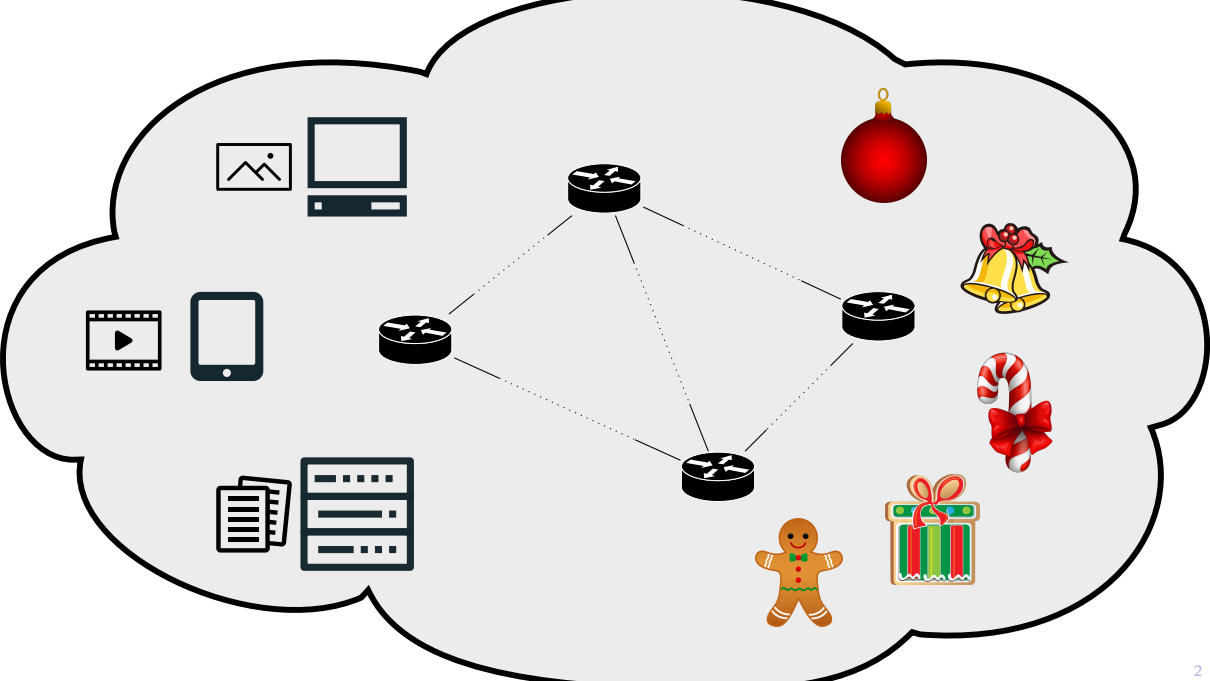
From PUF to Protected Key: Security Components for the IoT

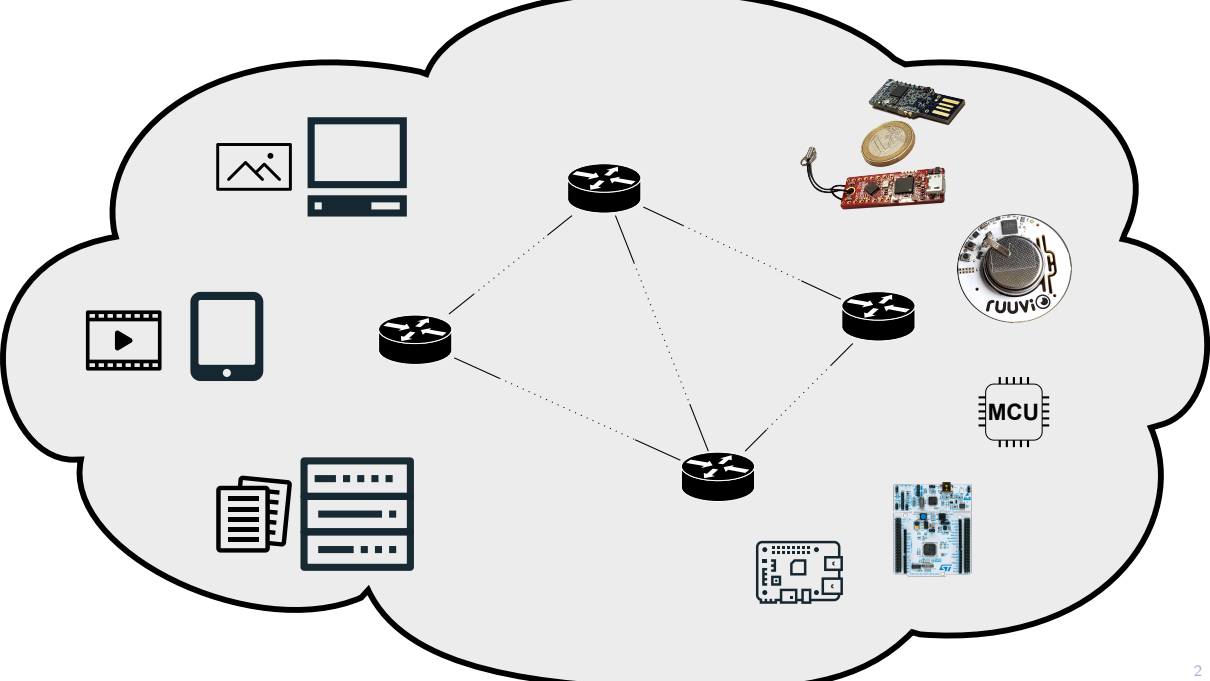
iNET Xmas '21

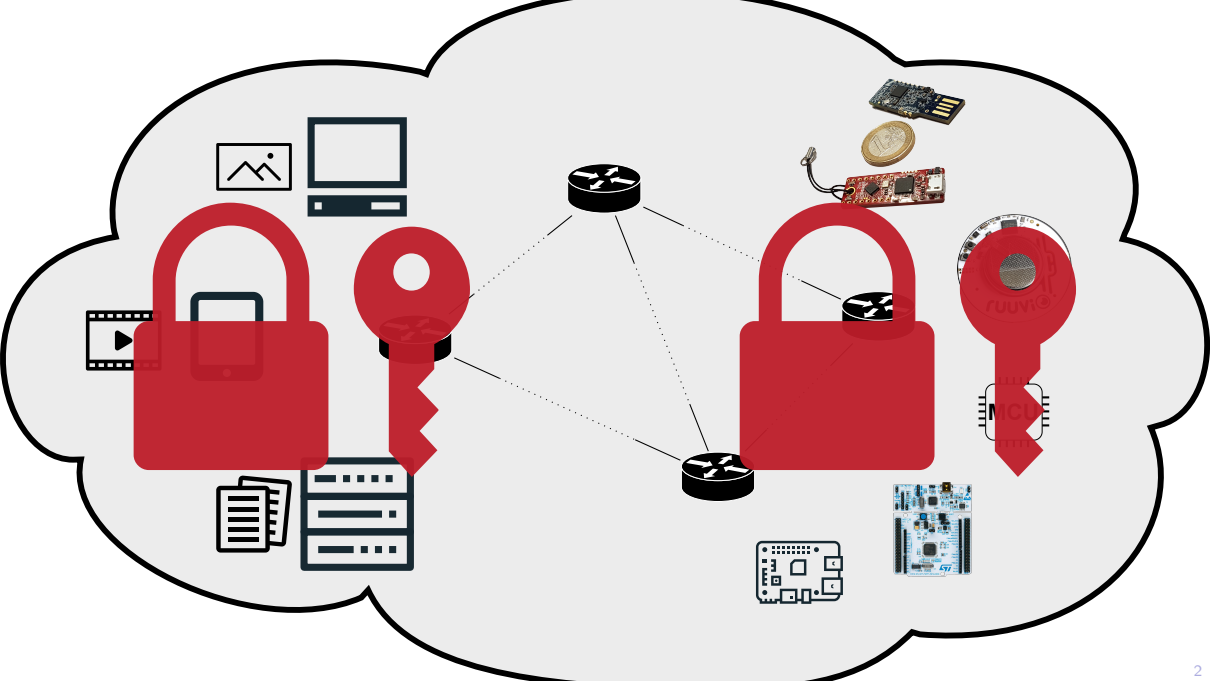
Peter Kietzmann
peter.kietzmann@haw-hamburg.de

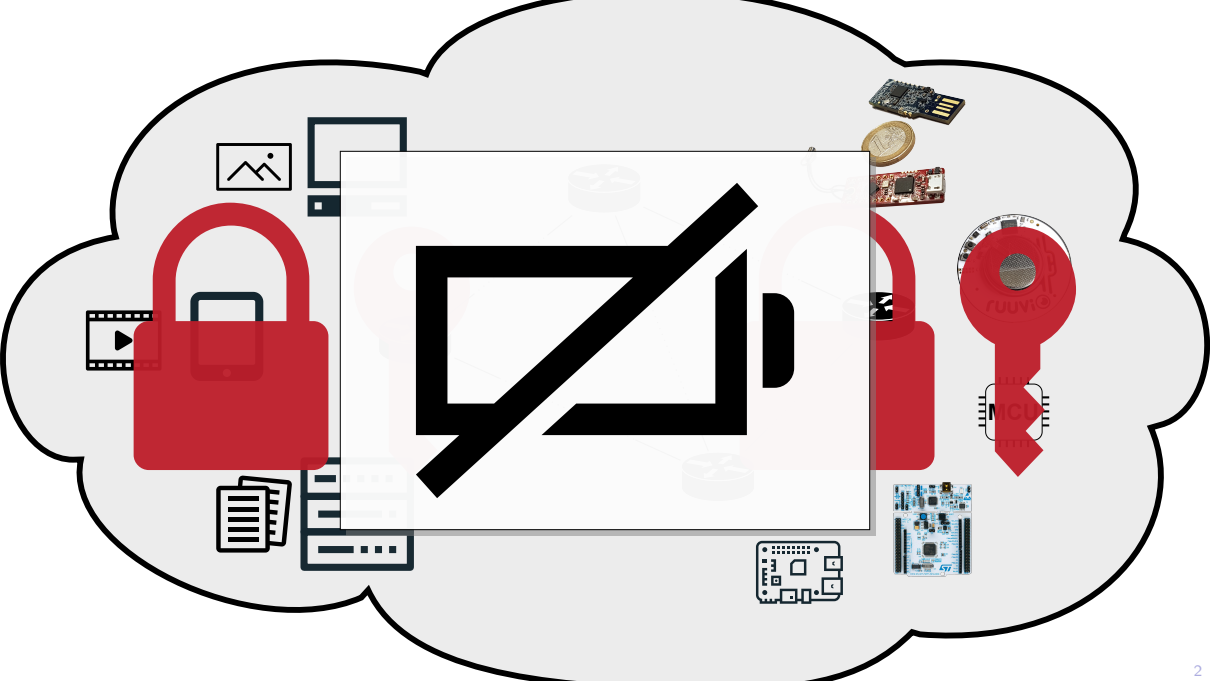
HAW Hamburg

December 13, 2021

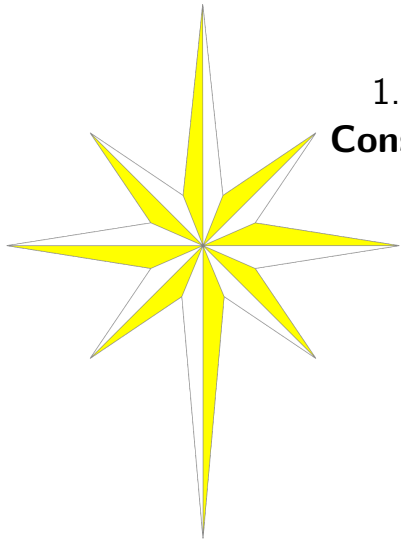



MCU


MCU

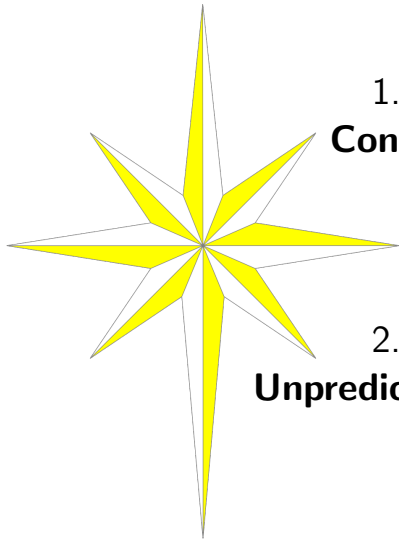

MCU

Outline – Advent Season for IoT Security



1. Advent
Constraints

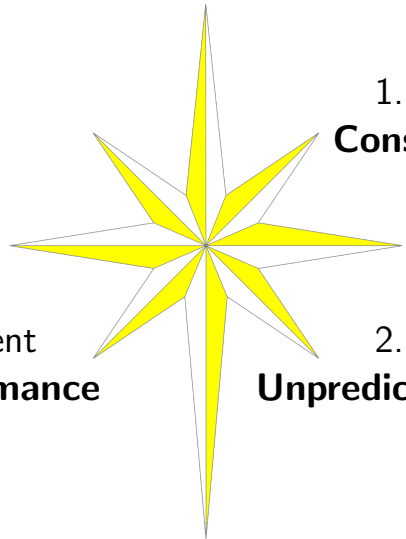
Outline – Advent Season for IoT Security



1. Advent
Constraints

2. Advent
Unpredictability

Outline – Advent Season for IoT Security



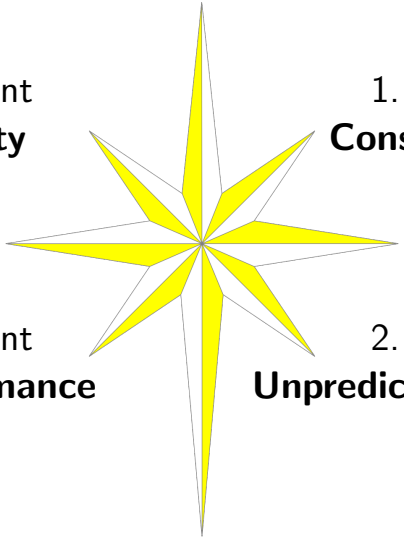
1. Advent
Constraints

3. Advent
Performance

2. Advent
Unpredictability

Outline – Advent Season for IoT Security

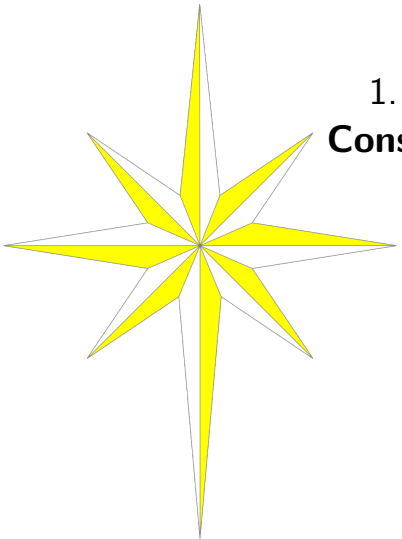
4. Advent
Usability



1. Advent
Constraints

3. Advent
Performance

2. Advent
Unpredictability



1. Advent

Constraints

Classes of Constrained Devices

RFC 7228

Name	data size (e.g., RAM)	code size (e.g., Flash)
Class 0, C0	<< 10 KiB	<< 100 KiB
Class 1, C1	~ 10 KiB	~ 100 KiB
Class 2, C2	~ 50 KiB	~ 250 KiB

Table 1: Classes of Constrained Devices (KiB = 1024 bytes)

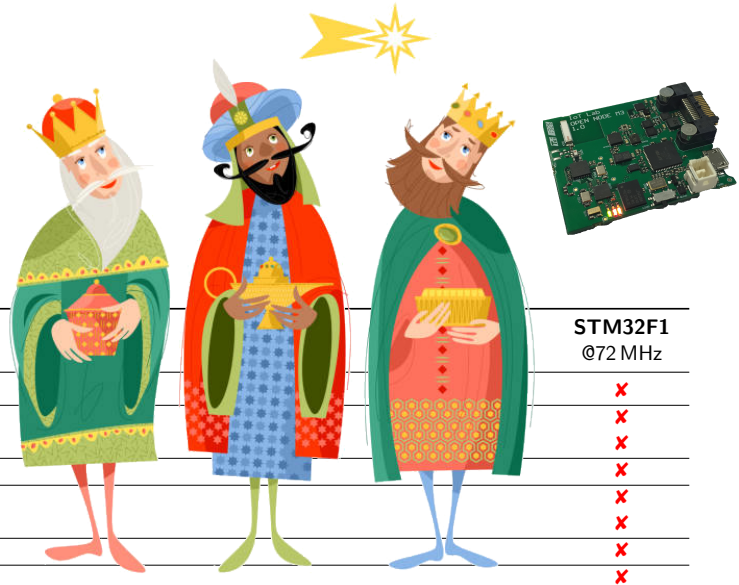
Platform Overview

Feature \ Device	**STM32F1** @72 MHz
TRNG	✘
SHA-256	✘
HMAC-SHA256	✘
AES-128	✘
ECC	✘
ECDSA / ECDH	✘
TEE	✘
Key Storage	✘

Platform Overview

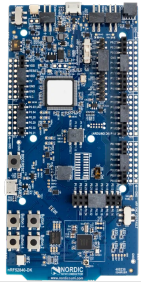
Feature \ Device	1
TRNG	
SHA-256 HMAC-SHA256	
AES-128	
ECC ECDSA / ECDH	
TEE	
Key Storage	

Platform Overview



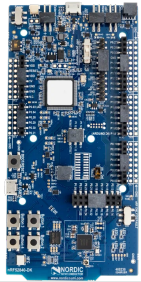
Feature \ Device	STM32F1 @72 MHz
TRNG	✘
SHA-256	✘
HMAC-SHA256	✘
AES-128	✘
ECC	✘
ECDSA / ECDH	✘
TEE	✘
Key Storage	✘

Platform Overview



Feature \ Device	**nRF52840** @64 MHz	**STM32F1** @72 MHz
TRNG	✓	✘
SHA-256	✓	✘
HMAC-SHA256	✓	✘
AES-128	**ECB**, CTR, **CBC**, ...	✘
ECC	**secp256**k/**r1**, ...	✘
ECDSA / ECDH	✓	✘
TEE	✘	✘
Key Storage	✘	✘

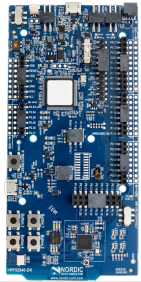
Platform Overview



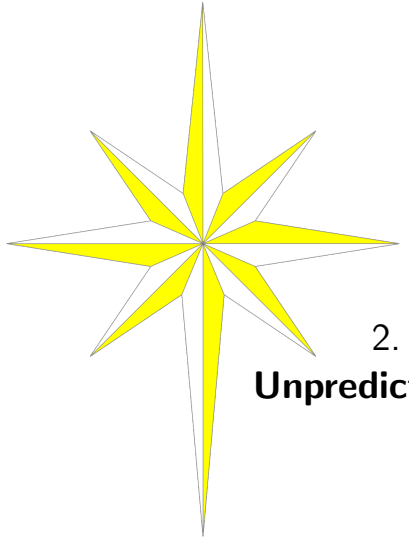
Feature \ Device	**nRF52840** @64 MHz	**nRF9160** @64 MHz	**STM32F1** @72 MHz
TRNG	✓	✓	✘
SHA-256	✓	✓	✘
HMAC-SHA256	✓	✓	✘
AES-128	**ECB**, CTR, **CBC**, ...	**ECB**, CTR, **CBC**, ...	✘
ECC	**secp256**k/**r1**, ...	**secp256**k/**r1**, ...	✘
ECDSA / ECDH	✓	✓	✘
TEE	✘	✓	✘
Key Storage	✘	✓	✘

Platform Overview

I2C



Feature \ Device	**ATECC608A** I2C@400kbps	**nRF52840** @64 MHz	**nRF9160** @64 MHz	**STM32F1** @72 MHz
TRNG	✓	✓	✓	✘
SHA-256	✓	✓	✓	✘
HMAC-SHA256	✓	✓	✓	✘
AES-128	**ECB**, GCM	**ECB**, CTR, **CBC**, ...	**ECB**, CTR, **CBC**, ...	✘
ECC	**secp256r1** (**P-256**)	**secp256**k/**r1**, ...	**secp256**k/**r1**, ...	✘
ECDSA / ECDH	✓	✓	✓	✘
TEE	(✓)	✘	✓	✘
Key Storage	✓	✘	✓	✘

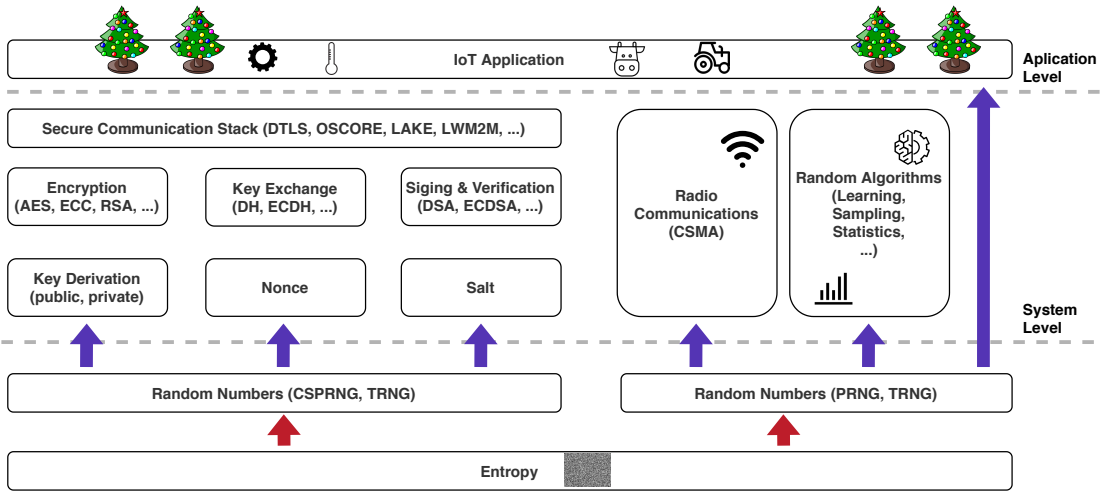


2. Advent

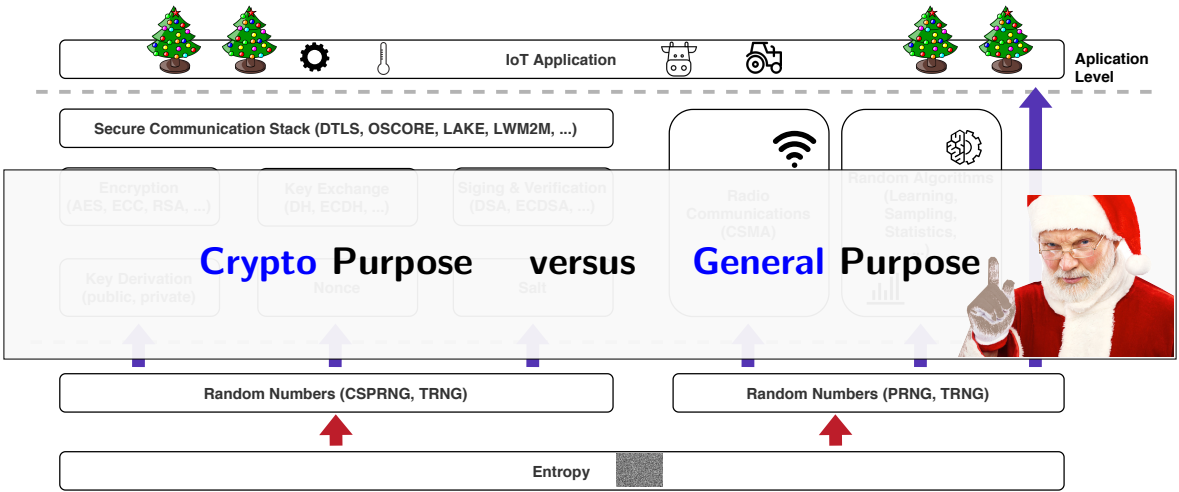
Unpredictability

Random Numbers

Applications of Random Numbers



Applications of Random Numbers



Randomness Requirements

Cryptographically Secure RNG

- Statistically indistinguishable from truly random
- Infeasible to guess future or past sequences
- High entropy seeding
- Feasible complexity for IoT devices

Optimizing parameters:
Health tests & re-seeding

General Purpose RNG

- Uniformly distributed & statistically independent numbers
- Differing start values across devices & resets
- Fast and efficient computation with little memory

Randomness Requirements

Cryptographically Secure RNG

- Statistically indistinguishable from truly random
- Infeasible to guess future or past sequences
- High entropy seeding
- Feasible complexity for IoT devices

Optimizing parameters:
Health tests & re-seeding

General Purpose RNG

- Uniformly distributed & statistically independent numbers
- Differing start values across devices & resets
- Fast and efficient computation with little memory

Randomness Requirements

C

... and what is Entropy?

"I thought of calling it 'information', but the word was overly used, so I decided to call it "uncertainty". [...] Von Neumann told me, 'You should call it entropy, for two reasons. In the first place your uncertainty function has been used in statistical mechanics under that name, so it already has a name. In the second place, and more important, nobody knows what entropy really is, so in a debate you will always have the advantage.' "

—— Conversation between Claude Shannon and John von Neuman

`https://www.jstor.org/stable/24923125`

Randomness Requirements

Cryptographically Secure RNG

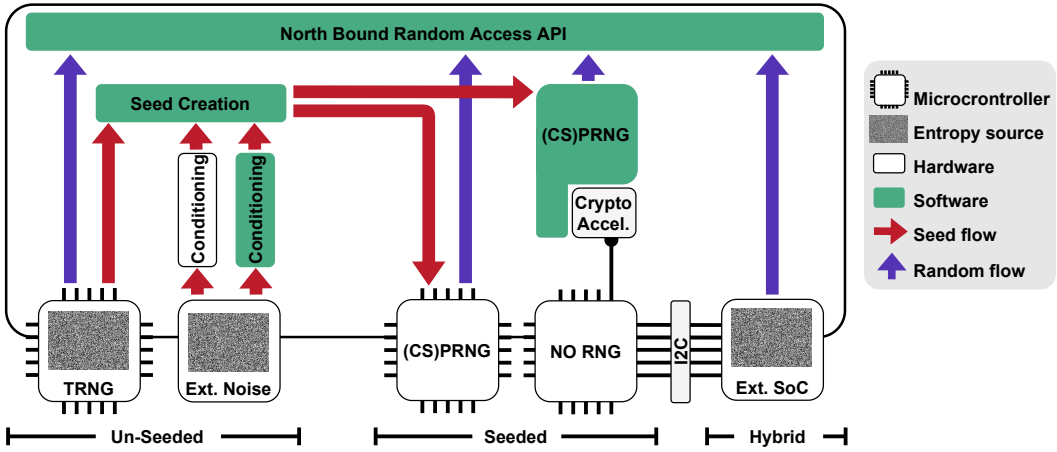
- Statistically indistinguishable from truly random
- Infeasible to guess future or past sequences
- High entropy seeding
- Feasible complexity for IoT devices

Optimizing parameters:
Health tests & re-seeding

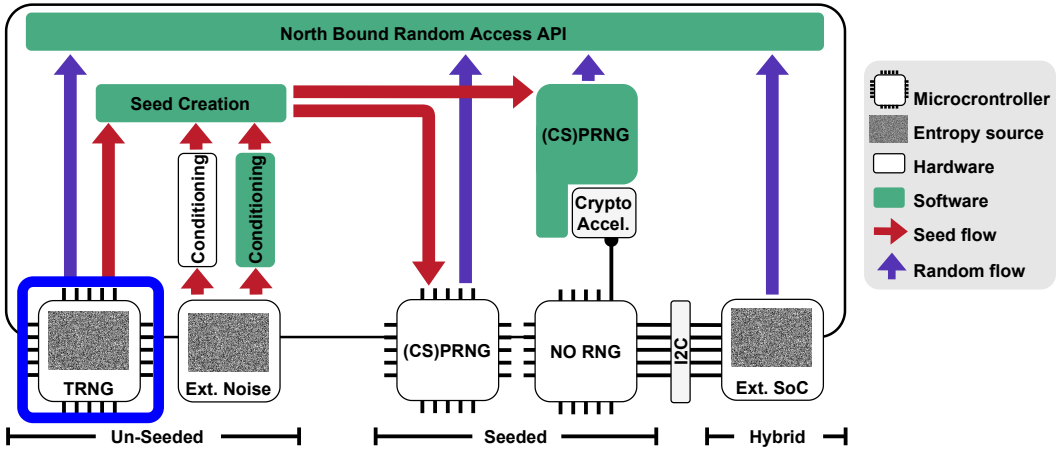
General Purpose RNG

- Uniformly distributed & statistically independent numbers
- Differing start values across devices & resets
- Fast and efficient computation with little memory

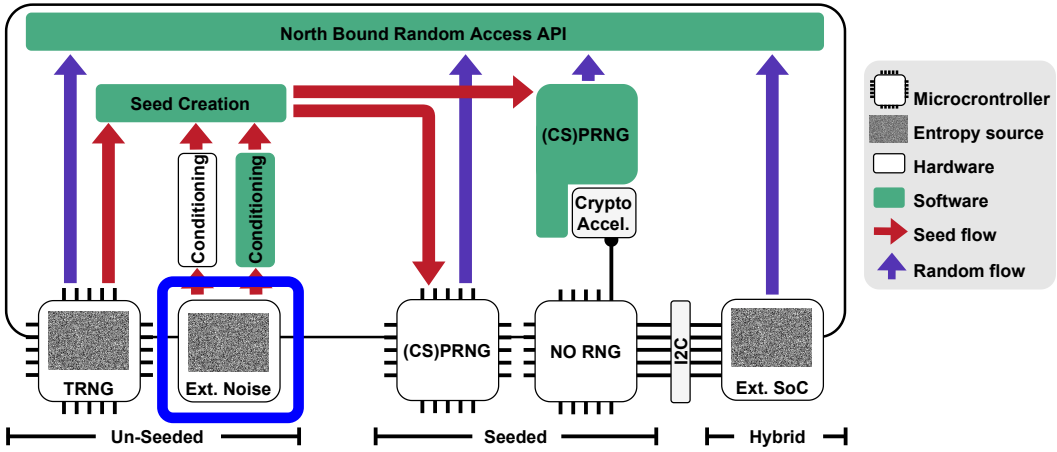
Randomness Sources in the IoT



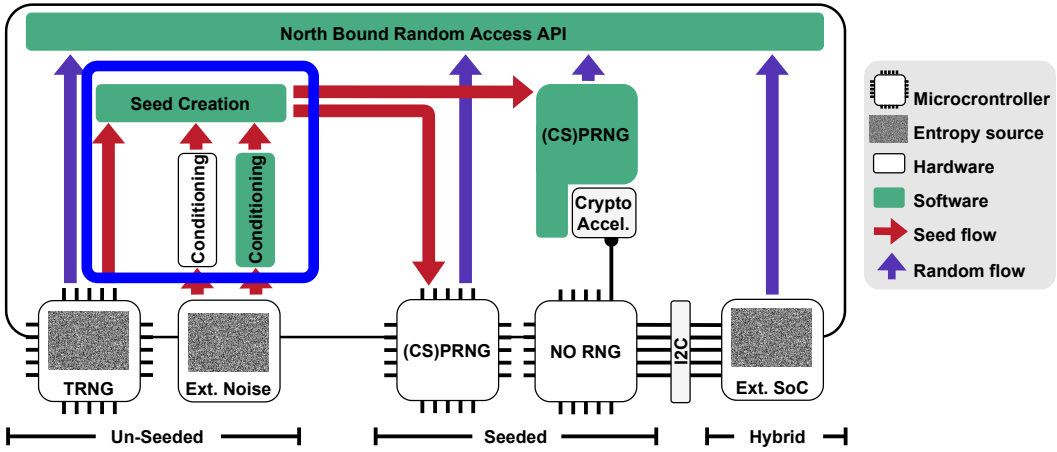
Randomness Sources in the IoT



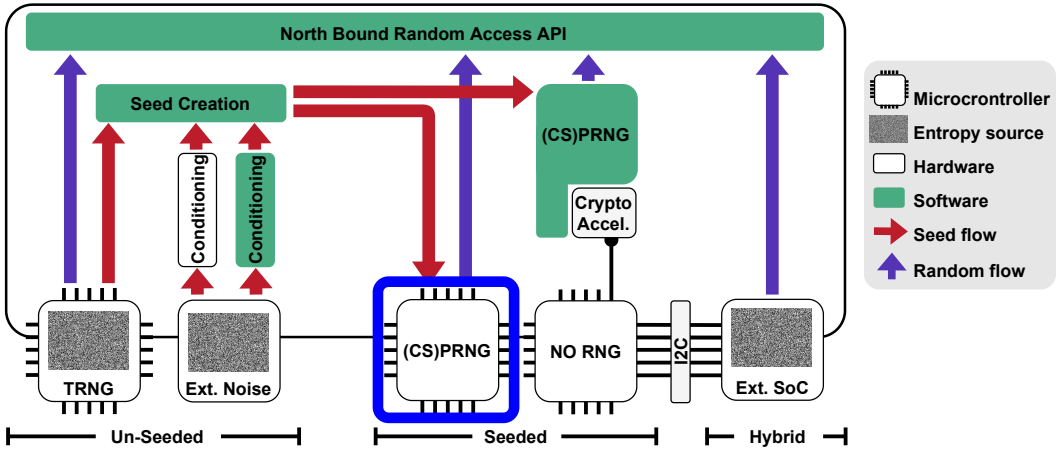
Randomness Sources in the IoT



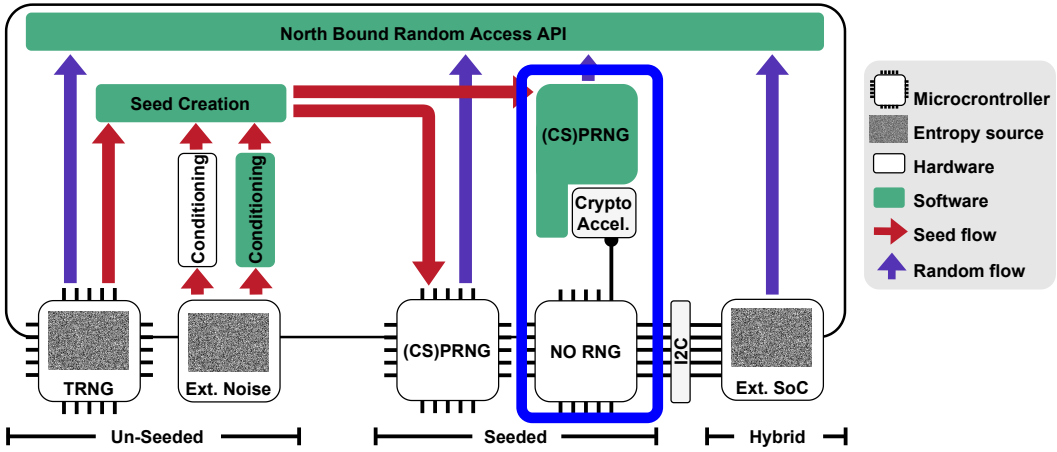
Randomness Sources in the IoT



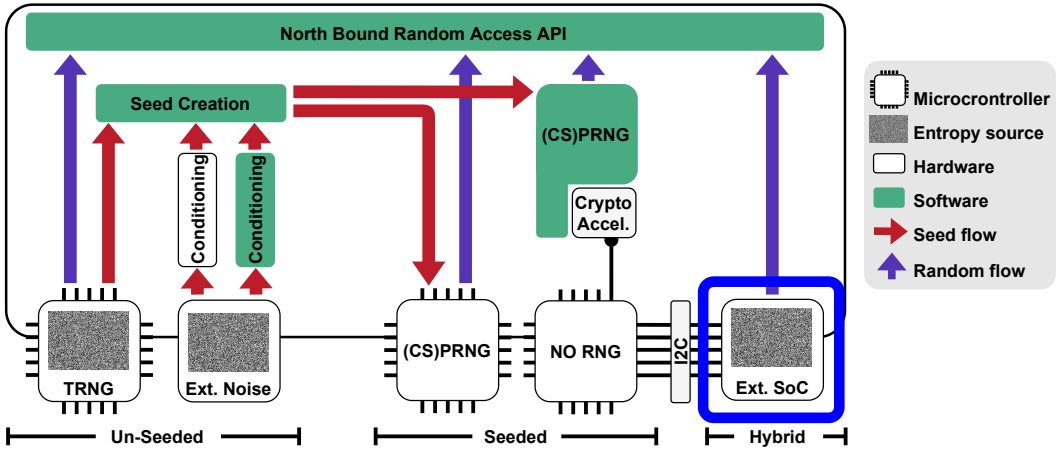
Randomness Sources in the IoT



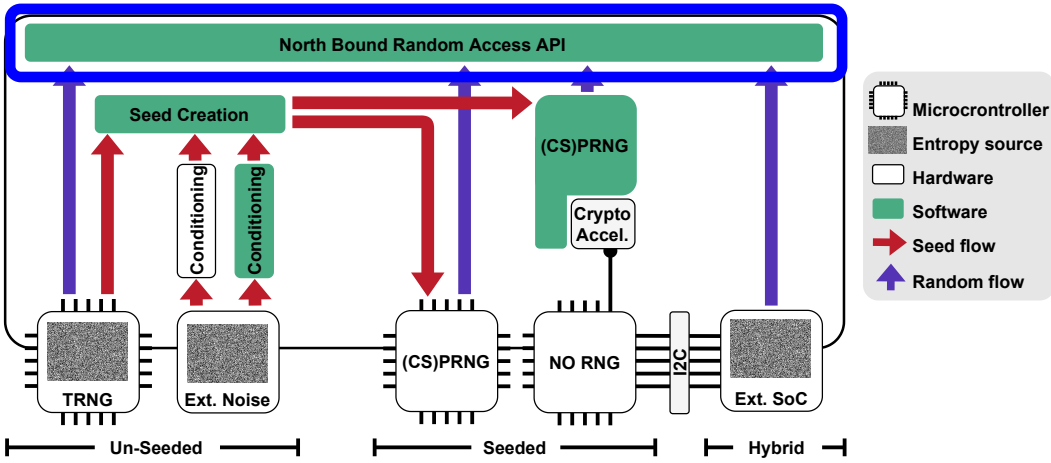
Randomness Sources in the IoT



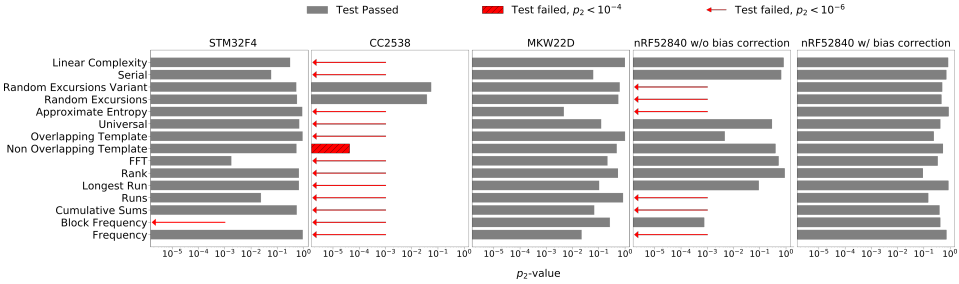
Randomness Sources in the IoT



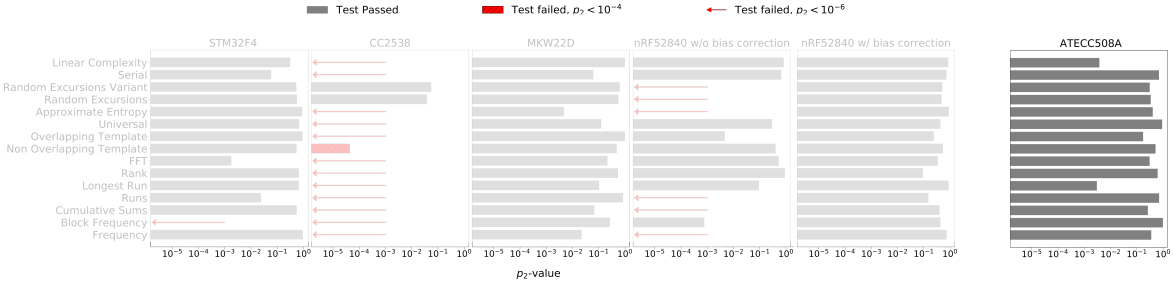
Randomness Sources in the IoT



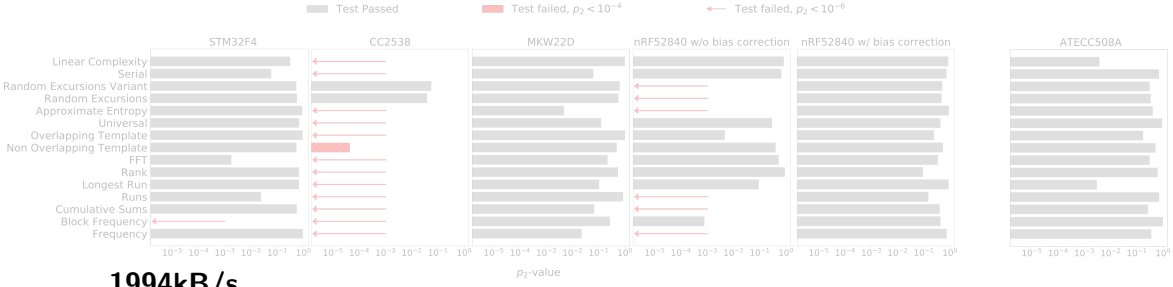
Hardware Based Randomness



Hardware Based Randomness



Hardware Based Randomness



1994kB/s

503kB/s

316kB/s

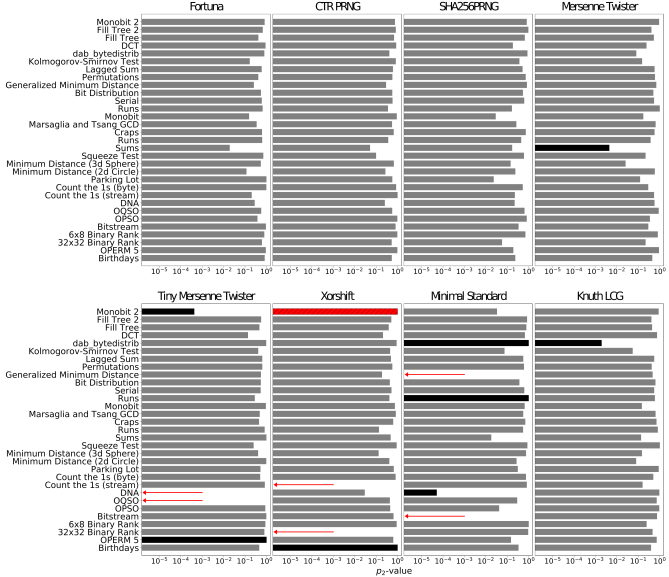
15kB/s

6kB/s

3kB/s

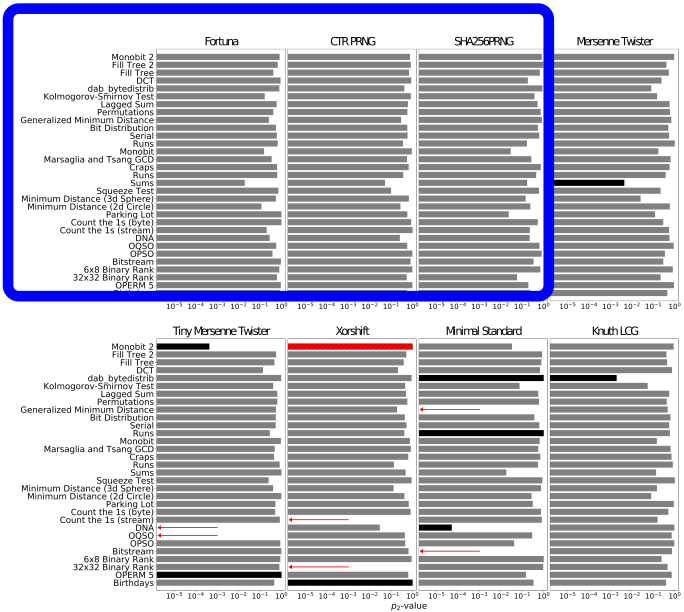
Hardware Based Randomness

Software Based Randomness

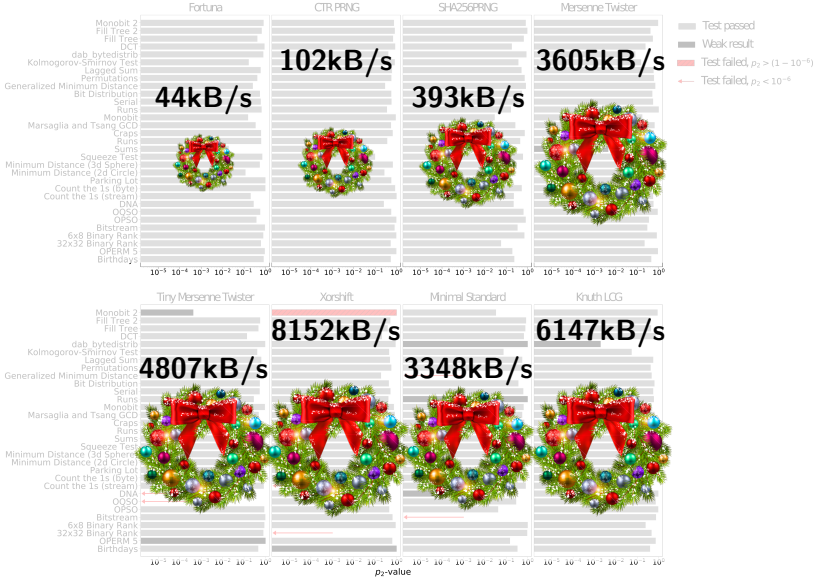


Software Based Randomness

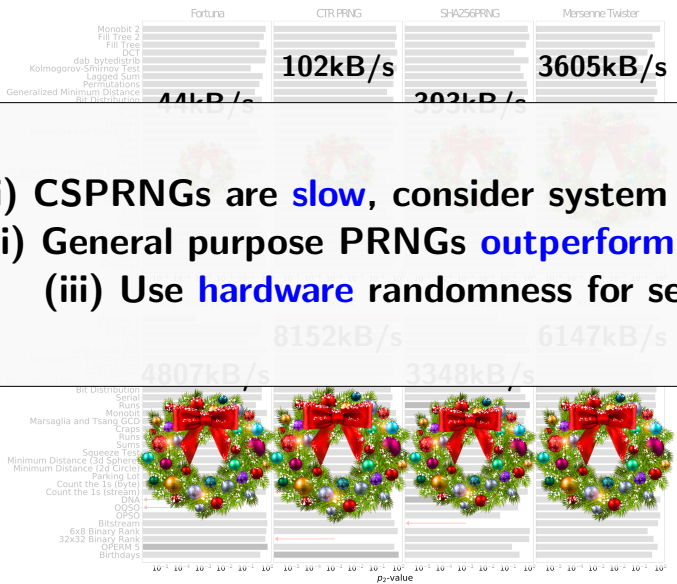
CSPRNGs



Software Based Randomness



Software Based Randomness

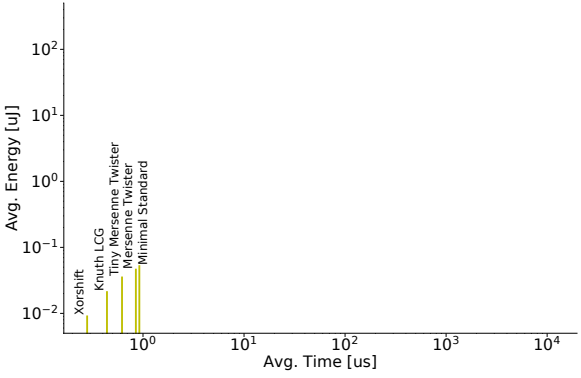


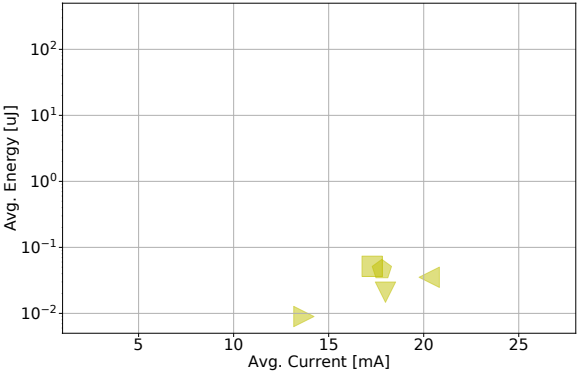
(i) CSPRNGs are slow, consider system resources

(ii) General purpose PRNGs outperform hardware

(iii) Use hardware randomness for seeding

Hardware versus Software Randomness





Hardware-generated:

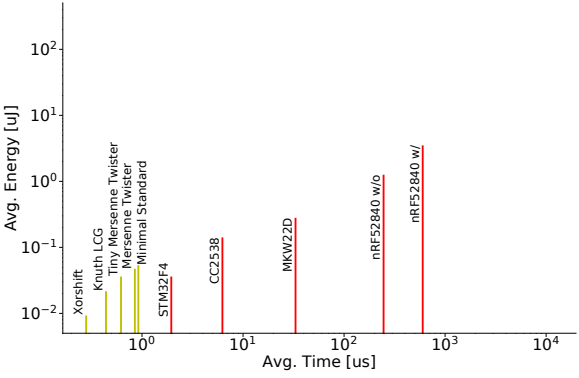
CSPRNGs on STM32F4:

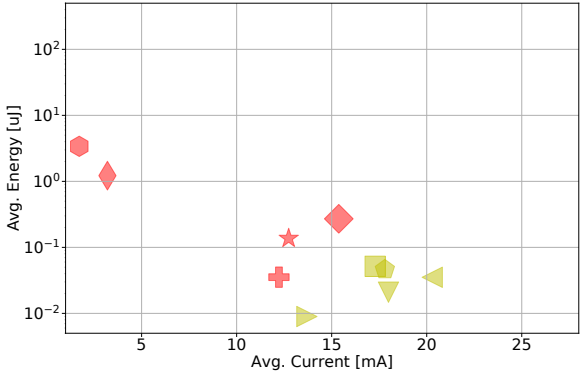
PRNGs on STM32F4:

- Mersenne Twister
- Tiny Mersenne Twister
- Xorshift
- Minimal Standard
- Knuth LCG

Other Platforms:

Hardware versus Software Randomness





Hardware-generated:

- STM32F4 (TRNG)

CSPRNGs on STM32F4:

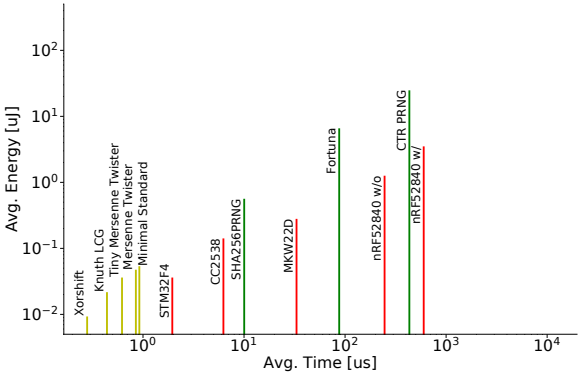
PRNGs on STM32F4:

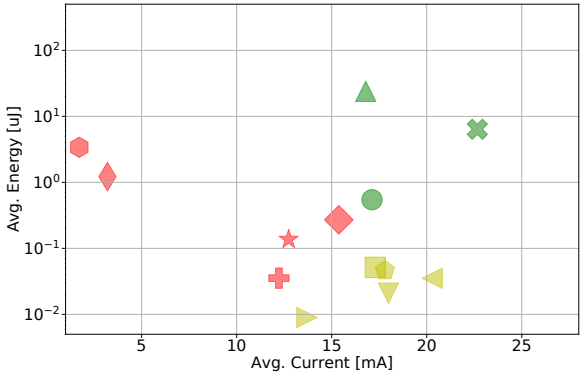
- Mersenne Twister
- Tiny Mersenne Twister
- Xorshift
- Minimal Standard
- Knuth LCG

Other Platforms:

- CC2538 (HWPRNG)
- MKW22D (TRNG)
- nRF52840 (TRNG) w/o bias correction
- nRF52840 (TRNG) w/ bias correction

Hardware versus Software Randomness





Hardware-generated:

- STM32F4 (TRNG)

CSPRNGs on STM32F4:

- Fortuna
- CTR PRNG
- SHA256PRNG

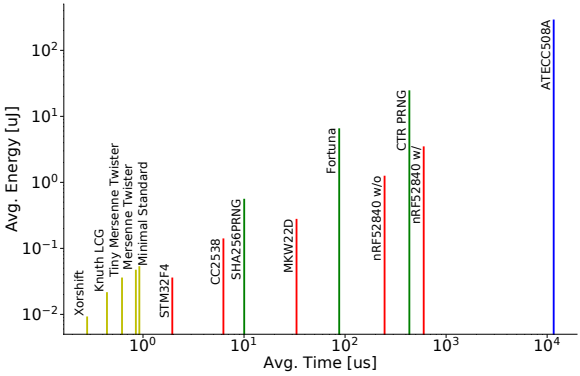
PRNGs on STM32F4:

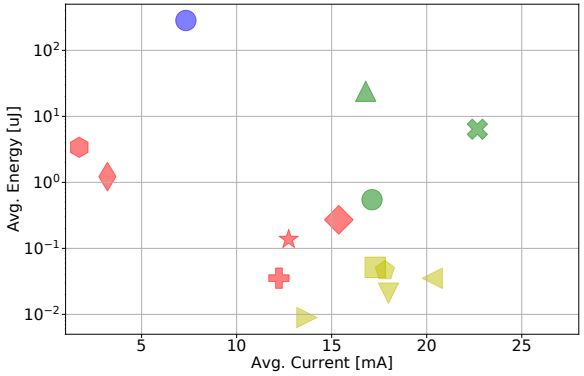
- Mersenne Twister
- Tiny Mersenne Twister
- Xorshift
- Minimal Standard
- Knuth LCG

Other Platforms:

- CC2538 (HWPRNG)
- MKW22D (TRNG)
- nRF52840 (TRNG) w/o bias correction
- nRF52840 (TRNG) w/ bias correction

Hardware versus Software Randomness





Hardware-generated:
- STM32F4 + ATECC508A
- STM32F4 (TRNG)

CSPRNGs on STM32F4:
- Fortuna
- CTR PRNG
- SHA256PRNG

PRNGs on STM32F4:
- Mersenne Twister
- Tiny Mersenne Twister
- Xorshift
- Minimal Standard
- Knuth LCG

Other Platforms:
- CC2538 (HWPRNG)
- MKW22D (TRNG)
- nRF52840 (TRNG) w/o bias correction
- nRF52840 (TRNG) w/ bias correction

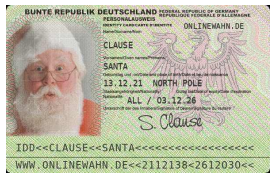

PUF!!! for the Commons

Physical Unclonable Functions

- Digital device fingerprint based on manufacturing variations
- Intrinsic properties are hidden in physical structure
  $\rightarrow$ Hard to predict, guess, and unclonable
- Secret identifies a device
- Like fingerprint, PUF affected by noise

Physical Unclonable Functions

- Digital device fingerprint based on manufacturing variations
- Intrinsic properties are hidden in physical structure
  $\rightarrow$ Hard to predict, guess, and unclonable
- Secret identifies a device
- Like fingerprint, PUF affected by noise



+

PUF Applications & Parameters

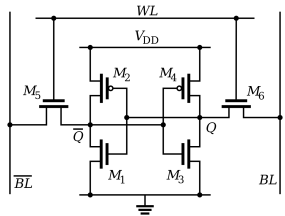
	Applications	Requirements
Noise	▶ RNG, PRNG seeding, ...	▶ Intra-device variations
Identity	▶ Secret key generation ▶ Secure (key) storage ▶ Device identification, authentication ▶ App–to–device binding (i.e., secure boot)	▶ Inter-device variations ▶ Unique ▶ Unpredictable ▶ Unclonable ▶ Reproducible

Practical PUFs in the IoT

The SRAM PUF

- Variety of PUFs: Ring oscillators latency, MEMS, Flip-flops, Megnetics, Optics, ...
- **SRAM**: Available on 'all' IoT devices
- Startup state of uninitialized memory cells: 0, 1, or fluctuating
  $\rightarrow$ Unique pattern + random flips

Secret persists only short time after startup :)



SRAM Evaluation

Inter-Device SRAM Analysis with 700 Nodes

Hamming Distance: Example I

Input A:	1	0	1	0
Input B:	1	0	1	0
Hamm. Dist.:	0 Bit			
Frac. Hamm. Dist.:	0			

SRAM Evaluation

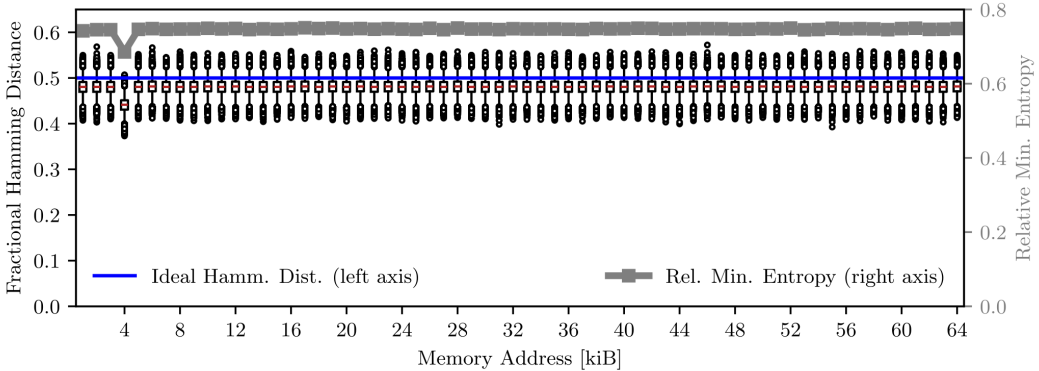
Inter-Device SRAM Analysis with 700 Nodes

Hamming Distance: Example II

Input A:	1	0	1	0
Input B:	0	1	0	1
Hamm. Dist.:	4 Bit			
Frac. Hamm. Dist.:	1			

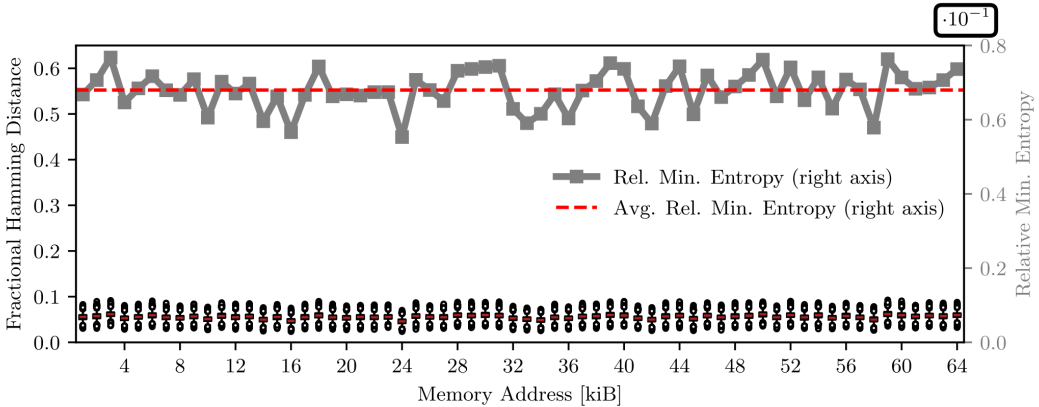
SRAM Evaluation

Inter-Device SRAM Analysis with 700 Nodes



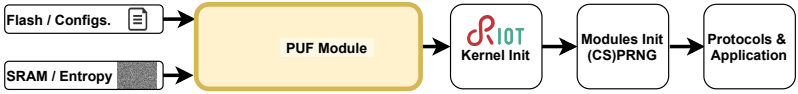
SRAM Evaluation

Intra-Device SRAM Analysis on one Node



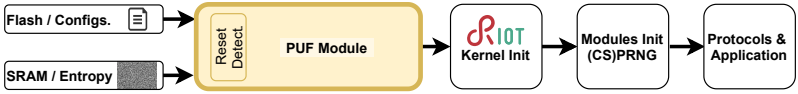
Integration of PUFs in RIOT

- Startup:
  - Module gets configurations & SRAM memory
  - `reset_handler` startup point → Untouched, uninitialized memory
- Reset detection prevents PUF on absent power-off cycle
- Generation of:
  - General purpose **seed** (from simple DEK hash)
  - Cryto secure **seed** (from SHA256 hash)
  - Secure **key** (from *Fuzzy Extractor*)
- Seeds & keys stored in `.noinit` to persist startup



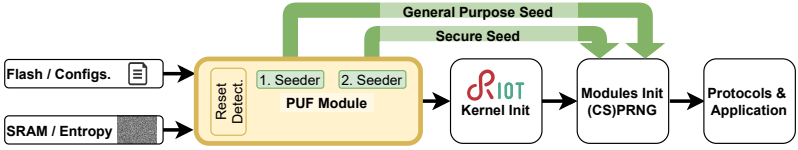
Integration of PUFs in RIOT

- Startup:
  - Module gets configurations & SRAM memory
  - `reset_handler` startup point → Untouched, uninitialized memory
- Reset detection prevents PUF on absent power-off cycle
- Generation of:
  - General purpose **seed** (from simple DEK hash)
  - Cryto secure **seed** (from SHA256 hash)
  - Secure **key** (from *Fuzzy Extractor*)
- Seeds & keys stored in `.noinit` to persist startup



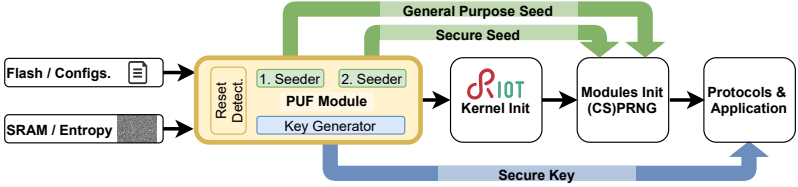
Integration of PUFs in RIOT

- Startup:
  - Module gets configurations & SRAM memory
  - `reset_handler` startup point → Untouched, uninitialized memory
- Reset detection prevents PUF on absent power-off cycle
- Generation of:
  - General purpose **seed** (from simple DEK hash)
  - Cryto secure **seed** (from SHA256 hash)
  - Secure **key** (from *Fuzzy Extractor*)
- Seeds & keys stored in `.noinit` to persist startup



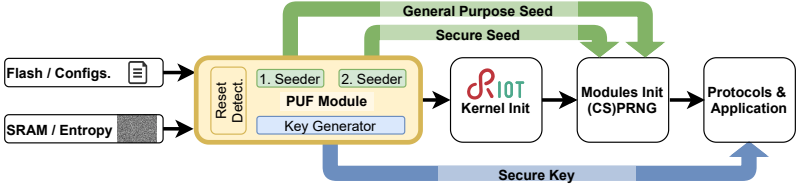
Integration of PUFs in RIOT

- Startup:
  - Module gets configurations & SRAM memory
  - `reset_handler` startup point → Untouched, uninitialized memory
- Reset detection prevents PUF on absent power-off cycle
- Generation of:
  - General purpose **seed** (from simple DEK hash)
  - Cryto secure **seed** (from SHA256 hash)
  - Secure **key** (from *Fuzzy Extractor*)
- Seeds & keys stored in `.noinit` to persist startup



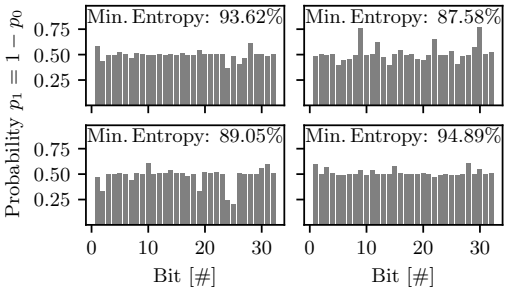
Integration of PUFs in RIOT

- Startup:
  - Module gets configurations & SRAM memory
  - `reset_handler` startup point → Untouched, uninitialized memory
- Reset detection prevents PUF on absent power-off cycle
- Generation of:
  - General purpose **seed** (from simple DEK hash)
  - Cryto secure **seed** (from SHA256 hash)
  - Secure **key** (from *Fuzzy Extractor*)
- Seeds & keys stored in `.noinit` to persist startup



PUF Seed Evaluation

General Purpose Seeds



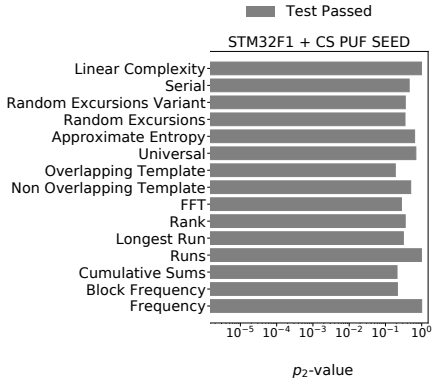
PUF Seed Evaluation

General Purpose Seeds

PUF Seed Evaluation

General Purpose Seeds

Cryptographically Secure Seeds



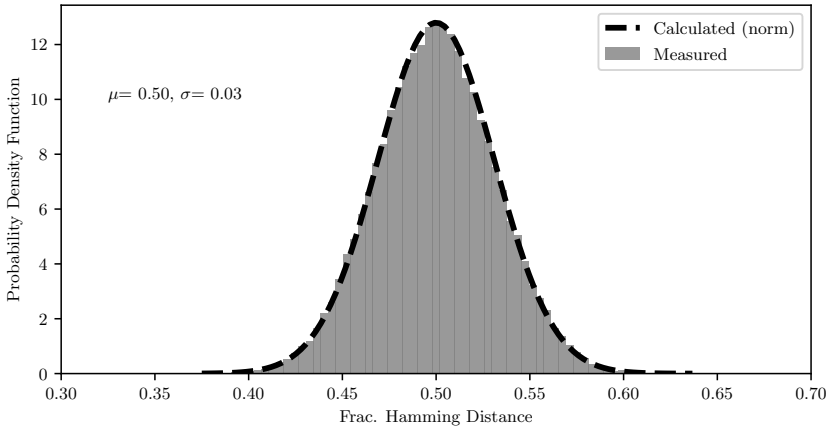
PUF Seed Evaluation

General Purpose Seeds

Cryptographically Secure Seeds

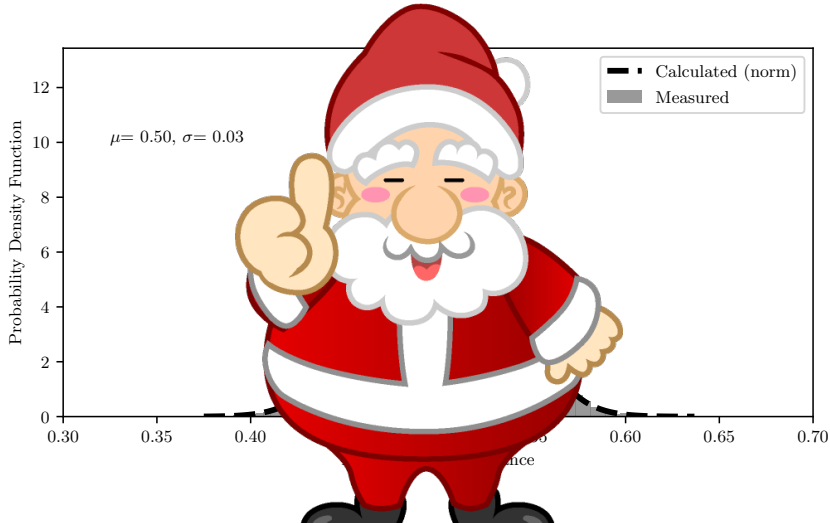
PUF Key Evaluation

Uniqueness of Keys between > 300 Nodes



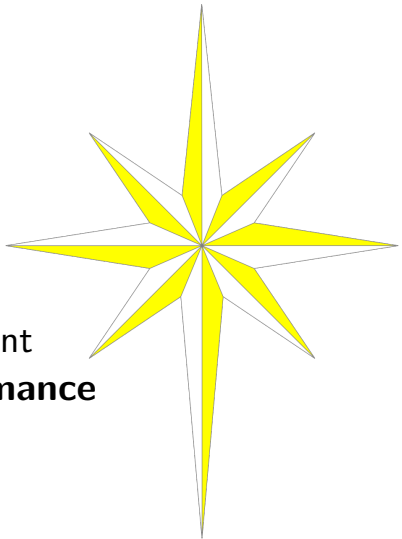
PUF Key Evaluation

Uniqueness of Keys between > 300 Nodes



3. Advent

Performance



Symmetric Crypto - Processing Time for 512 Byte Inputs

Software

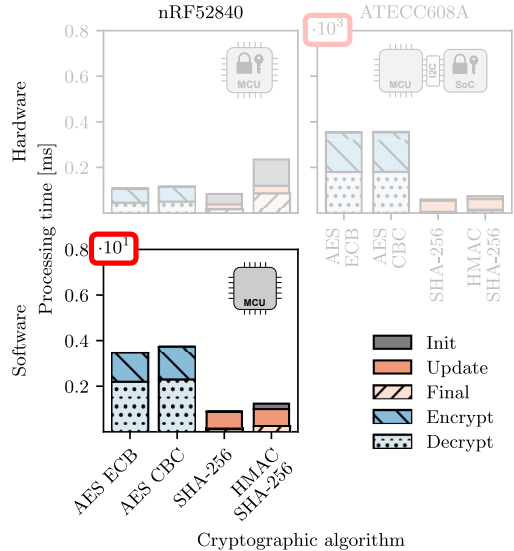
- Exec. on nRF52840 CPU
- Balanced RIOT core implementation
- Ciphers ↑ hashes for `memcpy`

nRF52840 (Hardware)

- Exec. on nRF52840 accelerator
- Flexible and highly configurable

ATECC608A (Hardware)

- Exec. on ext. device connected to nRF52840
- Secure memory for encryption keys



Symmetric Crypto - Processing Time for 512 Byte Inputs

Software

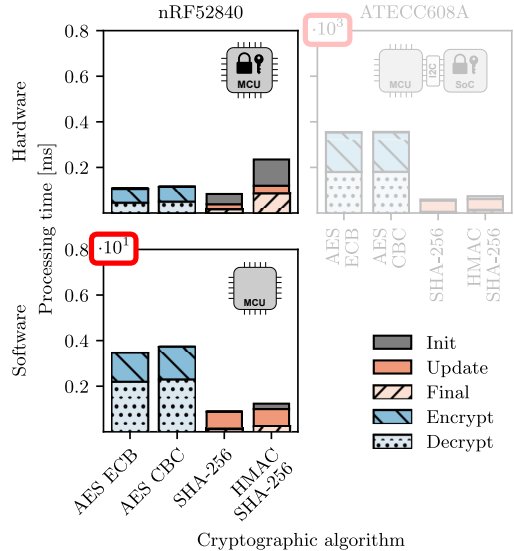
- Exec. on nRF52840 CPU
- Balanced RIOT core implementation
- Ciphers ↑ hashes for `memcpy`

nRF52840 (Hardware)

- Exec. on nRF52840 accelerator
- Flexible and highly configurable

ATECC608A (Hardware)

- Exec. on ext. device connected to nRF52840
- Secure memory for encryption keys



Symmetric Crypto - Processing Time for 512 Byte Inputs

Software

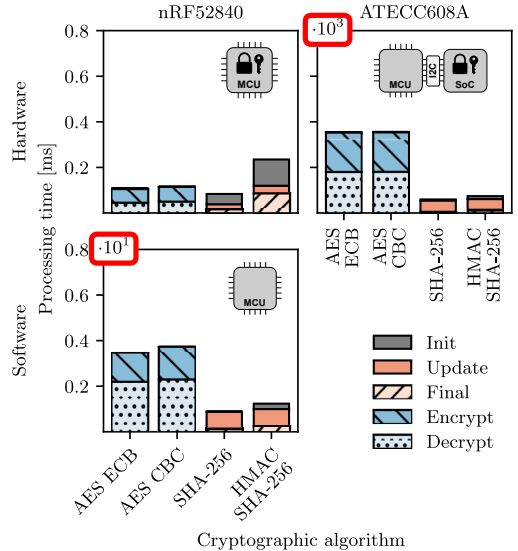
- Exec. on nRF52840 CPU
- Balanced RIOT core implementation
- Ciphers ↑ hashes for `memcpy`

nRF52840 (Hardware)

- Exec. on nRF52840 accelerator
- Flexible and highly configurable

ATECC608A (Hardware)

- Exec. on ext. device connected to nRF52840
- Secure memory for encryption keys



Symmetric Crypto - Processing Time for 512 Byte Inputs

nRF52840 ATECC608A

When implemented in hardware:
Ciphers gain a factor of 10–30, Hashes gain a factor of 5–10

External device reveals severe overhead by
(i) hardware control and (ii) I2C transport

AES ECB AES CBC SHA-256 HMAC SHA-256

Cryptographic algorithm

ECC Crypto - Processing Time

ECDSA / ECDH on NIST P-256

uECC

- Minimal, optimized library
- Static lookup tables

Relic

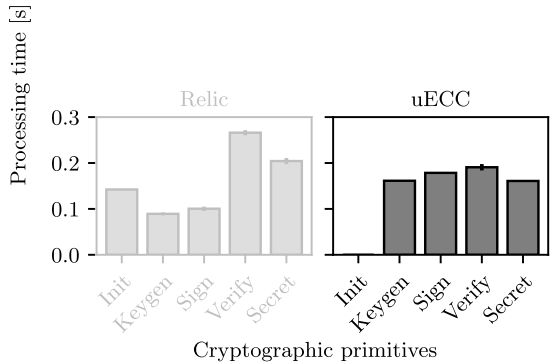
- Feature-rich crypto-toolkit
- Flexible and highly configurable

ATECC608A

- Constrained to single curve
- Secure memory for private keys

nRF52840

- Hardware support for > 15 elliptic curves



ECC Crypto - Processing Time

ECDSA / ECDH on NIST P-256

uECC

- Minimal, optimized library
- Static lookup tables

Relic

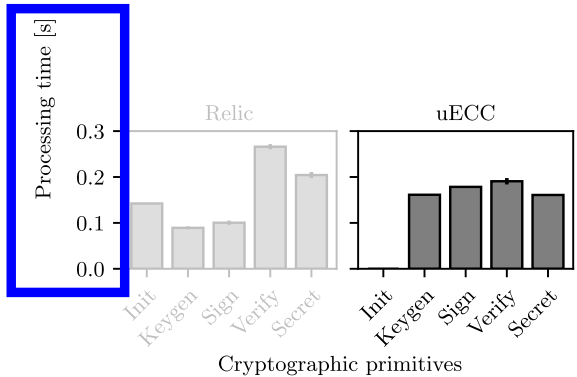
- Feature-rich crypto-toolkit
- Flexible and highly configurable

ATECC608A

- Constrained to single curve
- Secure memory for private keys

nRF52840

- Hardware support for > 15 elliptic curves



ECC Crypto - Processing Time

ECDSA / ECDH on NIST P-256

uECC

- Minimal, optimized library
- Static lookup tables

Relic

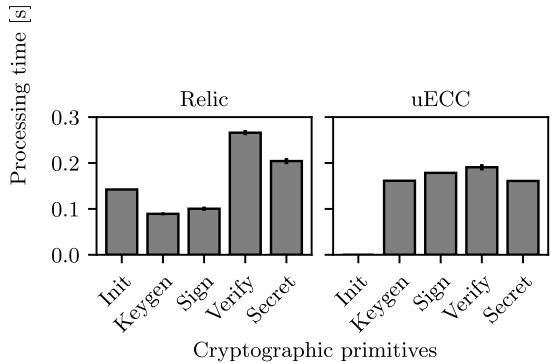
- Feature-rich crypto-toolkit
- Flexible and highly configurable

ATECC608A

- Constrained to single curve
- Secure memory for private keys

nRF52840

- Hardware support for > 15 elliptic curves



ECC Crypto - Processing Time

ECDSA / ECDH on NIST P-256

uECC

- Minimal, optimized library
- Static lookup tables

Relic

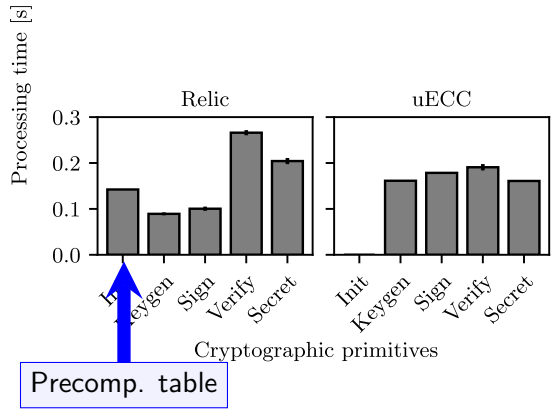
- Feature-rich crypto-toolkit
- Flexible and highly configurable

ATECC608A

- Constrained to single curve
- Secure memory for private keys

nRF52840

- Hardware support for > 15 elliptic curves



ECC Crypto - Processing Time

ECDSA / ECDH on NIST P-256

uECC

- Minimal, optimized library
- Static lookup tables

Relic

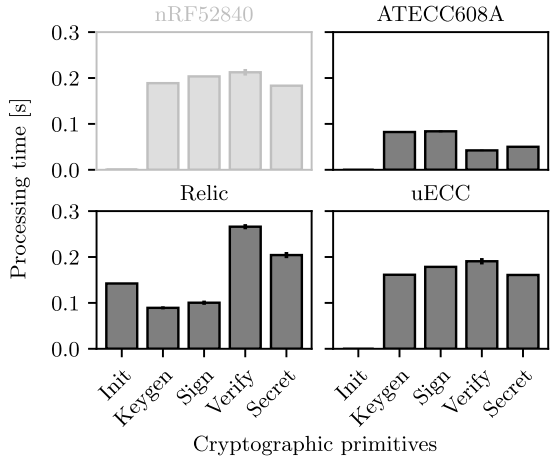
- Feature-rich crypto-toolkit
- Flexible and highly configurable

ATECC608A

- Constrained to single curve
- Secure memory for private keys

nRF52840

- Hardware support for > 15 elliptic curves



ECC Crypto - Processing Time

ECDSA / ECDH on NIST P-256

uECC

- Minimal, optimized library
- Static lookup tables

Relic

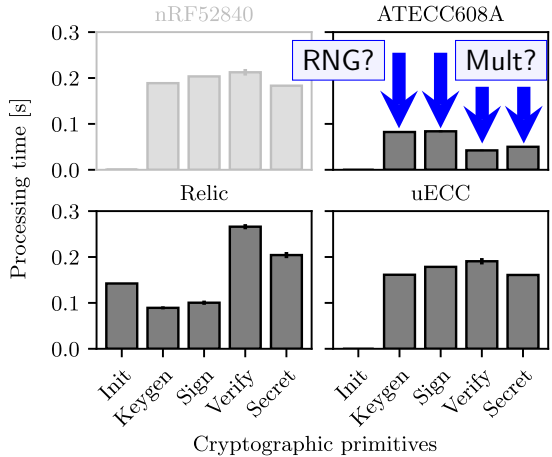
- Feature-rich crypto-toolkit
- Flexible and highly configurable

ATECC608A

- Constrained to single curve
- Secure memory for private keys

nRF52840

- Hardware support for > 15 elliptic curves



ECC Crypto - Processing Time

ECDSA / ECDH on NIST P-256

uECC

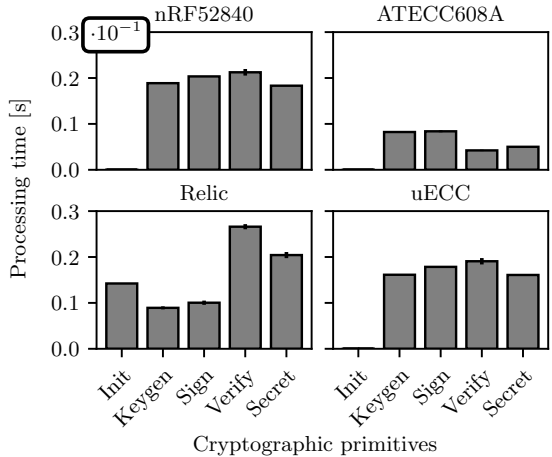
- Minimal, optimized library
- Static lookup tables

Relic

- Feature-rich crypto-toolkit
- Flexible and highly configurable

ATECC608A

- Constrained to single curve
- Secure memory for private keys

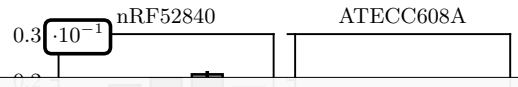


nRF52840

- Hardware support for > 15 elliptic curves

ECC Crypto - Processing Time

ECDSA / ECDH on NIST P-256



Configurability & algorithmic choice affect software performance

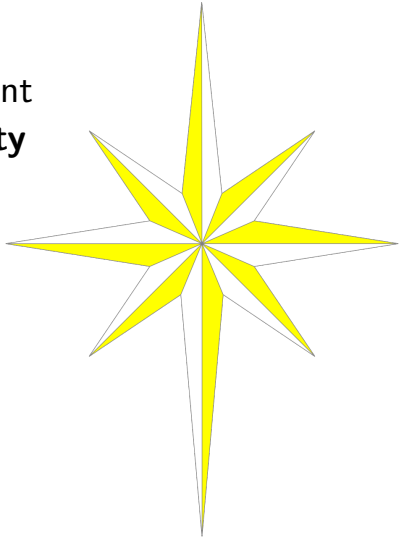
External device is on par with software

Peripheral accelerator gains by one order of mag.


AND THE ENERGY CONSUMPTION
BEHAVES SIMILARLY
memegenerator.net

4. Advent

Usability



Related Work on Crypto Usability

Developers are Not the Enemy!: The Need for Usable Security APIs (Green *et al.*)

- Operations should be high level & easy to use without crypto expertise or documentation.
- To strengthen security, professionals must focus on creating developer-centric approaches.

Usability Smells: An Analysis of Developers' Struggle With Crypto Libraries (Patnaik *et al.*)

- Crypto APIs should provide clear documentation, mark insecure algorithms, provide examples.
- Improvements to usable crypto libraries appear to be paying off with fewer usability smells.

Comparing the Usability of Cryptographic APIs (Acar *et al.*)

- Protecting and handling key material should not be a user responsibility.
- Security and usability are inherently linked.

How Usable are Rust Cryptography APIs? (Mindermann *et al.*)

- Need for extensive docu., usage recommendations, up-to-date example code and secure defaults.
- Poor usability of cryptographic APIs is a severe source of vulnerabilities.

Related Work on Crypto Usability

Developers are Not the Enemy!: The Need for Usable Security APIs (Green *et al.*)

- Operations should be high level & easy to use without crypto expertise or documentation.
- To strengthen security, professionals must focus on creating developer-centric approaches.

Usability Smells: An Analysis of Oauth Libraries (Patnaik *et al.*)

- Crypto APIs should provide clear documentation, mark insecure algorithms, provide examples.
- Improvements to usable crypto libraries appear to be paying off with fewer usability smells.

Comparing the Usability of Cryptographic APIs (Acar *et al.*)

- Protecting and handling key material should not be a user responsibility.
- Security and usability are inherently linked.

Security depends on:

(i) High level API design
(ii) Default configurations
(iii) Examples & documentation

How Usable are Rust Cryptography APIs? (Mindermann *et al.*)

- Need for extensive docu., usage recommendations, up-to-date example code and secure defaults.
- Poor usability of cryptographic APIs is a severe source of vulnerabilities.

Crypto Requirements in an IoT OS

Usability

- Developer friendly "standard" APIs with save defaults
- Reduce decision space to prevent choice of insecure params

Crypto Requirements in an IoT OS

Usability

- Developer friendly "standard" APIs with save defaults
- Reduce decision space to prevent choice of insecure params

Documentation & Tests

- Existing documentation of widely used APIs
- Existing test cases (like NIST)

Crypto Requirements in an IoT OS

Usability

- Developer friendly "standard" APIs with save defaults
- Reduce decision space to prevent choice of insecure params

Documentation & Tests

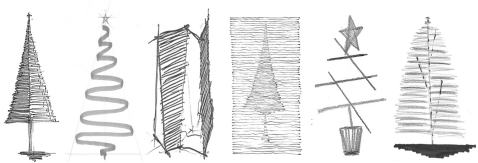
- Existing documentation of widely used APIs
- Existing test cases (like NIST)

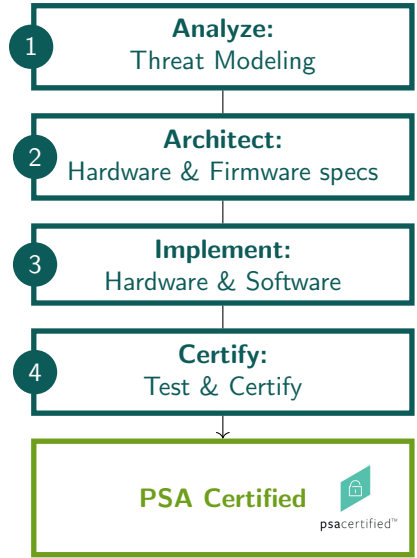
Abstraction

- Hide backend details from user $\rightarrow$ **Key IDs**
- Agnostic OS level API and backend integration
- Feature modeling and default selection

What is PSA?

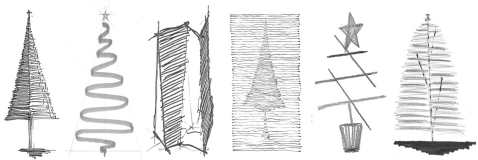
- ARM **P**latform **S**ecurity **A**rchitecture
- Framework for development of secure IoT systems
- Threat models, asset tracker, design docs., APIs, ...
- PSA Crypto: Hw./Sw. implementations
- Open source test suite for verification
- Implementations can be PSA certified

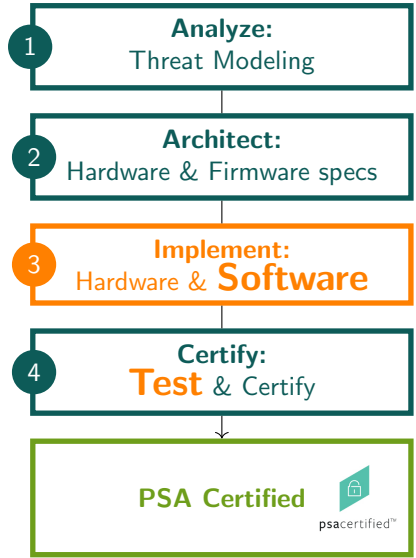




What is PSA?

- ARM Platform Security Architecture
- Framework for development of secure IoT systems
- Threat models, asset tracker, design docs., APIs, ...
- PSA Crypto: Hw./Sw. implementations
- Open source test suite for verification
- Implementations can be PSA certified

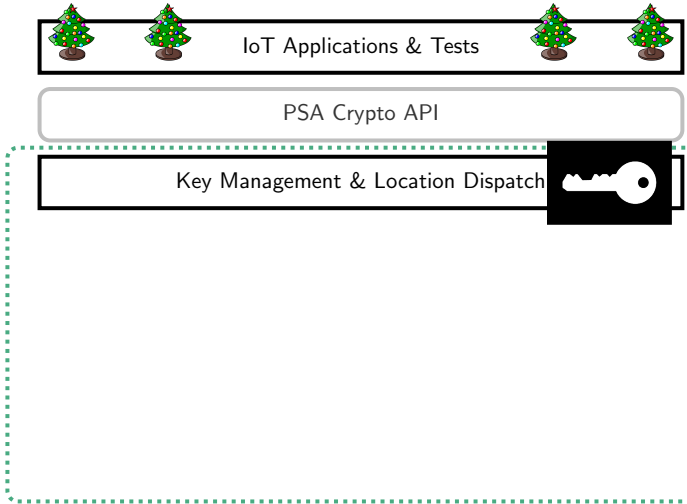




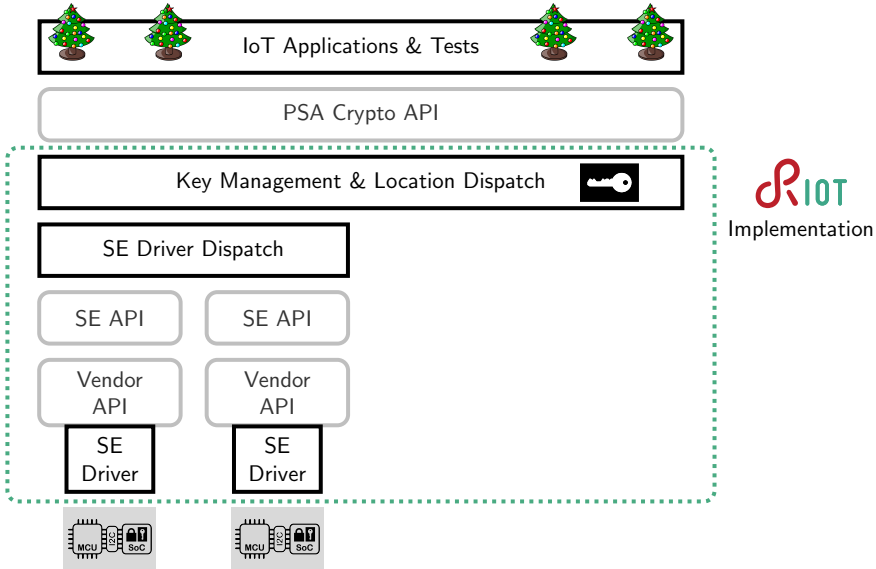

SANTA
CLAUSE
IS COMING
TO RIOT

PSA Crypto Implementation Structure

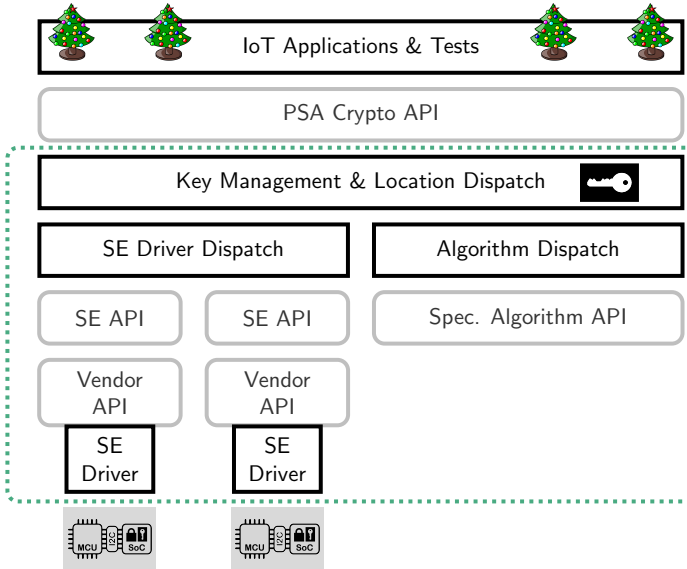
PSA Crypto Implementation Structure



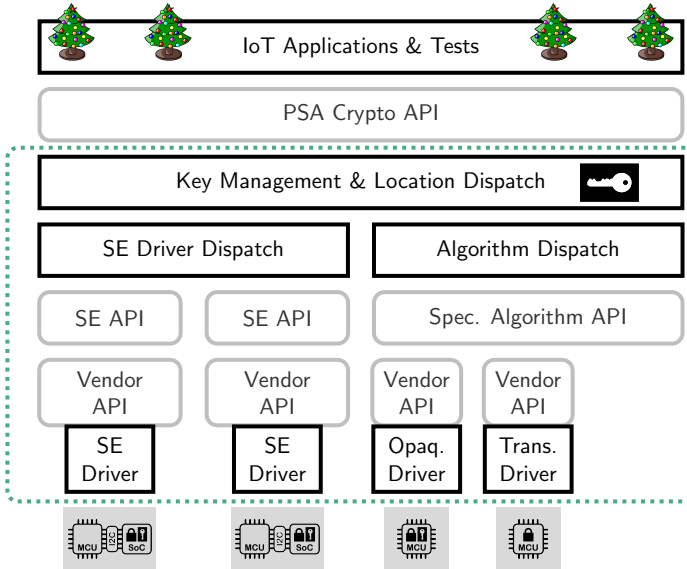
PSA Crypto Implementation Structure



PSA Crypto Implementation Structure

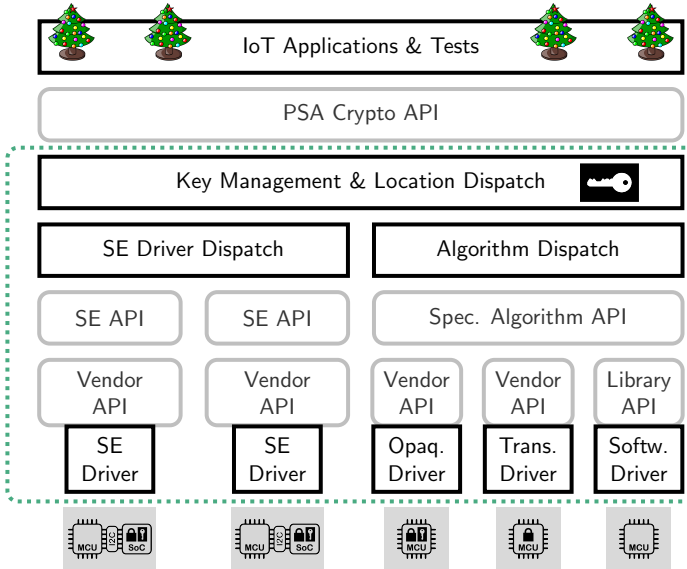


PSA Crypto Implementation Structure



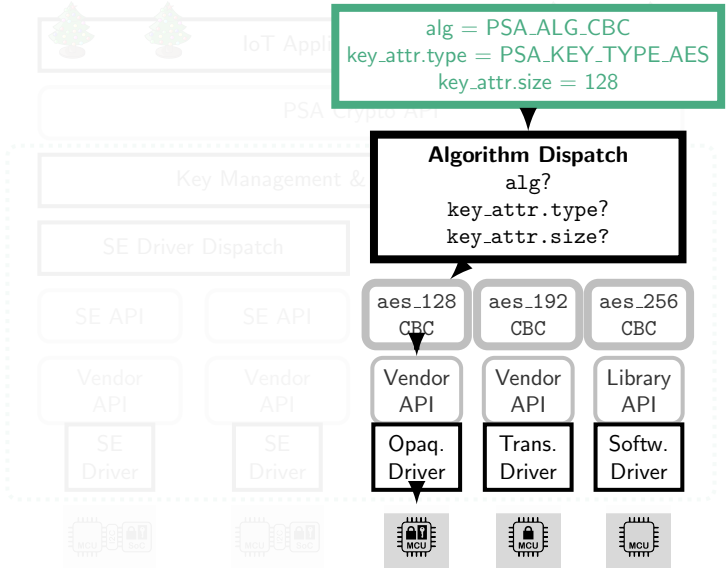
RIOT Implementation

PSA Crypto Implementation Structure



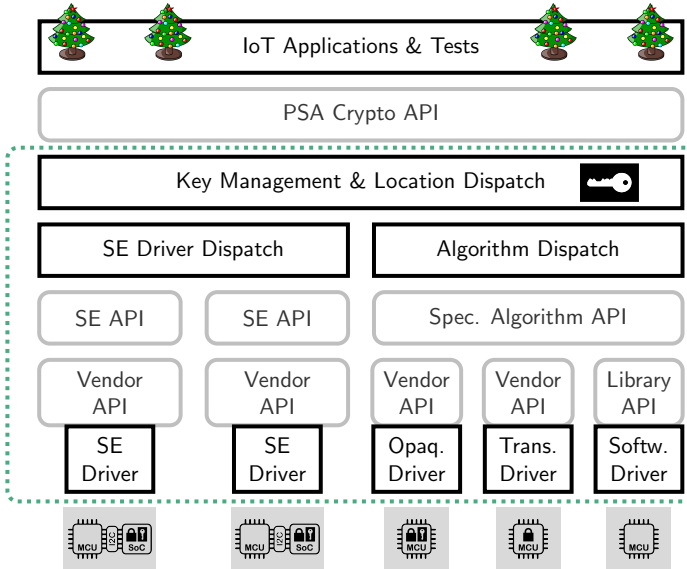
RIOT Implementation

PSA Crypto Implementation Structure



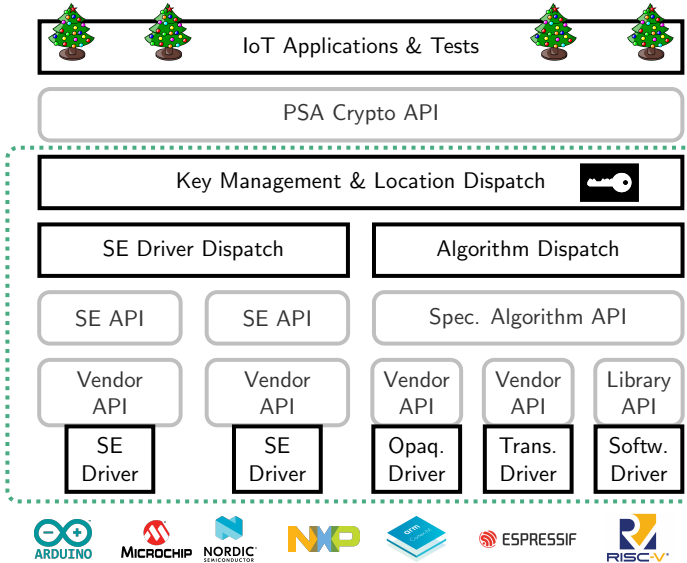
RIOT Implementation

PSA Crypto Implementation Structure

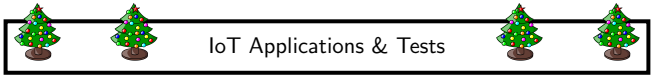


RIOT Implementation

PSA Crypto Implementation Structure



PSA Crypto Implementation Structure



PSA Crypto API

?

SILICON LABS

Other PSA implementations
e.g., provided by vendors
integrated as a RIOT `pkg`

ARM mbed

Linaro Connect


SECURITY...
IS
EVERYBODY'S
BAG.