PUFs in RIOT: Releasing Recent Crypto-fundamentals into the Wild

Peter Kietzmann, advised by Thomas C. Schmidt
Hamburg University of Applied Sciences (HAW Hamburg)
{peter.kietzmann, t.schmidt}@haw-hamburg.de

***Abstract*—Internet-connected devices need to provision crypto components in order to communicate securely in the network. Random number generation, authentication, or secret key generation rely on decent platform resources that provide unpredictable but reliable high-entropy numbers. Common solutions require a certain level of hardware and software complexity and thus device cost that are typically unavailable in the IoT.**

Physically unclonable functions (PUFs) are a promising class of solutions to this problem. They extract output from individual hardware properties that persist due to manufacturing inaccuracies. The extracted physical characteristics act as a digital fingerprint and can be used to generate random numbers but also to produce device unique secrets, which enhances security in various application scenarios. To make these crypto fundamentals widely available on low cost IoT nodes, we introduce and thoroughly evaluate a PUF module in the operating system RIOT.

I. Introduction & Related Work

The IoT interconnects off-the-shelf microcontrollers with the global Internet. RIOT [1] is the open source operating system that brings standard networking to many IoT nodes. Like in most OSes, networking, cryptography, and other daily routines require local input of very low predictability. Generating unclonable individual numbers on microcontrollers, though, is surprisingly hard, since most devices run on a deterministic clock and show almost no distinguishing features. Furthermore, the minority of these constrained and low-power IoT platforms provide secure hardware components.

Physically unclonable functions (PUFs) are a promising class of solutions to this problem. They extract unique output from individual hardware properties like a digital fingerprint. Similar to a human fingerprint, PUF responses are affected by noise which can be used as entropy source on a single device.

The field of PUF based applications is manifold and comprises device identification or authentication. Security mechanisms like DTLS or HMAC rely on pre-established keys, which can be generated from a PUF. Alternatively, a PUF can be used to decrypt a pre-shared encrypted key that is stored on non-volatile memory. Furthermore, device specific configuration parameters can be secured in an encrypted way to generate individual application to device bindings.

In [2] secure applications based on PUFs were analyzed and evaluated on different microcontroller types. One valuable use of PUFs is random number generation to properly seed pseudorandom number generators (PRNGs) which was analyzed in detail in [3]. The authors of [4] presented a secure approach to reliably generate crypto-keys from PUF responses by removing the noise component and in [5] a similar approach was implemented on an FPGA and analyzed in terms of resource utilization. First attacks against certain PUF based key-generators were already presented in [6] and [7].

To the best of my knowledge, there is no lightweight, open source, operating system integration of such PUF primitives. To enable security fundamentals for constrained microcontrollers that lack secure hardware, this work focuses an easy-to-use integration of PUFs based on uninitialized SRAM memory patterns to (i) properly seed PRNGs and (ii) for secret key generation in the IoT operating system RIOT. The focus lies on SRAM PUFs because this component is one of the few entropy sources that is present on all supported devices. With the integration of crypto-primitives in RIOT we contribute to a reliable, robust and secure Internet of Things.

II. Research Challenges & Contributions

A. Preliminary Analysis

Transistor variations of memory cells lead to different states after device power-on. The startup state of multiple memory blocks form a device-unique pattern plus additional noise. It is worth noting that memory properties may depend on environmental parameters and thus, they should be evaluated for each individual deployment. To prove usability of SRAM mounted on typical RIOT platforms, I analyzed inter- and intra- device variations between multiple PUF responses of various microcontrollers at ambient conditions. In more detail, I calculated (i) the minimum entropy as a measure of randomness and (ii) the hamming weight to determine bias between multiple startups of one device as well as (iii) the fractional hamming distance between different device responses to depict uniqueness. Results for (i) and (ii) indicate existence of a relative min. entropy around 5 % and unbiased patterns. A relative fractional distance of approximately 50 % in (iii) indicates uniqueness between device responses. More detailed analyses were published in [8].

B. Random Seeder

The newly introduced RIOT module to create random seeds hooks in the startup code even before the OS kernel initialization in order to get uninitialized memory state. A PUF measurement is compressed by the lightweight DEK hash to build a high entropy 32-bit number that is stored at a pre-allocated RAM section. Afterwards the kernel is initialized,

followed by automatic initialization of RIOT modules which includes PRNG seeding with the PUF based seed value.

To prove decent memory length for seed generation I evaluated the minimum seed entropy for varying input lengths and platforms. All measurements converge to approximately 31-Bit entropy with 1 kB SRAM which is a huge improvement considering that RIOT currently uses the CPUID for seeding.

Finally, I applied the NIST Statistical Test Suite [9] to a sequence of 1 Mega seeds, generated by a STM32F4 microcontroller. Thereby, the dataset was divided into 32 bitstrings of length 1 Mbit. The test suite consists of 15 tests that compare bitstrings against the hypothesis of perfect randomness. A test passes if the p-value lies within the significance level $[\alpha; 1-\alpha]$ with $\alpha = 0,01$. The random sequences successfully passed all tests.

C. Secret Generation

PUF measurements are noisy and not uniformly distributed. In order to reliably generate error-free and reproducible PUF responses on each device startup, a method called *secure sketch* is used to eliminate random bitflips by means of error correction codes. A *randomness extractor* generates uniformly distributed and compressed high entropy values from corrected PUF responses by incorporating a secure one-way hash function. Together, both components are known as *fuzzy extractor*. Deployment of such an extractor consists of two phases:

a) Enrollment: During enrollment a reference PUF measurement is taken and redundancy is added according to the *code offset method* for later error correction. This procedure generates a public (non-secret) helper data set but it needs to be processed in a trusted environment. The helper data is stored in non-volatile memory of the IoT device.

b) Reconstruction: The reconstruction is done during startup when the device is deployed. It utilizes the public helper data and a noisy PUF measurement together, to decode and consequently correct errors. From there, a secret key can be reliably constructed by applying a one-way function for privacy amplification. One major advantage of this mechanism is the unnecessity to store secrets long term, as they are only derived during startup.

The helper data generation is executed once for the IoT device life-time and requires a dedicated measurement firmware on that device, whereas the actual helper is calculated on an external machine and written to the IoT devices non-volatile memory afterwards. Once the individual helper is available, the actual IoT firmware can be flashed which may or may not utilize the secret generator. To simplify deployment process I implemented a build tool in RIOT that automatically flashes the measurement software, generates helper data and re-flashes the device with the required application.

III. Research Agenda

A. Entropy Source Optimization

SRAM PUFs are not only prone to attacks when physical device access is given, they may also be affected by aging processes of memory cells which would reduce intra-device entropy. Furthermore, this technique requires a power-down of the memory at the order of seconds. Thus, generating a new seed is impossible on the fly. However, some platforms provide different clock sources for high-precision or low-precision and low-power timers. I plan to experiment with a PUF that relies on jitters between different clock sources. Thereby, one mayor task will be the design of a proper PUF challenge and the minimum time needed to generate a high entropy number of defined bit length. Furthermore, this PUF needs a generic integration next to the currently implemented SRAM PUF infrastructure. Long-term, I plan to accumulate multiple PUF sources to increase reliability.

B. Deployment Simplification

Deployment of an individual secret generator requires two phases. Currently this incorporates two flash processes with external helper data calculation in between. However, some low-power microcontrollers allow for powering-off only parts of the memory block in order to reduce current consumption. Theoretically, this enables helper data generation on the controller itself. I plan to implement a transparent solution for that, to simplify industrial deployment in future.

C. Security Analysis

The helper data is stored on non-volatile memory and thus, it is not a secret. Ideally, it does not reveal information about the PUF response itself. In practice, the helper data relies on the XORed reference PUF response and a sequence of code words, according to the *code offset*. This may introduce entropy leakage, especially when trivial but resource efficient coders were used. In future I will analyze this leakage effect as well as other vulnerabilities of helper data in more detail and try to find a solution that trades-off robustness and security against resource efficiency in our RIOT implementation.

References

[1] E. Baccelli et al., "RIOT: an Open Source Operating System for Low-end Embedded Devices in the IoT," *The IEEE IoT Journal*, 2018.

[2] A. Schaller, "Lightweight Protocols and Applications for Memory-Based Intrinsic Physically Unclonable Functions Found on Commercial Off-The-Shelf Devices," Doctoral Dissertation, TU Darmstadt, 2017.

[3] A. Van Herrewege et al., "Secure prng seeding on commercial off-the-shelf microcontrollers," in *Proc. 3rd Intern. WS on Trustworthy Embedded Devices, TrustED '13*, ACM, 2013, pp. 55–64.

[4] Y. Dodis et al., "Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data," *SIAM Journal on Computing*, vol. 38, no. 1, pp. 97–139, 2008.

[5] C. Bösch et al., "Efficient Helper Data Key Extractor on FPGAs," in *Cryptographic Hardware and Embedded Systems - CHES*. Springer, 2008, pp. 181–197.

[6] J. Delvaux et al., "Attacking PUF-Based Pattern Matching Key Generators via Helper Data Manipulation," in *Topics in Cryptology — CT-RSA 2014*. Springer, 2014, pp. 106–131.

[7] C. Helfmeier et al., "Cloning Physically Unclonable Functions," in *IEEE International Symposium on Hardware-Oriented Security and Trust - HOST*, 2013, pp. 1–6.

[8] P. Kietzmann et al., "A PUF Seed Generator for RIOT: Introducing Crypto-Fundamentals to the Wild," in *Proc. of 16th ACM International Conference on Mobile Systems, Applications - MobiSys, Poster Session*. NY, USA: ACM, June 2018.

[9] L. Bassham et al., "SP 800-22 Rev. 1a. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications," Gaithersburg, MD, US, Tech. Rep., National Institute of Standards & Technology, 2010.