PUF for the Commons: Enhancing Embedded Security on the OS Level

Peter Kietzmann, Thomas C. Schmidt, *Member, IEEE,* and Matthias Wählisch, *Member, IEEE*.

Abstract—Security is essential for the Internet of Things (IoT). Cryptographic operations for authentication and encryption commonly rely on random input of high entropy and secure, tamper-resistant identities, which are difficult to obtain on constrained embedded devices. In this paper, we design and analyze a generic integration of physically unclonable functions (PUFs) into the IoT operating system RIOT that supports about 250 platforms. Our approach leverages uninitialized SRAM to act as the digital fingerprint for heterogeneous devices. We ground our design on an extensive study of PUF performance in the wild, which involves SRAM measurements on more than 700 IoT nodes that aged naturally in the real-world. We quantify static SRAM bias, as well as the aging effects of devices and incorporate the results in our system. This work closes a previously identified gap of missing statistically significant sample sizes for testing the unpredictability of PUFs. Our experiments on COTS devices of 64 kB SRAM indicate that secure random seeds derived from the SRAM PUF provide 256 Bits-, and device unique keys provide more than 128 Bits of security. In a practical security assessment we show that SRAM PUFs resist moderate attack scenarios, which greatly improves the security of low-end IoT devices.

Index Terms—Physically Unclonable Functions, Embedded Security, Large-scale SRAM Analysis, Internet of Things, Operating Systems

1 Introduction

The Internet of Things (IoT) comprises billions of constrained devices, but the low-cost IoT hardware is challenged by basic security operations. High entropy seeds for secure random number generation [9] and secure hardware identities form the minimal set of primitives that bootstrap the cryptographic subsystem needed for protecting basic services of networked nodes. These numbers must remain secret to prevent information leakage of past and future transactions, and require resistance against readout or tampering. Supplementary hardware security modules (*e.g.*, secure elements) can overcome these challenges but increase device cost. In practice, many large-scale IoT deployments consist of cheap embedded devices without hardware security features, and readily threaten the IoT [10] as well as the global Internet [11].

Physical unclonable functions (PUFs) utilize intrinsic hardware variations, which are a promising source of (*i*) random variations on one device, and (*ii*) unpredictable secrets between devices that become reproducible by excluding the variations from (*i*). A prevalent type of PUF input is SRAM. After powering up the hardware, SRAM provides a digital fingerprint from the patterns of uninitialized memory. SRAM is available on almost every IoT platform and can be exploited without additional hardware. This makes the technology particularly attractive for low-cost devices. Secret values are generated only during system startup and consumed quickly after to lower the risk of a compromise.

- *Peter Kietzmann and Thomas C. Schmidt are with the Department Informatik, HAW Hamburg, Berliner Tor 7, 20099 Hamburg, Germany. E-mail: {peter.kietzmann, t.schmidt}@haw-hamburg.de.*
- *Matthias Wählisch is with the Faculty of Computer Science, TU Dresden, Helmholtzstr. 10, 01069 Dresden, and also with the Barkhausen Institut, 01187 Dresden, Germany. E-mail: m.waehlisch@tu-dresden.de*

Manuscript received Jan. 17, 2023.

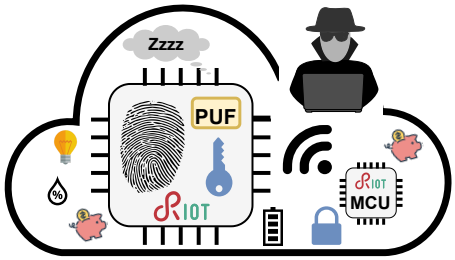


Fig. 1: PUF security services provided by an operating system enable lightweight crypto-operations on low-cost hardware in the IoT.

Consequently, SRAM secrets remain absent during regular node operations.

There have been concerns, though, that the physical layout of SRAM [12] as well as hardware aging [13] may introduce systematic biases. Quantifying these subtle statistical effects requires a comparative analysis between large quantities of devices [14], which we contribute in this paper. Our large-scale evaluation of more than 700 nodes clearly shows a localized bias for certain bits in the SRAM response. An attacker who tries to predict this pattern by using a large number of measurements from similar devices could reach an advantage in guessing the bit values at certain positions. We quantify the remaining entropy of secrets derived by these biased pattern and identify secret generation schemes that are able to mitigate this weakness.

We further analyse hardware aging by extensive measurements on nodes that naturally aged in the real-world environment of an open access testbed. We find address specific wear-out effects that link to past experiment executions. Our findings shall motivate testbed operators as well as

software developers to invoke anti-aging strategies to their firmware.

In this paper, we design and evaluate PUFs for the multi-purpose operating system RIOT [15] for constrained IoT devices [16]. Our configurable integration enables (*i*) portability across heterogeneous platforms with differing hardware capabilities, and (*ii*) the replacement of modular building blocks in order to adapt to differing environmental conditions, security requirements, or the energy availability on battery driven nodes. To the best of our knowledge, a consistent PUF integration into a commodity IoT operating system is yet missing, even though IoT deployments increasingly rely on some (open source) operating system (OS). IoT applications built on top of an OS benefit from reduced implementation overhead and enhanced dependability as they reutilize existing, well-tested code such as network stacks, drivers, or crypto-libraries.

We argue that operating system software should provide crypto-primitives for PUF functions (see Figure 1) to make its security benefits accessible to a wide range of IoT devices. The application of software PUFs is not limited to low-cost platforms, though, but can also assist security hardware, which is occasionally vulnerable [17].

The remainder of this paper is structured as follows. After providing background on PUFs and discussing prior work (§2), we present our evaluation methodology (§3) and perform an empirical SRAM evaluation on 708 devices (§4), in which we identify static bias and stress marks introduced by past utilization. We derive requirements for supporting SRAM PUFs on the OS-level, and introduce the RIOT OS integration (§5). In a base-line evaluation of our solution (§6), we quantify the uniqueness of SRAM PUF generation. Our second analysis concentrates on the quality of random seed and key generation, as well as its performance overhead (§7). A subsequent security analysis reveals that the SRAM PUF is secure under moderate attacker assumptions (§8). Finally, we conclude and outline future research directions (§9).

2 Problem Statement and Related Work

Pappu *et al.* [18] are the first to introduce "physical one-way functions" and the notion of a PUF dates back to Gassend *et al.* [19]; both describe a technique to uniquely identify and authenticate individual integrated circuits. The research community identified PUFs as an attractive solution for the IoT [20], because the intrinsic hardware variations can feed security primitives on low-end devices without increasing hardware cost. Orthogonal to PUFs that utilize variations of low-cost, multi-purpose building blocks, research also advances in the field of hardware security at the transistor level [21], [22], [23]. PUFs can be distinguished into two classes [18], [24]. They either process many inputs (*i.e.,* challenges) to produce varying outputs (*i.e.,* responses), or only few inputs which produce few, or only one response. More precisely, a PUF is denoted as strong if it provides a large challenge-response space, whereas a weak PUF provides only few challenge-response pairs that typically scale linearly or polynomially with the design size [25]. Hence, a memory readout which produces one response can be classified as a weak PUF.

A variety of use cases for PUFs emerged, such as secure key storages [26], communication protocols [27], [28], supply chain security [29], remote attestation [30], firmware updates [31], or generic trust anchors [32].

The security of these applications as derived from PUFs is only as strong, as the secrets extracted from the underlying hardware variations. In this section, we review the fundamental properties, basic assessment measures, and pose the question of potential weaknesses in PUFs.

2.1 Properties of Uninitialized SRAM

Reading out uninitialized SRAM produces a digital fingerprint. Manufacturing processes introduce variations in the silicon of transistors that construct a memory cell. When powered on, some cells drift to the logical state 1, others to 0, and cells without bias fluctuate according to environmental conditions. The resulting patterns require a careful assessment between devices (inter-device) in order to estimate their uniqueness, and between power-cycles on one device (intra-device), in order to quantify the (random) noise. This noise can be utilized as an entropy source, or needs to be removed for reliably reproducing an exact version of the pattern.

Aging Bias. Uniformly random variations result in an equal proportion of stable cells that power up with 0 or 1 on a single SRAM pattern. Aging and utilization, however, skew this distribution, due to drifting voltage threshold values of the transistors that form a memory cell [33]. The increased probability for one symbol (1 or 0) introduces a bias, which in turn benefits an attacker, who tries to guess bit values. Guin *et al.* [34] find a bias of up to 54 % under artificial aging. Holcomb *et al.* [8] counter that 'normal' use patterns of intermittently powered devices prevent an identical skew.

The state-of-the-art motivates additional and more realistic analyses of bias and aging effects on platforms that naturally aged while executing real-world IoT applications over a long period of time (see Section 4).

Static Bias. In contrast to an aging bias, real-world PUFs may be affected by a static bias at certain bit positions [35]. Rahman *et al.* [36] observe systematic correlation between SRAM patterns across chips, and cell-neighborhood interactions due a systematic physical arrangement on the silicon. Both effects reduce the device uniqueness. An attacker, who owns a chip of the same type, could utilize a local measurement to guess bit values at specific positions with a better chance than 50 %, which facilitates prediction of a secret value derived thereof.

Conditioning the SRAM resolves systematic bias. Bit selection [37], [38] is an approach to exclude biased bit addresses of the SRAM, but adds enrollment complexity for each individual node. Storing a bit mask for cell selection requires additional memory, which conflicts with limited memory resources on IoT devices. Instead, extending the SRAM PUF input increases the total amount of unbiased bits that generate a secret, which ideally prevents successful guesswork. Increasing the length, though, threatens *fuzzy extraction* (see Section 2.4) which may leak information about the secret, and requires a careful assessment of (*i*) the

remaining entropy as well as (*ii*) the increase in processing overhead on resource constrained nodes.

Environmental Bias. PUFs are subject to noise, which is affected by environmental operating conditions. Related work analyzes SRAM startup patterns under varying voltages and temperatures. A body of work shows that SRAM PUFs are robust against variations of the supply voltage [1], [2], [3], [4], tested at $\pm 10\,\%$ of the nominal value. Adjusting the voltage ramp-up speed [5], [6] can mitigate effects of temperature variations, which affect the SRAM startup behavior more severely. These solutions require special hardware, though.

The effect of temperature variations on the startup pattern of SRAM is commonly analyzed within the industrial operating temperature range of -40°C to +80°C. Leest *et al.* [7] quantify the **intra-device** relation between startup pattern of a device, which shows that the min. entropy is minimal at a low temperature of -40°C, and gains up to $\approx$2 % of noisy cells when the temperature increases to +80°C. When quantifying the required SRAM length for seed generation, the lower bound should serve as a conservative starting point, while higher temperatures improve seed generation due to higher entropy.

Schrijen *et al.* [1] compare the intra-device hamming distance of SRAM patterns taken under different temperatures, compared to an enrollment readout at ambient temperature. The startup noise between patterns of the same device almost doubles when increasing the operation temperature from 20°C to 80°C. Claes *et al.* [3] present an increase of the fractional hamming distance from 0.06 to 0.1 at the extreme operating temperatures of -40/+80°C. Katzenbeisser *et al.* [2] present similar results but find that the SRAM PUF is more robust against varying operational conditions compared to other PUFs (*e.g.,* arbiter or flip-flop PUFs). Overdesigning the error correction code can mitigate this effect, but increases the computational complexity— sometimes in conflict with IoT device constraints as well as the remaining key entropy.

Holcomb *et al.* [8] show that the increase in noise which is introduced by temperature variations overrules a predictable aging effect of NBTI. This has a positive effect on the **inter-device** uniqueness, which increases with the absence of an identical skew.

2.2 Empirical Evaluation of PUFs

The common measure to quantify the unpredictability of a pattern is given by the min. entropy metric:

$$H_{min}(p_{max}) = -log_2(p_{max}) \tag{1}$$

For a single bit, $p_{max} = max(p, 1-p)$, *i.e.,* the maximum probability for attaining one (p) or zero ($1-p$) at the same SRAM bit position. An ideal probability of $p_{max} = 0.5$ maximizes the min. entropy to $H_{min} = 1$. This metric is used to assess intra-device variations across multiple pattern of the same device, or inter-device variations between the pattern of multiple devices. Random noise increases the intra-device min. entropy after a power-cycle, which facilitates seed generation, but challenges a reliable key construction. Over-dimensioning the *fuzzy extractor* (see Section 2.3) can mitigate this effect, but increases the computational complexity—in conflict with IoT device constraints.

Schrijen *et al.* [1] present intra-device measurements across SRAM technologies in differing setups and find that variations across SRAM of different vendors are not significant. Katzenbeisser *et al.* [2] show that the inter-device min. entropy is invariant to temperature. This enables longer repetition codes to correct multiple errors, which would otherwise leak secret information in the case of a low inter-device entropy (*cf.* Section 7).

The inter-device min. entropy assesses device uniqueness and the impact of bias. The literature reports inter-device min. entropy values from 0.7 [3] to 0.9 [39] between SRAM patterns. Quantifying this metric requires multiple samples which is particularly challenging since it involves many nodes.

Min. Entropy Convergence. The maximum probability p_{max} in Equation 1 can be empirically sampled from a limited number of probes n (*i.e.,* nodes). Then the empirical estimator

$$H'_{min}(i,n) = -log_2\left[max\left(\frac{i}{n}, 1-\frac{i}{n}\right)\right] \tag{2}$$

with i positive events (ones) in n samples from individual nodes converges to the min. entropy in Equation 1. Statistical convergence, however, is slow. According to the central limit theorem [40],

$$|H_{min}(p_{max}) - H'_{min}(i,n)| \sim \frac{\sigma}{\sqrt{n}} \ (\text{as } n \to \infty), \tag{3}$$

where the dispersion $\sigma = \sigma_{H'_{min}} \approx 1$.

Hence, estimating the inter-device bias from 100 samples of SRAM PUFs still includes an error of 10%. Accordingly, the largest available SRAM evaluation of 144 nodes [41] bears an uncertainty of more than 8% This shows the need for significantly larger samples in order to approximate the inter-device min. entropy accurately, which we will present in Section 4.

Bit-Aliasing. Maiti *et al.* [12] introduce *bit-aliasing* to quantify systematic inter-device bias (cf. Section 2.1) among 125 FPGAs that implement a ring oscillator (RO) PUF. Large-scale evaluations of RO PUFs on 217 FPGAs [42], [43], and 133 ASICs [44] show that the location of cells within the FPGA affect performance properties. The bit-alias of uninitialized SRAM between 50 [4] and 144 [41] devices reveals a slight double-peaked distribution of the bit-alias scores due to SRAM layout systematics, but seem to miss convergence due to an insufficient sample size.Wilde *et al.* [14] identify a research gap in convergence and deduce that qualified inter-device bit-alias measurements require more than 600 devices to converge with an error below 5%.

Quantifying possible inter-device correlations using hundreds of devices demands for high cost and engineering efforts. In the subsequent analyses, we will tackle these challenges by taking advantage of a large-scale testbed.

2.3 Random Seed and Key Generation

SRAM PUFs promise to support bootstrapping security on embedded IoT nodes by deriving random seeds and private keys from uninitialized memory. Commercial IoT platforms more and more provide isolated PUF circuits for this purpose,

but an open software implementation that enables PUF-functionality without dedicated PUF-circuitry is missing. To enable software-based SRAM PUFs on a wide range of heterogeneous IoT platforms, the hardware abstraction layer of an operating system can enable low-level hardware access and facilitate PUF-based seed and key generation.

Seed Generation. Random numbers are essential for security. Commonly, a sequence generated by a true random number generator (TRNG) acts as seed or refresh value for a pseudo-random number generator (PRNG) as well as a cryptographically secure PRNG (CSPRNG). Van der Leest *et al.* [7] derive the min. entropy of repeated SRAM startup patterns on a device for creating a random seed value. The concept was applied to off-the-shelf MCUs [45] and revealed a diverse picture. Not all embedded SRAM technologies are qualified to produce high entropy seeds. SRAM, so the lessons learned from this study, must be analyzed prior to deployment. Krentz *et al.* [46] propose an SRAM seeding mechanism and add antenna noise to uninitialized memory pattern. The combined values are conditioned with a van Neumann extractor, which introduces variable runtime overhead.

SRAM must be uninitialized to obtain entropy between power-cycles, which is why the PUF operation should only take place during system startup before the memory has been utilized. This startup sequence, however, might be executed without a cold boot, possibly leading to zero-entropy seeding. Hence, a PUF implementation needs to ensure a preceding power-off cycle.

Key Generation. A reliable key generation depends on the removal of random noise. The related concept of *fuzzy extraction* was first presented in the context of biometric authentication systems [47], [48] to reliably reconstruct an exact version of a reference measurement. Fuzzy extractors are based on error correction codes. Error correction schemes for PUFs [49] were evaluated on an FPGA [50]. For complexity reasons, not all codes are applicable to low-end devices with its constrained resources. Korenda *et al.* [38] reduce the computational requirements by identifying stable values before encoding, which reduces the error probability. Leest *et al.* [51] propose specific hardware implementations for soft-decision decoders, which improve the correction capabilities and require only half of the PUF bits for secret generation compared to hard decision decoders.

A deployment of a fuzzy extractor proceeds in two phases, *enrollment* and *reconstruction*. The enrollment is a trusted process and produces *helper data* [52], which is later used to reconstruct the PUF value. Helper data is publicly stored in non-volatile memory.

A PUF response does not contain maximum entropy, which flaws its immediate use as a cryptographic key. Besides, it may be too long or too short, adjusting its length to include a required amount of entropy. For mitigation, a compression scheme can be used to create a key with maximized entropy and to preserve forward secrecy of the PUF response. Practical implementations [53], [54], [55] employ a cryptographic hash function that compresses the lengthy PUF response.

Error correction [52], and crypto-processing [56] quickly exceed the computational-, and energy resources on constrained embedded devices. A modular and configurable PUF implementation should ease the deployment under varying environmental conditions and adjust to the capabilities of heterogeneous platforms (*e.g.,* processing power, availability of crypto-acceleration).

2.4 Security Analysis of PUFs

The related work presents threats to PUFs mainly from three angles.

Analytical Attacks. Public helper data techniques leak information if the PUF is biased [57]. For the *code offset* method, helper data lengths should be kept small to avoid information disclosure–in conflict with PUF bias which may increase the required length. Koeberl *et al.* [39] conservatively estimate the entropy loss during helper data construction for varying error correction codes in the fuzzy extractor, but were criticized to be overly pessimistic [52]. Maes *et al.* [58] present methods that calculate the entropy leakage exactly, and de-biasing which resolves bias on an FPGA. Liu *et al.* [59] present countermeasures to bias on an MCU.

Modeling Attacks. PUFs are susceptible to modeling attacks [60], [61], [62]. Rührmair *et al.* [63] apply machine learning to challenge-response pairs of PUFs with many inputs and predict their outputs, which requires the ability to eavesdrop PUF responses. Strieder *et al.* [64] exploit helper data of PUFs with many inputs for training. PUFs with few (or only one) input are less vulnerable to learning attacks due to restricted input/output variables.

Hardware (Invasive) Attacks. Helfmeier *et al.* [65] cloned SRAM of a common IoT device using a focused ion beam instrument. Zeitouni *et al.* [66] present a side-channel analysis on an SRAM PUF, using remanence decay. Both attacks require physical control of the device under attack.

These analyses are tied to specific algorithms, or dedicated PUF implementations in hardware or software. A practical threat model that analyses the remaining security risks of SRAM PUFs on low-end hardware from the perspective of an IoT operating system is missing. We will fill this gap in Section 8.

3 Experiment Setup

We want to analyze the properties of uninitialized SRAM on a large scale, and assess our measurements on IoT-typical constrained hardware. Therefore, we chose an existing testbed as an evaluation environment (Section 3.1), which provides many off-the-shelf nodes (Section 3.2) and grants open (remote) access for reproducibility. The drawback of this approach, however, is that we cannot vary the operational conditions of nodes.

On the software side of our experiments (Section 3.3), we chose the open source IoT operating system RIOT [15] for three reasons. (*i*) RIOT is an off-the-shelf OS that is used in many IoT deployments [67] with support of numerous heterogeneous platforms. In RIOT, PUF support brings benefit to a broad range of systems and applications. (*ii*) It provides support for the FIT IoT-LAB testbed nodes. This allows us to easily benefit from the existing tools and facilities. (*iii*) An active open source community, which had first hand experiences with initial SRAM PUF trials [68], facilitates code contributions.

3.1 Testbed Environment

We conduct our experiments on the FIT IoT-LAB testbed [69] to attain a large number of nodes. The testbed consists of seven sites with different topologies and a total number of more than 1500 nodes of 25 architectures. The *M3* nodes make up the majority (≈800 nodes) and reflect properties of commercial off-the-shelf class 2 IoT devices [16]. Each node is attached to a control node which provides a power monitor (INA220), allowing to measure the operational voltage and current that flows to the MCU and the external board components. Nodes are deployed across facilities of INRIA in France. Hence, all our experiments are conducted under environmental conditions of work offices.

We use 708 *M3* nodes in our experiments. To automate experiment control, we utilize the command-line interface `iotlabcli`. Nodes serial outputs are piped to individual log files. Note, when reproducing the experiments, high data volumes are generated, while testbed users have limited disk quota. Data compression, moving files periodically, and asking for increased quota can assist.

3.2 Hardware Platform

Testbed. *M3* nodes consist of a 32-bit ARM Cortex-M3 CPU, integrated into the STM32F103REY MCU, which runs at max. 72 MHz and provides 64 kB embedded SRAM, and 512 kB internal flash. The MCU offers common features that we exploit: (*i*) Low-power standby mode turns off the whole SRAM. All content in SRAM and registers are lost, except for the backup domain. (*ii*) Real-time clock remains operable during standby, to trigger an interrupt for wakeup. (*iii*) Power control registers indicate whether the MCU has been in standby after a system restart. But this MCU lacks hardware security features, *i.e.,* a random number generator, crypto-accelerator, and secure key storage. *M3* nodes additionally connect external components via SPI: An 16 MB external NOR flash allows storing data persistently, and a low-power radio which is the only alternative to gather entropy on this board, by sampling antenna noise.

The microchips of the *M3* nodes in the FIT IoT-LAB testbed originate from two lots and four wafers, two of which build the majority of devices. We conducted several experiments to find a systematic variation. But we could not find significant differences between these batches, hence, we treat them equally in our evaluation and exclude the results of the batch comparisons.

Local. To evaluate the PUF performance on heterogeneous IoT devices with varying architectures, we also perform local experiments on two different off-the-shelf IoT platforms, and measure the processing time on a single device per platform: The *ESP32*, which consists of an Xtensa 32-bit CPU with 520 kB SRAM, 4 MB flash, and operates at max. 240 MHz. The *HiFive* which consists of a RISC-V RV32IMAC CPU that provides 16 kB SRAM, 4 MB off-chip flash, and operates at max. 320 MHz.

3.3 Software Platform

We base our PUF implementation on RIOT 2022.01. It supports different architectures (8–32-bit CPUs), over 150 MCUs, and nearly 250 IoT boards. The OS provides multi-threading with preemption, power management, and a hardware abstraction layer to enable portability. We utilize and complement these features in our implementation (Section 5). RIOT provides its own IPv6 network stack (`GNRC`) and supports multiple low-power radios as well as wired interfaces. For the *M3* nodes, we added drivers to access power control registers and the external flash memory. To broaden our experimental basis, we integrated the PUF initialization to the *ESP32* and *HiFive* architectures.

In our experiments, we trigger repeated power cycles on the nodes. For this, we utilize existing RIOT interfaces, namely, the power management (PM) interface to enter standby, which turns-off the SRAM, and the real-time clock (RTC) to generate a future wakeup interrupt.

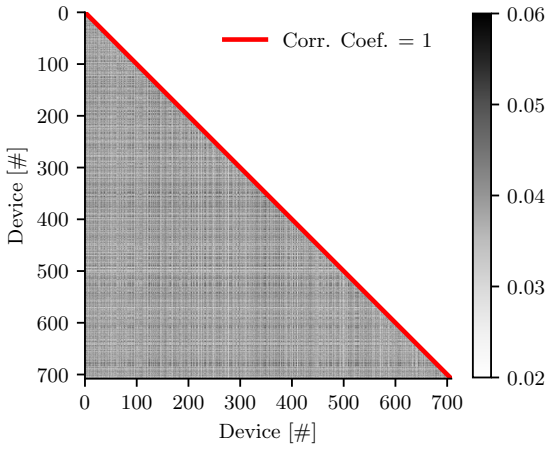


Fig. 2: SRAM correlation between 708 nodes. The Pearson product-moment correlation coefficient of each pair is encoded in gray intensity. Autocorrelation results in a coefficient of one.

4 Large Field Study of Uninitialized SRAM

4.1 Inter-device Correlation

We want to analyze the similarity between individual SRAM patterns. Therefor we read the whole memory of 708 available *M3* nodes and compute the Pearson product-moment correlation coefficient which is defined as:

$$r(a,b) = \frac{\sum_{i=1}^{m}(a_i-\bar{a})(b_i-\bar{b})}{\sqrt{\sum_{i=1}^{m}(a_i-\bar{a})^2}\sqrt{\sum_{i=1}^{m}(b_i-\bar{b})^2}} \tag{4}$$

where a and b denote the SRAM pattern of two devices with a length of m=64 kB. Figure 2 presents the matrix of correlation coefficients between the SRAM readout of all node pairs, as a measure of linear dependency between nodes. A coefficient of 1 indicates perfect correlation (pairs are equal), -1 represents negative correlation (pairs are opposite), and 0 means (linear) independence. All coefficients are small with a small positive bias (0.02–0.06), which indicates high independence between the memory patterns and motivates their usage as PUF source. Certain samples, however, indicate a slightly increased coefficient when compared to others. To better understand these correlations, we chose to further analyze

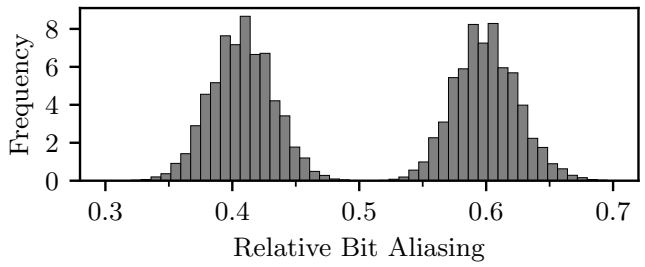


Fig. 3: Distribution of bit-alias values between 708 nodes.

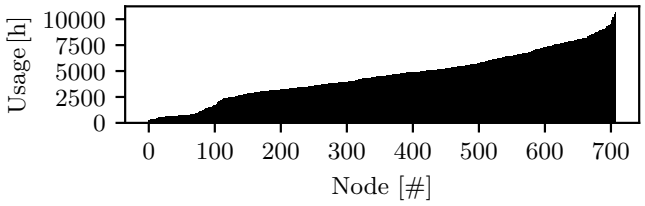


Fig. 4: *M3* node active experiment operation time in hours. Nodes are ranked according to their utilization.

the inter-device relations with a metric that incorporates the bit locality, *e.g.,* the bit-alias [12] quantifies inter-device bias (*cf.* Section 4.2).

4.2 Analysis of Static Bias

We calculate the bit-alias which quantifies the proportion of zeros and ones at every bit position j in the memory pattern between n devices:

$$\hat{p}_j = \frac{1}{n}\sum_{i=1}^{n} p_{j,i}, \tag{5}$$

where $p_{j,i}$ denotes the measured bit probability at position j of device i. If the probability for attaining one or zero is unbiased, the expectation value at each bit position equals 0.5 and the distribution of the empirical $\hat{p}_j$-values follows a normal (error) distribution. Our evaluation indicates repetitive pattern with (multiples) of 32 Bit blocks. Rahman *et al.* [36] find similar effects and relate this to the physical layout of the SRAM. Figure 3 displays the histogram of the bit-alias metric for all bit positions of the 64 kB memory. It reveals a bimodal distribution with peaks around 0.4 and 0.6. Previous work [41] suggested a double-peak distribution, but its sample size was too small [14]. To the best of our knowledge, our results show the first SRAM evaluation of the bit-alias with an error around 3 %.

Inter-device correlations in regions of SRAM can be beneficial for an attacker. Analysing a large set of equally produced devices may assist prediction of SRAM bit values at certain positions. In detail, a deviation of ≈ 0.1 from the ideal bit probability of $p_j = 0.5$ increases the chance of guessing the correct value by 10 %. This lowers the inter-device entropy, which we quantify in Section 6.1, and requires careful consideration when generating keys (see Section 7.2). While not all SRAM technologies seem to be affected by this inter-device correlation, a pre-selection of uncorrelated bits for the enrollment process can mitigate this effect [36].

4.3 Analysis of Aging

The MCU age is noted on the chip package and our local *M3* sample devices indicate a production date in January 2012. This is in line with testbed statistics that date back the first experiment to September 2012. We further managed to get experiment metadata from the testbed team. Figure 4 displays the active utilization time of our test nodes since their deployment until the end of 2021. The majority of our publicly accessible nodes have been operated 2.5–8 thousand hours since their deployment. Thus, in contrast to prior work, we analyze devices that naturally aged under real-world conditions.

We want to analyze whether certain devices or memory blocks show anomalous behavior caused by aging or wear-out from similar firmware images. To that end, we quantify the intra-device bias by calculating the relative hamming weight:

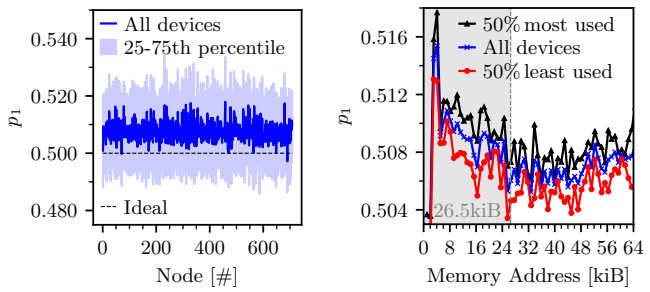
$$HW(r) = |\{r_i \neq 0 : 1 \leq i \leq m\}| \cdot \frac{1}{m} \tag{6}$$

where r denotes the bit value of one device at position i in a block of the length m. Hence, the hamming weight reflects the proportion of ones (p_1). The proportion of ones and zeros should be equal ($p_1 = 1 - p_0 = 0.5$) without bias. Figure 5a displays the intra-device measurements across the whole memory of all boards (m=64 kB) which shows an average hamming weight of $0.508 \pm 0.003(\sigma)$. This slight (positive) bias is the effect of aging and is still small compared to the results of Guin *et al.* [34] who find biases of up to 0.54 after 336 hours (14 days) of stressed operation.

Figure 5b displays the hamming weight separated into memory blocks (m=1024 Bytes). We show average values across all devices, and two subsamples that include 50 % of the most and least used devices. (*i*) An increase at $\approx$ 4 kB is introduced by the bootloader. In real-world implementations, this can barely be avoided and PUFs should exclude SRAM at that region. (*ii*) The bias of heavier utilized devices increases less used ones by ≈ 0.0025, which confirms aging by operation, with a small magnitude. (*iii*) Besides (*ii*), the first $\approx$ 26.5 kB of memory exhibit a higher skew compared to the remaining. Common firmware sizes of large-scale networking experiments on these testbed nodes report (*e.g.,* [70]) memory requirements of 22–28 kB in RAM, which matches the region of systematic wear-out. Hence, we report strong indications of visible wear-out effect by long-term testbed utilization and avoid that memory region in our PUF design (Sections 5.4 and 5.5). Operating systems likely organize the program memory from the start of the address space. At the same time, real- world firmware images do not necessarily utilize the whole memory (uniformly), which fosters bespoke unbalanced wear-out effects. Testbed operators as well as PUF developers should include anti-aging techniques in the future to mitigate wear-out patterns of various characteristics in practice.

5 PUF Design for the RIOT OS

A wide availability of PUFs requires grounding in the ecosystem of an OS. The heterogeneity of supported platforms



(a) Avg. and 25-75th perc. weight by device. (b) Avg. weight by address.

Fig. 5: 64 kB SRAM is split and analyzed in blocks of 1024 Bytes. The relative hamming weight is displayed for every device (5a) and memory address (5b); the latter distinguishes the half most/least used devices.

requires an integration into the configuration and the build system to adjust the diverse device properties. OS tests and tools provide useful interfaces to verify PUF viability and to assess crucial configuration parameters (*e.g.*, required SRAM lengths). PUFs bootstrap system security and must therefore extend the OS startup code, module initialization, and finally the secure operation. Figure 6 presents an overview of our PUF integration in RIOT for creating (*i*) a simple seed for general purpose PRNG initialization, (*ii*) a secure seed for CSPRNG initialization, and (*iii*) a secret key. In addition, to ensure qualified PUFs, we provide a soft-reset detection mechanism that prevents initialized SRAM (*i.e.*, caused by insufficient power-off cycles) from generating seeds or keys.

5.1 Compile-time Configuration

RIOT supports many boards of largely varying hardware capabilities [71] that demand for a systematic compile-time modeling of its features. This modeling enables extensible code paths where possible, and facilitates reduced feature sets on platforms without certain hardware capabilities. RIOT uses a feature modeling based on Kconfig [56], [72]. Kconfig allows defining symbols that represent features, based on which dependencies and conditional default values are defined. For the PUF module, a platform can indicate *capabilities* as follows. `HAS_PM` enables low-power mode, and `HAS_PM_TIMER` enables programmatic wake-up from low-power mode. `HAS_PM_INDICATION` enables additional power-cycle detection during soft-reset detection. `HAS_CRYPTO_ACCEL` enables crypto hardware acceleration (future work).

Both seeders (Section 5.4) and the key generator (Section 5.5) provide *configurations* for PUF algorithms: (*i*) separate start addresses in SRAM, (*ii*) length of the considered SRAM blocks, (*iii*) choice of a cryptographic hash function,(*iv*) configuration of the error correction code for the key generator. Default values are chosen according to our evaluation.

5.2 Integration into OS Startup Routine

System Reset. RIOT provides a `reset_handler`, which is the start point after every system reset. A default startup routine follows four steps. (*i*) The data section is loaded from flash to RAM. (*ii*) The *.bss* (block starting symbol) section (used for uninitialized data) is set to zero. (*iii*) The MCU and board specific components are initialized. (*iv*) The OS kernel is loaded and (*v*) `auto_init` initializes modules prior to starting applications.We perform our PUF initialization prior to step (*i*), to obtain a pristine response of uninitialized memory.

Linker Attributes and Erasure. To prevent PUF outputs from erasure by the subsequent startup routine, a *.noinit* section in the linker script of every supported CPU architecture defines a PUF attribute with which we declare variables used to store the PUF seeds and keys. Seeds and the key are consumed during `auto_init` and unavailable by the end of the initialization (see Section 5.6).

Startup Delay. PUF execution adds a delay (see Section 7.3) to the system startup, which is primarily introduced by resource intensive crypto-operations on constrained devices [56]. If available, crypto-accelerators can reduce that time. When operated in software, the execution may degrade due to the early PUF execution on perhaps uninitialized system clocks, prior to MCU initialization. An interface that allows conditional PUF execution during the next reset can mitigate this affect in the future.

5.3 Detection of Soft Resets

Memory must be uninitialized for PUF operations, which is achieved by a sufficiently long power-off cycle. Short resets can occur, however. In such cases, the PUF procedure must not be executed to prevent duplicate seeds and false key construction. Kietzmann *et al.* [9] present a simple detection mechanism to catch soft-resets. In a nutshell, the soft-reset writes a memory marker to a known address. On soft-reset, the marker will persist in memory. Conversely, a sufficient power-off cycle changes the value of the marker and enables PUF operation. One caveat of this approach is a false negative decision, which can be triggered by only a single bit flip in the marker variable, while old values stay in memory. In a baseline experiment we analyzed the hamming distance between the marker variable and the expected value and decreased the duration of the low-power cycle. Our results show notable signs of memory retention (decrease of the hamming distance) below 3 ms, which should be excluded. Longer cycles, however, may still incur memory retention, sometimes in the order of seconds [1].

Distance Detection. A future soft-reset detection stage could improve the false negative sensitivity and incorporate a distance metric to the detection algorithm. Additionally, an individual marker per device could utilize the inverse startup pattern, maximizing the range of potentially flipping bits, which increases granularity.

Sleep State Report Interface. The memory marker is at risk to be manipulated during runtime, by software defects, or intentionally by adversaries that manage to execute malicious code (Section 8). This can cause an undetected soft-reset, resulting in zero-entropy seeding and false key reconstruction. We extend the power management (PM) API in RIOT by a function to report the preceding state after a reset. Only if the marker-based detection fails *and* the system starts from deep sleep, PUF operation is executed. Not all platforms support this feature, unfortunately.

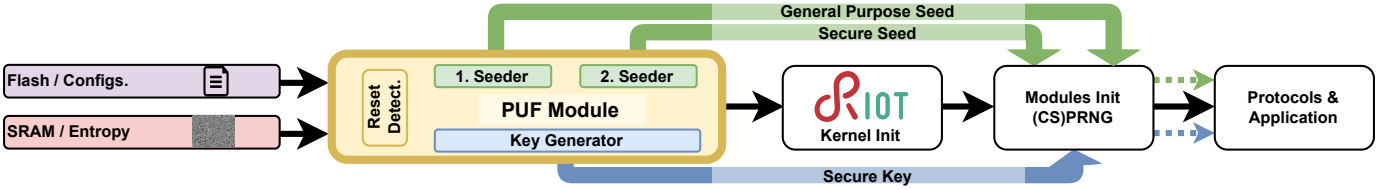


Fig. 6: Integration of the SRAM PUF module in the IoT operating system RIOT.

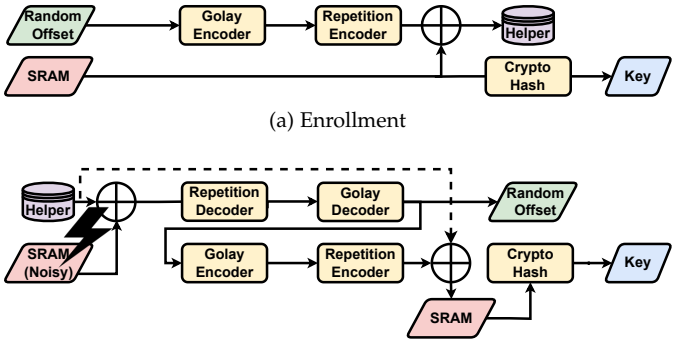
5.4 Random Seed Generation

Uninitialized SRAM contains randomness for the seed generator and is compressed to provide a concise value of maximized entropy. We provision two seed generation functions that take as input the SRAM start address and considered memory length. By default, we utilize a randomly chosen start address in the center of the memory map, to circumvent systematic wear-out effects that likely occur in the beginning of the RAM (*cf.* Section 4.3), and we locate regions for both seed functions successively. Addresses and lengths can be configured, though. A dynamic mechanism could thus mitigate potential aging phenomena.

Construction. Our first seed is extracted by the lightweight DEK hash [73] and compressed to an integer value, which is utilized to seed a non-secure general purpose PRNG. The second seed is created for security purposes and bases on compression by a cryptographic hash (SHA256 by default). Hence, the size of the seed corresponds to the digest length. It can be utilized to feed an entropy accumulator, or the CSPRNG initialization directly. Potential CSPRNG re-seeding [9], however, requires a power-cycle to obtain fresh entropy from the SRAM.

General Purpose vs Secure Seeds. General purpose seeds must not be used in cryptographic contexts due to insufficient entropy and lack of forward secrecy. Conversely, cryptographic seeds can be used for general purpose, but exhibit higher cost (see Section 7.3). The same seed must not be used for both types of generators [9], since typical PRNGs are invertible, hence, their outputs disclose information about the initial value. Similarly to PRNGs, our general purpose seed generator is invertible. Consequently, this seed can disclose information about the initial PUF response. Hence, cryptographic seed- and key generators should never operate on a memory region that was used by the simple seeder before.

Secure Seeds on Soft Reset. A fresh and secure seed that was used on CSPRNG initialization should be disguised after use to preserve privacy. Hence, we hash it after CSPRNG seeding and keep the updated value in memory, for a future soft-reset. This prevents backtracking of former random sequences. A future soft-reset adds a soft-reset counter and re-hashes it. This provides statistical variation among soft-resets (general purpose seeds follow that same procedure). Disclosure of the updated seed, however, makes future sequences predictable. Hence, a status indication field (using the *.noinit* PUF attribute) can report the PUF status persistently. CSPRNG initialization can follow its own policy to accept or reject seeding after soft-reset.



(a) Enrollment

(b) Reconstruction

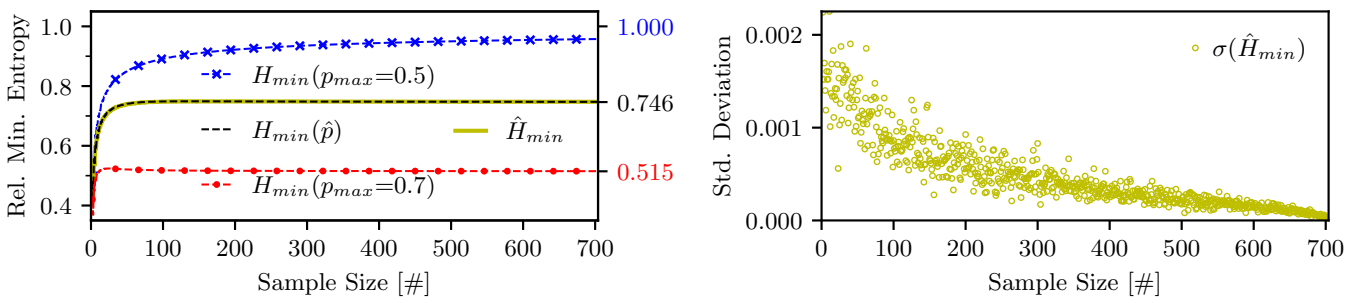
Fig. 7: A fuzzy extractor based on the code-offset construction. Offset is created at random. Deployments consist of enrollment, and reconstruction during regular device operation.

5.5 Key Generation

Our key generator follows the approach of the code offset method [47]. Deployments of such a system consist of two phases, namely the enrollment (Figure 7a), which has to be executed in a trusted environment, and the reconstruction (Figure 7b), which reflects regular device operation.

Enrollment. Our key generator provides two enrollment options. (*i*) Helper data is calculated on the device itself. This greatly simplifies a deployment and allows for re-enrollment during deployment time (*e.g.*, via firmware updates). Re-enrollment must be authenticated, though, to prevent invalidation of intact helper data. Self-assessment takes a reference measurement utilizing a low-power power-cycle. A true randomness source is required to generate the random code offset [47] (*cf.* Figure 7). We utilize the PUF based secure seed (see Section 5.4) to initialize a crypto-secure SHA256PRNG, which provides unpredictable code offsets of configurable lengths. (*ii*) Helper data is calculated externally, which is convenient for devices with very limited hardware resources. Thereby, a reference SRAM readout is transmitted via UART and an external (trusted) party deals with code offset generation and encoding. In turn, helper data is formatted into a header file that is part of the subsequent compilation of the firmware. This option requires individual compilation for every device to deploy.

For the error correction scheme, we rely on lightweight alternatives, namely, a concatenation of the Golay [74]- and repetition codes, which provide output bit error probabilities of approx. 10^{-5} to 10^{-7} for common PUF failure rates and



(a) Convergence of the min. entropy estimators for different bias values

(b) Standard deviation of min. entropy measurements

Fig. 8: Expectation and measurement of the min. entropy for varying max. probabilities (p_{max}) and increasing sample sizes.

lengths [54]. Our modular OS integration allows a seamless replacement of corrections codes in the future.

Reconstruction. A device can reconstruct the key after a power-off cycle, utilizing the helper data. After error correction, the key is calculated by a secure hash (SHA256 by default) and stored in a reserved key variable (see Section 5.2) to prevent overwriting by subsequent OS startup code. Isolated memory resources are more secure and could hold keys in future, if available on the hardware platform.

5.6 Access to PUF Primitives

Random seeds and the secret key are not directly accessible by the user to prevent unauthorized readout, misuse, or tampering. Instead, this vulnerable data is utilized during module initialization, before application code starts in `main`. Seeds are consumed on (CS)PRNG initialization and further processed to obfuscate secret start values, as well as to prepare for the case of a future soft-reset (Section 5.4). As a result, application code simply faces a readily usable (CS)PRNG. The secret key can be utilized for the initialization of consuming modules, *e.g.*, as a master key for deriving additional keys that bootstrap security protocols, or to decrypt secured storage. By the end of the module initialization, the PUF derived key is erased, to prevent direct access by the `main` application. Hence, it does not persist through a soft-reset but requires a real power-off cycle to be re-generated. Alternatively, the key can be stored in isolated memory with controlled access in the future.

6 Evaluation of OS-integrated SRAM PUFs

6.1 Estimation of the Min. Entropy Convergence

Bitwise Inter-device Minimal Entropy. We want to evaluate the unpredictability of uninitialized SRAM between multiple devices using the min. entropy. (*i*) Based on experiment data, we measure the relative frequency $p_{max} = max(p, 1-p)$ for attaining one (p) or zero ($1-p$) at the same SRAM bit position of the different devices. Based on a vector of p_{max} values for every bit position, we evaluate the empirical min. entropy for varying sample sizes (*cf.* Equation 2) For this, we pick ten sets of devices randomly, and calculate their average min. entropy and p-values. (*ii*) An estimator theoretically calculates the expected min. entropy or the empirical estimator as a function of the sample size, *i.e.*, the number of nodes, and the maximum probability for logical zero or one.

Robustness of Estimator. To assess the validity of our min. entropy measurements, we evaluate its convergence rate. We compare our measurements with a sequence of perfect Bernoulli trials and quantify the convergence for different values of p_{max} (*cf.* Section 2.2).

Figure 8a presents the results with convergence limits labeled at the right y-axis. For different p_{max} values, the estimated convergence rate varies. Exemplary, a p_{max} of 0.7 decreases the number of samples needed for convergence, but it also decreases the relative min. entropy $H_{min}(p_{max} = 0.7)$ down to ≈ 0.5. In contrast, the ideal case of $p_{max} = 0.5$ should converge to $H_{min}(p_{max} = 0.5) \approx 1$, which however does not occur within 700 displayed samples. This demonstrates the need for large sample sizes.

In our measurements, we find a relative frequency of $\hat{p}_1 = 0.596$, which slowly converges to a min. entropy of $\hat{H}_{min} \approx 0.749$ after more than 125 samples. The standard deviation of our measurements $\sigma(\hat{H}_{min})$ yields $2.3 \cdot 10^{-3}$ at max. (Figure 8b), and decreases with increasing sample sizes. A comparison of measurement results with our empirical estimator shows almost perfect agreement. We conclude that our measurements with 708 nodes are empirically robust.

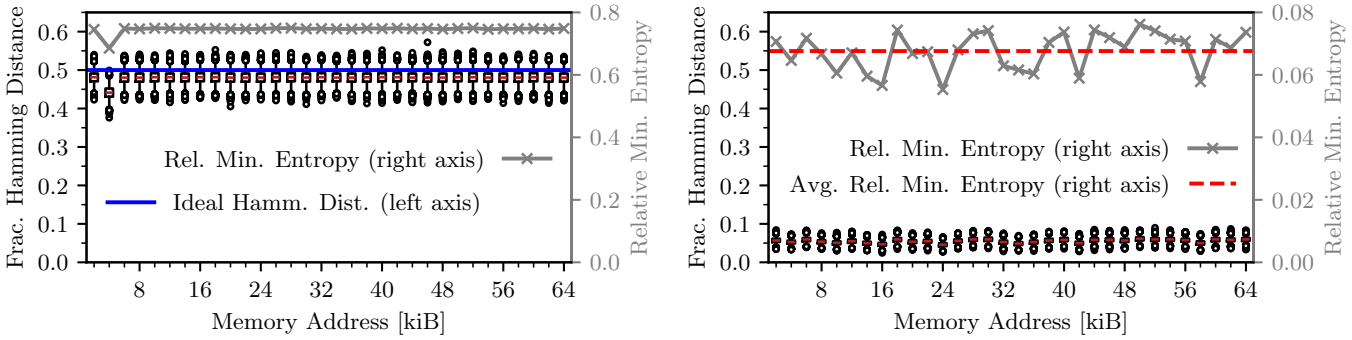
6.2 Blockwise Evaluation of the Uniqueness

Evaluation between Devices. We want to quantify the device uniqueness and analyze the fractional hamming distance [75] between devices and blocks, as a preparation to derive unpredictable secrets:

$$HD(r_a, r_b) = |\{r_{a,i} \neq r_{b,i} : 1 \leq i \leq m\}| \cdot \frac{1}{m} \qquad (7)$$

where $r_{a,i}$ and $r_{b,i}$ denote the bit values of two devices at position i in a block of the length m=1024 Bytes. Figure 9a displays our results for the fractional hamming distance between unique device pairs. Assuming a location-independent occurrence of zeros and ones, the ideal distance is 0.5. Our measurements fluctuate around an average value of 0.48 except for the block at 4 kB (bootloader, *cf.* Section 4.3), a slight deviation from the optimum case. Based on these results, we consider memory pattern as unique.

Figure 9a additionally presents the blockwise min. entropies (*cf.* Equation 1) between all devices as a lower bound



(a) Between 708 nodes.

(b) Between ≈ 700 reboots on one device.

Fig. 9: SRAM evaluation. A total of 64 kB SRAM is split and analyzed in blocks of 1024 Bytes. Boxes show fractional hamming distances between all blocks (left y-axis) and lines show min. entropies (right y-axis). IQR: 25th–75th percentile, whiskers: Q1-1.5·IQR and Q3+15·IQR.

of its uniqueness. The min. entropy is commonly used to determine input lengths in crypto-contexts (*e.g.*, key lengths). Our results reveal a min. entropy of ≈ 75 % for each block, which is in agreement with Section 6.1 and sufficient to derive unique secrets. As an example, a naive key generator would require 171 Bits of uninitialized memory to create a 128 Bit maximum entropy key.

Evaluation on a Single Device. We apply the same methodology to ≈ 700 readouts on the same device to quantify its initial randomness, required to derive distinct seeds. Thereby we utilize a low-power cycle with a sleep delay of one second. Figure 9b presents our results for the blockwise hamming distances and min. entropies. The intra-device hamming distances reveal a different picture than the inter-device analysis. Even though a majority of bits remain stable over retries, a small portion adds noise, which leads to intra-device distances of ≈ 0.06 (average). This behavior remains stable among all memory blocks. Bit flips lead to an intra-device min. entropy of 6.8 %±0.51(σ), which supports seed generation. Conversely, a reproducible key generator must eliminate these. To dimension sufficient correction schemes, we also search for the bit error probability in every block and between all measurements, and find the maximum at p_e=0.086.

7 Analysis of Seed and Key Generation

7.1 Analysis of Random Seeds

We evaluate the quality of seeding and generate two seeds on each startup, (*i*) a secure 256 Bit seed with maximum entropy, (*ii*) a 32 Bit general purpose seed for non security purposes. Our evaluation program triggers periodic power-off cycles of 1 sec. over two days, which results in ≈ 180 k values per device, and 45.1 Mbit secure / 5.7 Mbit general purpose seed bits.

Secure Seeds. We calculate the required bits from SRAM based on the intra-device min. entropy of ≈ 7 % as obtained in Section 6.2. We account for the entropy loss using the leftover hash lemma [76] ($L = log_2(1/\epsilon)$ with $\epsilon = 2^{-256}$ close to uniform) while targeting at 256 Bit entropy in our final seeds. This requires a minimum of 7314 Bits/914 Bytes of uninitialized memory. We conservatively chose 1024 Bytes. It is worth noting that SRAM portions should be chosen based on a deployment specific initial evaluation of SRAM properties. All seed values are unique and uniformly distributed due to the properties of the SHA256 hash.

General Purpose Seeds. A min. entropy of 7 % requires a minimum of 457 Bits/57 Bytes of SRAM to provide 32 Bit of seed entropy. Conservatively, we choose 128 Bytes with well aligned values in return. Figure 10 presents the probabilities of p for every bit in the 32 Bit seed, from two sample devices. They roughly follow a normal distribution and provide 89–95 % min. entropies, which we consider sufficient for non-security purposes.

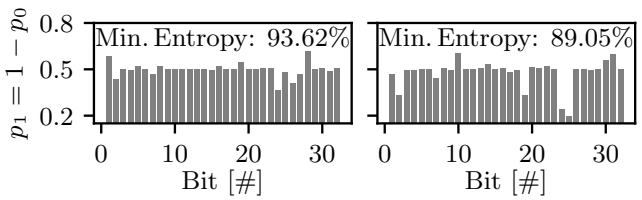


Fig. 10: Evaluation of general purpose seeds. Index based distribution of bit probabilities (p) throughout 32 Bit integer values; min. entropy across ≈ 180 k measurements per device.

7.2 Analysis of the Fuzzy Extractor for Key Generation

Figure 11 visualizes the fuzzy extractor properties for varying configurations (*cf.* Section 5.5). Similar to the seed evaluation, every configuration produces ≈ 180 k values. We vary the code offset from 9 to 24 Bytes on the y-axis, and the number of repetitions by the repetition error-correction code between 1–13 on the x-axis. A repetition of 1 reflects a single occurrence of the code word. The Golay code is active in all cases. Lower right triangles in Figure 11 (blue) encode the length of required SRAM bits. The same length is required for helper data on non-volatile memory.

Remaining Key Entropy. A naïve estimation of the entropy of a key output would multiply the SRAM length by the inter-

device min. entropy to determine its cryptographic strength. For example, a code offset of 9 Bytes with repetition 1 leads to an SRAM length of 18 Bytes/144 Bits; multiplied with a min. entropy of ≈ 0.75 would then yield 108 Bits of entropy in the SRAM used for key derivation. For biased SRAM, however, publicly available helper data leak information about the generated key [39], [52], [58], which is due to the concatenation of the two error correction codes as part of the fuzzy extractor. This leakage further reduces the remaining entropy in the key and requires additional random code offset- and SRAM bits to compensate. Maes *et al.* [58] derived methods for calculating the leakage and the remaining entropy as a function of bias, which reflects the average-case resistance against brute force attacks [77].

We determine the remaining entropy for varying fuzzy extractor configurations and for our measured SRAM bias of $\hat{p}_1 = 0.596$ (*cf.* Section 6.1). Figure 11 visualizes the results for various configurations of the fuzzy extractor. The upper left triangles (red) reflect the remaining key entropy after fuzzy extraction. Increasing code offsets increase the required SRAM length (*i.e.*, initial entropy) and the remaining entropy in the extracted keys. Increasing repetitions unsurprisingly increase the required input lengths too, whereas the remaining entropy shows a reversed trend and increases with fewer repetitions. Code offsets of 24 Bytes expose remaining entropies from 182 Bits (1 repetition) down to 82 Bits (13 repetitions). A random code offset of 24 Bytes with 5 repetitions provides 144 Bits of remaining entropy and meets the recommended security strength [78] of 128 Bits key entropy.

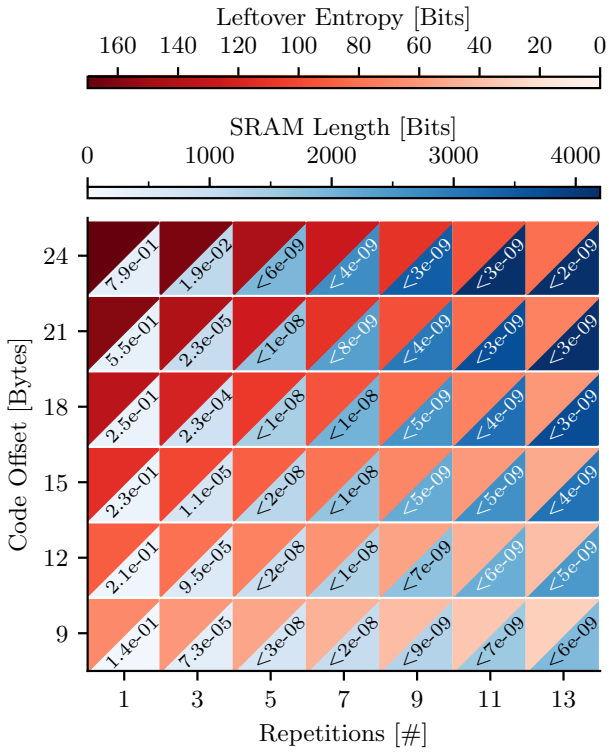


Fig. 11: SRAM length, remaining entropy, and measured reconstruction failure rate for different configurations of the fuzzy extractor.

Reliability. Figure 11 also presents the empirical reconstruction failure rate, which is introduced by bit errors between SRAM readouts that cannot be corrected by the fuzzy extractor. Increasing code offsets increases the error rate (notable in Figure 11 following repetitions 1 and 3 for bottom to top). Following repetitions fewer than five, all fuzzy extractor configurations reveal a notable failure probability, which contradicts the common key reconstruction error rate of 10^{-6} [24], [50], [51], [58]. Five or more Repetitions expose errors smaller than $3 \cdot 10^{-8}$. In our measurements, no uncorrected bit error occurred in reconstructed outputs and the error values represent the multiplicative inverse of all successfully reconstructed bits.

Discussion. Increasing the SRAM length is undesirable since memory is sparse on very constrained IoT devices. Repetitions should remain few to avoid entropy loss. Conversely, multiple repetitions are required to provide an acceptable reconstruction rate, in particular for deployments of large SRAM noise level [1]. A code offset of 24 Bytes and five repetitions preserves sufficient key entropy on our *M3* nodes at a failure rate that meets the requirements for a PUF design. Other fuzzy extractor configurations either sacrifice reliability by an intolerable reconstruction failure rate at the required level of security, or they sacrifice the remaining key entropy. Our overall balanced strategy provides highly unique and reliable device identities at an acceptable security level. Preprocessing of the SRAM pattern as proposed in [36], [37], [59] can further reduce the required SRAM length and increase the remaining key entropy from biased SRAM PUFs, which promises to improve the performance at the same or better security strength.

7.3 Resource Overhead

Processing Time. We measure processing times on *M3* nodes and compare the PUF performance with two different off-the-shelf IoT platforms: *ESP32* and *HiFive* (see Section 3).

First, we analyze the startup latency of two RIOT applications executed on the *M3* node, without the PUF module. (*i*) `Hello world` is a minimal single-threaded application and introduces a startup latency of 1.1 ms. (*ii*) `gnrc_networking` is the standard IPv6 networking application which initializes many modules in 8 threads prior to execution of application code. This requires 10.8 ms for startup. The latter case excludes seed generation. Here, we utilize a static CSPRNG seed, since microcontroller lacks an entropy source (without the PUF).

Table 1 presents the processing overhead of (*i*) soft-reset detection, (*ii*) common routines, and (*iii*) both seed generators. Soft-reset detection is mandatory with our PUF module and adds a small overhead of $< 8\,\mu$s on the *M3* node. *ESP32* adds $\approx 23\,\mu$s and *HiFive* surprisingly requires ≈ 65 times longer than *M3*. This is an effect of PUF operation prior to system clock initialization. Common processing adds a negligible overhead on all platforms. General purpose seed generation ($\approx$ 0.02–0.13 ms) is lean compared to secure seed generation (up to 14 ms on *M3*) which is comparable to `gnrc_networking`, though, seed generation from real entropy sources is slow in general [9]. Secure seeds take $\pm$ 12 ms

TABLE 1: Additional operating system startup latencies introduced by soft-reset detection and generation of two seeds.

Platform	Soft-reset detection [ms]	Common routines [ms]	Seed generation [ms]	
			simple	secure
M3	$7.65 \cdot 10^{-3}$	$3.00 \cdot 10^{-3}$	0.13	13.65
ESP32	$22.59 \cdot 10^{-3}$	$0.47 \cdot 10^{-3}$	0.02	1.37
HiFive	$494.91 \cdot 10^{-3}$	$1.50 \cdot 10^{-3}$	0.08	27.03

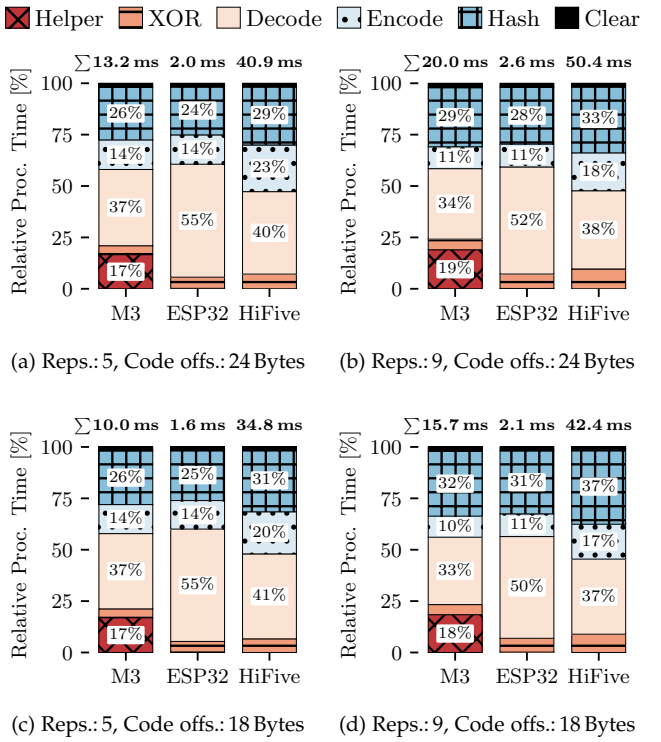
on *ESP32* and *HiFive*. Flash memory access during SHA256 computation is slower on *HiFive* due to a serial interface. In agreement with previous measurements, processing times do not directly reflect CPU frequency. Initializing clocks prior to PUF execution can improve performance in the future, but requires rearrangement of the OS startup routine. Exemplary, we rearrange the startup code for the *M3* platform and find a speedup of almost 7 times, though, system clock speed increased by a factor of 9, comparing the hardware default state (8 MHz) and the RIOT configuration (72 MHz).

Next, we look at the processing overhead of the fuzzy extractor and focus on reconstruction since enrollments happen rarely. We present four relevant configurations for key construction in Figure 12. 'Helper' contains readout of the helper data from flash. 'XOR' contains the overhead from bitwise xor operation at the input and output of the fuzzy extractor (Figure 7a). 'Decode' includes overhead of the concatenated Golay- and repetition decoder, and 'Encode' includes renewed encoding of the corrected code offset. 'Hash' calculates a digest over the reconstructed PUF measurement. Finally, 'Clear' contains the overhead of resetting vulnerable data structures after usage.

The absolute latency (numbers above bars) depends on the SRAM length and requires 10–20 ms on *M3* in all presented cases. The order of magnitude compares to `gnrc_networking` and the secure seed generator. Reconstruction and seed generation add to the existing startup latency, though. Other platforms reflect results from Table 1 and take 1.6–2.6 ms (*ESP32*) and 35–50 ms (*HiFive*) respectively. Readout of the helper data is only notable on the *M3* ($\approx$17 %) due to its slow NOR flash. The relative processing time for fuzzy extraction increases almost linearly with longer code offsets (Figure 12 bottom to top). Increasing the number of repetitions (Figure 12 left to right) also increases the relative hashing time for a reduction in decoding. Longer inputs affect the cryptographic hash efforts moderately more than the simple decoder.

In summary, the collection of PUF features moderately delays the startup routines of our sample applications. This motivates our modular design, which allows for selective configuration of PUF features. Furthermore, a positive soft-reset detection skips parts of the PUF execution. The order of tens of milliseconds is still small compared to the required SRAM power-off time (1 second has proven suitable for different platforms) to generate a fresh memory pattern. In practice, most IoT applications only awake a few times per hour or day, which obviates the latency overhead.

Energy Consumption. Table 2 presents the average current flow and the energy consumption of PUF execution on the



(a) Reps.: 5, Code offs.: 24 Bytes (b) Reps.: 9, Code offs.: 24 Bytes

(c) Reps.: 5, Code offs.: 18 Bytes (d) Reps.: 9, Code offs.: 18 Bytes

Fig. 12: Additional OS startup latency introduced by PUF reconstruction for four configurations of the fuzzy extractor on different boards. Unlabeled bars relate to proportions below 10%.

M3 node, including the generation of two seeds following four configurations of our fuzzy extractor, in agreement with Figure 12. To compare these results, we have added a *Comparative case* which acts as a simple alternative for seed- and key inclusion without the PUF. Thereby, we create two seeds by requesting noise from the external radio module (without further conditioning) on the *M3* board and assume a pre-provisioned key to persist in external flash memory, which we import. The quality of randomness, unpredictability, and uniqueness, in this case does not compare to the PUF. PUF execution prior to systems initialization drains a small current of less than 8.3 mA on average, whereas the energy consumption ranges from 680–942 μJ, depending on the input length. This is in line with the processing time. Our *Comparative case* drains more than five times higher current compared to the PUF, for two reasons. First, this case additionally requires the radio module to be powered. Second, it is operated after systems initialization. This speeds up the execution time ($\approx$3 ms), however, a higher clock speed further increases the MCU current, leading to a total energy consumption of $\approx$ 440 μJ. In summary, the energy demands of the PUF are on the same order of magnitude compared to a simplified use case, and increase the consumption by a factor between 1.5 and 2. Therefore, our PUF contributes conditioned seeds and uniform keys across devices, at little current drain, without depending on additional external hardware modules on the boards.

TABLE 2: Current- and energy consumption of the PUF for four configurations of the fuzzy extractor, and a comparative use case—measured on an *M3* node.

Reps. [#]	**5**	**9**	**5**	**9**	*Comparative*
Code offs. [B]	**24**	**24**	**18**	**18**	*case*
Avg. Current [mA]	8.26	8.19	8.28	8.07	*44.72*
Energy [μJ]	757.49	941.88	681.06	807.21	*439.52*

8 Security Analysis

PUFs need to maintain unpredictability and unclonability. A secret is embedded in the chip, hence, no seed or key is stored during device sleep, the prevalent state of a battery-driven IoT device. Secrets only persist during a short time after system startup, reducing the attack vector to a limited time. Practical attacks, however, can still exploit a number of vectors. We identify (*i*) assets, (*ii*) attackers, and (*iii*) attack surfaces of our PUF module, and present (*iv*) threats. Risks arise from the combination of specific hardware capabilities, the deployment consideration, application requirements, and related attacker assumptions. Hence, we are aiming to provide an overview of the prevalent risks, together with a series of mitigations.

8.1 Assets

The most vulnerable resources of the SRAM PUF are the uninitialized memory pattern (**A1**), the output of the PUF, namely secure seeds (**A2**), and the key (**A3**). These assets must preserve confidentiality and integrity. In our implementation, the memory marker (**A4**) (*e.g.,* for soft-reset detection, see Section 5) persist after OS startup and is vulnerable because it controls the next reset behavior, *i.e.,* can instruct to skip or execute the PUF on a future reset. Hence, this data must preserve integrity. Non-volatile memory (**A5**) stores helper data that is required for key reconstruction. Although helper data is considered public, it is still susceptible. It must preserve integrity and availability to reconstruct the PUF correctly. Authenticity is also desired, but conventionally very challenging to achieve.

8.2 Adversaries

We distinguish two types of adversaries. First, **software** attackers that try to compromise, manipulate, or analyze the system under attack without hardware access. This includes crypto-analysis and the application of learning algorithms. Software attackers exploit software backdoors, weak implementations, or software bugs to reveal secret information, or disturb code execution. Considering networked nodes in the IoT, attackers can be in wireless reach or connected remotely. Second, **hardware** attackers that have direct physical device access. We distinguish two types of hardware attackers: *Non-invasive* attackers try to interface the device during sleep or operation. They utilize interfaces such as system peripherals, or try to manipulate the device operation conditions. *Invasive* hardware attackers have deep knowledge and access to advanced techniques to gather or manipulate information on the silicon level. We exclude *invasive* attacks from the reminder of this section because they are (*i*) rare due to high financial and knowledge requirements and (*ii*) very specific to chip constructions, and so are mitigations, which contradicts our goal to improve the security of cheap, heterogeneous, and possibly already deployed devices.

8.3 Surfaces

We categorize the attack surfaces into three groups. (*i*) The communication interface (**S1**), *e.g.,* the low-power radio can act as an entry point to inject malicious inputs, or be used for (crypto-) analysis of protocols that make use of random numbers derived by the PUF seed, or the key derived by the fuzzy extractor. This interface also acts as entry point for software updates (future work). (*ii*) I/Os provide an interface to the MCU (**S2**). Peripherals such as UART, SPI, or GPIO can revel system internals through logging output, and open an attack vector for interaction with the system. More crucial, debugging interfaces such as JTAG open a direct interface to the chip memory. (*iii*) The physical presence of a device (**S3**) provides a surface to operational conditions (*e.g.,* temperature, magnetic field) and the power supply.

8.4 Threats & Mitigations

We classify threats using STRIDE [80] which defines six categories of security threats: Spoofing identity (**S**), Tampering with data (**T**), Repudiation (**R**), Information disclosure (**I**), Denial of service (**D**), and Elevation of privilege (**E**). Table 3 summarizes our results and presents mitigations for hardware (**T0–T2**) and software (**T3–T6**) adversaries.

T0. An attacker manages to read non-volatile memory, by (physically) connecting to the flash memory. Without the PUF, persistent keys would be stored as plain text, directly disclosing the secret. PUFs provide additional security by storing only the public helper data in flash. This attack, however, may disclose information in cases of high bias. Hence, helper data readout should still be impractical.

T1. An attacker manages to read/write data such as the uninitialized SRAM pattern, seeds, or keys. Debug interfaces can directly interact with the processor. Adversaries that manage to connect to the debug lines and initiate a debug session, can halt the CPU during startup to read out memory. If the PUF primitive is used for authentication, this enables spoofing and elevation of privileges without repudiation. Tampering can invalidate operation leading to denial of service which, however, is simple to achieve with physical device access. It is noteworthy that PUFs do not introduce additional threats compared to pure software solutions.

T2. An attacker manages to tamper by manipulating environmental operation conditions of the device. Common examples vary the temperature or control the power supply, *e.g.,* the power-off time, operation voltage, or startup slope. This affects random physical processes, including but not limited to SRAM startup state. A reduction of entropy disqualifies seeds and discloses information, especially in combination with T0. False key reconstruction can lead to denial of service. Without the PUF, applications require alternative sources for seed generation, or sometimes use TRNGs permanently which are similarly affected by the

TABLE 3: Threat overview of the SRAM PUF integration.

No.	Threat description	Asset (§8.1)	Adversary (§8.2)	Surface (§8.3)	STRIDE (§8.4)	Mitigation
T0	Readout public helper data.	A5	Hardware	S2	I	• Conserv. entropy estim. during enrollment.
T1	Read/write data.	A1–5	Hardware	S2	STRIDE	• Enable debug port lock. • Use one-time program. memory / write protect. • Cut input/output connections. • Deploy device with tamper protect. enclosure.
T2	Manipulate operational conditions.	A1	Hardware	S3	TID	• Additional entropy sources for seed generation. • Sensors to monitor environ. conditions [79].
T3	(Crypto-)analysis of network traffic.	A1–3	Software	S1	I	• Conserv. entropy estim. during enrollment. • Short error correction codes.
T4	Readout public/secret data.	A5	Software	S1	I	• Clear memory after usage. • Separate mem. for non-/secure seeds and key.
T5	Overwrite control data.	A4–5	Software	S1	TRID	• Enable hardware assisted soft-reset detection.
T6	Control operational conditions.	A1-3	Software (&Hardware)	S1	TID	• Enable hardware assisted voltage detection.

environment. PUFs thus act as an additional source of entropy to increase seed security.

T3. An attacker monitors (secured) network traffic that utilizes random numbers or keys. This attack might be complemented by owning and analyzing the SRAM on a device of the same type, exploiting bias to predict the initial SRAM pattern. Crypto-analysis of the output of known algorithms can disclose information of secrets derived from insufficient entropy. Combined with T0, learning attacks become a risk [64] in these cases. Without the PUF, however, random numbers are unavailable on platforms without a TRNG which fully prevents security. Keys are sometimes shipped by the vendor and reutilized across devices, leading to zero entropy on large quantities of nodes [81], [82]. PUFs enable security contributing a uniformly random seed and a unique key that is derived from individual device variations.

T4. An attacker manages to read data structures through software backdoors, which challenges privacy regardless of PUFs (*e.g.,* compromise of keys in working memory). At the time that the network interface is up and running, SRAM is not uninitialized anymore, and vulnerable seeds should be cleared. A state compromise of a non-forward secure PRNG, however, potentially allows backtracking of the initial SRAM pattern and discloses information. Without the PUF, initial secrets are likely stored persistently in plain text. PUFs reduce this attack surface to helper data disclosure (see T0).

T5. An attacker manages to overwrite vulnerable data structures (*e.g.,* forcing buffer overflow). Tampering with the soft-reset memory marker (Section 5.3) can trigger a false negative detection on next reset, which leads to zero entropy seeding and defect key reconstruction. Similarly, tampering helper data is a risk. This causes information disclosure and enables denial of service without repudiation. Interfering with code execution threatens code execution regardless of PUFs, though.

T6. Combines T2 and T5. An attacker manages to tamper with the voltage supply through **software** interfaces–likely present in low-power OSes for undervolting. Dynamic adjustments during program execution that do not affect startup conditions after reset (sleep) are uncritical, since the PUF is processed before operation. Adjustments that persist after reset, however, are crucial. A lower voltage causes the reduction of SRAM entropy which disqualifies seeds, leading to information disclosure. Alternative random sources might be similarly affected by this attack (see T2).

Threat discussion. PUFs provide non-uniform keys across devices, which means that each device has to be attacked separately, rather than attacking one and owning all devices. The success of a hardware attacker depends on (*i*) the device accessibility and (*ii*) the additional security features of the chip. Hardware attacks are typically small-scale, which contradicts the large-scale characteristics of common IoT deployments. A *non-invasive* hardware attack requires high efforts for a single device, whereas many threats can be mitigated by standard hardware features. High security applications, however, should design specific hardware-security features and consider device enclosures.

Software attacks are more likely in the IoT since devices become accessible remotely through the network. Thereby, the attack surface reduces notably, compared to hardware attacks. Presuming an adequate enrollment, the prevalent software-based threat is given by information disclosure and tampering through a software backdoor (T4–T6). These threats, however, do not assault the PUF in particular, but generally impede operation of this constrained device class. Hence, the PUF adds a layer of security in practice. To reduce this attack surface, vendors include trusted execution environments (*e.g.,* ARM TrustZone, RISC-V PMP) on modern IoT platforms, that allow for code isolation and privileged memory access. Privileged PUF operations can improve security by separating user facing, networking, or driver code from secure operations. Conversely, secure processing environments require a root of trust, which can be assisted with a PUF. Hence, both features could complement each other in the future.

9 Conclusion and Outlook

This paper started from the observation that many commodity IoT devices provide little to no hardware security features, sometimes not even a source of randomness. We presented the first comprehensive PUF integration into an IoT operating system to fill this gap and broadly enhance embedded security. Our PUF proposal uses uninitialized SRAM, which is available on common IoT platforms, and is portable due to an integration below the hardware abstraction layer of the open-source operating system.

We evaluated SRAM PUF on typical class 2 devices in an open testbed using 708 nodes. This is, to the best of our knowledge, the first empirical PUF study with several hundreds constrained IoT nodes, albeit prior work [14] proved the need for large sample sizes for the subtle analysis of SRAM bias. Our analysis revealed four key insights.

(*i*) An inter-device distance of $\approx$48 % between node pairs shows high uniqueness, which enables the generation of unpredictable keys. Still, the physical SRAM layout introduces inter-device bias, which becomes visible when analyzing high numbers of nodes. This reduces the inter-device min. entropy to $\approx$75 %, and thereby the number of unpredictable bits per node. Key generation relies on public helper data, which may reveal information about the SRAM pattern in the case of bias. Our analysis of the entropy leakage identified a fuzzy extractor configuration that results in 144 Bits of remaining key entropy at a failure rate of $6{\cdot}10^{-9}$. (*ii*) An intra-device min. entropy of $\approx$7 % allows for secure seed generation on startup. (*iii*) The uninitialized SRAM properties of real-word aged, heavily utilized testbed nodes are still sufficient to achieve (*i*) and (*ii*). (*iv*) A configurable OS integration can seamlessly provide PUF services to the IoT at moderate start-up overhead while shielding soft resets.

We could also show that a number of hardware-based *non-invasive* attacks against SRAM PUFs heavily depend on the availability of platform features such as device pinouts or debug port locks. The availability of PUFs upgrades the security of commercial off-the-shelf devices without cryptographic hardware and strengthens the resistance against the more dangerous software attacks from remote parties throughout the Internet. Contributing non-uniform keys across devices, our PUF integration reduces the efficacy of these attacks, since each node needs to be attacked individually, rather than attacking one and owning all.

This work opens four future research directions. First, pre-processing of biased SRAM pattern may increase the security of keys while reducing the fuzzy extractor overhead, but it adds a layer of complexity to the generation process, which needs careful evaluation on resource constrained devices. Second, an aging detection and an anti-aging stage may observe and mitigate entropy loss on degrading nodes. Third, the PUF functions can be extended to include trusted execution environments, which become increasingly available on modern hardware. Fourth, integrated analysis tools may improve estimates of entropy and SRAM length. We hope this will ease deployment efforts toward a future, more secure IoT.

Acknowledgments. We would like to thank Nils Wisiol for his careful feedback, which has significantly helped to improve the paper. This work was supported in part by the German Federal Ministry for Education and Research (BMBF) within the project *PIVOT: Privacy-Integrated design and Validation in the constrained IoT.*

Availability of software and reproducibility. We support reproducible research ([83], [84]) and utilize open source software and open testbed platforms. All of our work is publicly released. The code of the software components, pre-compiled binary images, the implementation of the estimator, documentation, data sets and related tools are available on GitHub at https://github.com/inetrg/IEEE-TDSC-PUF23.

References

[1] G.-J. Schrijen and V. van der Leest, "Comparative analysis of SRAM memories used as PUF primitives," in *DATE '12: Design, Automation Test in Europe Conference Exhibition*. Piscataway, NJ, USA: IEEE, 2012, pp. 1319–1324.

[2] S. Katzenbeisser, Ü. Kocabaş, V. Rožić, A.-R. Sadeghi, and I. V. C. Wachsmann, "PUFs: Myth, Fact or Busted? A Security Evaluation of Physically Unclonable Functions (PUFs) Cast in Silicon," in *Cryptographic Hardware and Embedded Systems (CHES '12)*, E. Prouff and P. Schaumont, Eds. Berlin, Heidelberg: Springer–Verlag, 2012, pp. 283–301.

[3] M. Claes, V. van der Leest, and A. Braeken, "Comparison of SRAM and FF PUF in 65nm Technology," in *Information Security Technology for Applications*, P. Laud, Ed. Berlin, Heidelberg: Springer–Verlag, 2012, pp. 47–64.

[4] M. Barbareschi, E. Battista, A. Mazzeo, and N. Mazzocca, "Testing 90 nm microcontroller SRAM PUF quality," in *10th International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS'15)*. Piscataway, NJ, USA: IEEE, 2015.

[5] M. Cortez, S. Hamdioui, V. van der Leest, R. Maes, and G.-J. Schrijen, "Adapting voltage ramp-up time for temperature noise reduction on memory-based PUFs," in *International Symposium on Hardware-Oriented Security and Trust (HOST'13)*. Piscataway, NJ, USA: IEEE, 2013, pp. 35–40.

[6] J. Lee, D.-W. Jee, and D. Jeon, "Power-up control techniques for reliable SRAM PUF," *IEICE Electronics Express*, vol. 16, no. 13, 2019.

[7] V. van der Leest, E. van der Sluis, G.-J. Schrijen, PimTuyls, and H. Handschuh, *Efficient Implementation of True Random Number Generator Based on SRAM PUFs*. Berlin, Heidelberg: Springer, 2012, pp. 300–318.

[8] D. E. Holcomb, W. P. Burleson, and K. Fu, "Power-Up SRAM State as an Identifying Fingerprint and Source of True Random Numbers," *IEEE Transactions on Computers*, vol. 58, no. 9, pp. 1198–1210, 2009.

[9] P. Kietzmann, T. C. Schmidt, and M. Wählisch, "A Guideline on Pseudorandom Number Generation (PRNG) in the IoT," *ACM Comput. Surv.*, vol. 54, no. 6, pp. 112:1–112:38, July 2022. [Online]. Available: https://dl.acm.org/doi/10.1145/3453159

[10] D. Kumar, K. Shen, B. Case, D. Garg, G. Alperovich, D. Kuznetsov, R. Gupta, and Z. Durumeric, "All Things Considered: An Analysis of IoT Devices on Home Networks," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 1169–1185.

[11] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, "Understanding the Mirai Botnet," in *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, Aug. 2017, pp. 1093–1110.

[12] A. Maiti, J. Casarona, L. McHale, and P. Schaumont, "A large scale characterization of RO-PUF," in *International Symposium on Hardware-Oriented Security and Trust (HOST'10)*. Piscataway, NJ, USA: IEEE, 2010, pp. 94–99.

[13] S. V. Kumar, C. H. Kim, and S. S. Sapatnekar, "Impact of NBTI on SRAM read stability and design for reliability," in *7th International Symposium on Quality Electronic Design (ISQED'06)*. Los Alamitos, CA, USA: IEEE Computer Society, 2006.

[14] F. Wilde and M. Pehl, "On the Confidence in Bit-Alias Measurement of Physical Unclonable Functions," in *International New Circuits and Systems Conference (NEWCAS'19)*. Piscataway, NJ, USA: IEEE, 2019.

[15] E. Baccelli, C. Gündogan, O. Hahm, P. Kietzmann, M. Lenders, H. Petersen, K. Schleiser, T. C. Schmidt, and M. Wählisch, "RIOT: an Open Source Operating System for Low-end Embedded Devices in the IoT," *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 4428–4440, December 2018. [Online]. Available: http://dx.doi.org/10.1109/JIOT.2018.2815038

[16] C. Bormann, M. Ersue, and A. Keranen, "Terminology for Constrained-Node Networks," IETF, RFC 7228, May 2014.

[17] The Hacker News, "A Critical Random Number Generator Flaw Affects Billions of IoT Devices," https://thehackernews.com/2021/08/a-critical-random-number-generator-flaw.html, last accessed 29-03-2022, 2021.

[18] R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld, "Physical One-Way Functions," *Science*, vol. 297, no. 5589, pp. 2026–2030, 2002.

[19] B. Gassend, D. Clarke, van Marten Dijk, and S. Devadas, "Silicon Physical Random Functions," in *Proc. of the 9th ACM Conference on Computer and Communications Security (CCS '02)*. New York, NY, USA: ACM, 2002, pp. 148–160.

[20] M. G. Samaila, M. Neto, D. A. B. Fernandes, M. M. Freire, and P. R. M. Inácio, "Challenges of securing Internet of Things devices: A survey," *Security and Privacy*, vol. 1, no. 2, p. e20, 2018.

[21] S. Satpathy, S. K. Mathew, V. Suresh, M. A. Anders, H. Kaul, A. Agarwal, S. K. Hsu, G. Chen, R. K. Krishnamurthy, and V. K. De, "A 4-fJ/b Delay-Hardened Physically Unclonable Function Circuit With Selective Bit Destabilization in 14-nm Trigate CMOS," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 4, pp. 940–949, 2017.

[22] S. Taneja, V. K. Rajanna, and M. Alioto, "In-Memory Unified TRNG and Multi-Bit PUF for Ubiquitous Hardware Security," *IEEE Journal of Solid-State Circuits*, vol. 57, no. 1, pp. 153–166, 2022.

[23] Y. He, D. Li, Z. Yu, and K. Yang, "ASCH-PUF: A "Zero" Bit Error Rate CMOS Physically Unclonable Function With Dual-Mode Low-Cost Stabilization," *IEEE Journal of Solid-State Circuits*, pp. 1–11, early access, Jan. 2023.

[24] J. Guajardo, S. S.Kumar, G.-J. Schrijen, and P. Tuyls, "FPGA Intrinsic PUFs and Their Use for IP Protection," in *Cryptographic Hardware and Embedded Systems (CHES'07)*, P. Paillier and I. Verbauwhede, Eds. Berlin, Heidelberg: Springer–Verlag, 2007, pp. 63–80.

[25] T. McGrath, I. E. Bagci, Z. M. Wang, U. Roedig, and R. J. Young, "A PUF taxonomy," *Applied Physics Reviews*, vol. 6, no. 1, p. 011303, 2019.

[26] I. Eichhorn, P. Koeberl, and V. van der Leest, "Logically Reconfigurable PUFs: Memory-Based Secure Key Storage," in *Proc. of the 6th ACM Workshop on Scalable Trusted Computing (STC '11)*. New York, NY, USA: ACM, 2011, pp. 59–64.

[27] U. Chatterjee, R. S. Chakraborty, and D. Mukhopadhyay, "A PUF-Based Secure Communication Protocol for IoT," *ACM Trans. Embed. Comput. Syst.*, vol. 16, no. 3, 2017.

[28] G. Bianchi, A. L. Rosa, and G. Restuccia, "RIOT-AKA: cellular-like authentication over IoT devices," in *29th IEEE Int. Conf. on Network Protocols (ICNP'21)*. Piscataway, NJ, USA: IEEE, 2021, pp. 1–6.

[29] A. Falcone, C. Felicetti, A. Garro, A. Rullo, and D. Saccà, "PUF-Based Smart Tags for Supply Chain Management," in *16th International Conference on Availability, Reliability and Security (ARES'21)*. New York, NY, USA: ACM, 2021.

[30] S. Schulz, A.-R. Sadeghi, and C. Wachsmann, "Short Paper: Lightweight Remote Attestation Using Physical Functions," in *Proc. of the 4th ACM Conference on Wireless Network Security (WiSec '11)*. New York, NY, USA: ACM, 2011, pp. 109–114.

[31] M. A. Prada-Delgado, A. Vázquez-Reyes, and I. Baturone, "Trustworthy firmware update for Internet-of-Thing Devices using physical unclonable functions," in *Global Internet of Things Summit (GIoTS '17)*. Piscataway, NJ, USA: IEEE, 2017, pp. 1–5.

[32] Marten van Hulst, "Anchoring TrustZone with SRAM PUF," https://community.arm.com/arm-community-blogs/b/architectures-and-processors-blog/posts/anchoring-trustzone-with-sram-puf, last accessed 09-29-2021, 2019.

[33] R. Faraji and H. R. Naji, "Adaptive Technique for Overcoming Performance Degradation Due to Aging on 6T SRAM Cells," *IEEE Transactions on Device and Materials Reliability*, vol. 14, no. 4, pp. 1031–1040, 2014.

[34] U. Guin, W. Wang, C. Harper, and A. D. Singh, "Detecting Recycled SoCs by Exploiting Aging Induced Biases in Memory Cells," in *International Symposium on Hardware Oriented Security and Trust (HOST'19)*. Piscataway, NJ, USA: IEEE, 2019, pp. 72–80.

[35] F. Wilde, C. Frisch, and M. Pehl, "Efficient Bound for Conditional Min-Entropy of Physical Unclonable Functions Beyond IID," in *International Workshop on Information Forensics and Security (WIFS'19)*. Piscataway, NJ, USA: IEEE, 2019.

[36] M. T. Rahman, A. Hosey, Z. Guo, J. Carroll, D. Forte, and M. Tehranipoor, "Systematic Correlation and Cell Neighborhood Analysis of SRAM PUF for Robust and Unique Key Generation," *Journal of Hardware and Systems Security*, vol. 1, no. 2, pp. 137–155, 2017.

[37] K. Xiao, M. T. Rahman, D. Forte, Y. Huang, M. Su, and M. Tehranipoor, "Bit selection algorithm suitable for high-volume production of SRAM-PUF," in *International Symposium on Hardware-Oriented Security and Trust (HOST'14)*. Piscataway, NJ, USA: IEEE, 2014, pp. 101–106.

[38] A. R. Korenda, F. Afghah, B. Cambou, and C. Philabaum, "A Proof of Concept SRAM-based Physically Unclonable Function (PUF) Key Generation Mechanism for IoT Devices," in *Workshop on Security Trust and Privacy in Emerging Cyber-Physical Systems (SECON'19)*. Piscataway, NJ, USA: IEEE, 2019.

[39] P. Koeberl, J. Li, A. Rajan, and W. Wu, "Entropy loss in PUF-based key generation schemes: The repetition code pitfall," in *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST '14)*. Piscataway, NJ, USA: IEEE, 2014, pp. 44–49.

[40] W. Feller, *An Introduction to Probability Theory and Its Applications*, 2nd ed. New York: Wiley & Sons, 1971, vol. 2.

[41] F. Wilde, "Large Scale Characterization of SRAM on Infineon XMC Microcontrollers as PUF," in *Proc. of the 4th Workshop on Cryptography and Security in Computing Systems (CS2'17)*. New York, NY, USA: ACM, 2017, pp. 13–18.

[42] R. Hesselbarth, F. Wilde, C. Gu, and N. Hanley, "Large scale RO PUF analysis over slice type, evaluation time and temperature on 28nm Xilinx FPGAs," in *International Symposium on Hardware-Oriented Security and Trust (HOST'18)*. Piscataway, NJ, USA: IEEE, 2018, pp. 126–133.

[43] C. Gu, C.-H. Chang, W. Liu, N. Hanley, J. Miskelly, and M. O'Neill, "A large-scale comprehensive evaluation of single-slice ring oscillator and PicoPUF bit cells on 28-nm Xilinx FPGAs," *Journal of Cryptographic Engineering*, vol. 11, no. 3, pp. 227–238, 2021.

[44] M.-D. Yu, R. Sowell, A. Singh, D. M'Raïhi, and S. Devadas, "Performance metrics and empirical results of a PUF cryptographic key generation ASIC," in *International Symposium on Hardware-Oriented Security and Trust (HOST'12)*. Piscataway, NJ, USA: IEEE, 2012, pp. 108–115.

[45] A. van Herrewege, V. van der Leest, A. Schaller, S. Katzenbeisser, and I. Verbauwhede, "Secure PRNG Seeding on Commercial Off-the-shelf Microcontrollers," in *3rd International Workshop on Trustworthy Embedded Devices (TrustED '13)*. New York, NY, USA: ACM, 2013, pp. 55–64.

[46] K. Krentz, C. Meinel, and H. Graupner, "Secure self-seeding with power-up SRAM states," in *ISCC '17: Symposium on Computers and Communications*. Heraklion, Greece: IEEE, 2017, pp. 1251–1256.

[47] A. Juels and M. Wattenberg, "A Fuzzy Commitment Scheme," in *Proc. of the 6th ACM Conference on Computer and Communications Security (CCS '99)*. New York, NY, USA: ACM, 1999, pp. 28–36.

[48] Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith, "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.

[49] M.-D. Yu and S. Devadas, "Secure and robust error correction for physical unclonable functions," *IEEE Design & Test of Computers*, vol. 27, no. 1, pp. 48–65, 2010.

[50] M. Hiller, L. Kürzinger, and G. Sigl, "Review of error correction for PUFs and evaluation on state-of-the-art FPGAs," *Journal of Cryptographic Engineering*, vol. 10, no. 3, pp. 229–247, 2020.

[51] V. van der Leest, B. Preneel, and E. van der Sluis, "Soft Decision Error Correction for Compact Memory-Based PUFs Using a Single Enrollment," in *Cryptographic Hardware and Embedded Systems (CHES '12)*, E. Prouff and P. Schaumont, Eds. Berlin, Heidelberg: Springer–Verlag, 2012, pp. 268–282.

[52] J. Delvaux, D. Gu, D. Schellekens, and I. Verbauwhede, "Helper Data Algorithms for PUF-Based Key Generation: Overview and Analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 6, pp. 889–902, 2015.

[53] G. S. Edward and S. Devadas, "Physical Unclonable Functions for Device Authentication and Secret Key Generation," in *Proc. of the 44th Annual Design Automation Conference (DAC '07)*. New York, NY, USA: ACM, 2007, pp. 9–14.

[54] C. Bösch, J. Guajardo, A.-R. Sadeghi, J. Shokrollahi, and P. Tuyls, "Efficient Helper Data Key Extractor on FPGAs," in *Cryptographic*

Hardware and Embedded Systems - CHES 2008. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 181–197.
[55] R. Maes, A. V. Herrewege, and I. Verbauwhede, "PUFKY: A Fully Functional PUF-Based Cryptographic Key Generator," in *Cryptographic Hardware and Embedded Systems (CHES '12)*, E. Prouff and P. Schaumont, Eds. Berlin, Heidelberg: Springer–Verlag, 2012, pp. 302–319.
[56] P. Kietzmann, L. Boeckmann, L. Lanzieri, T. C. Schmidt, and M. Wählisch, "A Performance Study of Crypto-Hardware in the Low-end IoT," in *International Conference on Embedded Wireless Systems and Networks (EWSN'21)*. New York, USA: ACM, February 2021. [Online]. Available: https://dl.acm.org/doi/10.5555/3451271.3451279
[57] M.-D. M. Yu, D. M'Raïhi, S. Devadas, and I. Verbauwhede, "Security and Reliability Properties of Syndrome Coding Techniques Used in PUF Key Generation," 2013.
[58] R. Maes, V. van der Leest, E. van der Sluis, and F. Willems, "Secure key generation from biased PUFs: extended version," *Journal of Cryptographic Engineering*, vol. 6, no. 2, pp. 121–137, 2016.
[59] H. Liu, W. Liu, Z. Lu, Q. Tong, and Z. Liu, "Methods for Estimating the Convergence of Inter-Chip Min-Entropy of SRAM PUFs," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 2, pp. 593–605, 2018.
[60] F. Ganji, S. Tajik, F. Fäßler, and J.-P. Seifert, "Strong Machine Learning Attack Against PUFs with No Mathematical Model," in *Cryptographic Hardware and Embedded Systems (CHES'16)*. Berlin, Heidelberg: Springer–Verlag, 2016, pp. 391–411.
[61] J. Shi, Y. Lu, and J. Zhang, "Approximation Attacks on Strong PUFs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2138–2151, 2020.
[62] N. Wisiol, B. Thapaliya, K. T. Mursi, J.-P. Seifert, and Y. Zhuang, "Neural Network Modeling Attacks on Arbiter-PUF-Based Designs," *IEEE Transactions on Information Forensics and Security*, 2022.
[63] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber, "Modeling Attacks on Physical Unclonable Functions," in *Proc. of the 17th ACM Conference on Computer and Communications Security (CCS'10)*. New York, NY, USA: ACM, 2010, pp. 237–249.
[64] E. Strieder, C. Frisch, and M. Pehl, "Machine Learning of Physical Unclonable Functions using Helper Data: Revealing a Pitfall in the Fuzzy Commitment Scheme," *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES '21)*, vol. 2021, no. 2, pp. 1–36, 2021.
[65] C. Helfmeier, C. Boit, D. Nedospasov, and J.-P. Seifert, "Cloning Physically Unclonable Functions," in *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST '13)*. Piscataway, NJ, USA: IEEE, June 2013, pp. 1–6.
[66] S. Zeitouni, Y. Oren, C. Wachsmann, P. Koeberl, and A.-R. Sadeghi, "Remanence Decay Side-Channel: The PUF Case," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 6, pp. 1106–1116, 2016.
[67] Eclipse Foundation, "IoT & Edge Developer Survey Report," https://outreach.eclipse.foundation/iot-adoption-2019, last accessed 03-12-2022, 2019.
[68] P. Kietzmann, C. Gündogan, T. C. Schmidt, and M. Wählisch, "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*. New York, NY, USA: ACM, June 2018. [Online]. Available: https://doi.org/10.1145/3210240.3210805
[69] C. Adjih, E. Baccelli, E. Fleury, G. Harter, N. Mitton, T. Noel, R. Pissard-Gibollet, F. Saint-Marcel, G. Schreiner, J. Vandaele, and T. Watteyne, "FIT IoT-LAB: A large scale open experimental IoT testbed," in *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*. Piscataway, NJ, USA: IEEE Press, Dec 2015, pp. 459–464.
[70] C. Gündogan, P. Kietzmann, M. Lenders, H. Petersen, T. C. Schmidt, and M. Wählisch, "NDN, CoAP, and MQTT: A Comparative Measurement Study in the IoT," in *Proc. of 5th ACM Conference on Information-Centric Networking (ICN)*. New York, NY, USA: ACM, September 2018, pp. 159–171. [Online]. Available: https://doi.org/10.1145/3267955.3267967
[71] L. Boeckmann, P. Kietzmann, L. Lanzieri, T. C. Schmidt, and M. Wählisch, "Usable Security for an IoT OS: Integrating the Zoo of Embedded Crypto Components Below a Common API," in *International Conference on Embedded Wireless Systems and Networks (EWSN'22)*. New York, USA: ACM, October 2022, pp. 84–95. [Online]. Available: https://dl.acm.org/doi/10.5555/3578948.3578956
[72] The Linux Kernel Development Community, "Kconfig Language," https://www.kernel.org/doc/html/latest/kbuild/kconfig-language.html, last accessed 28-09-2020, 2020.
[73] D. E. Knuth, *The Art of Computer Programming (Second Edition)*. Reading, MA, USA: Addison Wesley, 2009.
[74] M. J. E. Golay, "Notes on Digital Coding," *Proc. of the Institute of Radio Engineers (IRE '49)*, vol. 37, pp. 657–657, 1949.
[75] R. W. Hamming, "Error detecting and error correcting codes," *The Bell System Technical Journal*, vol. 29, no. 2, pp. 147–160, 1950.
[76] B. Barak, Y. Dodis, H. Krawczyk, O. Pereira, K. Pietrzak, F.-X. Standaert, and Y. Yu, "Leftover Hash Lemma, Revisited," in *Advances in Cryptology (CRYPTO '11)*, P. Rogaway, Ed. Berlin, Heidelberg: Springer–Verlag, 2011, pp. 1–20.
[77] J. L. Massey, "Guessing and entropy," in *Proceedings of IEEE International Symposium on Information Theory (ISIT'94)*, 1994, p. 204.
[78] E. Barker, "Recommendation for Key Management," National Institute of Standards and Technology, Gaithersburg, MD, US, Tech. Rep. NIST SP 800-57 Part 1, May 2020.
[79] M. T. H. Anik, J.-L. Danger, S. Guilley, and N. Karimi, "Detecting Failures and Attacks via Digital Sensors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 7, pp. 1315–1326, 2021.
[80] L. Kohnfelder and P. Garg, "The threats to our products," Microsoft, Tech. Rep., 1999. [Online]. Available: https://adam.shostack.org/microsoft/The-Threats-To-Our-Products.docx
[81] E. Ronen, A. Shamir, A.-O. Weingarten, and C. O'Flynn, "IoT Goes Nuclear: Creating a ZigBee Chain Reaction," in *IEEE Symposium on Security and Privacy (SP)*. Piscataway, NJ, USA: IEEE Press, 2017, pp. 195–212.
[82] The Hacker News, "Millions of IoT Devices Using Same Hard-Coded CRYPTO Keys," https://thehackernews.com/2015/11/iot-device-crypto-keys.html, last accessed 02-12-2022, 2015.
[83] ACM, "Result and Artifact Review and Badging," http://acm.org/publications/policies/artifact-review-badging, Jan., 2017.
[84] Q. Scheitle, M. Wählisch, O. Gasser, T. C. Schmidt, and G. Carle, "Towards an Ecosystem for Reproducible Research in Computer Networking," in *Proc. of ACM SIGCOMM Reproducibility Workshop*. New York, NY, USA: ACM, August 2017, pp. 5–8.

AUTHOR BIOGRAPHY

Peter Kietzmann is a PhD student of the Internet Technologies research group at the Hamburg University of Applied Sciences. His particular interests lie in radio technologies, embedded programming, and secure IoT protocols. In the German-French research project PIVOT (Privacy-Integrated design and Validation in the constrained IoT) he is currently involved, exploring the secure protection of data on low-end devices and low-power radio networks of the ultra-constrained IoT.

Thomas C. Schmidt studied mathematics, physics, and German literature at Freie Universität Berlin (FU Berlin), Berlin, Germany, and the University of Maryland at College Park, MD. He received the Ph.D. degree in mathematical physics from FU Berlin, in 1993. He is a Professor of computer networks and Internet technologies with the Hamburg University of Applied Sciences, Hamburg, Germany, where he heads the Internet Technologies Research Group. He was the Director of the HTW Computer Centre, Berlin. Since then, he has continuously conducted numerous national and international research projects. He was the Principal Investigator in a number of EU, nationally funded, and industrial projects, as well as a Visiting Professor with the University of Reading, Reading, U.K. He is also a co-founder and a coordinator of the open source community developing the RIOT operating system. His current research interests include development, measurement, and analysis of large-scale distributed systems like the Internet or its offsprings. Dr. Schmidt has served as a Co-Editor and a Technical Expert on several occasions and is actively involved in the work of IETF and IRTF, where he co-chaired the SAM RG.

Matthias Wählisch received the Ph.D. degree (with highest honors) in computer science from Freie Universität Berlin, Berlin, Germany. He is a full professor and holds the Chair of Distributed and Networked Systems at the Faculty of Computer Science at TU Dresden. He is a co-founder and a coordinator of several successful open source projects such as RIOT. His efforts are driven by trying to improve Internet communication based on sound research. He is the PI of several national and international projects. He has authored or co-authored over 130 peer-reviewed papers (e.g., in ACM HotNets, ACM IMC, and ACM CoNEXT). His current research interests include the design and evaluation of networking protocols and architectures, as well as Internet measurements and analysis. Dr. Wählisch was a recipient of the Young Talents Award of Leibniz-Kolleg Potsdam for outstanding achievements in advancing the Internet, and recipient of the Excellent Young Scientists Award for his contributions to the Internet of Things and their prospective entrepreneurial practice. He has been active in the IETF/ IRTF since 2005. He co-organized or co-chaired over 50 scientific events, including the IEEE ICNP Ph.D. Forum 2013, ACM IMC 2017, ACM SIGCOMM 2017, and ACM ICN 2022.