New Crypto-fundamentals in RIOT

Peter Kietzmann

peter.kietzmann@haw-hamburg.de

3rd get-together of the friendly Operating System for the Internet of Things

September 13, 2018

“Crypto-Fundamentals” ???

IoT requires security. . .

… as we just learned in *“Usable Security for RIOT and the Internet of Things”*

“Crypto-Fundamentals”???

IoT requires security. . .
... as we just learned in *“Usable Security for RIOT and the Internet of Things”*

Low-cost “COTS” devices. . .
. . . usually don’t provide secure hardware such as Trusted Platform Module, Intel SGX or ARM TrustZone to reduce cost

“Crypto-Fundamentals” ???

IoT requires security. . .
... as we just learned in *“Usable Security for RIOT and the Internet of Things”*

Low-cost “COTS” devices. . .
. . . usually don’t provide secure hardware such as Trusted Platform Module, Intel SGX or ARM TrustZone to reduce cost

Lack of computational power. . .
. . . and the absence of secure hardware require efficient software implementations to fit device constraints

“Crypto-Fundamentals” ???

IoT requires security. . .
... as we just learned in *“Usable Security for RIOT and the Internet of Things”*

Low-cost “COTS” devices. . .
. . . usually don’t provide secure hardware such as Trusted Platform Module, Intel SGX or ARM TrustZone to reduce cost

Security protocols require. . .
. . . certain resources such as high quality random numbers, salts, cryptographic keys

Lack of computational power. . .
. . . and the absence of secure hardware require efficient software implementations to fit device constraints

“Crypto-Fundamentals”???

IoT requires security. . .
... as we just learned in *“Usable Security for RIOT and the Internet of Things”*

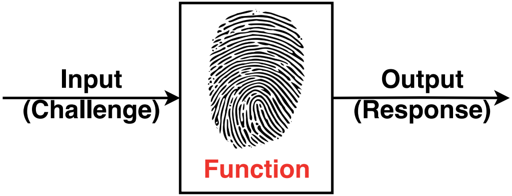
Low-cost “COTS” devices. . .
. . . usually don’t provide secure hardware such as Trusted Platform Module, Intel SGX or ARM TrustZone to reduce cost

Security protocols require. . .
. . . certain resources such as high quality random numbers, salts, cryptographic keys

Lack of computational power. . .
. . . and the absence of secure hardware require efficient software implementations to fit device constraints

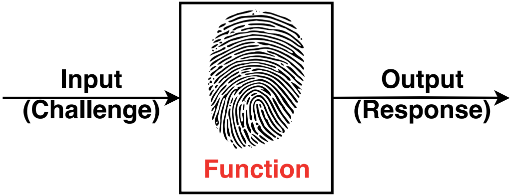
We introduce software fundamentals to address crypto requirements

Physical Unclonable Functions



Physical Unclonable Functions

- Digital fingerprint based on manufacturing process variations
- Extracted response identifies a device like human fingerprint
- The "secret" is hidden in **physical** structure
  $\rightarrow$ Hard to predict or **clone**



- A variety of PUFs exist based on:
  Component delays, magnetism, optics, uninitialized memory pattern, ...

Note: Like biometric data, PUF responses are affected by noise

PUF Applications & Parameters

	Applications	Quality Parameters
Noise	▶ RNG, PRNG seeding, ...	▶ Intra-device variations
Identity	▶ Identification, authentication ▶ Secret key generation or storage ▶ Unique app–to–device binding (i.e., secure boot)	▶ Reproducible ▶ Unique ▶ Unpredictable ▶ Unclonable

Literature & Recent Work

A. Schaller:

"Lightweight Protocols and Applications for Memory-Based Intrinsic Physically Unclonable Functions Found on Commercial Off-The-Shelf Devices" (2017)

Secure applications based on PUFs evaluated on multiple COTS

"A. Van Herrewege et al.: Secure PRNG Seeding on Commercial Off-the-Shelf Microcontrollers" (2013)

SRAM analysis of different COTS for PRNG seeding under varying environmental conditions

"Y. Dodis et al.: Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data" (2008)

Provide secure techniques to generate crypto-keys from noisy responses

"C. Bösch et al.:Efficient Helper Data Key Extractor on FPGAs" (2008)

Design and evaluation of key extractors on FPGAs

"J. Delvaux et al.: Attacking PUF-Based Pattern Matching Key Generators via Helper Data Manipulation" (2012)

Propose attacks and recovery from PUF-constructed keys

No lightweight, open source, operating system integration?

We implement SRAM based PUFs in RIOT for PRNG seeding and key generation

Outline

A Brief Introduction to PUFs

SRAM Memory Analysis of Standard RIOT Devices

A Seeder for Pseudo Random Number Generators

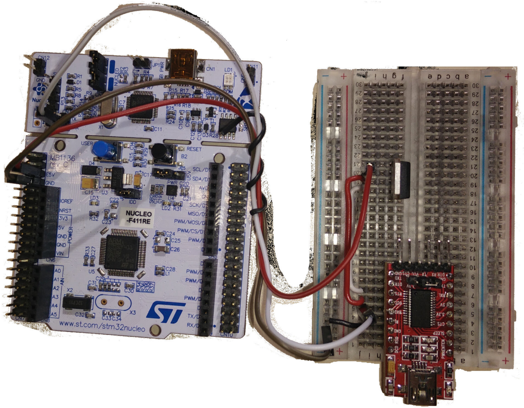
Cryptographic Key Generation from Noisy PUF Responses

Current Implementation Progress in RIOT

Next Steps, Future Plans, ...

SRAM Memory Analysis of Standard RIOT Devices

Experiment Setup



- Periodically power-on device and read SRAM blocks after boot
  $\rightarrow$ Power-down time $>$ RAM hold-time
- Transistor variations lead to different cell states on startup
  $\rightarrow$ Unique pattern $+$ noise
- Results depend on SRAM technologies, circuit and environment
  $\rightarrow$ Should be evaluated individually

Intra-Device Analysis

50 reads; 1kB SRAM; 5 SAMD21; Ambient Temperature

Quantify **randomness** by min. entropy: $H_{min} = -\sum_{i=1}^{n} \log_2(\max(p_0^i, p_1^i)) \cdot \frac{100\%}{n}$

n: memory length, $p_{0/1}$: low/high probabilities

Quantify **bias** by hamming weight: $W(a) = \|\{a_i \neq 0\}_{1 \leq i \leq n}\| \cdot \frac{100\%}{n}$

Device	A	B	C	D	E
Min. Entropy	4.16 %	5.46 %	5.28 %	4.68 %	5.48 %
Hamming Weight	50.7±3 %	49.5±3 %	51.3±6 %	49.8±4 %	53.1±3 %

→ **The SRAM memory is not biased and contains a random component**

Inter-Device Analysis

50 reads; 1kB SRAM; 5 SAMD21; Ambient Temperature

Quantify **uniqueness** by fractional hamming distance:

$$D(a, b) = \|\{a_i \neq b_i\}_{1 \leq i \leq n}\| \cdot \frac{100\%}{n}$$

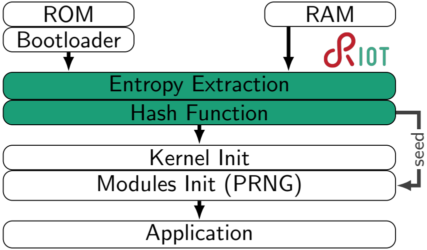
Device Pair	A–B	A–C	A–D	A–E
Hamming Distance	49.2±4 %	49.5±3 %	50.1±3 %	50.4±4 %

→ **The SRAM pattern do not correlate between devices**

A Seeder for Pseudo Random Number Generators

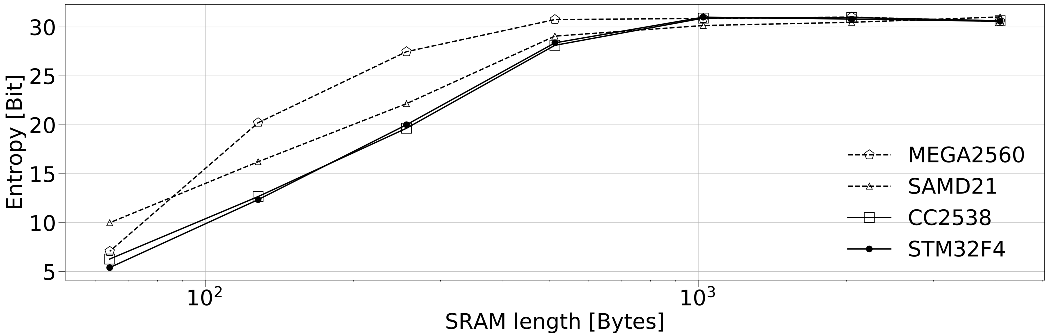
Seeder Architecture

- Module hooks into startup **before** `kernel_init`
- Patterns of uninitialized SRAM are hashed by DEK Hash
- 32-bit result is stored in pre-reserved RAM section
- Seeds PRNG **after** `kernel_init`



SRAM Memory Length

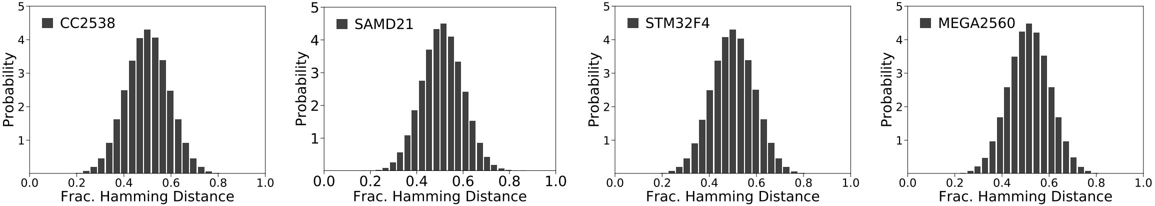
Min. Seed Entropy; Varying SRAM Lengths; Ambient Temperature



→ **Approximately 31 Bit entropy @ 1kB SRAM is a good fit**

Seed Distribution

Frac. Hamming Distances of Seeds; 1kB SRAM; Ambient Temperature



Distances follow a normal distribution with expectation value around 0.5

→ **We consider seeds as independent**

Reset Detection

- The SRAM needs to be **uninitialized** to provide highest intra-device entropy
  $\rightarrow$ device needs start from power-off
- That's not the "development" case where programmers press `reset`
- We implement a reset detection mechanism to report soft-resets
- A 32-bit marker is written to a specific location
- During the next reboot we test it's presence

Talk Progress

A Brief Introduction to PUFs

SRAM Memory Analysis of Standard RIOT Devices

A Seeder for Pseudo Random Number Generators

Cryptographic Key Generation from Noisy PUF Responses

Current Implementation Progress in RIOT

Next Steps, Future Plans, ...

Motivation

Problem:

1. PUF responses are error-prone
2. PUF responses are not distributed uniformly

Requirement:

1. We need reproducible PUF responses
2. We want to produce uniformly distributed secrets

Solution:

1. Remove errors from PUF measurements
2. Map the high-entropy input to a uniformly distributed output

Fuzzy Extractor

Mechanism

Secure Sketch:

- Reliably reconstruct response from a noisy measurement
- Uses error correction codes

Randomness Extractors:

- One way hash function to compress high entropy output
- The input sequence needs min. entropy

Fuzzy Extractor

Mechanism

Secure Sketch:

- Reliably reconstruct response from a noisy measurement
- Uses error correction codes

Randomness Extractors:

- One way hash function to compress high entropy output
- The input sequence needs min. entropy

Deployment

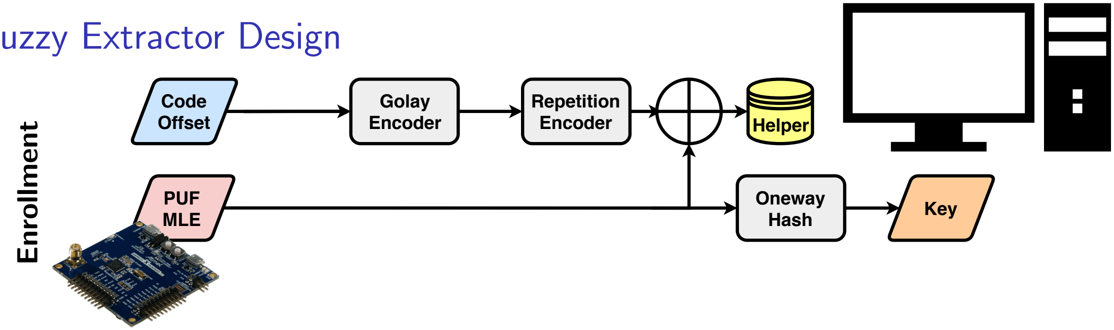
Enrollment:

- Encoding and helper data generation
- Uses a **reference** PUF response
- Executed in trusted environment

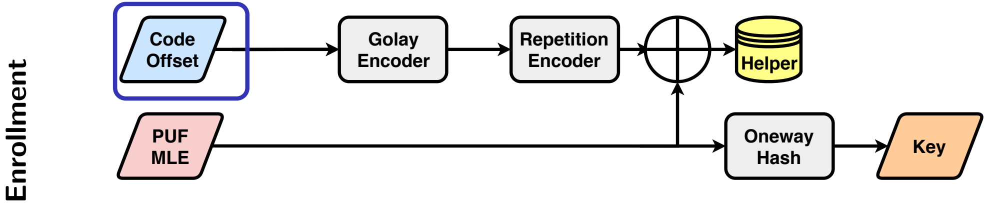
Reconstruction:

- Decodes corrupted input sequence
- Uses a noisy PUF **measurement**
- Executed on the device after startup

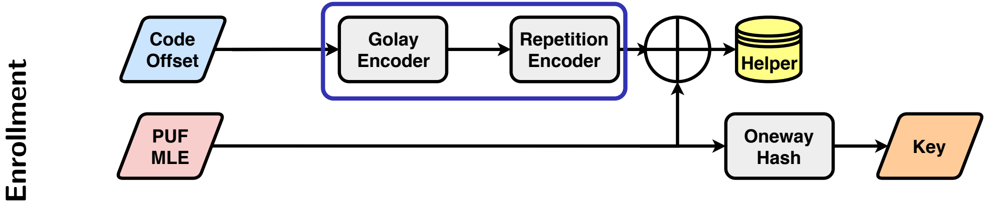
Fuzzy Extractor Design



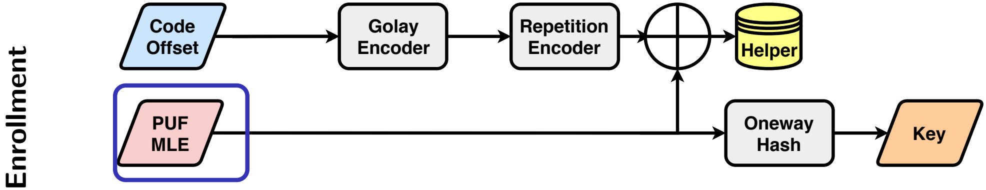
Fuzzy Extractor Design



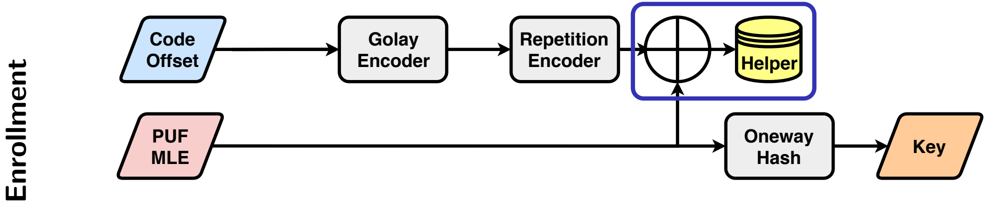
Fuzzy Extractor Design



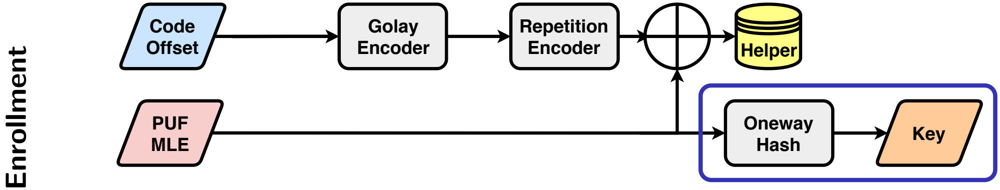
Fuzzy Extractor Design



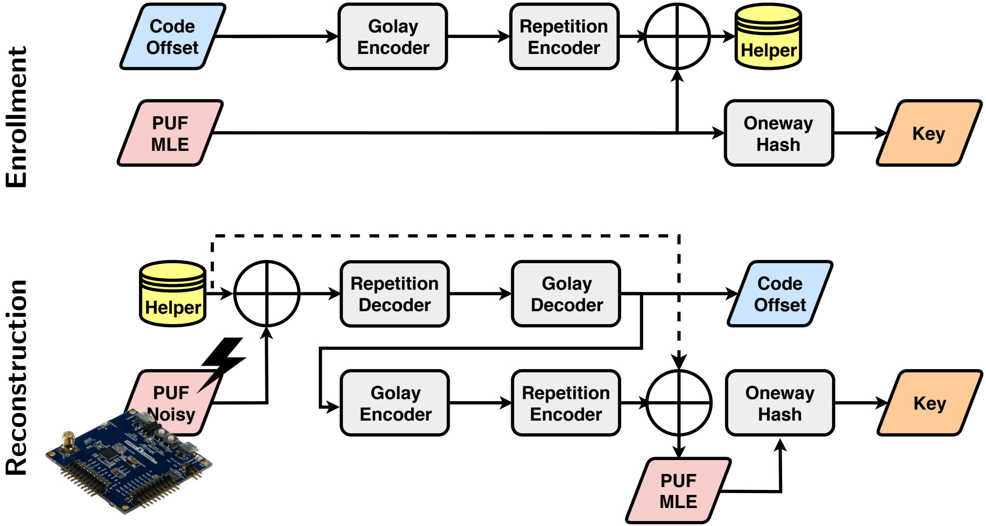
Fuzzy Extractor Design



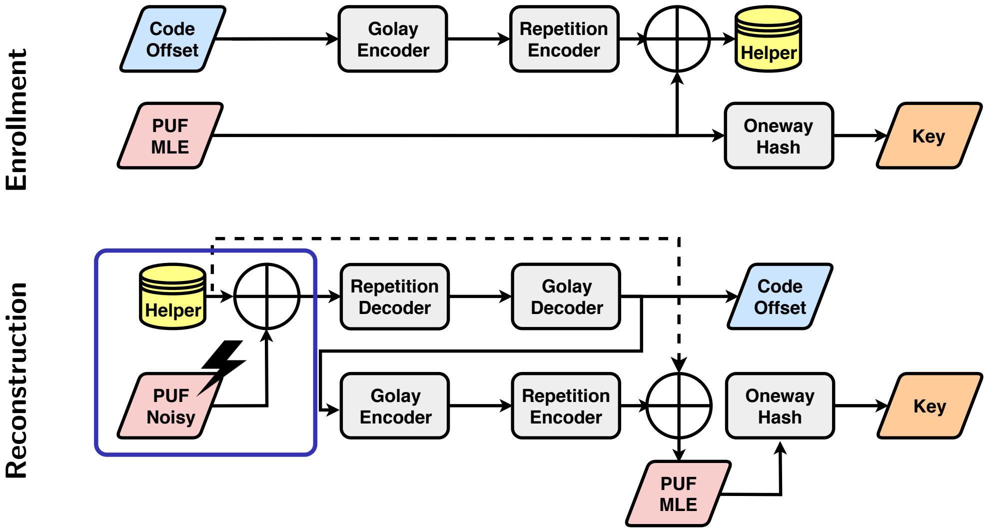
Fuzzy Extractor Design



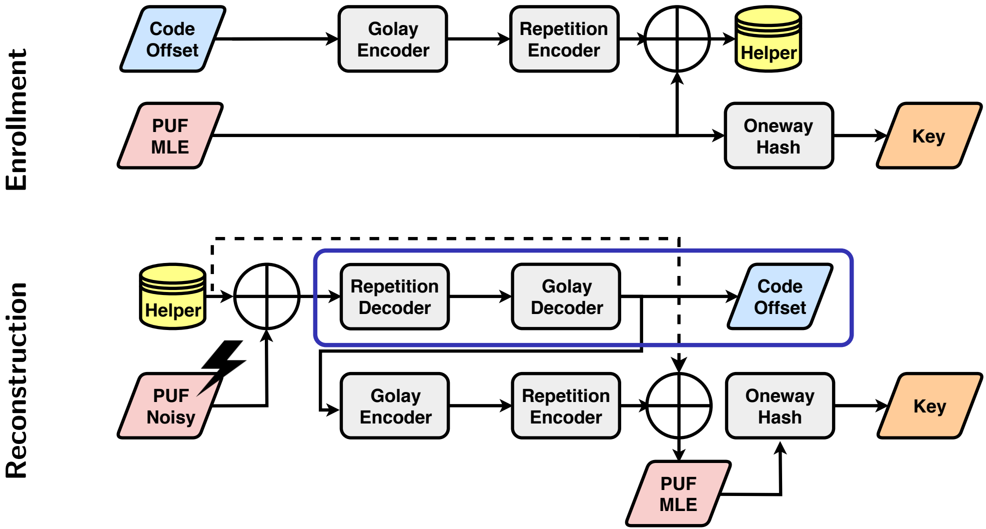
Fuzzy Extractor Design



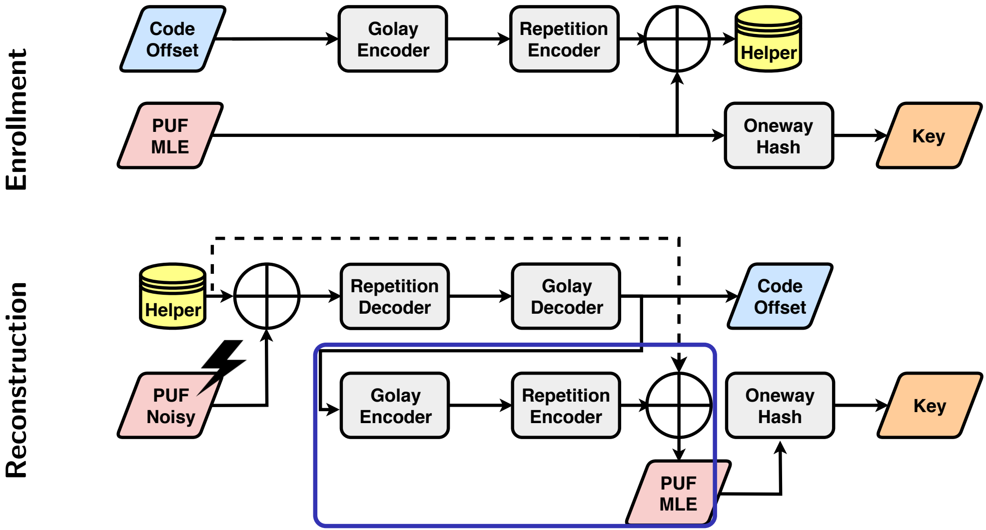
Fuzzy Extractor Design



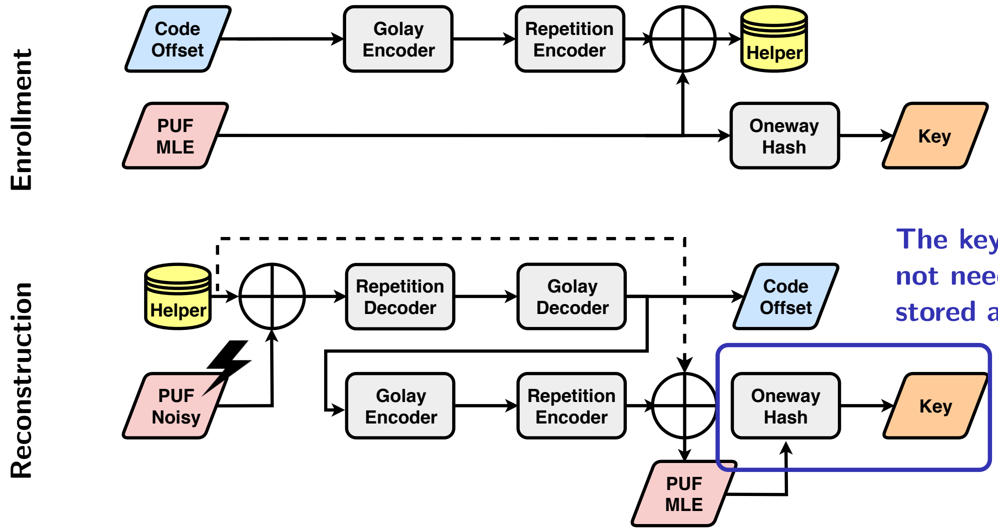
Fuzzy Extractor Design



Fuzzy Extractor Design



Fuzzy Extractor Design

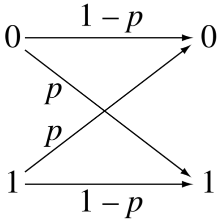


The key does not need to be stored anywhere!

Fuzzy Extractor Parameters

Error probability:

- Measured bit error probability: $\boxed{p_{max} = 0.1}$
  (literature calculates with $p_b = 0.15$)
- Calculated output error probability: $\boxed{P_{total} = 5.07 \times 10^{-7}}$
  (literature considered $P_{total} = 1 \times 10^{-6}$ as conservative)

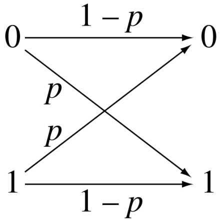


[1] *T.Ignatenko et al.: "Estimating the Secrecy-Rate of Physical Unclonable Functions with the Context-Tree Weighting Method"*

Fuzzy Extractor Parameters

Error probability:

- Measured bit error probability: $\boxed{p_{max} = 0.1}$
  (literature calculates with $p_b = 0.15$)
- Calculated output error probability: $\boxed{P_{total} = 5.07 \times 10^{-7}}$
  (literature considered $P_{total} = 1 \times 10^{-6}$ as conservative)

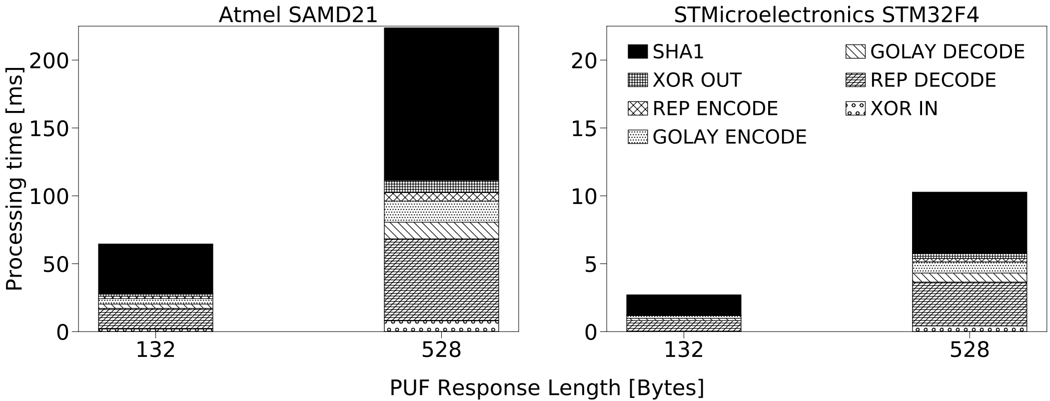


Min. length of PUF response:[1]

Secret Bits	Source Bits	Coded Source Bits	Coded Source Bytes
32	42	1056	132
128	171	3960	495
146	192	4224	528

[1] *T.Ignatenko et al.: "Estimating the Secrecy-Rate of Physical Unclonable Functions with the Context-Tree Weighting Method"*

Fuzzy Extractor Processing Time



Current Implementation Progress in RIOT

RIOT Implementation Progress

Component	Feature	Status
PRNG Seeder		
	Cortex-M	✓
	AVR8	✓
	Evaluation Tool	✓
Fuzzy Extractor		
	Cortex-M	✓
	AVR8	✗
	Helper Data generation tool	✓

Next Steps, Future Plans, ...

General:

- Implement the missing components :-) !
- Evaluate SRAM startup from low power wake-up

General:

- Implement the missing components :-) !
- Evaluate SRAM startup from low power wake-up

Random:

- Add “secure” seed for cryptographically secure PRNG
- Extend random API in various aspects
  - Enable parallel PRNGs
  - Application based seed provisioning
  - Event reporting, e.g., soft-reset detection
- Apply NIST statistical test suite to RIOT

General:

- Implement the missing components :-) !
- Evaluate SRAM startup from low power wake-up

Random:

- Add "secure" seed for cryptographically secure PRNG
- Extend random API in various aspects
  - Enable parallel PRNGs
  - Application based seed provisioning
  - Event reporting, e.g., soft-reset detection
- Apply NIST statistical test suite to RIOT

Fuzzy Extractor:

- Evaluate privacy of public Helper Data
- Measure bit error probability on embedded devices
- Implement build target for Helper Data generation & storage

BS - Error Correction Code

- Binary codes are noted as $[n, k, d]$ -codes with $n =$ code length, $k =$ encoded message length, $d =$ minimum distance of code words
- Concatenation of Golay and Repetition 11 code leads to $[264, 12, 77]$ -code
- Binary Symmetric Channel as model:

$$P_{total} = 1 - \sum_{i=1}^{t} \binom{n}{i} p_b^i (1 - p_b)^{n-i}$$

  with $t = (d_{min} - 1)/2$ correctable errors
- $t_{golay} = 3$, $t_{rep11} = 5$ and $p_b = 0.1$
- Total error by calculating inner code and apply error to outer code

BS - Length of PUF response

Secrecy rate:

- Universal hash function compresses PUF response bits
- Min. amount of compression (by hashing) is expressed by "secrecy rate" S_R
- Max. achievable secrecy rate given by mutual information between PUF responses during Enrollment and Reconstruction
- Common value is $S_R = 0.76$
  $\rightarrow$ For a secret of length 128 Bit, we need $S_R^{-1} \cdot 128 = 171$ source Bits
- Minimum number of source bits after encoding: $n\lceil 171/k \rceil$