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
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

---

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

---

**Transparent**

**Opaque**

**On-Chip**  **External**
The Zoo of Crypto Backends in the IoT

Driver APIs:
The Zoo of Crypto Backends in the IoT

Driver APIs:

Application
The Zoo of Crypto Backends in the IoT

Driver APIs:

- AES
- HASH
- AES
- ECC P256
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 Platform Security Architecture
- 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

Application

PSA Crypto API
Integration in RIOT

Application

PSA Crypto API

Key Management & Location Dispatch
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

| ID           | Lifetime | Type | Bits       | Policy                          |
|--------------|----------|------|------------|---------------------------------|
| psa_key_attributes_t |          |      |            |                                 |
# Key Attributes

## Lifetime

- **Location:**
  - Key storage location
  - Local volatile, persistent memory or protected hardware storage

- **Persistence:**
  - Volatile, persistent, read-only
Key Attributes
Policy

• Permitted algorithms

• Usage Flags:
  • Encrypt, Decrypt
  • Sign, Verify
  • Export, Copy, Cache
  • Derivation

ID
Lifetime
Type
Bits
Policy

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

SE Driver Dispatch

SE API

SE API

Vendor API

Vendor API

SE Driver

SE Driver

Vendor API

Vendor API

SE Driver

SE Driver

MCU

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

Algorithm Dispatch

SE API

SE API

Specific Algorithm API

Vendor API

Vendor API

Vendor API

Vendor API

SE Driver

Opaq. HW Driver

Trans. HW Driver

Trans. SW Library
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

Nordic nRF52840dk with ARM CryptoCell 310 accelerator

Microchip ATECC608A via I2C
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

AES-128 CBC

On-Chip Time [µs]

PSA Overhead
Key Import
Basic operation

CryptoCell
RIOT Software
ATECC608A
Processing Time

AES-128 CBC

PSA Overhead:

**SE**: max. 40 us (~0.1 %)

**Others**: max. 30 us (~10 %)
Memory Overhead

- AES RIOT Cipher
- HMAC SHA-256 CryptoCell
- ECDSA ATECC608A
- All three backends

ROM [KiB]

- PSA Crypto API
- Secure Element Management
- Key Slot Management
- Location/Algorithm Dispatcher
- Glue Code
- Backend

RAM [KiB]
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

Reduced crypto memory usage by 70%
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

ECDSA

- Generate
- Sign
- Verify

CryptoCell

- Generate Key
- Sign
- Verify

ATECC608A

- Generate Key
- Sign
- Verify

Time [µs]

- PSA Overhead
- Key Import
- Basic operation
Processing Time

ECDSA

PSA Overhead:

SE: max. 25 us (~0.025 %)

CryptoCell: max. 30 us (~0.1 %)
# Modeling Crypto Hardware Features

**Application**

| Algorithms: |
|------------|
| AES 128 CBC |
| AES 256 CTR |
Modeling Crypto Hardware Features

Application

Algorithms:
- AES 128 CBC
- AES 256 CTR

Build Appl.

CIPHER_AES_128_CBC

CIPHER_AES_256_CTR

KEY_SLOT_COUNT=2

HAS_PERIPH_AES_128
Modeling Crypto Hardware Features

Application

Algorithms:
• AES 128 CBC
• AES 256 CTR

Build Appl.

CIPHER_AES_128_CBC

CIPHER_AES_256_CTR

KEY_SLOT_COUNT=2

MAX_KEY_SIZE=16

Has HW AES 128?

Yes

Build AES 128 Hardware

HAS_PERIPH_AES_128

Application Algorithms:
• AES 128 CBC
• AES 256 CTR
Modeling Crypto Hardware Features

Application

Algorithms:
- AES 128 CBC
- AES 256 CTR

Build Appl.

CIPHER_AES_128_CBC

CIPHER_AES_256_CTR

KEY_SLOT_COUNT=2

Build AES 128

Has HW AES 128?

Yes

Build AES 128 Hardware

MAX_KEY_SIZE=16

No

Build AES 256

Has HW AES 256?

Yes

Build AES 256 Software

MAX_KEY_SIZE=32

No

Has HW AES 128?

Yes

Binary

HAS_PERIPH_AES_128

No

MAX_KEY_SIZE=32

Has HW AES 256?