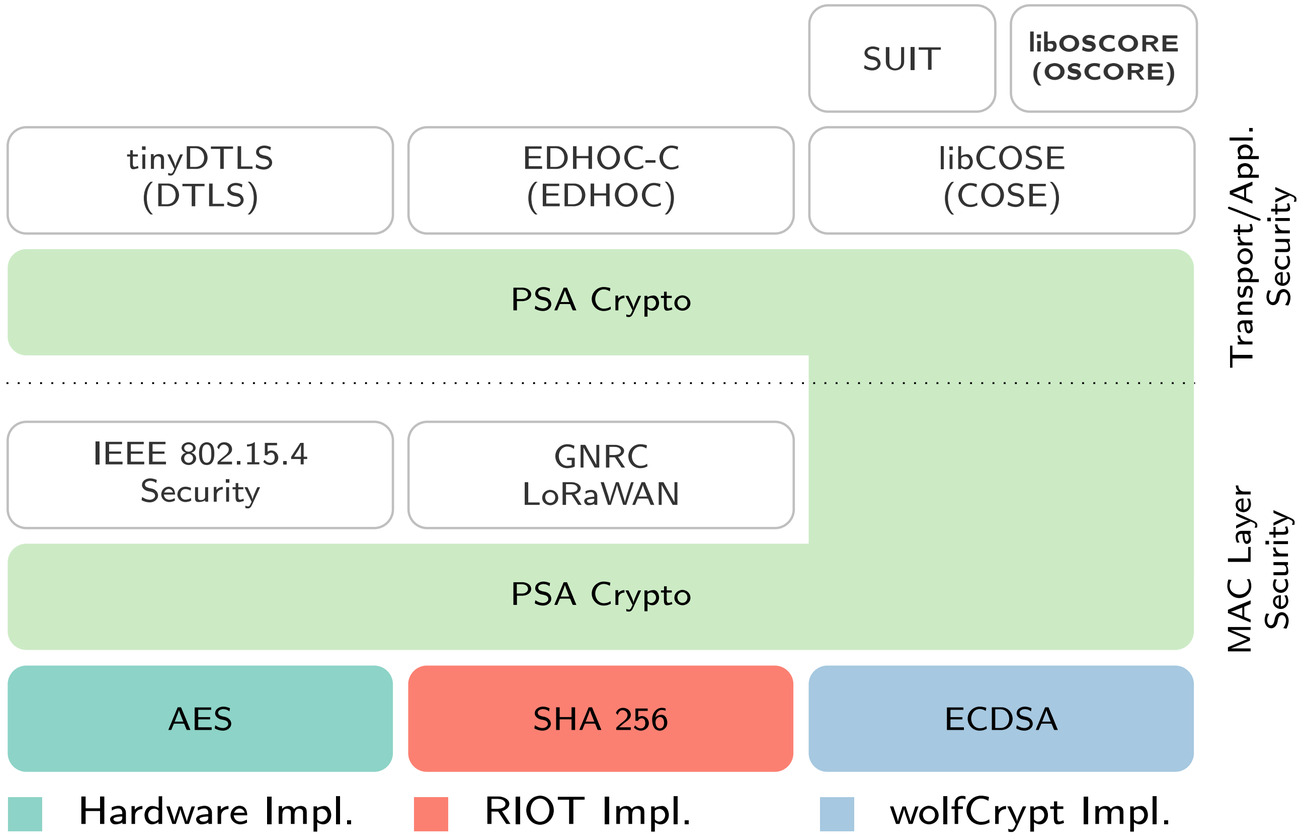


Crypto Code Size:

26.5 kB

Code Deduplication

Example: Secure Protocol Stack in RIOT



Crypto Code Size:

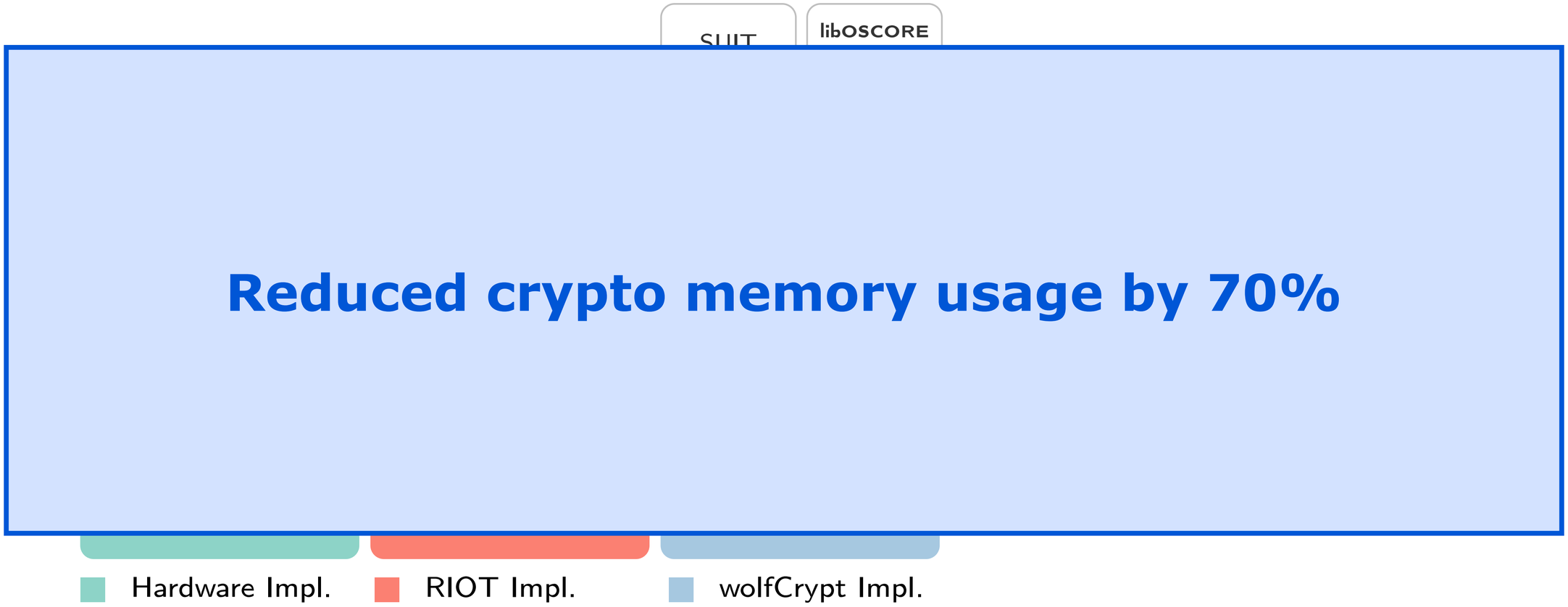
7 kB

PSA Code Size:

~ 8 kB

Code Deduplication

Example: Secure Protocol Stack in RIOT



Conclusion

- Enhanced RIOT with OS level crypto API with configurable backends
- ID based key management supports backends with and without protected key storage
- Kconfig build system:
  - Developers choose functionality, not implementation
  - System automatically builds best configuration
- Low overhead in processing time and memory

Outlook

- Step-by-step integration of additional backends in RIOT
- Support persistent key storage and minimal set of operations on all platforms
- Integrate Trusted Execution Environments

Contact

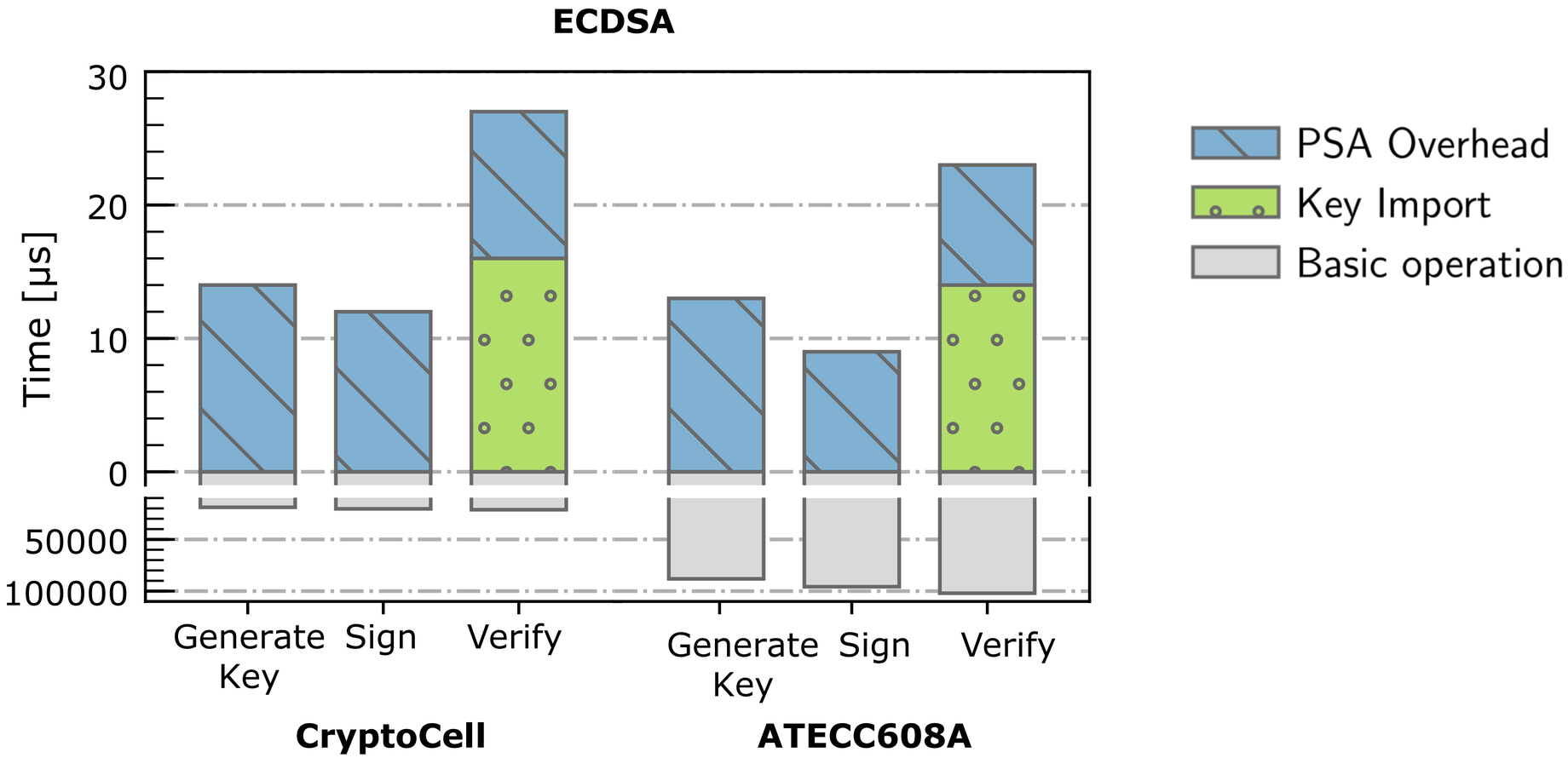
https://www.riot-os.org/

https://github.com/RIOT-OS/RIOT/pull/18547

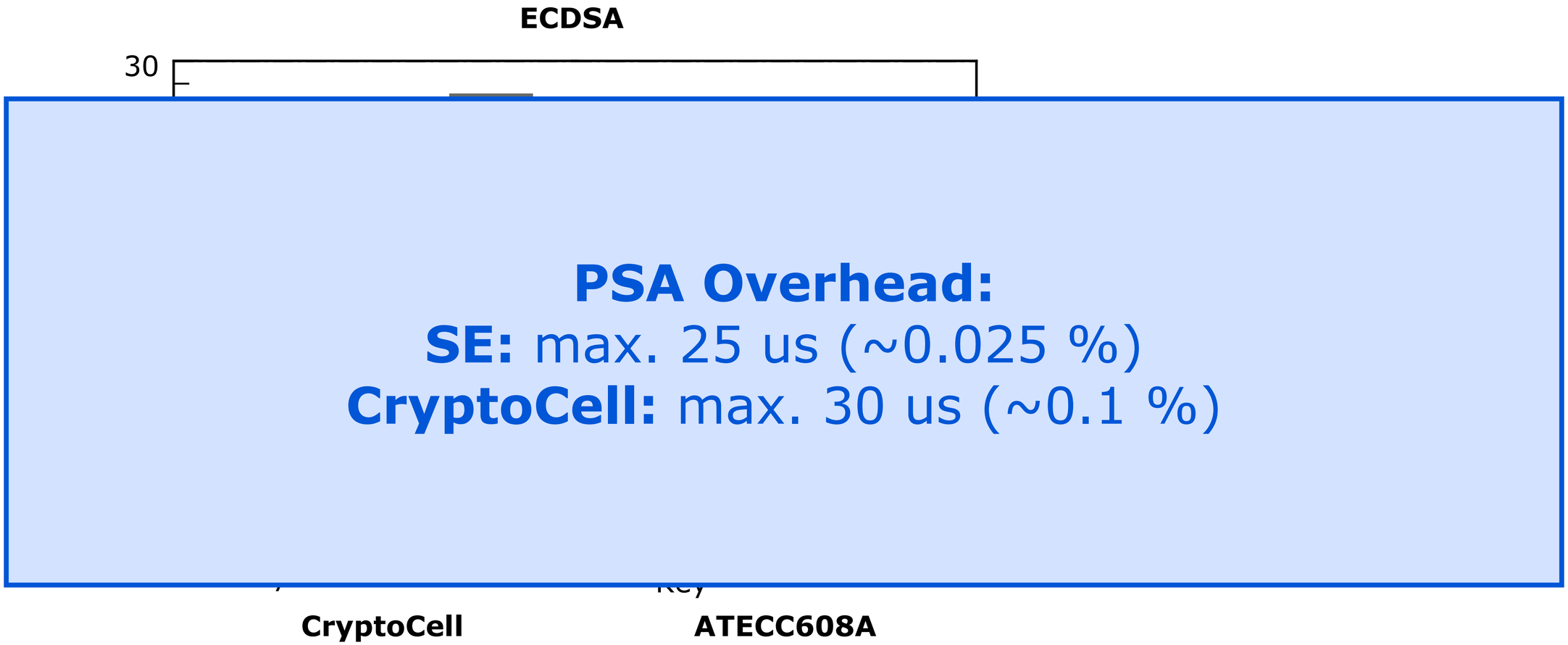
http://inet.haw-hamburg.de/

lena.boeckmann@haw-hamburg.de
peter.kietzmann@haw-hamburg.de
leandro.lanzieri@haw-hamburg.de
t.schmidt@haw-hamburg.de
m.waehlisch@fu-berlin.de

Processing Time



Processing Time

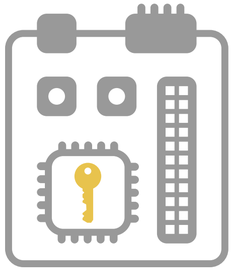


Modeling Crypto Hardware Features

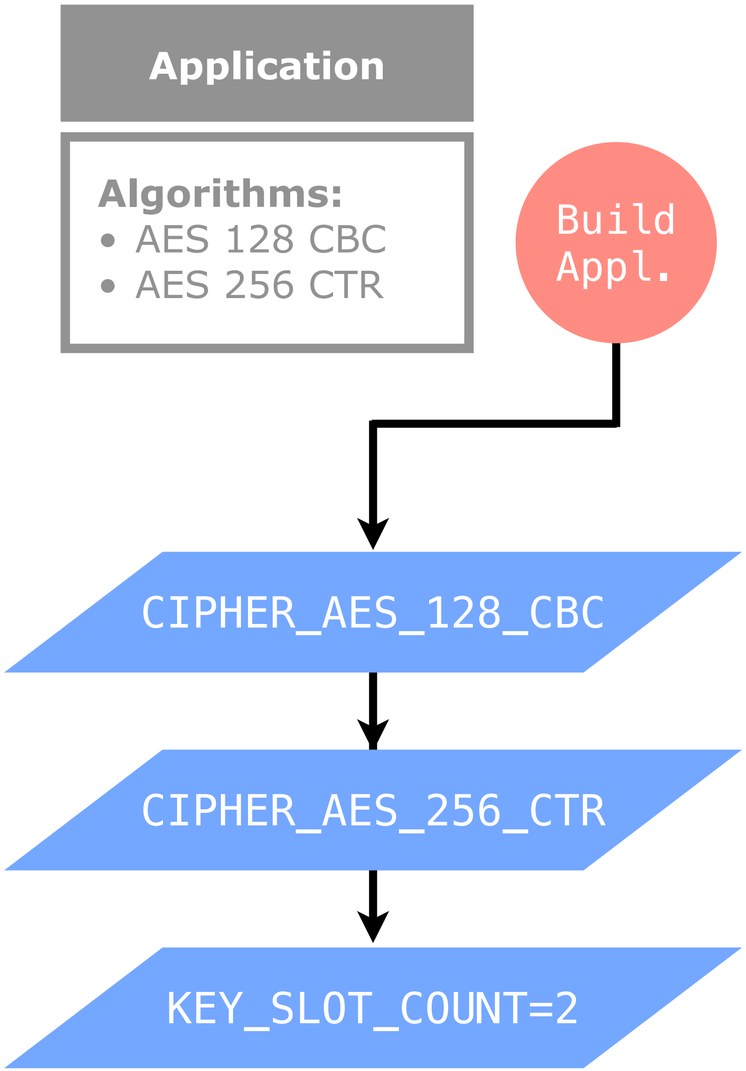
Application

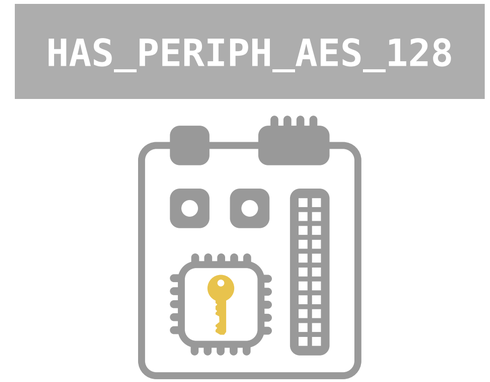
Algorithms:
- AES 128 CBC
- AES 256 CTR

`HAS_PERIPH_AES_128`

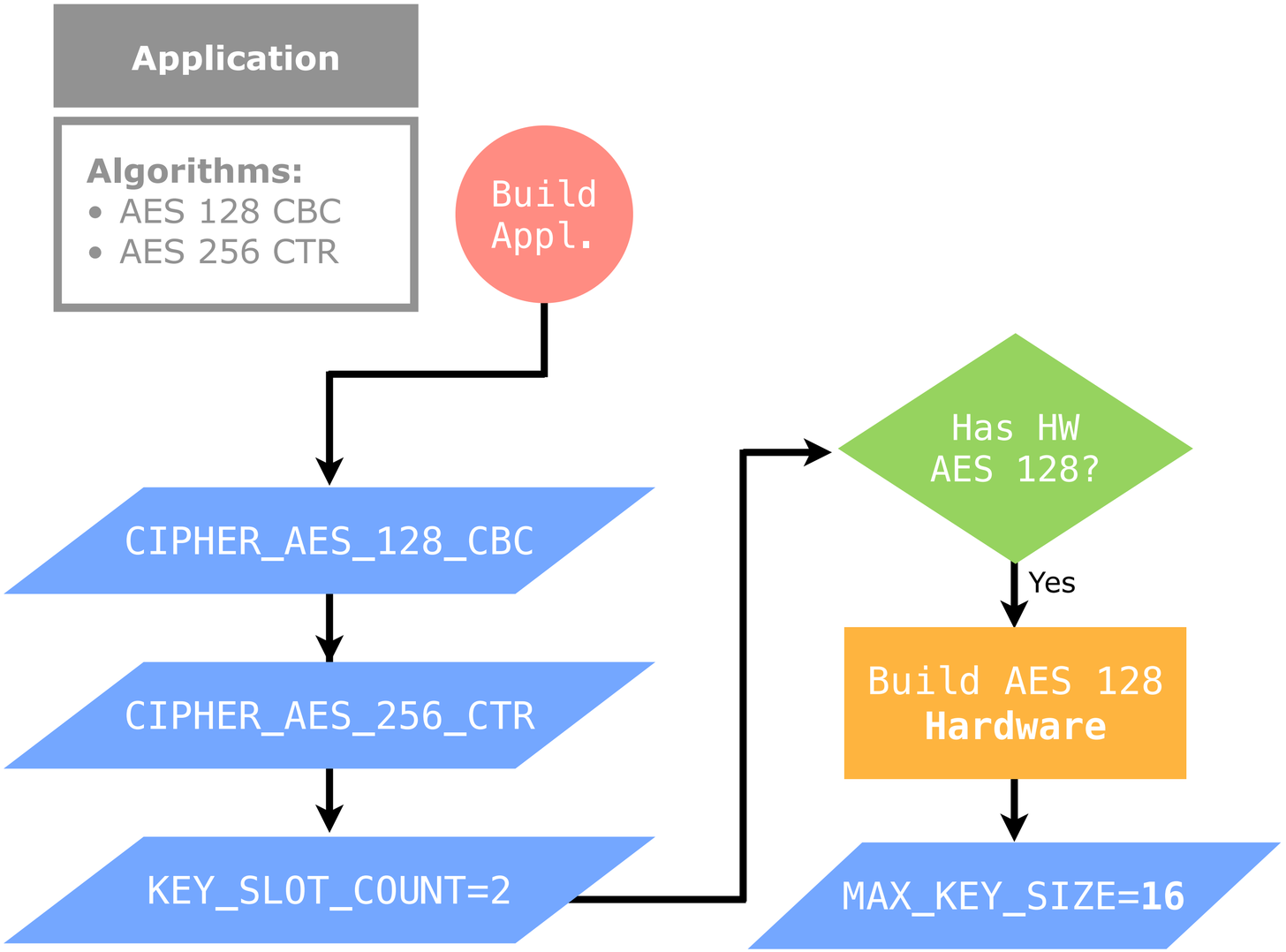


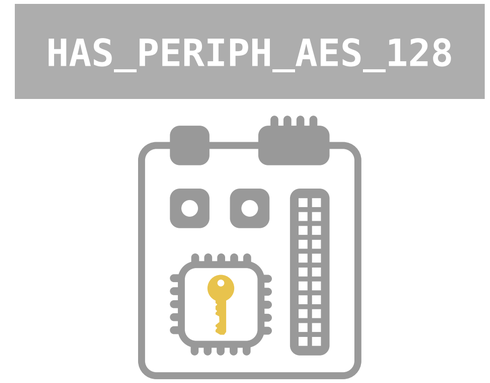
Modeling Crypto Hardware Features





Modeling Crypto Hardware Features





Modeling Crypto Hardware Features