HAW
HAMBURG

Integration of the PSA Crypto API with Configurable Backends in RIOT

Lena Boeckmann
lena.boeckmann@haw-hamburg.de
27.06.2022

Cryptography in the IoT

- > 50 billion IoT devices expected by 2025[1]
- Growing threat potential and increasing number of attacks
- Cryptography plays large role in securing IoT systems

[1] Christopher Bellman, Paul C. Von Oorschot. „Analysis, Implications, and Challenges of an Evolving Consumer IoT Security Landscape“. In: 2019 17th International Conference on Privacy, Security and Trust (PST). 2019

Cryptography in the IoT

- > 50 billion IoT devices expected by 2025[1]
- Growing threat potential and increasing number of attacks
- Cryptography plays large role in securing IoT systems

Problem

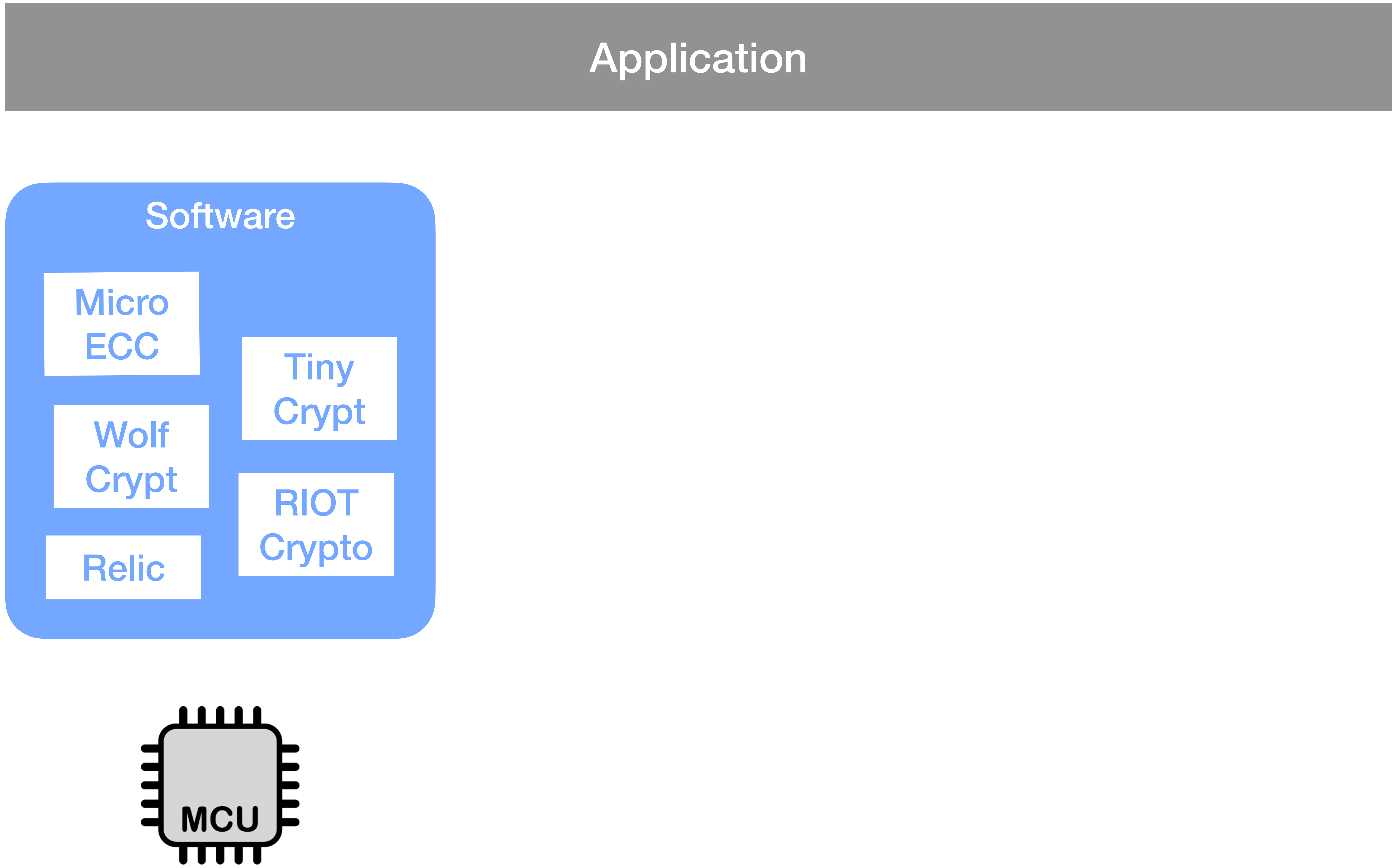
Crypto operations are inefficient and strain resources in constrained environments

[1] Christopher Bellman, Paul C. Von Oorschot. „Analysis, Implications, and Challenges of an Evolving Consumer IoT Security Landscape“. In: 2019 17th International Conference on Privacy, Security and Trust (PST). 2019

Cryptographic Backends in the IoT

Application

Cryptographic Backends in the IoT



Application

Software

Micro ECC

Tiny Crypt

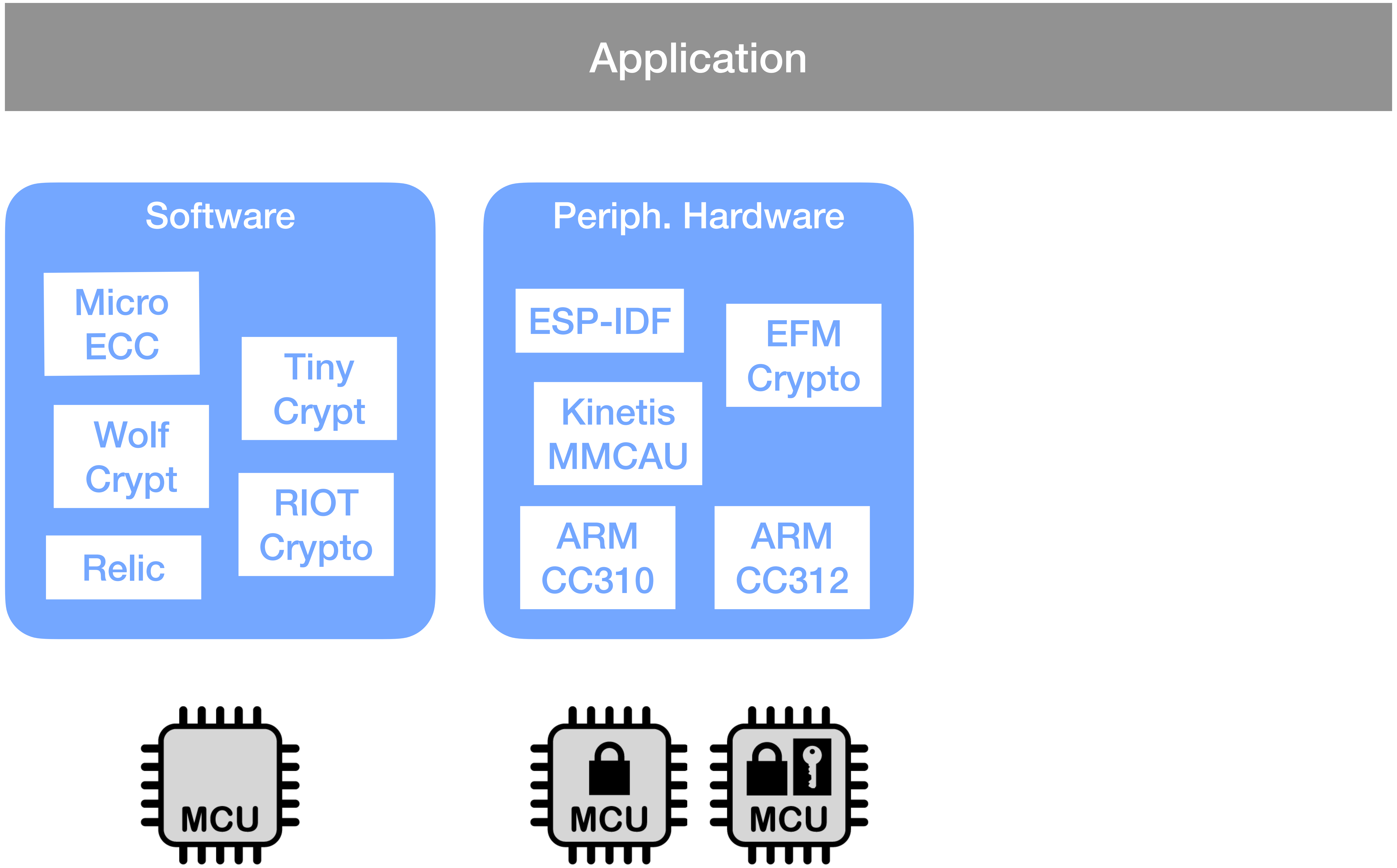
Wolf Crypt

RIOT Crypto

Relic

MCU

Cryptographic Backends in the IoT



Application

Software

Micro ECC

Tiny Crypt

Wolf Crypt

RIOT Crypto

Relic

Periph. Hardware

ESP-IDF

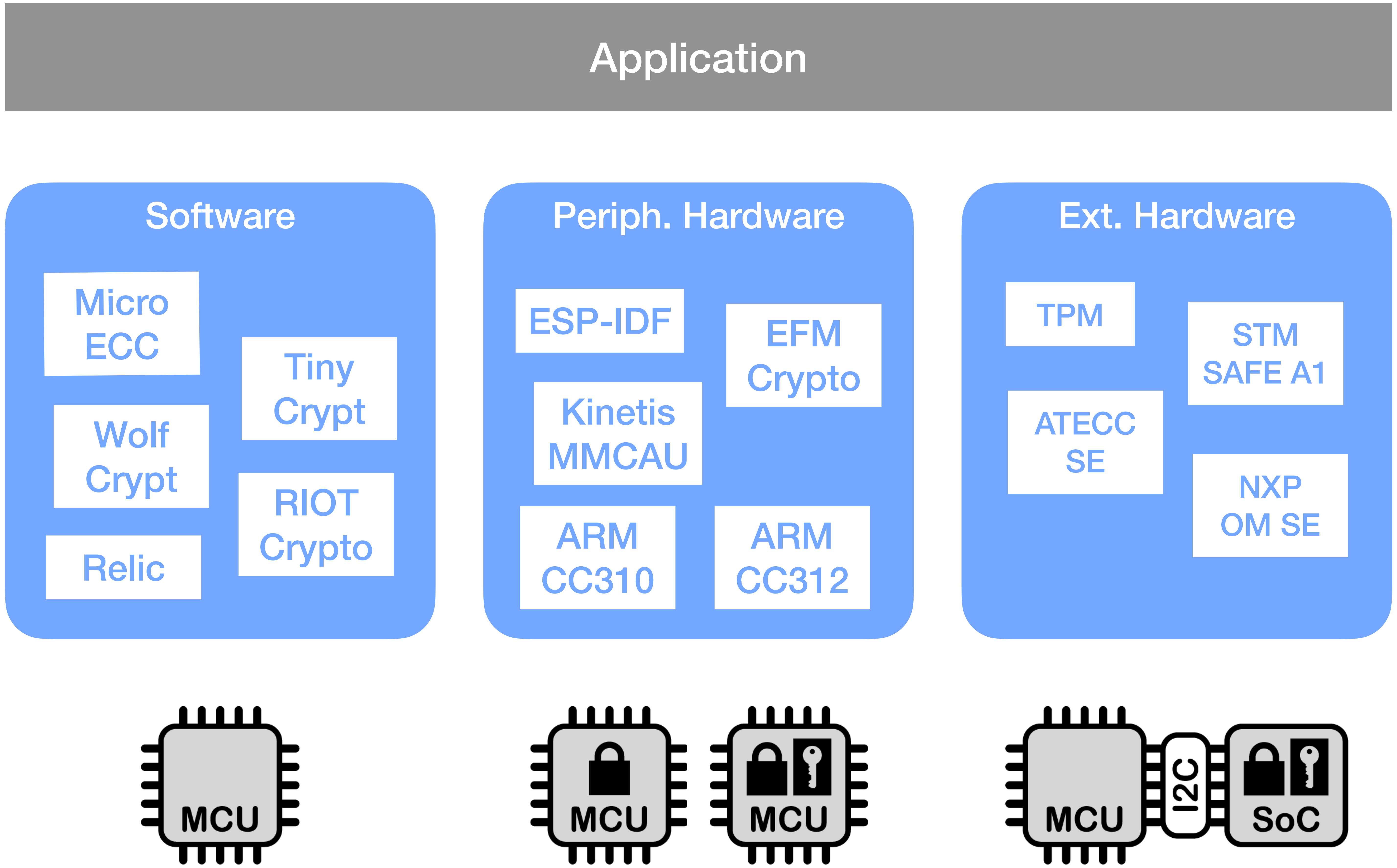
EFM Crypto

Kinetis MMCAU

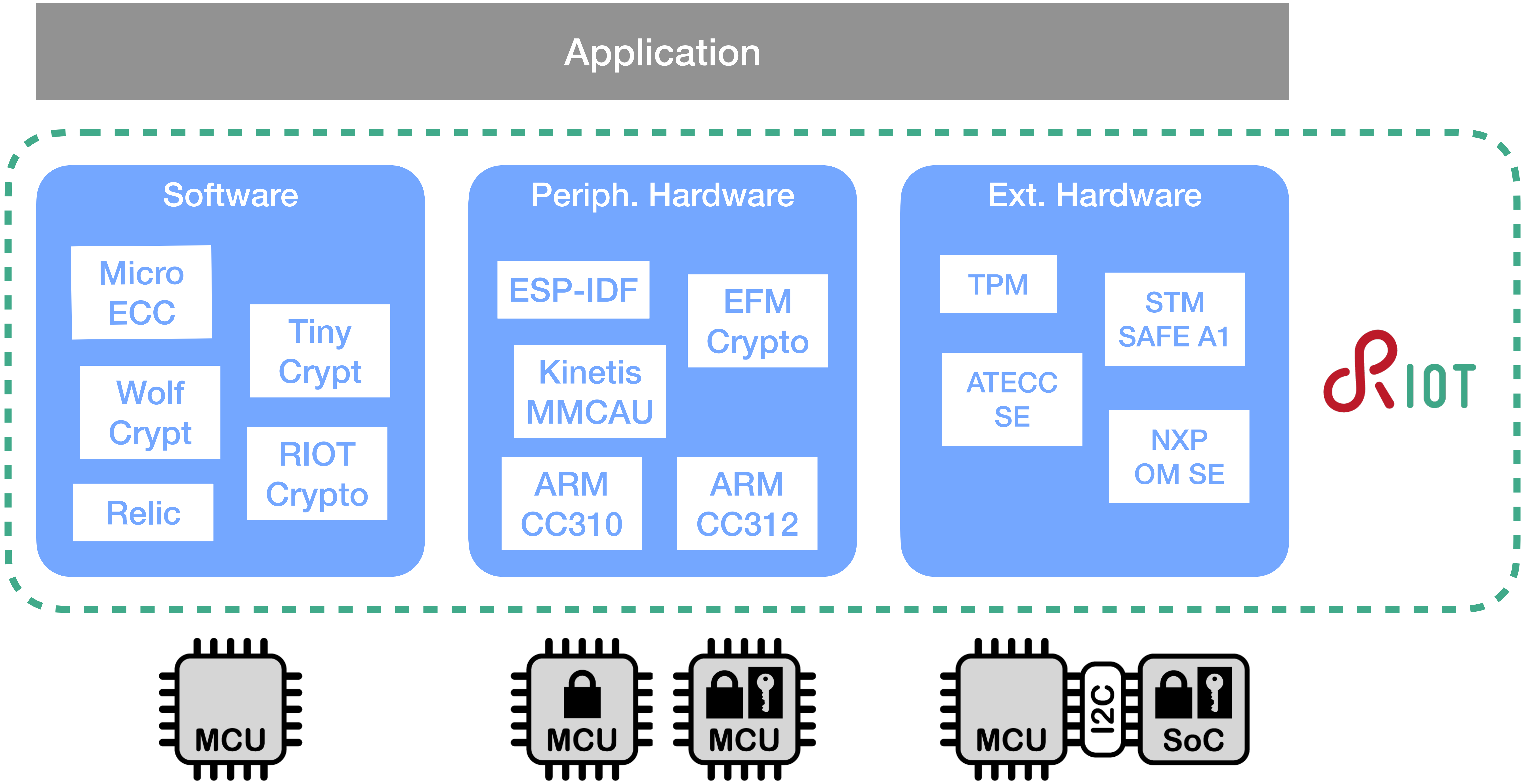
ARM CC310

ARM CC312

Cryptographic Backends in the IoT



Crypto Usage in RIOT



Application

Software

Micro ECC

Tiny Crypt

Wolf Crypt

RIOT Crypto

Relic

Periph. Hardware

ESP-IDF

EFM Crypto

Kinetis MMCAU

ARM CC310

ARM CC312

Ext. Hardware

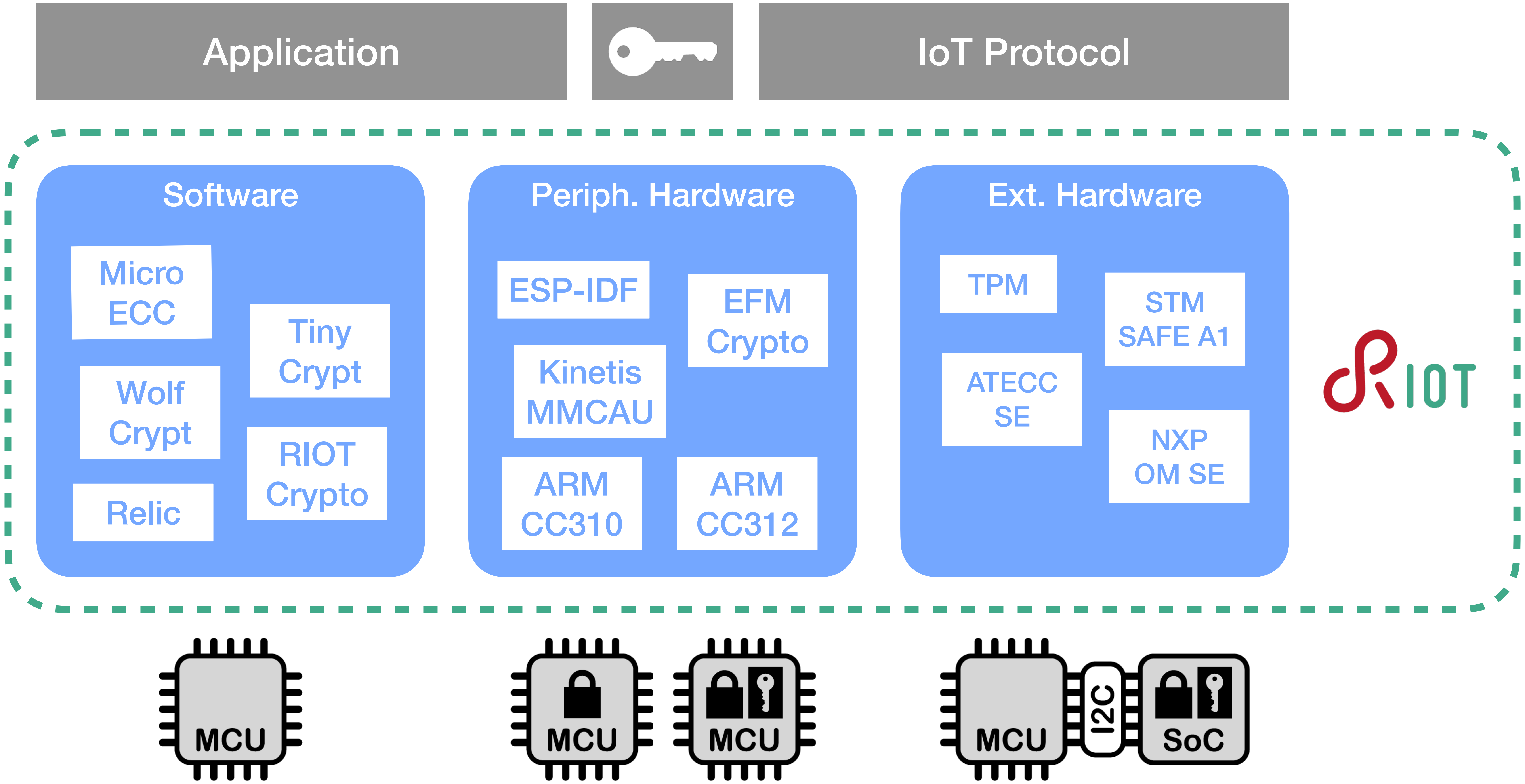
TPM

STM SAFE A1

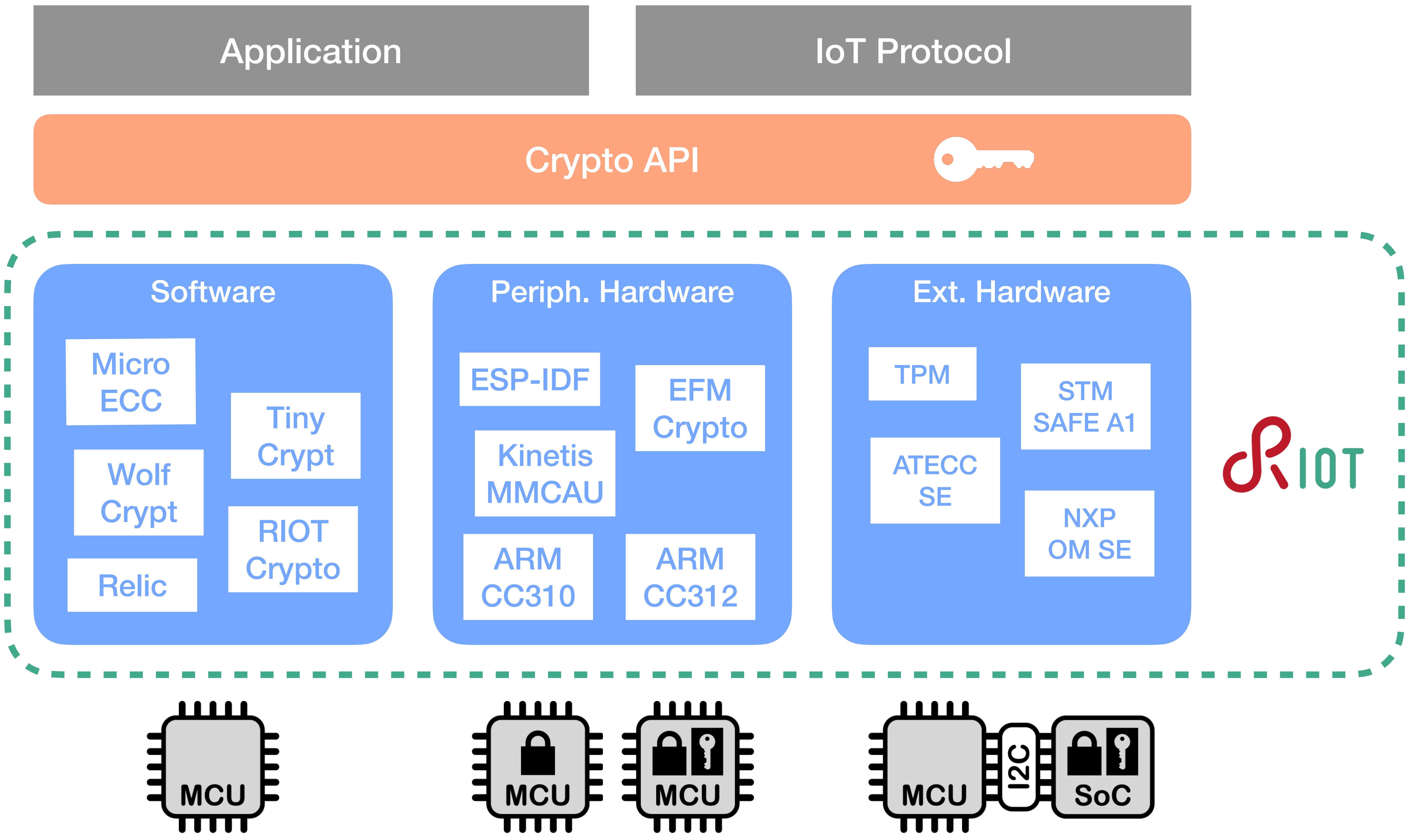
ATECC SE

NXP OM SE

Crypto Usage in RIOT



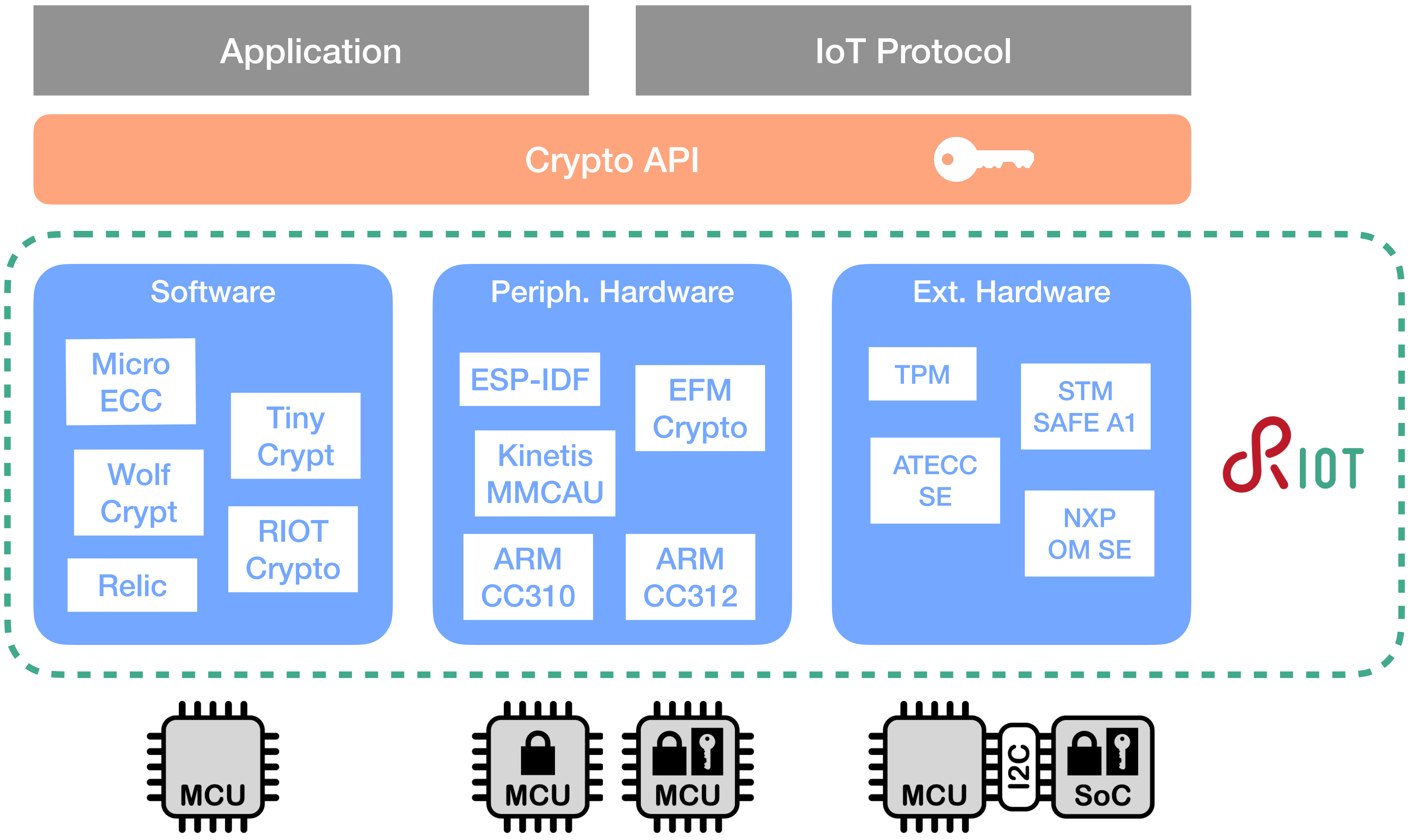
Crypto Usage in RIOT



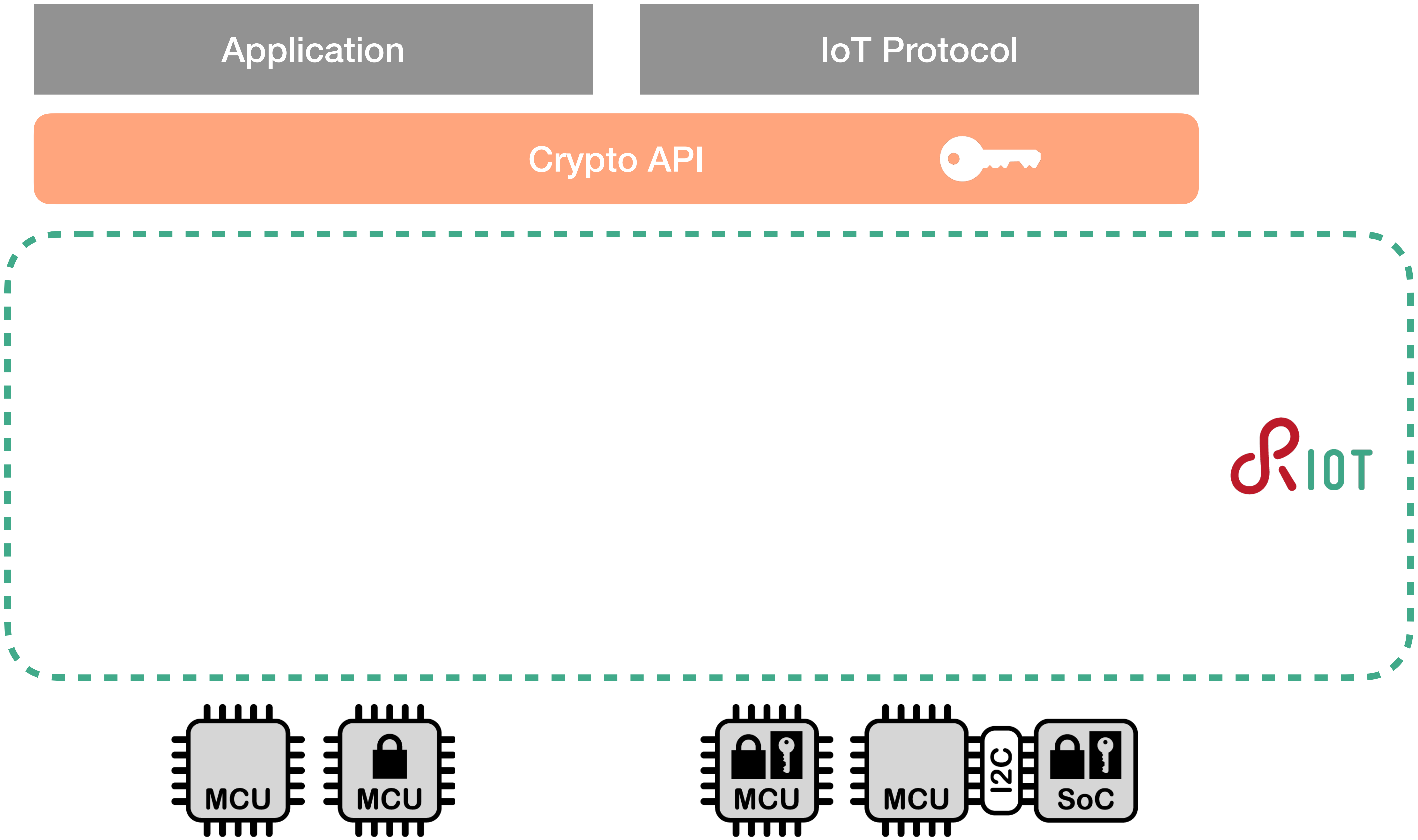
Outline

- Driver classification
- Requirements for a crypto API
- ARM PSA Crypto
- Integration in RIOT
- Evaluation
- Outlook

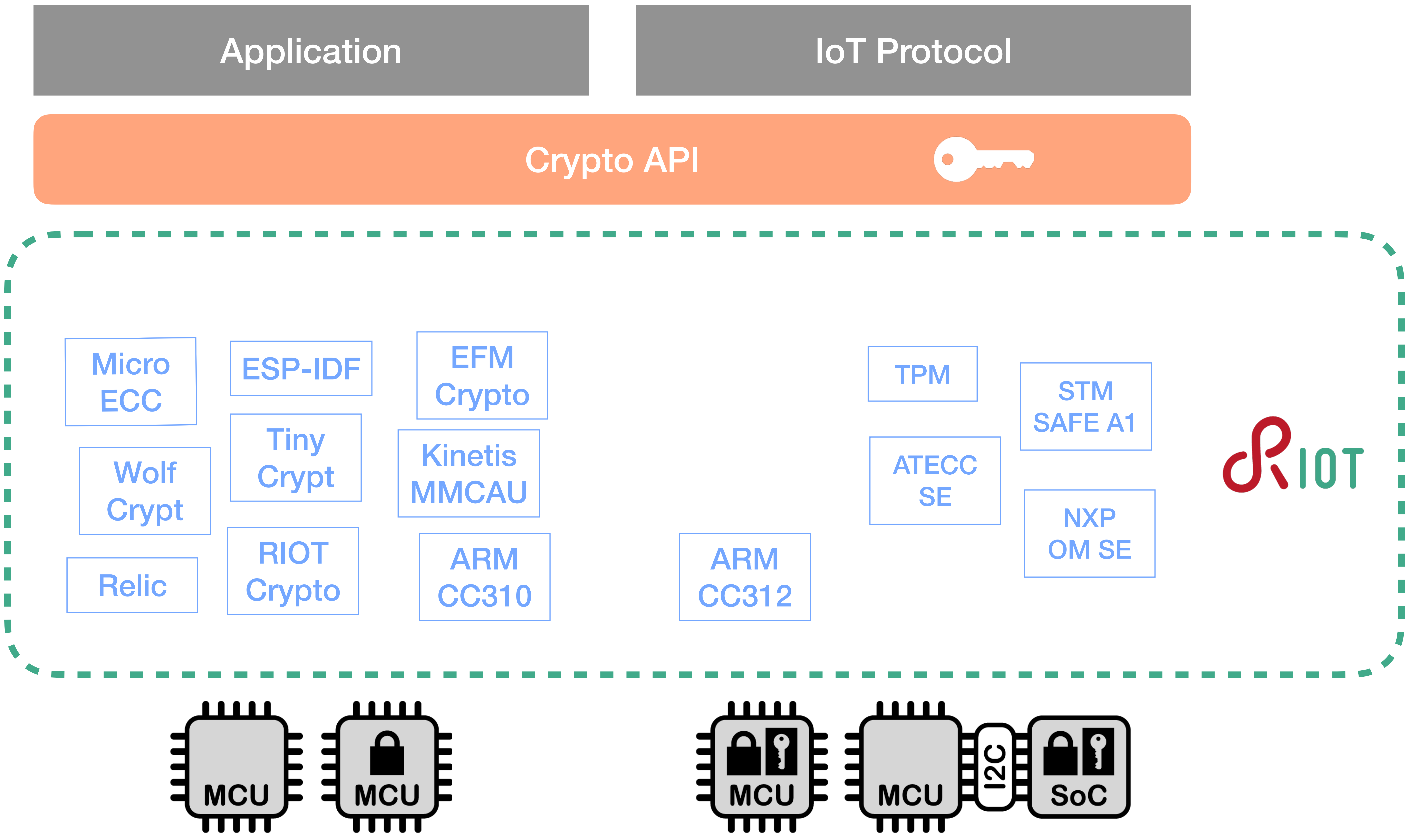
Driver Classification



Driver Classification



Driver Classification



Application

IoT Protocol

Crypto API

Micro ECC

ESP-IDF

EFM Crypto

Tiny Crypt

Kinetis MMCAU

Wolf Crypt

RIOT Crypto

Relic

ARM CC310

ARM CC312

TPM

ATECC SE

STM SAFE A1

NXP OM SE

RIOT

MCU

MCU

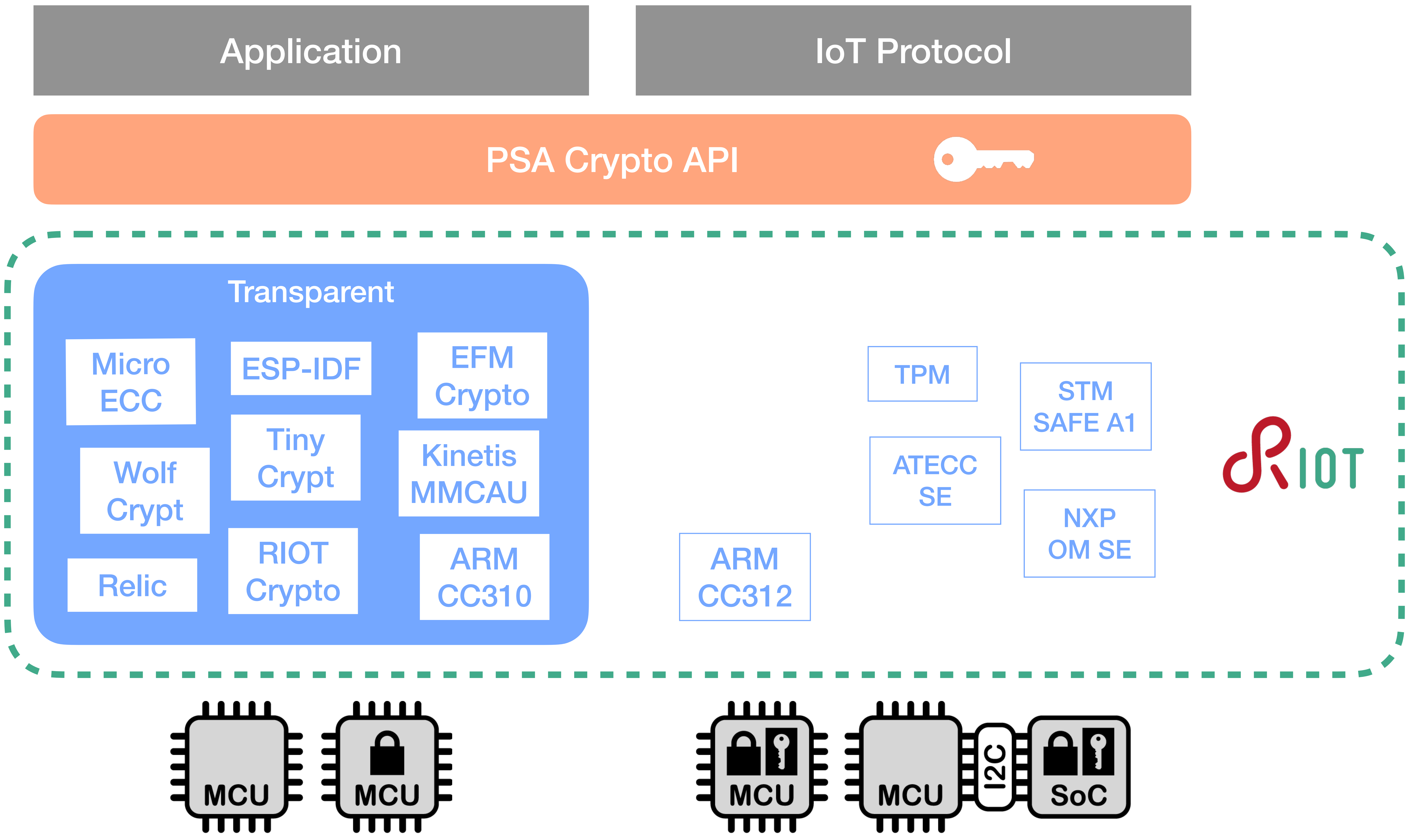
MCU

MCU

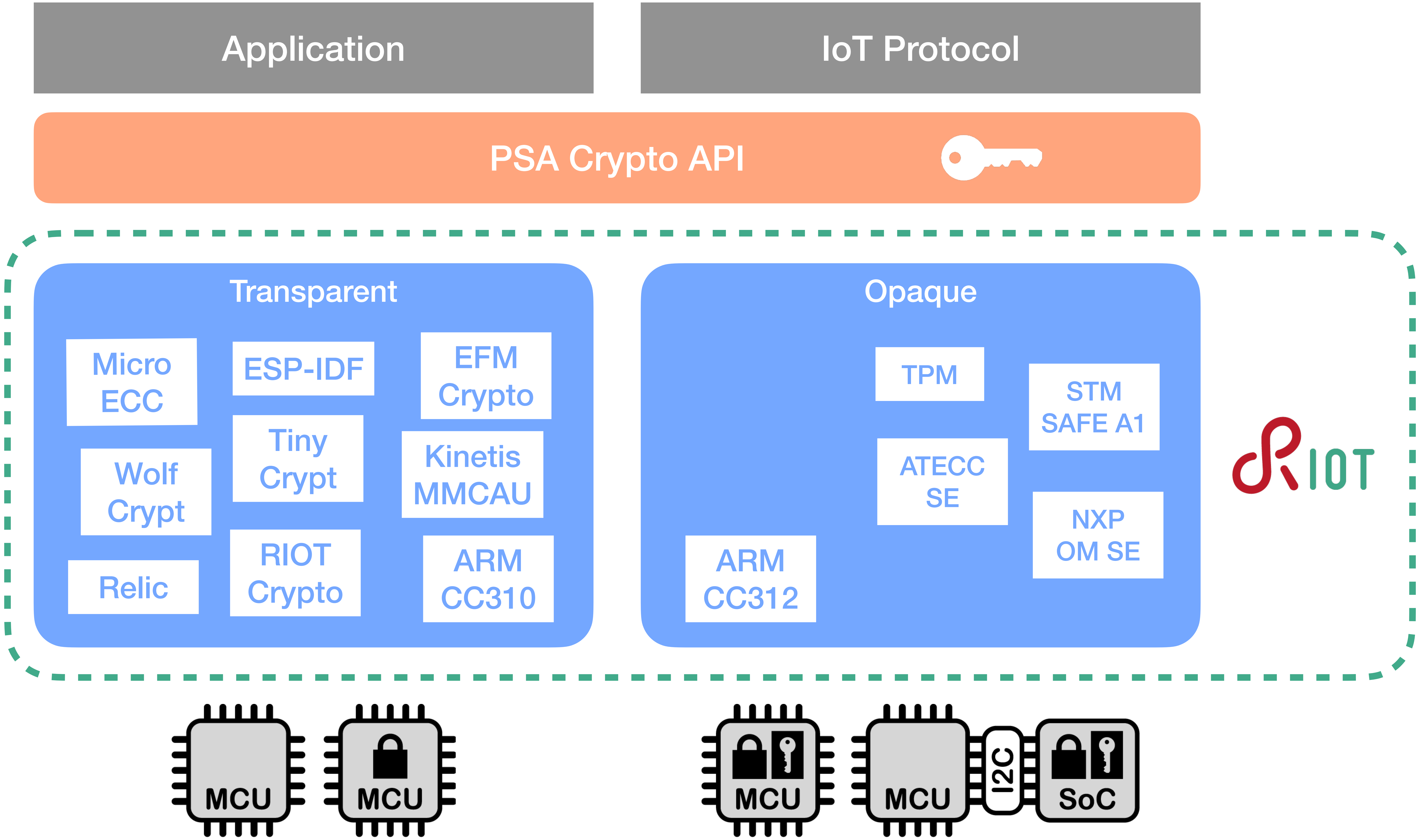
I2C

SoC

Driver Classification



Driver Classification



Requirements for a Crypto API

Portability

Platform:

- Support implementation agnostic development
- Exchange backends transparently

Application and OS:

- Switch to other OSes with same API
- Should be widely supported in the IoT

Extensive and Flexible

- Support all available algorithms in hardware and software
- Allow for any combination of drivers and libraries
- Indirect key access to support transparent and opaque backends

Usability

- Simplify development of secure applications
- Simple, usable interface
- Prevent misuse
- Good documentation and state-of-the-art examples
- Secure key handling and enforcing usage policies

ARM PSA Crypto API

What is PSA?

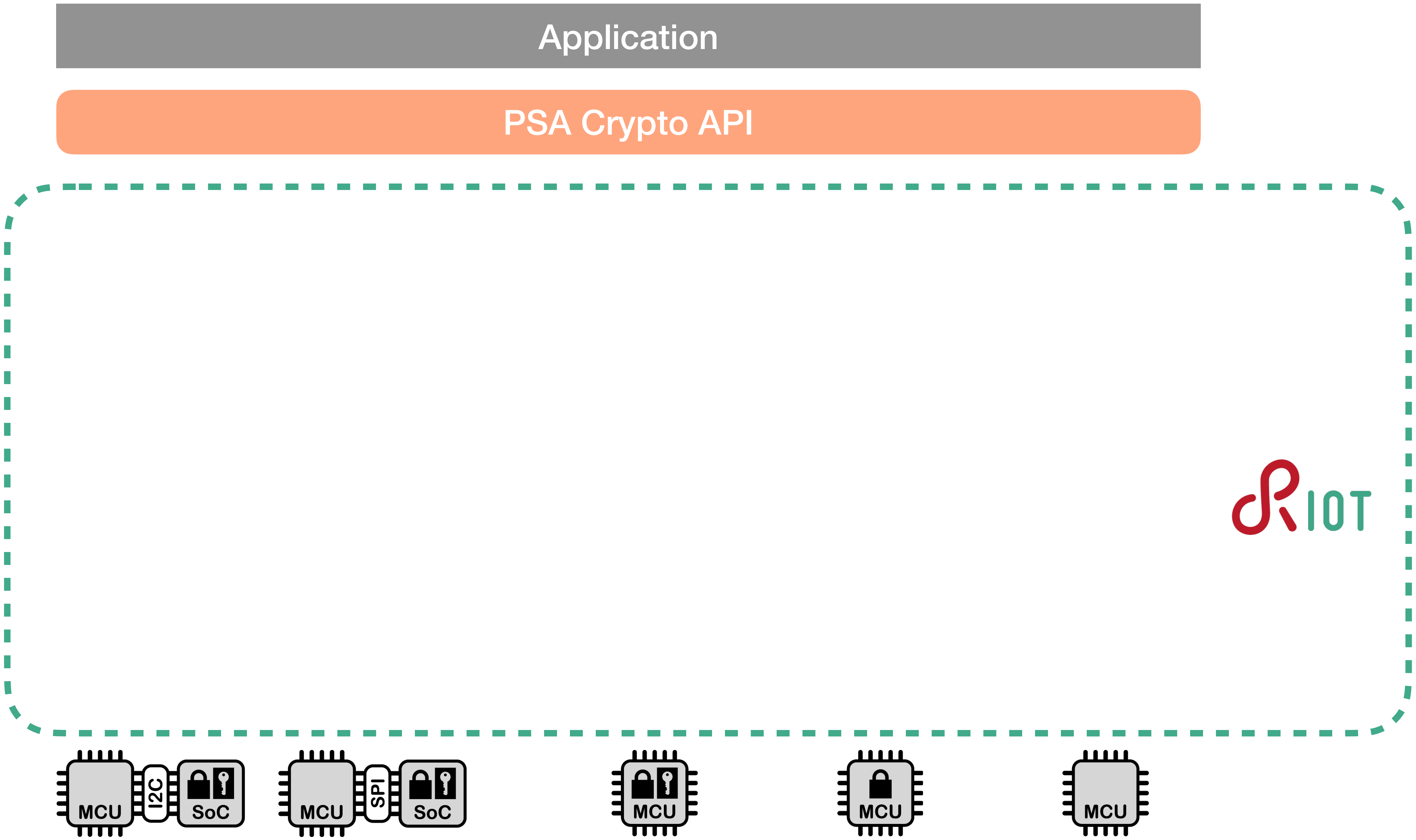
- ARM Platform Security Architecure
- Framework for secure IoT systems
- Standardized resources for hardware, firmware and software development
- Certification process
- PSA Crypto is one of four APIs designed for utilizing system security services

Why do we want this?

- Specifically designed for IoT
- Indirect, ID-based key management
- Supports all kinds of backends
- Secure element handling
- Testing
- Already supported by other OSes and libraries

Integration in RIOT

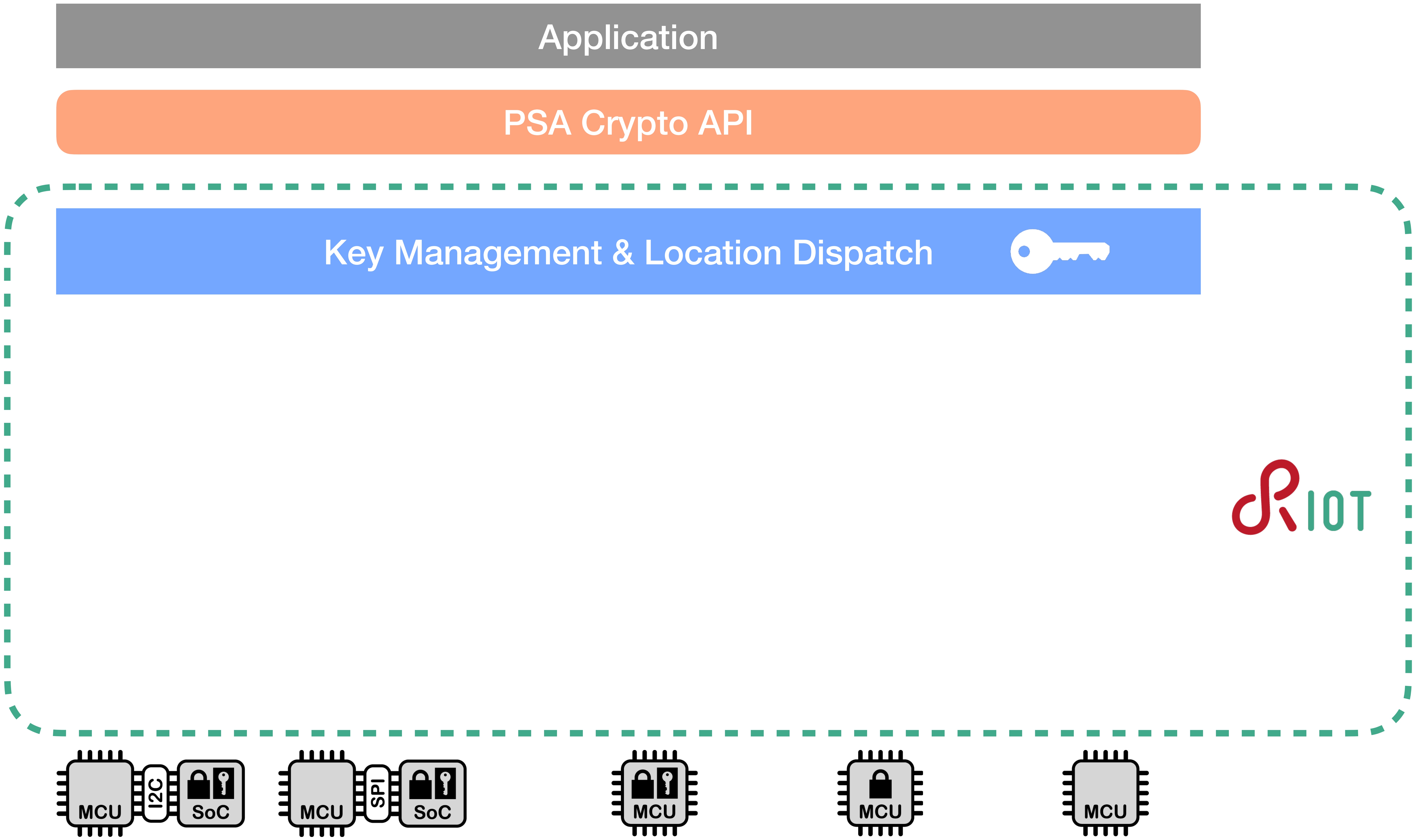
Integration in RIOT



Application

PSA Crypto API

Integration in RIOT



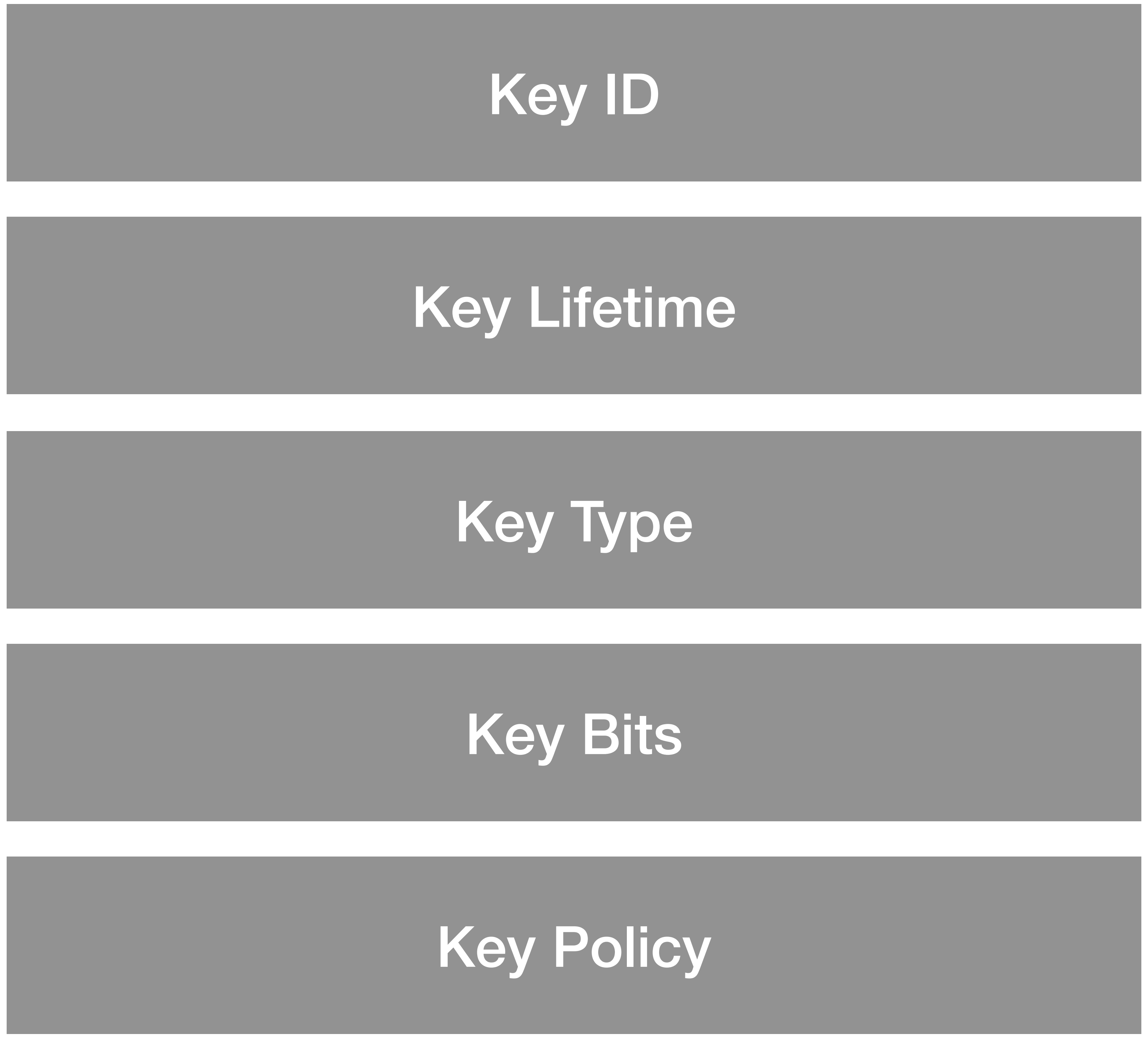
Application

PSA Crypto API

Key Management & Location Dispatch

Key Management

- Internal, ID-based
- Key attributes hold metadata (location, policies, etc.)
- Can't be changed without destroying key



Key ID

Key Lifetime

Key Type

Key Bits

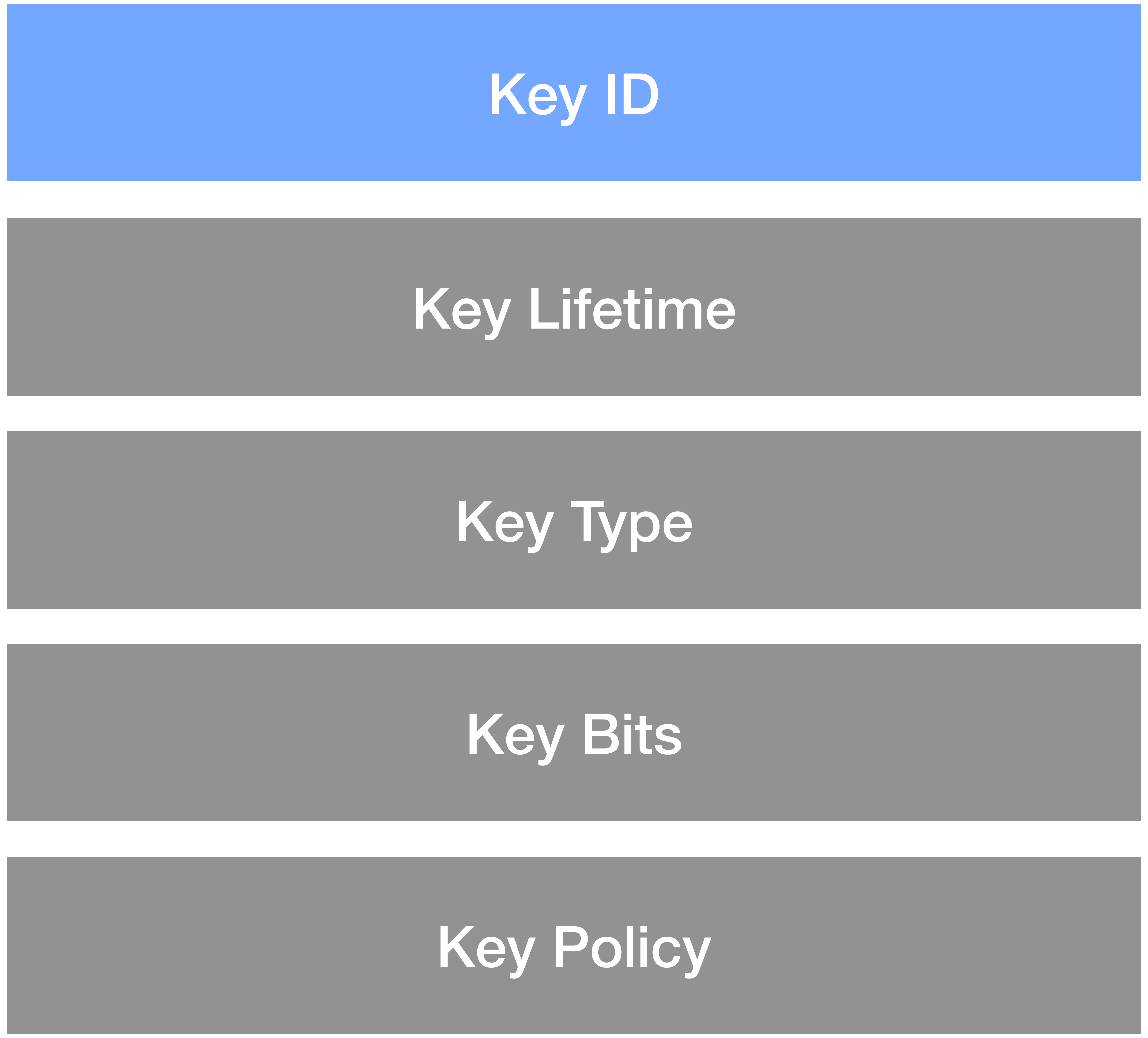
Key Policy

psa_key_attributes_t

Key Attributes

Key ID

- Assigned to each key
- Volatile keys: volatile ID assigned by key management
- Persistent keys: persistent ID specified by user



Key ID

Key Lifetime

Key Type

Key Bits

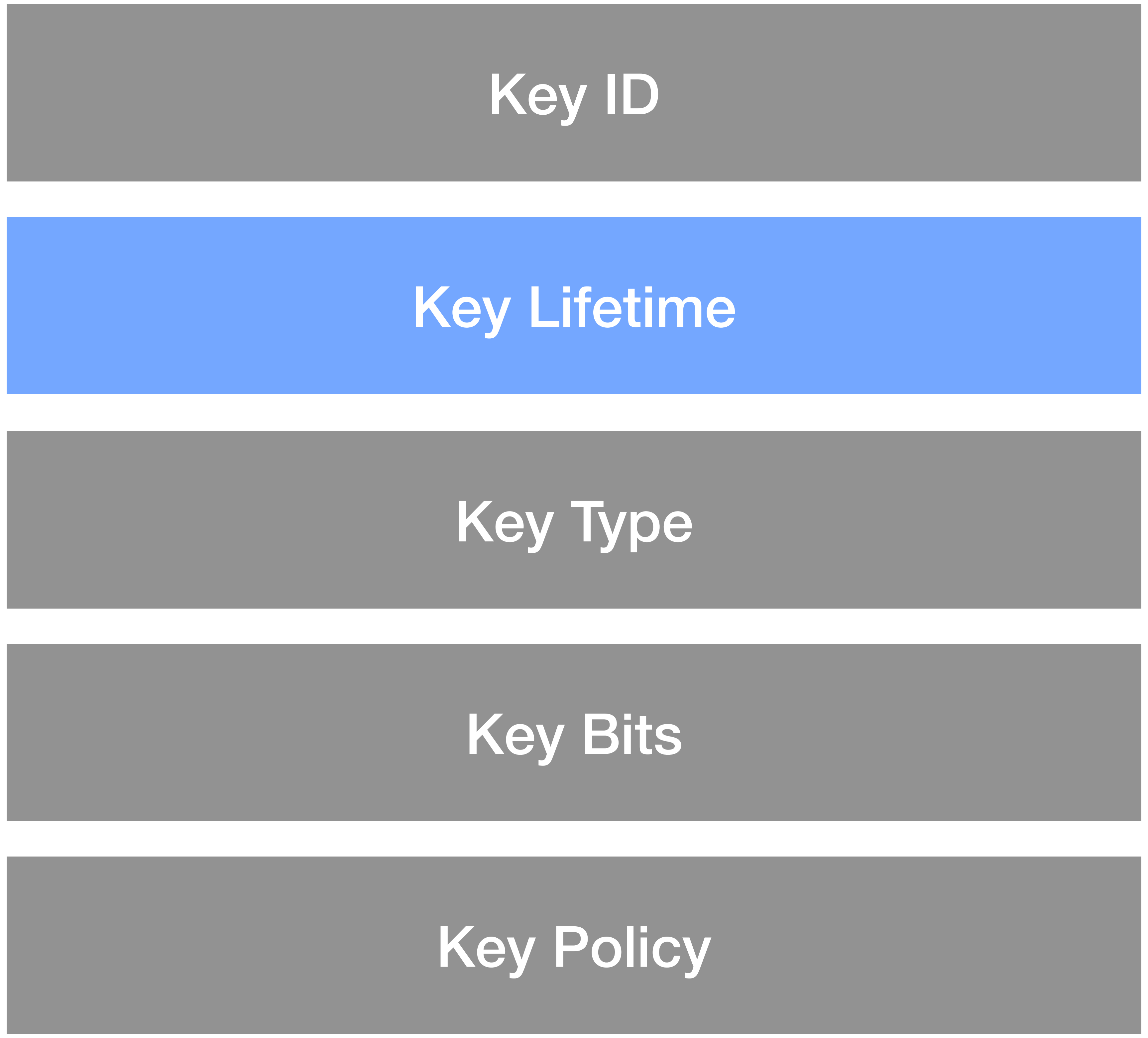
Key Policy

psa_key_attributes_t

Key Attributes

Key Lifetime

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



Key ID

Key Lifetime

Key Type

Key Bits

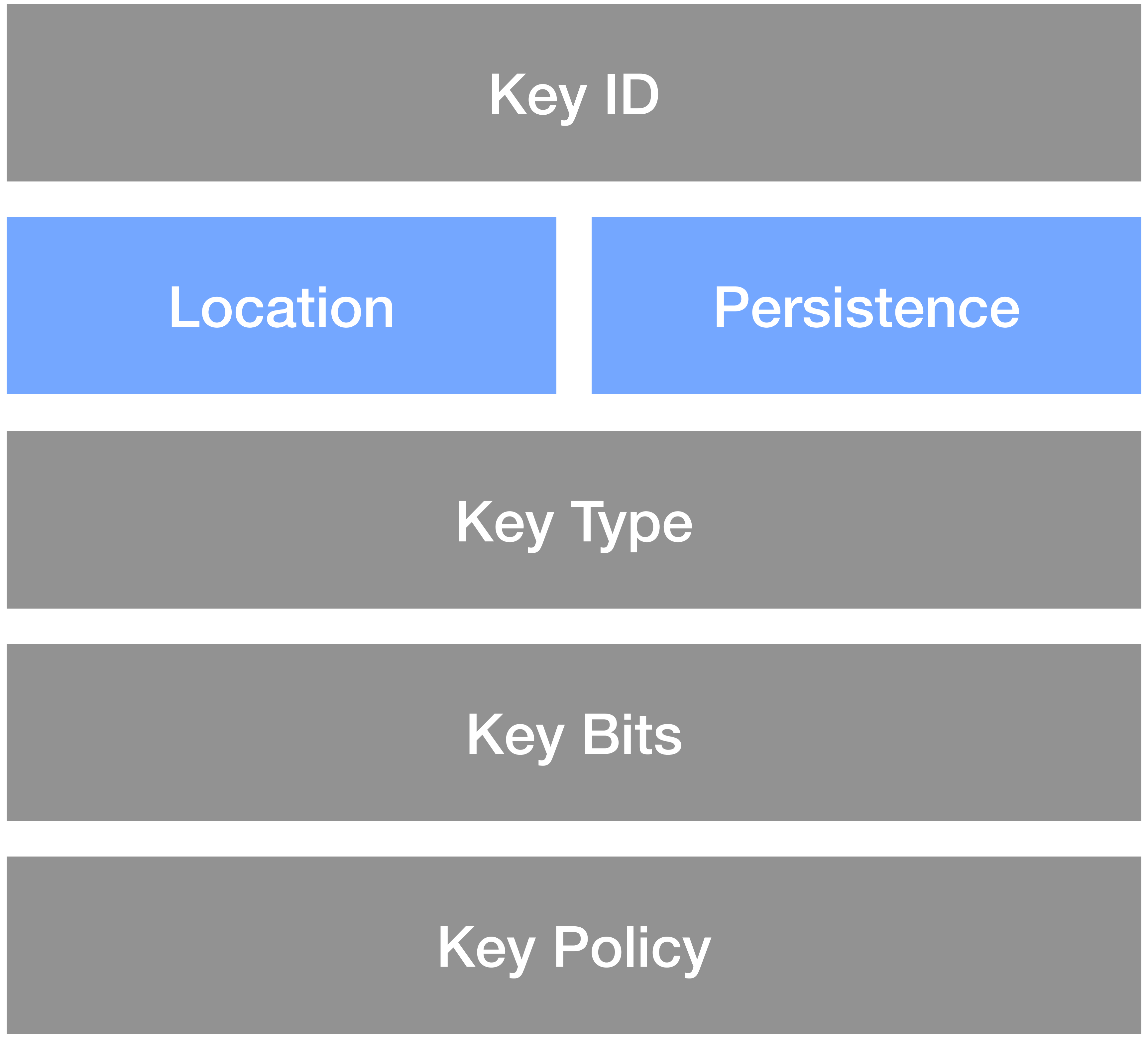
Key Policy

psa_key_attributes_t

Key Attributes

Key Lifetime

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

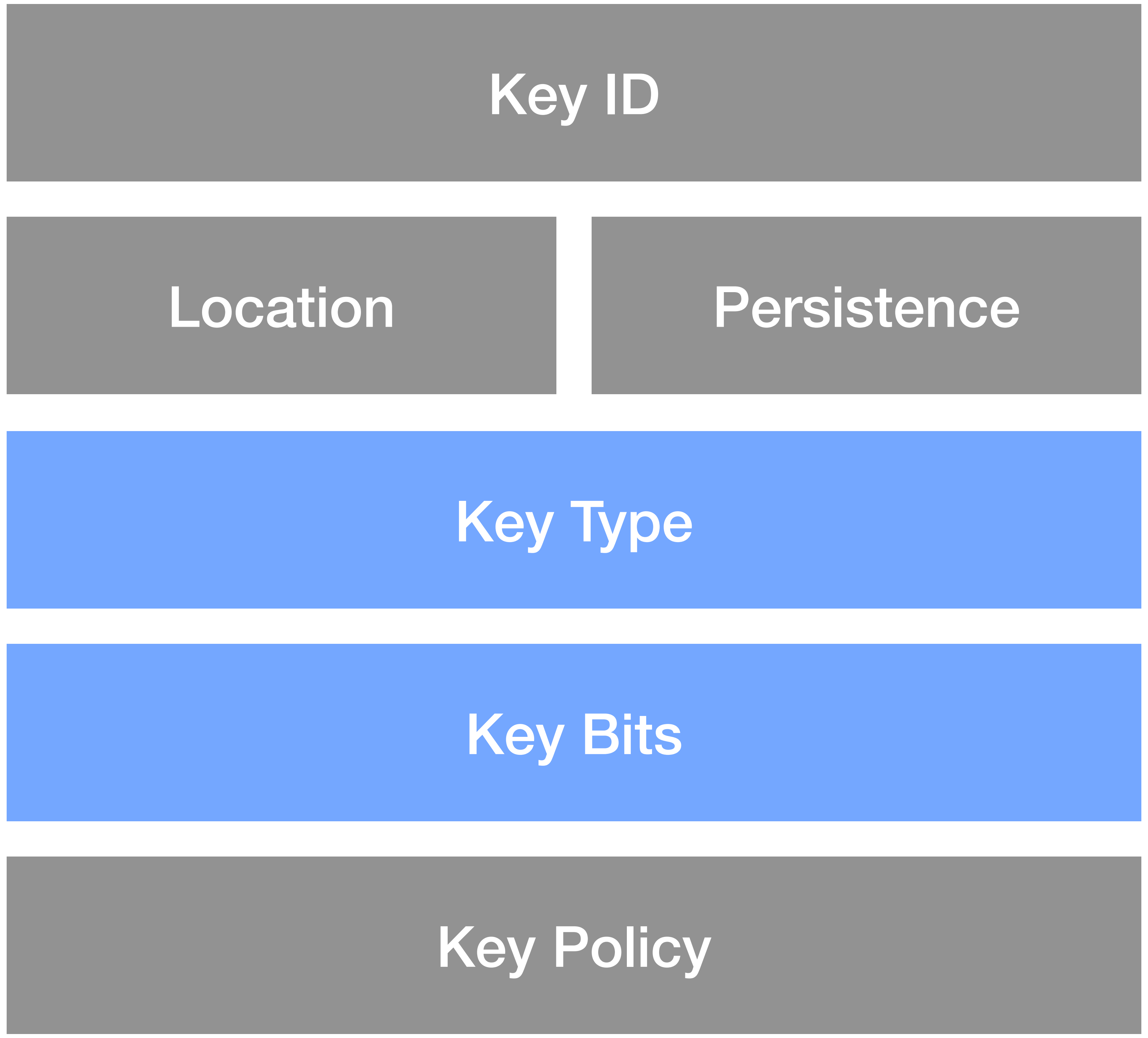


psa_key_attributes_t

Key Attributes

Key Type and Bits

- Type of key (e.g. AES, ECC-Family)
- Size of key in bits

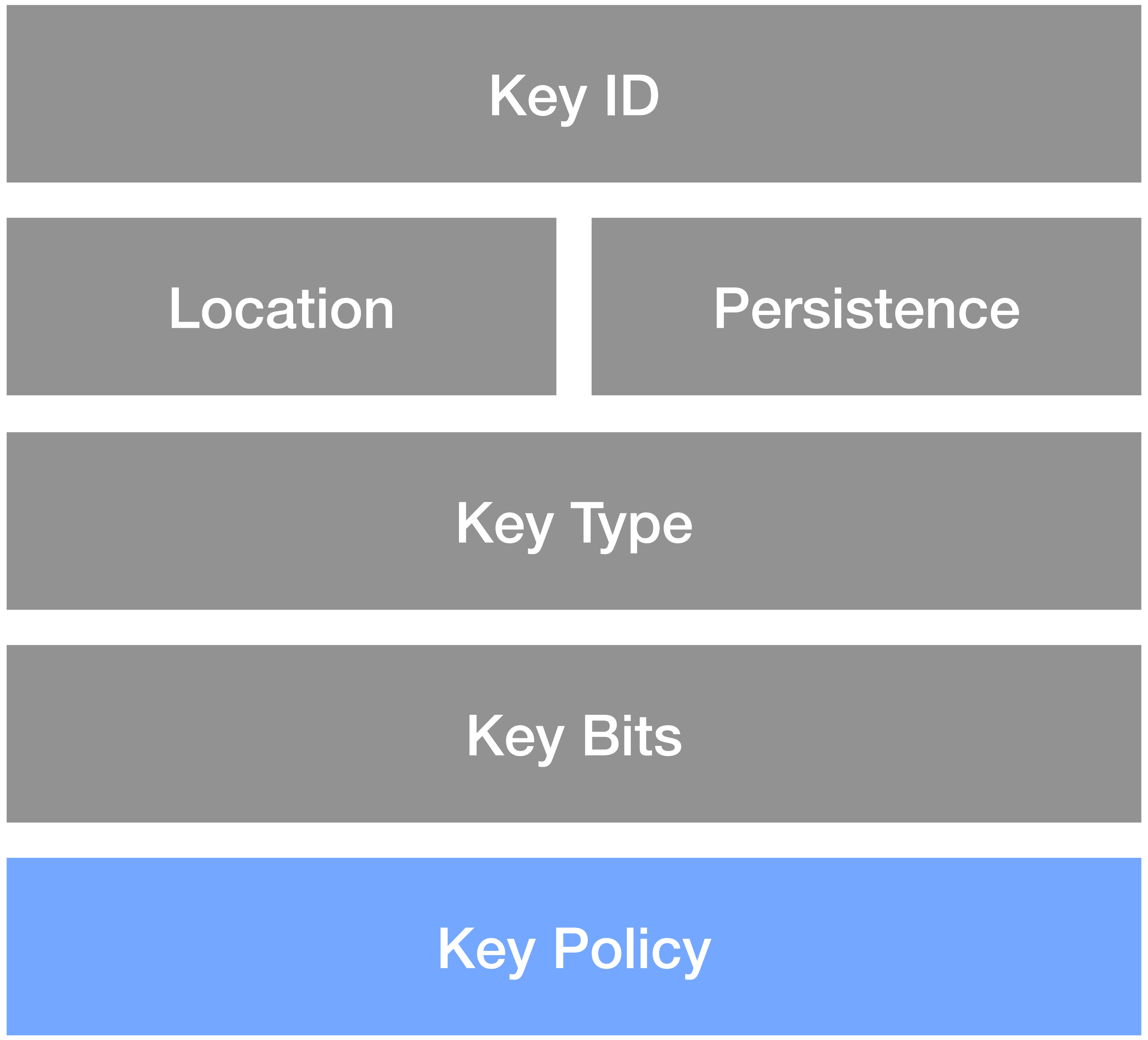


psa_key_attributes_t

Key Attributes

Key Policy

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



Key ID

Location

Persistence

Key Type

Key Bits

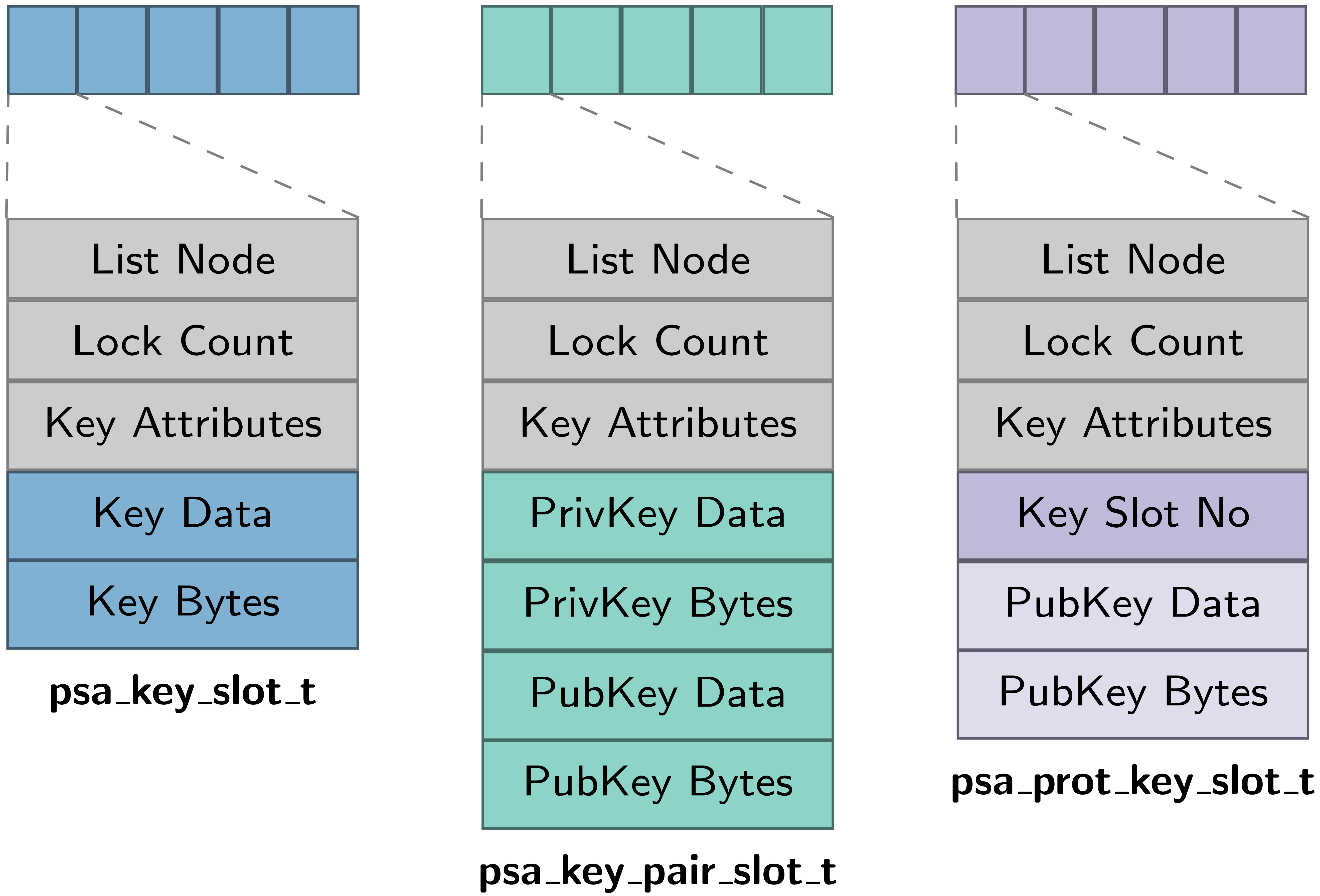
Key Policy

psa_key_attributes_t

Key Storage

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

Key Slots



List Node
Lock Count
Key Attributes
Key Data
Key Bytes

psa_key_slot_t

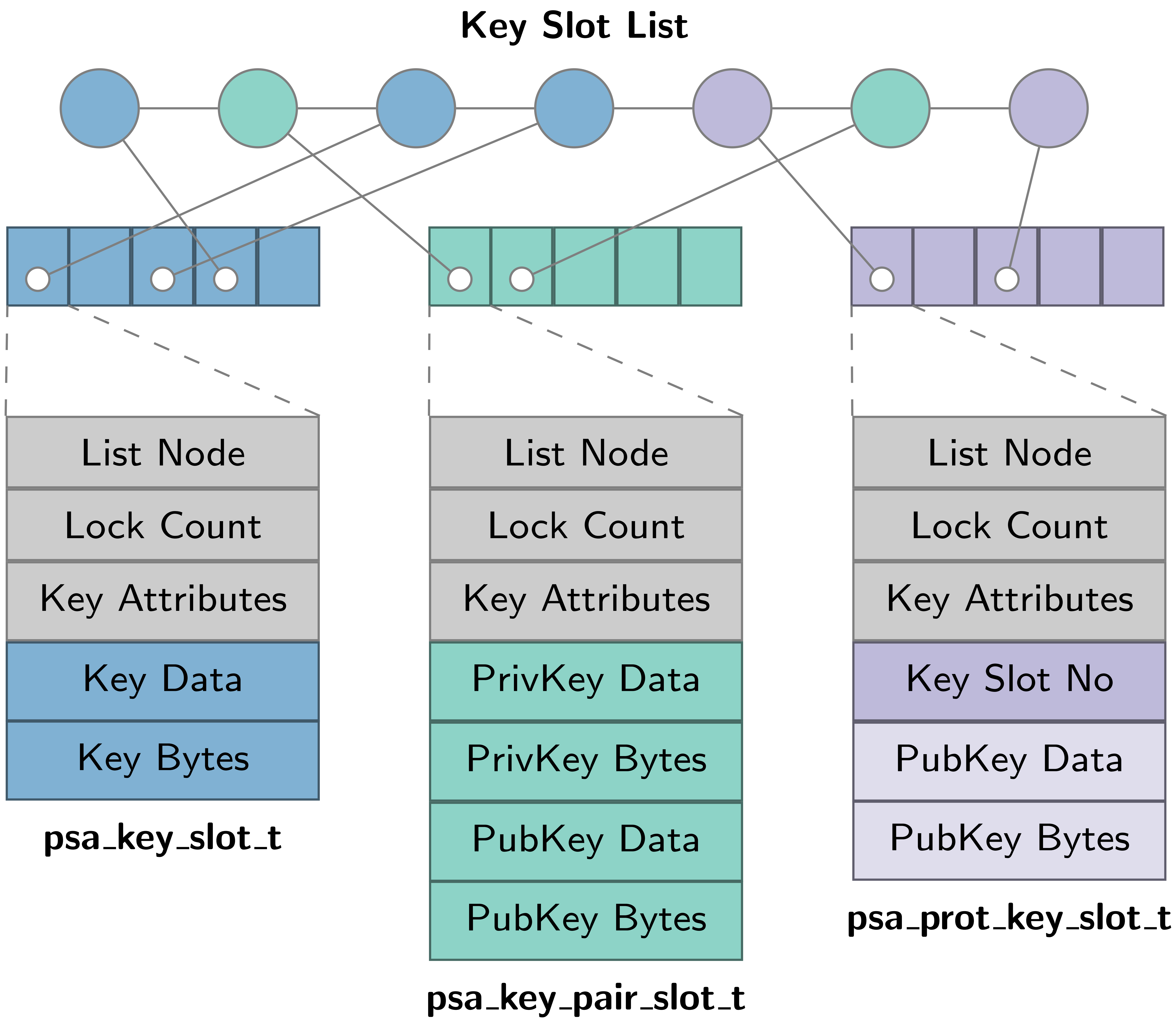
List Node
Lock Count
Key Attributes
PrivKey Data
PrivKey Bytes
PubKey Data
PubKey Bytes

psa_key_pair_slot_t

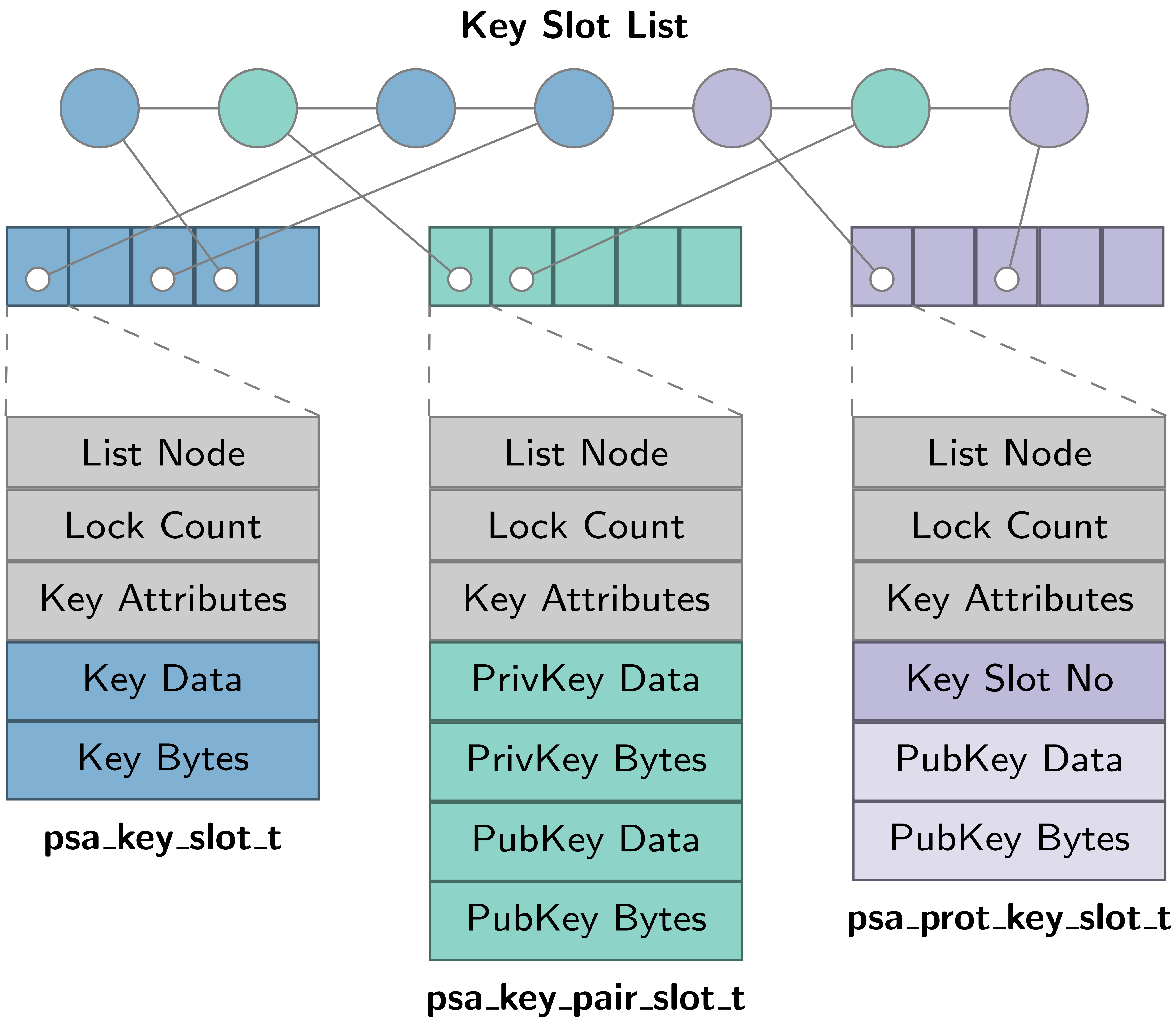
List Node
Lock Count
Key Attributes
Key Slot No
PubKey Data
PubKey Bytes

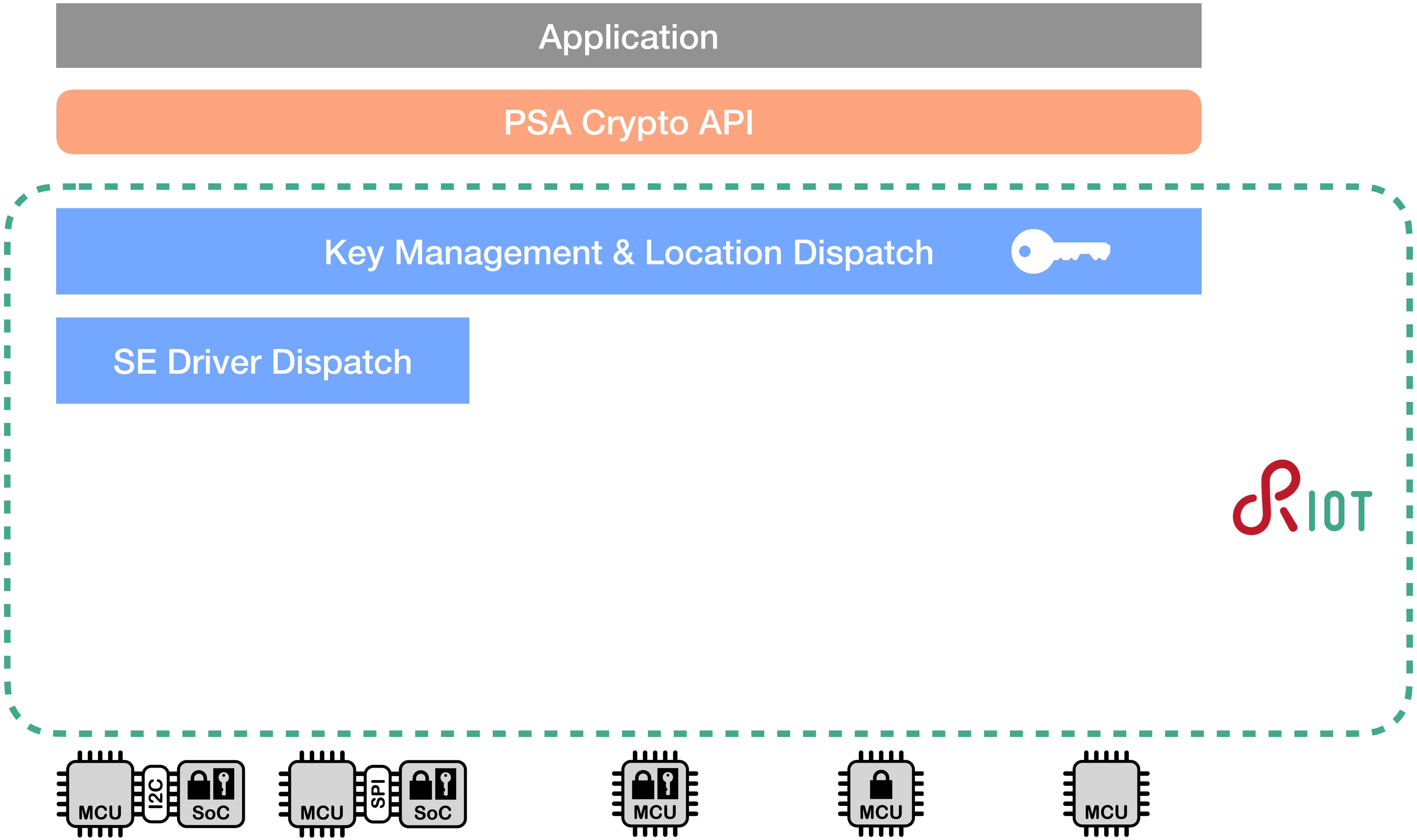
psa_prot_key_slot_t

Key Slots


Key Slot List
List Node
Lock Count
Key Attributes
Key Data
Key Bytes
psa_key_slot_t
List Node
Lock Count
Key Attributes
PrivKey Data
PrivKey Bytes
PubKey Data
PubKey Bytes
psa_key_pair_slot_t
List Node
Lock Count
Key Attributes
Key Slot No
PubKey Data
PubKey Bytes
psa_prot_key_slot_t

Key Slots


Key Slot List
List Node
Lock Count
Key Attributes
Key Data
Key Bytes
psa_key_slot_t
List Node
Lock Count
Key Attributes
PrivKey Data
PrivKey Bytes
PubKey Data
PubKey Bytes
psa_key_pair_slot_t
List Node
Lock Count
Key Attributes
Key Slot No
PubKey Data
PubKey Bytes
psa_prot_key_slot_t



Application

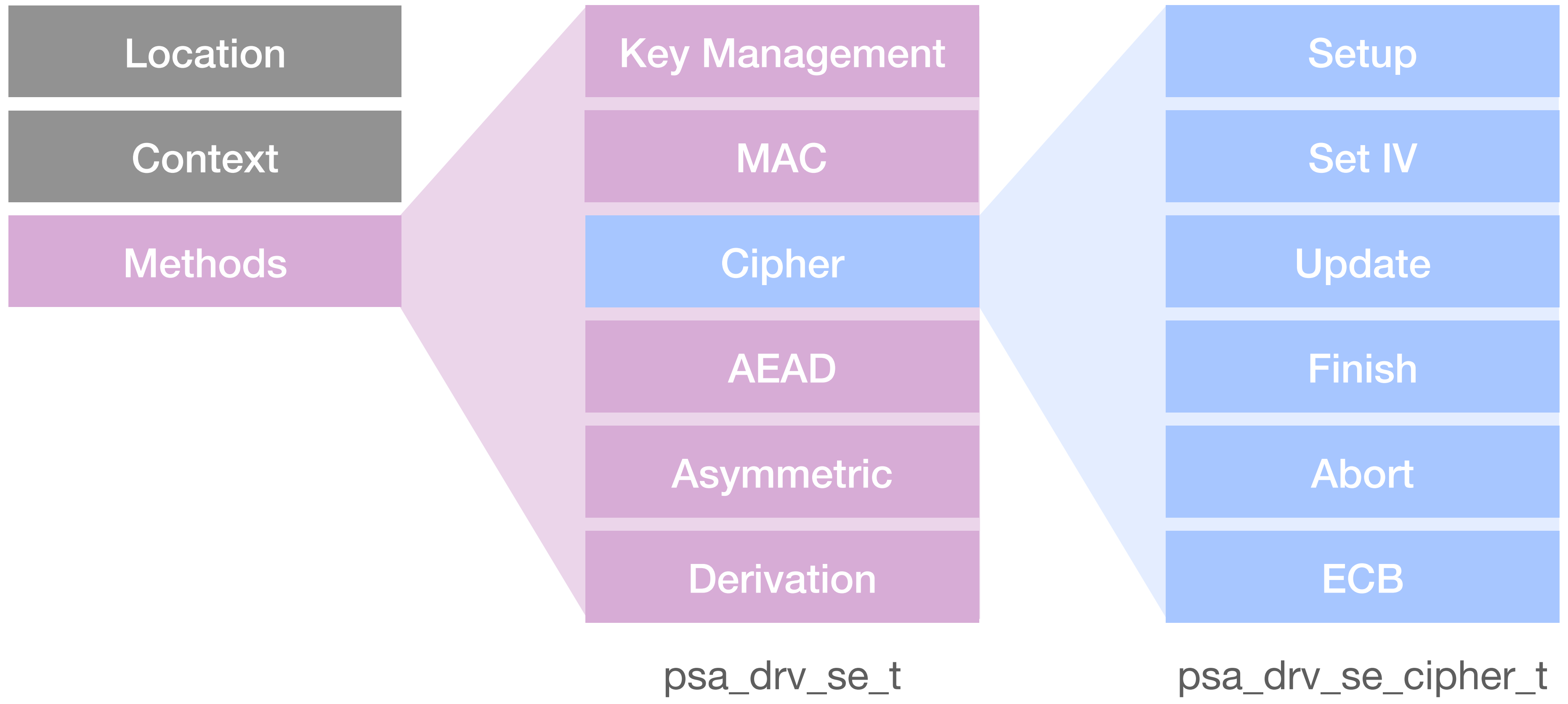
PSA Crypto API

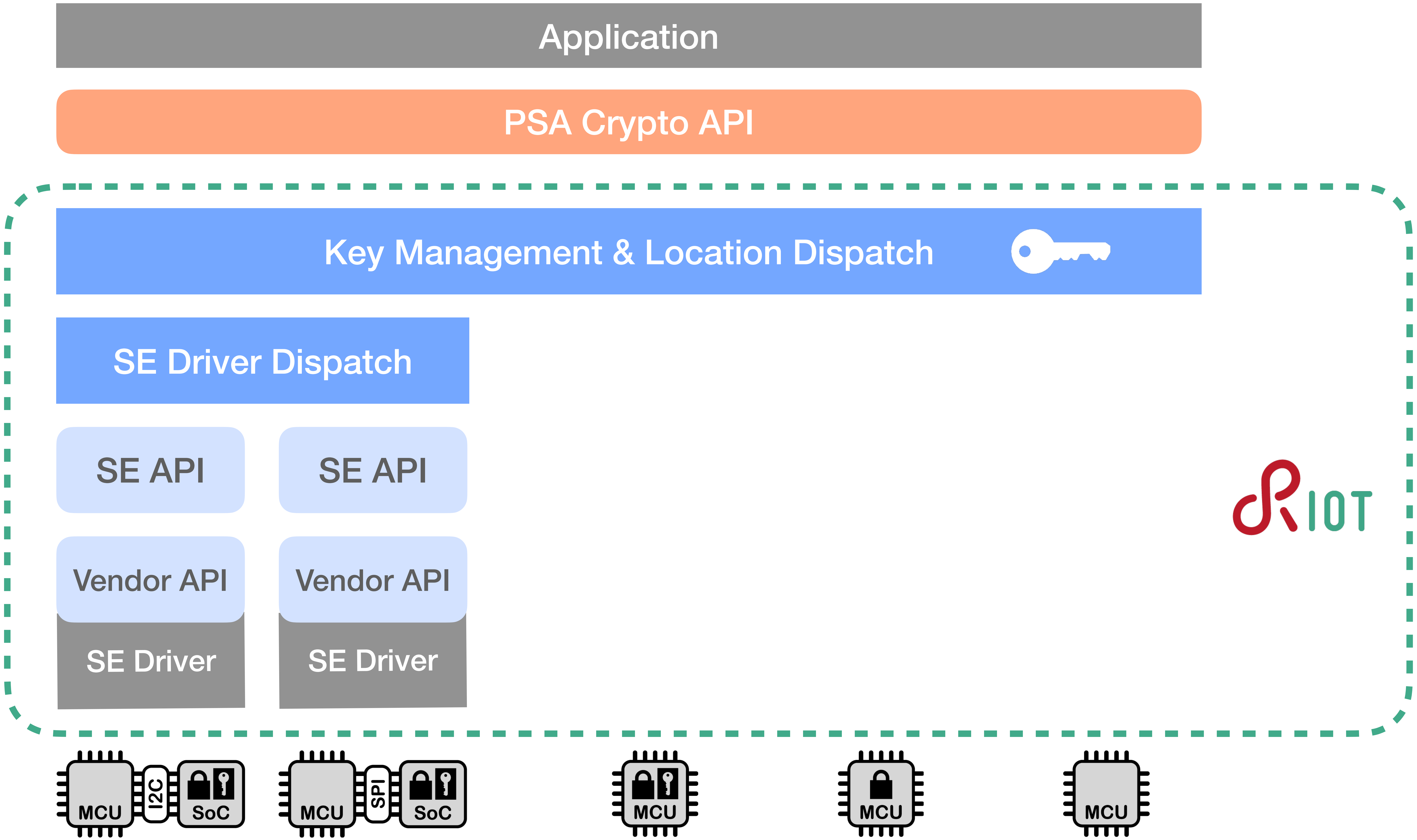
Key Management & Location Dispatch

SE Driver Dispatch

Secure Element Handling

- Connected SE devices get assigned a static location value
- Device drivers must implement generic SE interface and provide a structure with function pointers
- At startup:
  - OS function `auto_init` initializes and registers devices with SE management module
  - SE module stores function pointers, locations and some context data in global driver list
- At runtime, drivers associated to key location values are retrieved from list to perform operations





Application

PSA Crypto API

Key Management & Location Dispatch

SE Driver Dispatch

SE API

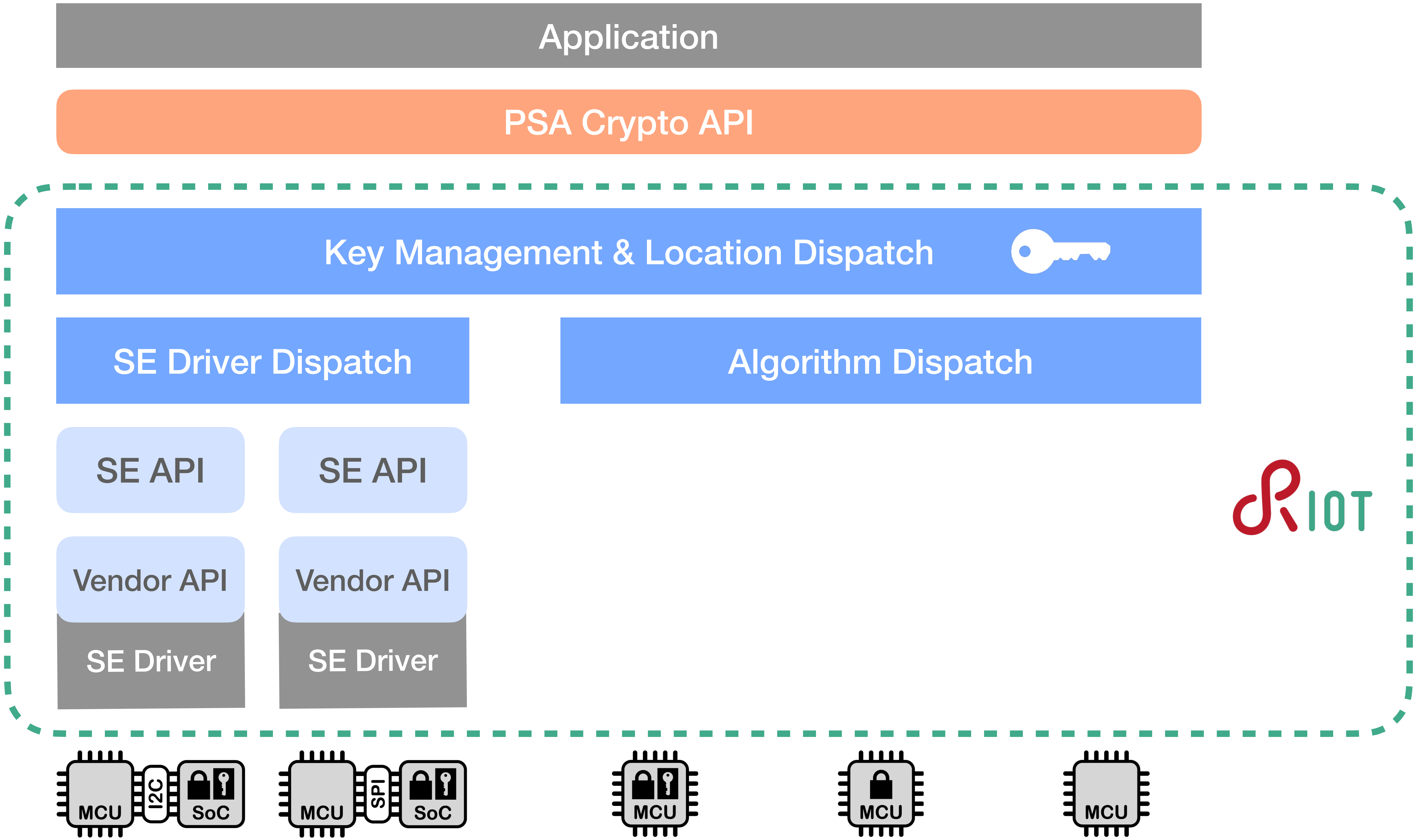
SE API

Vendor API

Vendor API

SE Driver

SE Driver



Application

PSA Crypto API

Key Management & Location Dispatch

SE Driver Dispatch

Algorithm Dispatch

SE API

SE API

Vendor API

Vendor API

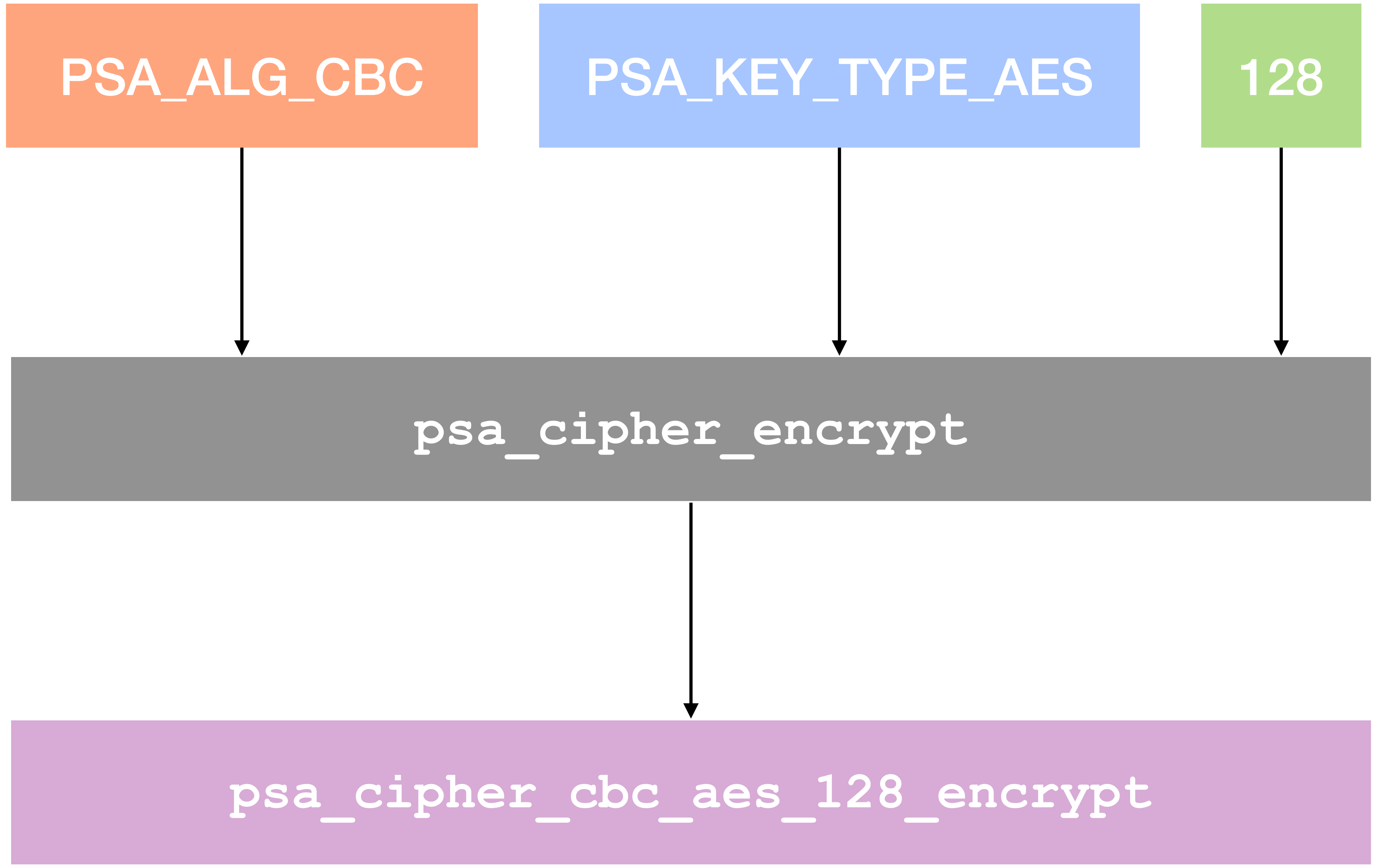
SE Driver

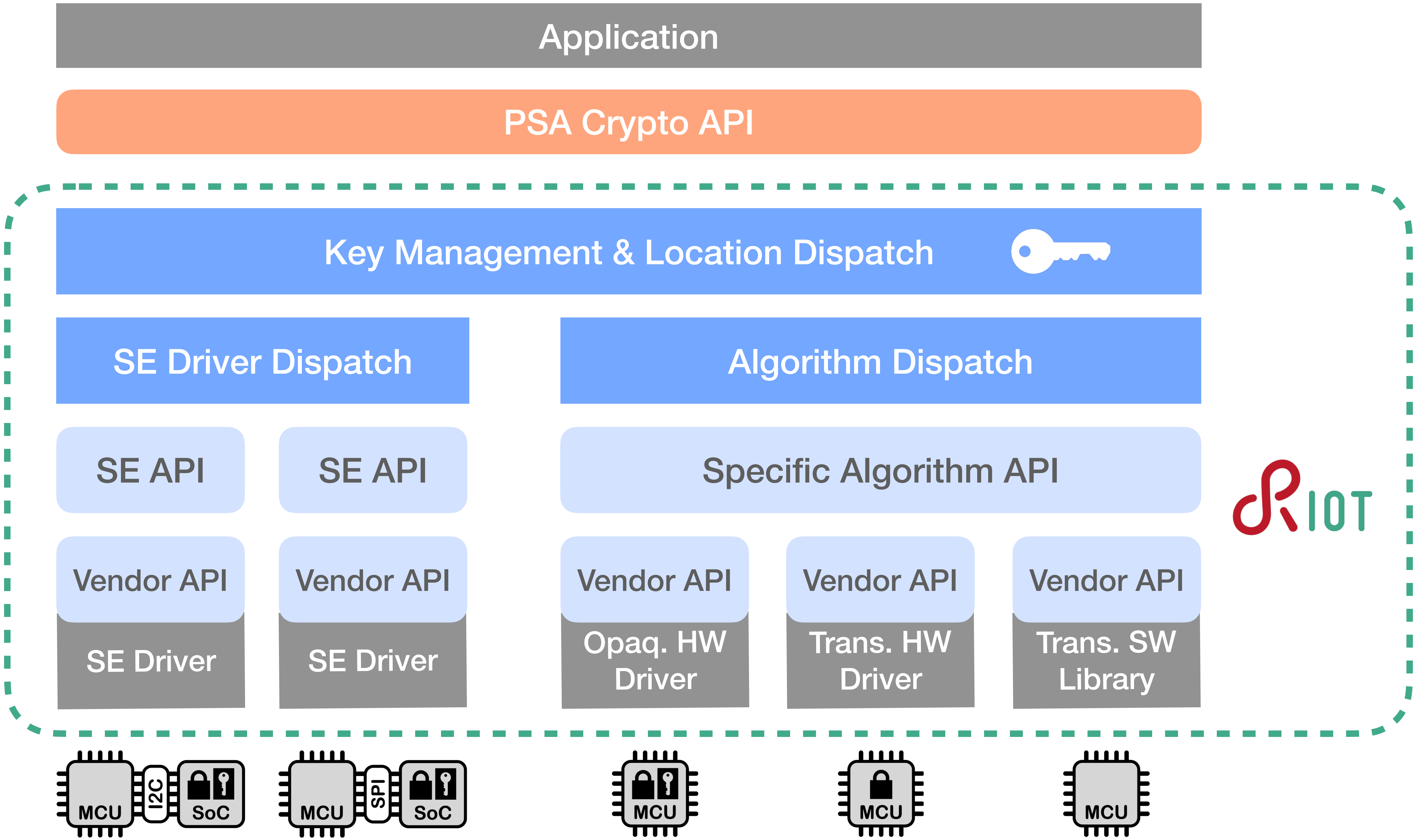
SE Driver

Algorithm Dispatcher

- Maps algorithm, key type and key size to specific algorithm API

- Transparent and opaque drivers and libraries implement algorithm specific API




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

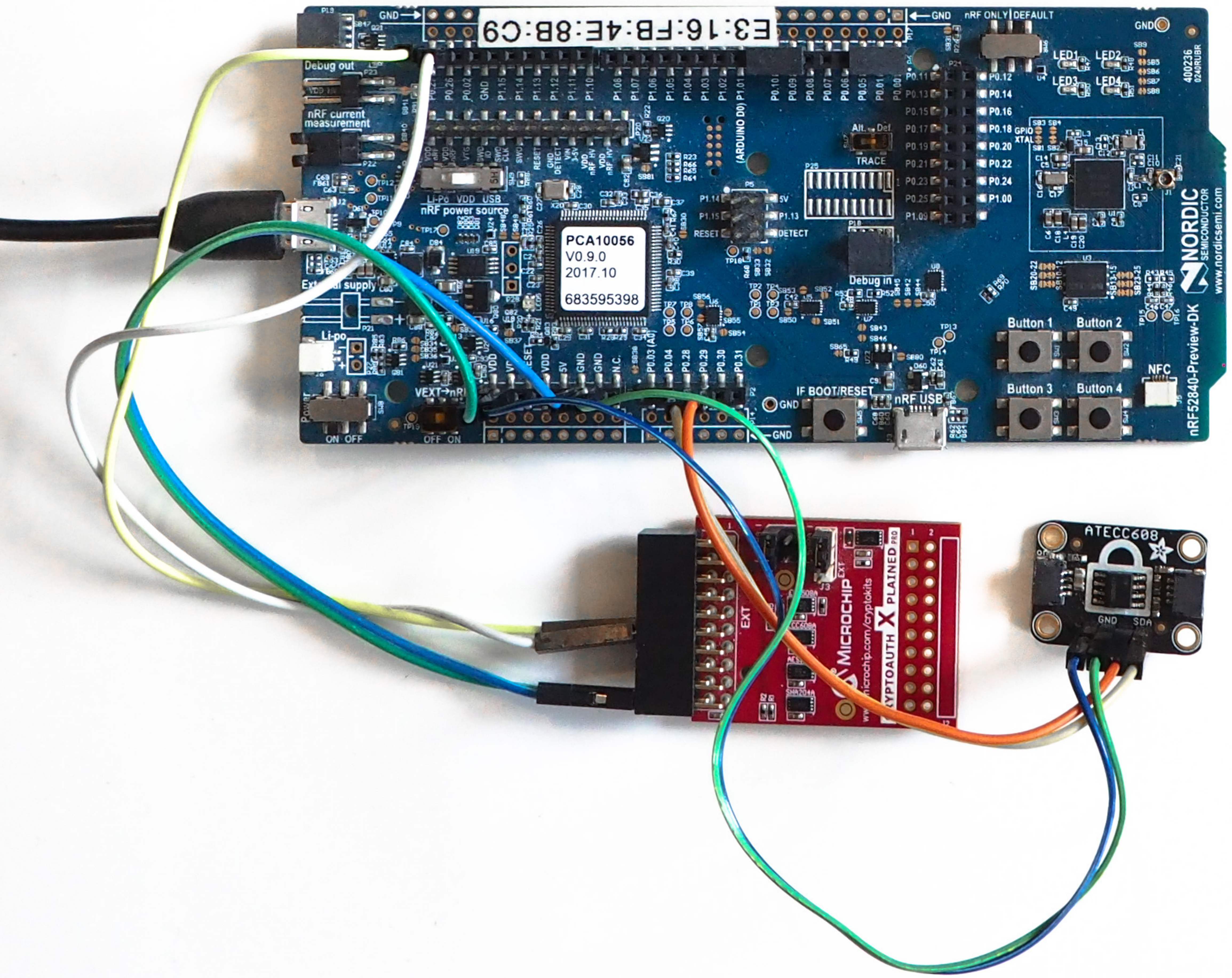
Kconfig for Backend Configuration

- Select build-time options, enable/disable features
- Use configuration file or GUI
- RIOT modules define menus and configuration symbols in Kconfig files
- Specified default selections or auto-selection in case of predefined conditions
- Automatic selection of hardware backends, if available (e.g. if CPU Kconfig defines `HAS_PERIPH_CIPHER_AES_128_CBC`)
- Developers can choose different backends
- Specify number of required key slots

Evaluation

- Processing Time
- Memory Overhead
- Code Deduplication
- Usability

Device Setup



- Nordic nRF52840dk with ARM CryptoCell 310 peripheral accelerator
- Microchip ATECC608A via I2C

Applications

HMAC SHA 256

- Import 32 byte key
- Compute MAC of 32 byte message

Backends:

- RIOT Hash module (SW)
- CryptoCell 310 (HW)
- ATECC608A (HW)

AES 128 CBC

- Import 16 byte key
- Encrypt 32 byte plaintext

Backends:

- RIOT Cipher module (SW)
- CryptoCell 310 (HW)
- ATECC608A (HW)

ECDSA

- Generate key pair with NIST P-256 curve
- Sign 127 byte message
- Import 64 byte public key
- Verify message signature

Backends:

- CryptoCell 310 (HW)
- ATECC608A (HW)

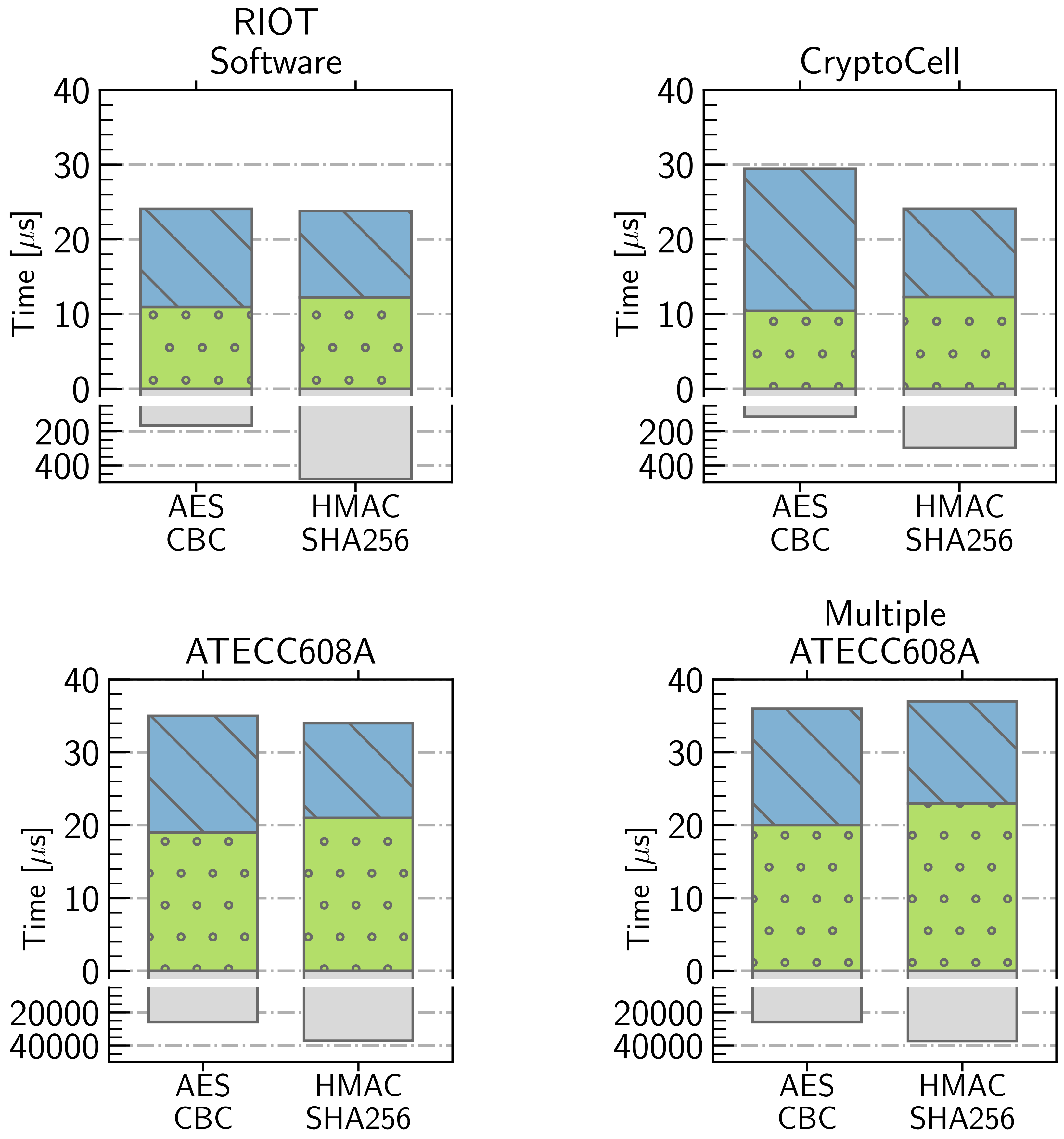
Processing Time

- Logic analyzer
- Toggle I/O pins via direct register access before and after function calls
- Complete processing of API functions and internal driver calls
- Mean over 1000 iterations

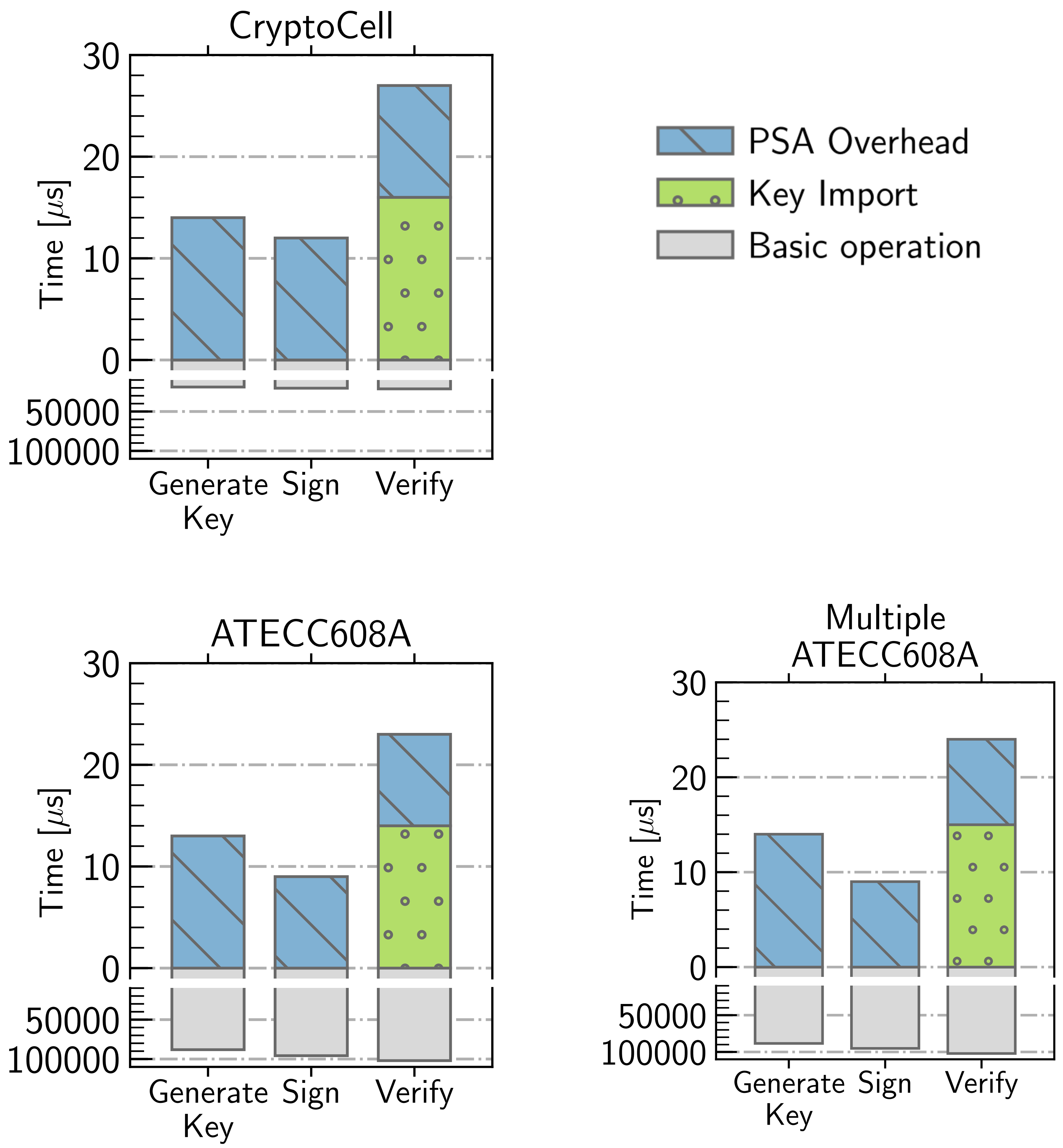
Processing Time

Results

Symmetric Crypto



Asymmetric Crypto

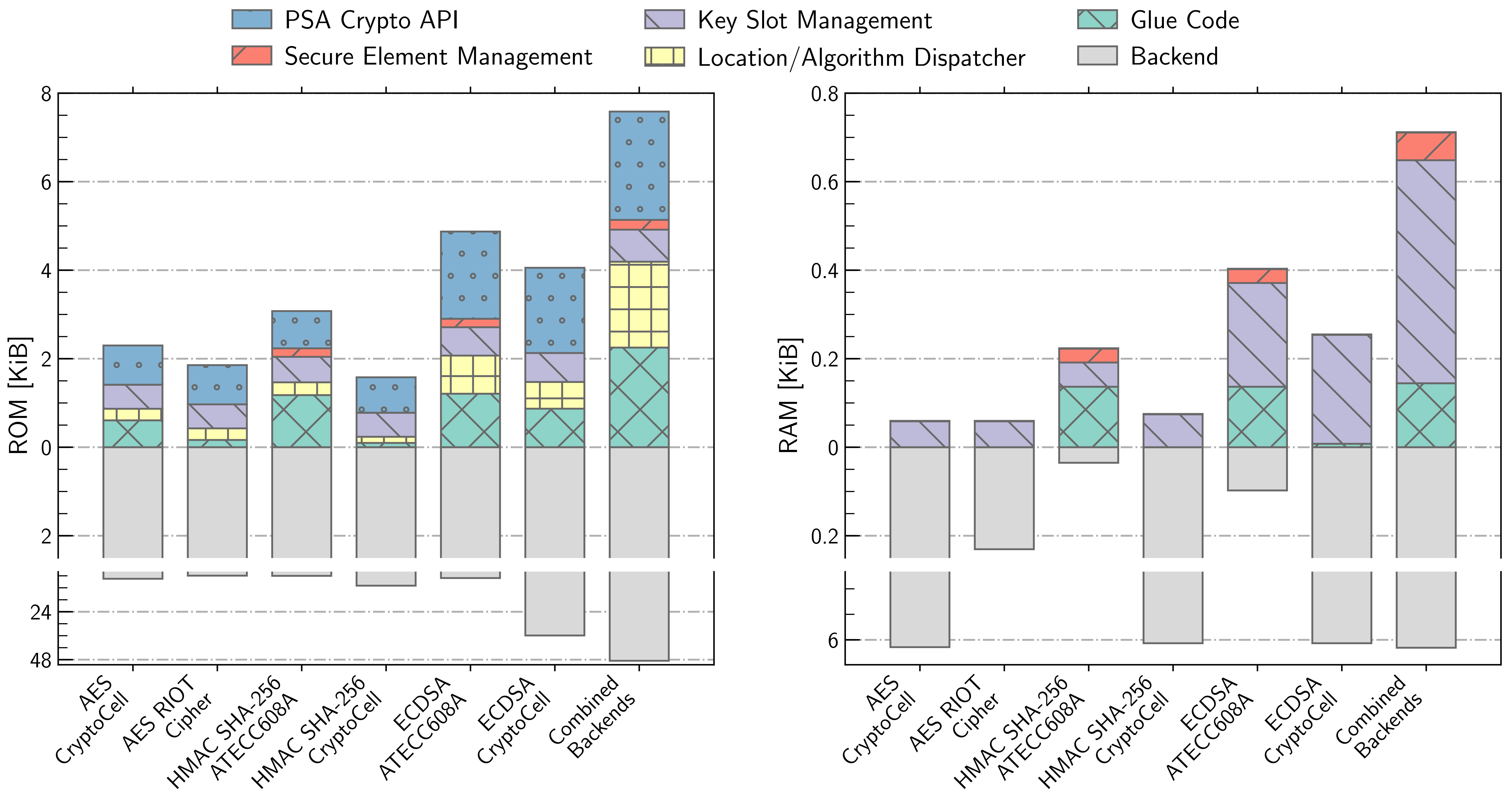


Memory Overhead

- Accumulation of crypto related objects in ELF file
- Ignores OS overhead
- Distinguished RAM and ROM

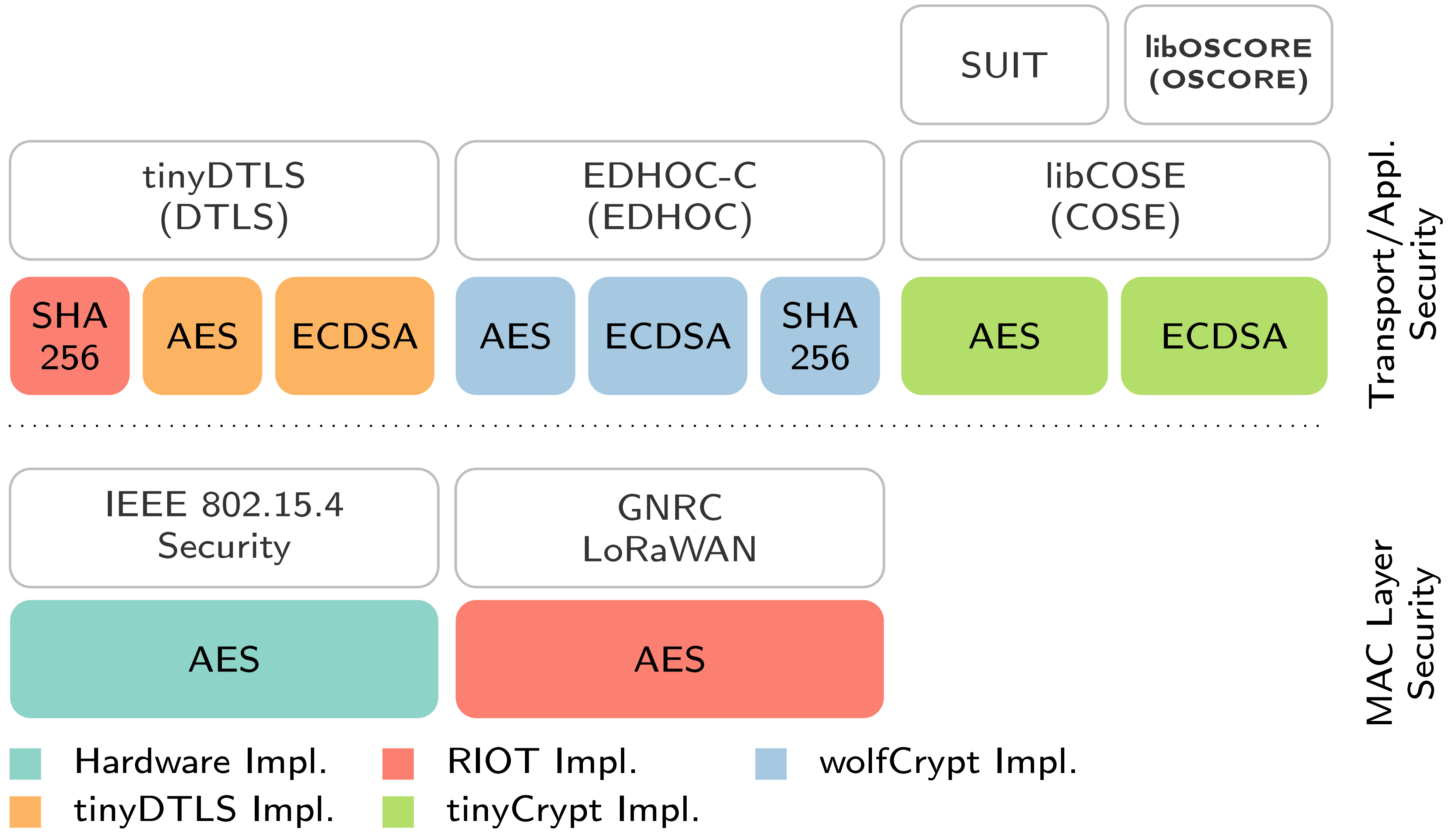
Memory Overhead

Results



Code Deduplication

Example: Secure Protocol Stack in RIOT



SUIT

libOSCORE (OSCORE)

tinyDTLS (DTLS)

EDHOC-C (EDHOC)

libCOSE (COSE)

SHA 256 | AES | ECDSA | AES | ECDSA | SHA 256 | AES | ECDSA

Transport/Appl. Security

IEEE 802.15.4 Security

GNRC LoRaWAN

AES | AES

MAC Layer Security

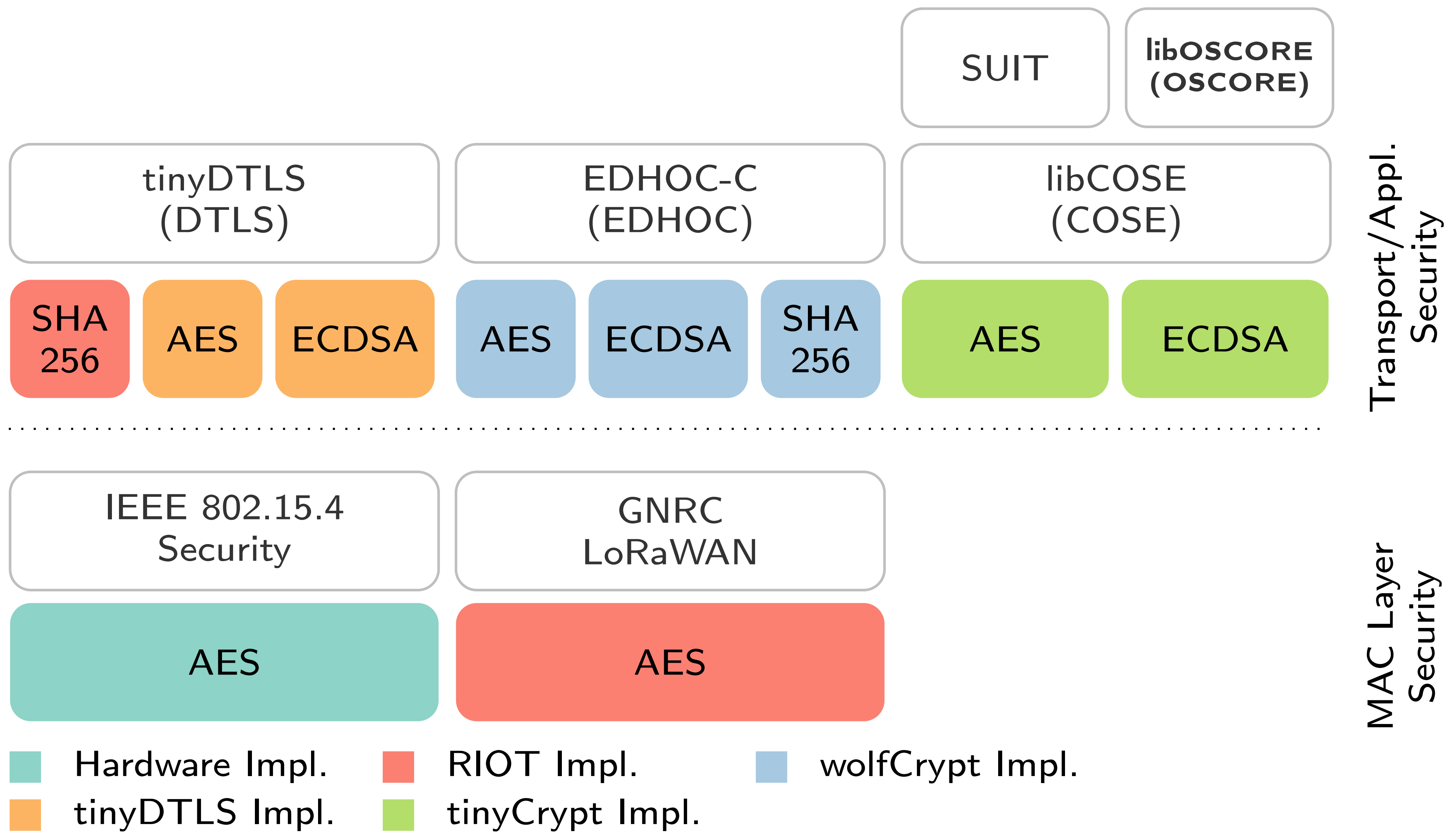
■ Hardware Impl. 🟥 RIOT Impl. 🟦 wolfCrypt Impl.

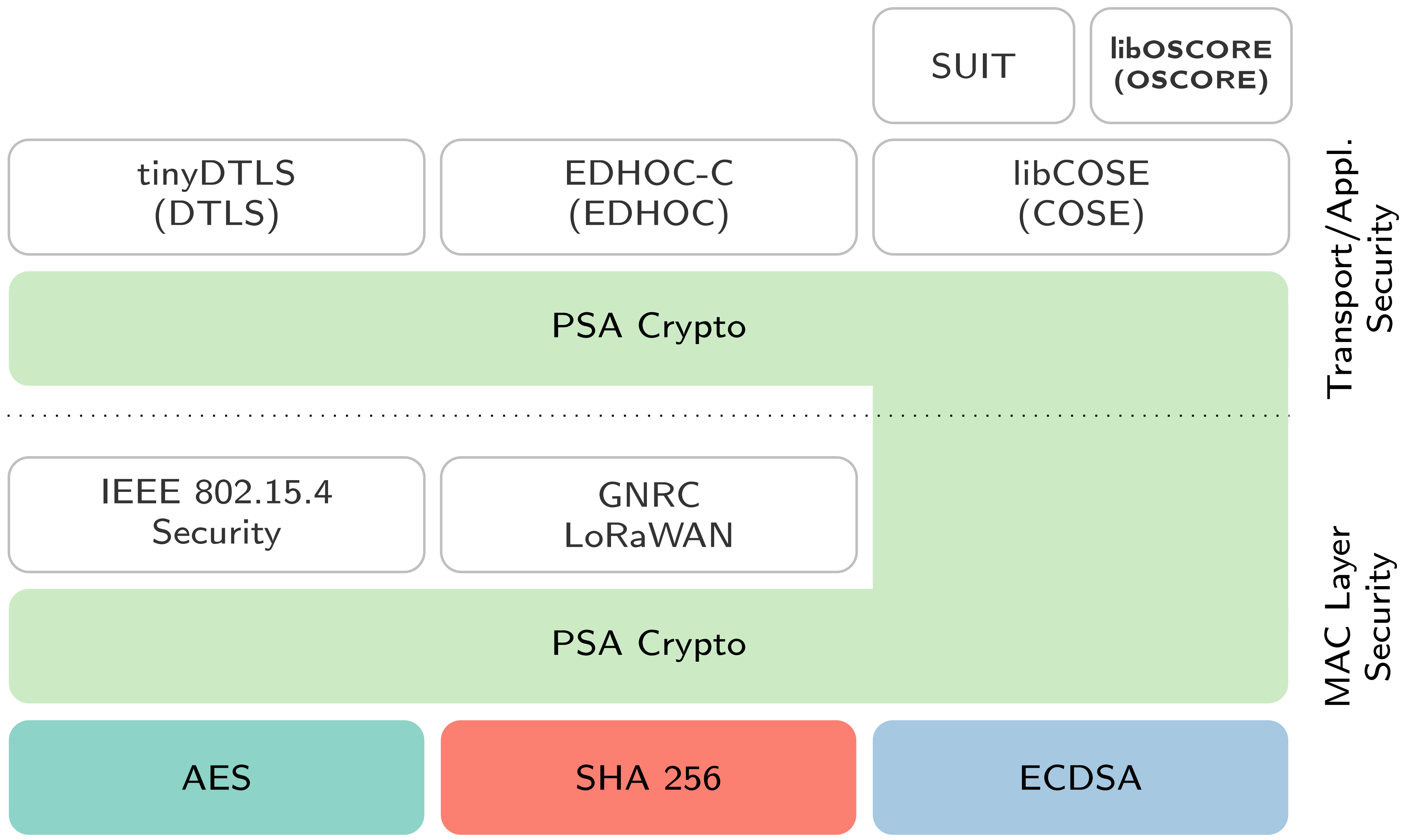
🟧 tinyDTLS Impl. 🟩 tinyCrypt Impl.

- Crypto: 26.5 kB ROM

Code Deduplication

Example: Secure Protocol Stack in RIOT





- Crypto: 26.5 kB ROM

- Crypto: 750 B in ROM
- Plus PSA overhead < 8 kB

Usability

- Enhanced usability through decreased code complexity
  - No driver specific code needed
  - No key handling needed
  - Misuse prevention through simplified functions

Usability

Example: CryptoCell Driver Code vs. PSA Code

CryptoCell Driver

```c
extern uint8_t * key;
extern size_t key_size;

int status;
uint8_t plaintext[] = {
            0x00, 0x01, 0x02, 0x03,
            0x04, 0x05, 0x06, 0x07,
            0x08, 0x09, 0x0A, 0x0B,
            0x0C, 0x0D, 0x0E, 0x0F };
uint8_t iv[16];
uint8_t output[32];
size_t output_length;
size_t size;
size_t offset = 0;
size_t length = sizeof(plaintext);

SaSiAesUserContext_t ctx;
SaSiAesUserKeyData_t user_key;
user_key.pKey = key;
user_key.keySize = key_size;

random_bytes(iv, sizeof(iv));
status = SaSi_AesInit(
            &ctx,
            SASI_AES_ENCRYPT,
            SASI_AES_MODE_CBC,
            SASI_AES_PADDING_NONE);
status = SaSi_AesSetKey(
            &ctx,
            SASI_AES_USER_KEY,
            &user_key,
            sizeof(user_key));
status = SaSi_AesSetIv(&ctx, iv);

do {
    if (length > MAX_AES_BLOCK) {
        size = MAX_AES_BLOCK;
        length -= MAX_AES_BLOCK;
    }
    else {
        size = length;
        length = 0;
    }
    status = SaSi_AesBlock(
                &ctx,
                (plaintext + offset),
                size,
                (output + offset));
    offset += size;
} while ((length > 0));

status = SaSi_AesFinish(
            &ctx, length,
            plaintext,
            sizeof(plaintext),
            output,
            &output_length);
```

PSA Crypto

```c
extern psa_key_id_t id;

psa_status_t status;
psa_algorithm_t algorithm =
            PSA_ALG_CBC_NO_PADDING;

uint8_t plaintext[] = {
            0x00, 0x01, 0x02, 0x03,
            0x04, 0x05, 0x06, 0x07,
            0x08, 0x09, 0x0A, 0x0B,
            0x0C, 0x0D, 0x0E, 0x0F };

size_t output_size =
        PSA_CIPHER_ENCRYPT_OUTPUT_SIZE(
            PSA_KEY_TYPE_AES,
            PSA_ALG_CBC_NO_PADDING,
            sizeof(plaintext))
uint8_t cipher_out[output_size];
size_t output_len;

status = psa_cipher_encrypt(
            id, algorithm,
            plaintext,
            sizeof(plaintext),
            cipher_out,
            output_size,
            &output_len);
```

What’s next?

- Persistent key storage
- Software assisted hardware crypto
- Trusted Execution Environment (TEE) integration