Secure Self-Seeding with Power-Up SRAM States

Konrad-Felix Krentz, Christoph Meinel
Hasso-Plattner-Institut, Germany
{konrad-felix.krentz,christoph.meinel}@hpi.de

Hendrik Graupner
Bundesdruckerei, Germany
hendrik.graupner@bdr.de

***Abstract*—Generating seeds on Internet of things (IoT) devices is challenging because these devices typically lack common entropy sources, such as user interaction or hard disks. A promising replacement is to use power-up static random-access memory (SRAM) states, which are partly random due to manufacturing deviations. Thus far, there, however, seems to be no method for extracting close-to-uniformly distributed seeds from power-up SRAM states in an information-theoretically secure and practical manner. Moreover, the min-entropy of power-up SRAM states reduces with temperature, thereby rendering this entropy source vulnerable to so-called freezing attacks. In this paper, we mainly make three contributions. First, we propose a new method for extracting uniformly distributed seeds from power-up SRAM states. Unlike current methods, ours is information-theoretically secure, practical, and freezing attack-resistant rolled into one. Second, we point out a trick that enables using power-up SRAM states not only for self-seeding at boot time, but also for reseeding at runtime. Third, we compare the energy consumption of seeding an IoT device either with radio noise or power-up SRAM states. While seeding with power-up SRAM states turned out to be more energy efficient, we argue for mixing both these entropy sources.**

***Keywords*—Self-configuration, true random number generation, pseudo-random number generation.**

I. Introduction

In the Internet of things (IoT), a lot of security issues persist. One of them is the generation of cryptographic random numbers, which is a preliminary to many cryptographic operations. More specifically, cryptographically-secure pseudo-random number generators (CSPRNGs) for IoT devices exist [1], [2], but there appears to be no established way to seed the CSPRNG of an IoT device.

Asking the user to preload his IoT devices with seeds is a straightforward solution [1], but is tedious if done via cables. To facilitate preloading such configuration settings, several wireless preloading schemes were devised. However, many of these schemes require cryptographic random numbers for securing the wireless preloading process itself, thus making them inapplicable to preloading seeds [3]–[5]. Moreover, a subtlety of preloading seeds is reboots. After a reboot, a CSPRNG must not start over with the same seed, as otherwise the same sequence of cryptographic random numbers is regenerated. Hence, some data has to be kept across reboots. Yet, the only non-volatile memory on most IoT devices is flash memory, which is slow, energy consuming, as well as prone to wear. After all, an IoT device may have to be able to reseed its CSPRNG for two reasons anyway. First, at some point, a seed is exhausted and has to be replaced [6]. Second, reseeding is also necessary to achieve forward and backward security [7]. Forward and backward security is the property of a CSPRNG that, if an attacker compromises the internal state of a CSPRNG, he is unable to predict past and future outputs of the CSPRNG, respectively [7].

On the other hand, it is also hard for an IoT device to generate seeds by itself since IoT devices usually lack common entropy sources, such as user interaction or hard disks. A promising replacement is to use power-up static random-access memory (SRAM) states, which are partly random due to manufacturing deviations [8]. There, however, seems to be no method for extracting close-to-uniformly distributed seeds from power-up SRAM states in an information-theoretically secure and practical manner. In fact, using a hash function, as suggested by van der Leest et al. [9], [10], is not information-theoretically secure, while the information-theoretically secure extractor that is used by Holcomb et al. is impractical due to its need for additional truly random bits [8]. Another unsolved issue with power-up SRAM states is that their min-entropy reduces with temperature. This enables freezing attacks, where an attacker exposes SRAM to low temperatures [10].

This paper makes three main contributions:

- We propose a method for extracting uniformly distributed seeds from power-up SRAM states in an information-theoretically secure, practical, as well as freezing attack-resistant manner.
- Although power-up SRAM states are deemed to be only suitable for self-seeding at boot time [8], we point out a trick that enables using power-up SRAM states for reseeding at runtime, too.
- Third, we provide a comparison of the energy consumption of seeding an IoT device either with radio noise or power-up SRAM states. Seeding with power-up SRAM states turned out to be more energy efficient, but we advise against using any of these two entropy sources alone since both have vulnerabilities.

II. Preliminaries

For generating cryptographic random numbers, an IoT device can follow three steps, namely entropy gathering, randomness extraction, and pseudo-random number generation.

A. Entropy Gathering

During entropy gathering, an IoT device draws a value from an entropy source. An *entropy source* X is a discrete random variable. We use $x \leftarrow X$ to denote that x is drawn at random according to the probability distribution of X. Note that X may be composed of multiple samples.

An attacker would try to predict the value of an entropy source by choosing its most likely value. Therefore, the *predictability* of an entropy source X is $\max_x \Pr[X = x]$. Correspondingly, the *min-entropy* $H_\infty(X)$ of X is

$$H_\infty(X) \overset{\text{def}}{=} -\log_2(\max_x \Pr[X = x]) \quad (1)$$

For comparison, the *Shannon entropy* $H_1(X)$ of X is

$$H_1(X) \overset{\text{def}}{=} -\sum_x \Pr[X = x] \log_2 \Pr[X = x] \quad (2)$$

While Shannon entropy is a measure of the average level of surprise when seeing a value $x \leftarrow X$, min-entropy is a measure of the minimum level of surprise when seeing a value $x \leftarrow X$.

B. Randomness Extraction

To serve as a seed, the value of an entropy source has to be distilled into a shorter close-to-uniformly distributed bit string. The functions for doing so are often referred to as extractors. Basically, an *extractor* Ext is a function $\text{Ext} : \{0,1\}^n \times \{0,1\}^d \rightarrow \{0,1\}^m$ that takes a value $\{0,1\}^n \ni x \leftarrow X$, as well as a public uniformly distributed *extractor seed* $s_{\text{Ext}} \in \{0,1\}^d$ as input, and outputs a close-to-uniformly distributed bit string $s \in \{0,1\}^m$, where $d < m$.

Extractor seeds are needed for randomization. If an extractor Ext was deterministic, there would be entropy sources with high min-entropy that Ext only maps to subsets of $\{0,1\}^m$ [11]. This raises a chicken-and-egg problem since, to be able to generate a seed, each IoT device has to have an individual extractor seed. Previous work came up with two solutions to obviate the need for individual extractor seeds. On the one hand, one can construct seedless extractors for entropy sources that follow specific distributions. For example, if an entropy source is composed of i.i.d. Bernoulli trials, one can use the seedless von Neumann extractor [12]. On the other hand, Barak et al. suggested using a static extractor seed so that a vendor can preload all its devices with the same extractor seed [13], [14]. For this, Barak et al. restrict themselves to an attacker that can not modify more than t boolean properties of the distribution of the employed entropy source, as well as require that these modifications do not cause the min-entropy of the employed entropy source to become lower than a threshold. Thus, information-theoretically secure randomness extraction without individual extractor seeds is feasible.

C. Pseudo-Random Number Generation

With a seed in place, an IoT device can efficiently generate a stream of cryptographic random numbers $r_1, r_2, \ldots$, e.g., by invoking a block cipher in output feedback mode (OFB) [1], [2], [15]. Concretely, let $\text{Enc} : \{0,1\}^k \times \{0,1\}^l \rightarrow \{0,1\}^l$ be a block cipher, where k denotes key length and l block size. Furthermore, let $s = s_{\text{Enc}} \| r_1$ be a seed, where $s_{\text{Enc}} \in \{0,1\}^k$ and $r_1 \in \{0,1\}^l$. OFB generates $r_{i+1} \in \{0,1\}^l$ as $\text{Enc}(s_{\text{Enc}}, r_i)$.

III. Related Work

Owing to the lack of common entropy sources on IoT devices, many related efforts aimed to find alternatives.

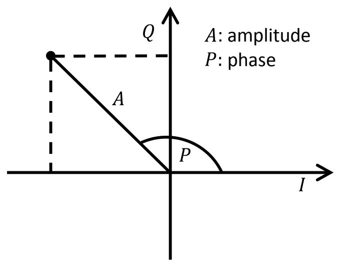


Fig. 1: Conversion of an I/Q reading to amplitude and phase

A. RAM

Holcomb et al. were first to consider power-up SRAM states as an entropy source [8]. The motivation behind using power-up SRAM states as an entropy source is that, after powering up, each SRAM cell eventually settles to either 0 or 1. The tendency of whether an SRAM cell settles to 0 or 1 mainly depends on the individual characteristics of each cell, but also on three other factors that an attacker might influence. First, when exposed to low temperatures, many SRAM cells become either 0-skewed or 1-skewed [8]. In effect, this reduces the min-entropy of the power-up SRAM states. Second, Negative Bias Temperature Instability (NBTI) is a burn-in effect that occurs when an SRAM cell stores the same bit for a long period of time. This issue can be sidestepped by using SRAM regions that are updated frequently [8]. Alternatively, we propose a different way to prevent NBTI from taking effect in Section V. Finally, Holcomb et al. found that an attacker may be able to predictably influence power-up SRAM states by tampering with the supply voltage [8]. In view of this third factor, as well as because further ways to influence power-up SRAM states could exist, we suggest mixing power-up SRAM states with other entropy sources.

Alternatively, Mowery et al. considered using dynamic RAM (DRAM) decays as an entropy source [16]. Indeed, they conclude that DRAM decays is a good entropy source on DRAM-based IoT devices. Yet, on battery-powered and energy-harvesting IoT devices, SRAM is often preferred over DRAM since SRAM is more energy efficient than DRAM.

B. Antennas

Furthermore, several manuals of IEEE 802.15.4 transceivers suggest using radio noise as an entropy source [17], [18]. Radio noise can be measured in one of two ways. First, a common feature of IEEE 802.15.4 transceivers is their support for energy scans, which return the average amplitude of the current channel over 8 symbol periods. Second, some transceivers also provide access to I/Q data [18]. I/Q data are cartesian representations of the amplitude and phase of the sinusoidal wave in the channel, as depicted in Fig. 1. However, the min-entropy of radio noise readings under interference and eavesdropping has never been investigated to the best of our knowledge, which leaves security concerns.

A similar approach is the gathering of entropy from incoming frames. Especially, received signal strength indicators (RSSIs) and link quality indicators (LQIs) of incoming IEEE 802.15.4 frames can serve as entropy sources [19], [20]. Also,

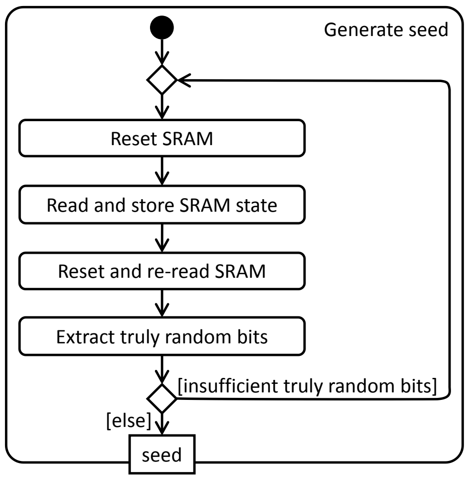

Generate seed
Reset SRAM
Read and store SRAM state
Reset and re-read SRAM
Extract truly random bits
[insufficient truly random bits]
[else]
seed

Fig. 2: Proposed method for extracting seeds from power-up SRAM states

erroneous IEEE 802.15.4 frames, i.e. IEEE 802.15.4 frames with false checksums, were considered as an entropy source [20], [21]. Yet, RSSIs, LQIs, as well as erroneous frames are susceptible to frame injection attacks [20]. Moreover, RSSIs are vulnerable to eavesdropping, too [20].

C. Sensors

Hennebert et al. considered sensor readings as entropy sources [22]. Unfortunately, they found sensor readings to be weak entropy sources. Only special sensors, such as accelerometers and microphones, serve well as entropy sources [23], [24], but are not available on all IoT devices.

D. Clocks

By contrast, quite ubiquitous entropy sources are those based on clocks. For instance, a straightforward approach is to read the least significant bit (LSB) of one clock at intervals that are measured by another clock [25]. Besides, Mowery et al. proposed using execution times as an entropy source [16]. However, it remains to be investigated how suitable these entropy sources for cryptographic purposes are [16], [25].

IV. Proposed Method

Current methods for extracting close-to-uniformly distributed seeds from power-up SRAM states (i) read a region of the SRAM at start up and (ii) pass the result to a hash or extractor function. By contrast, our method is based on the idea of sampling the power-up SRAM state at least two times in a row. This can be done by resetting the SRAM in between, e.g., by issuing a cold reboot or by using special sleep modes like we do in our implementation.

Specifically, our proposed method for extracting uniformly distributed seeds from power-up SRAM states involves four steps, as shown in Fig. 2. Step 1 is to reset the SRAM to account for the occasion that our method may be executed after a warm reboot. Step 2 is to read a region of the SRAM and to store the result. Step 3 is to reset the SRAM and to re-read the SRAM region again. Step 4 is to compare, for each

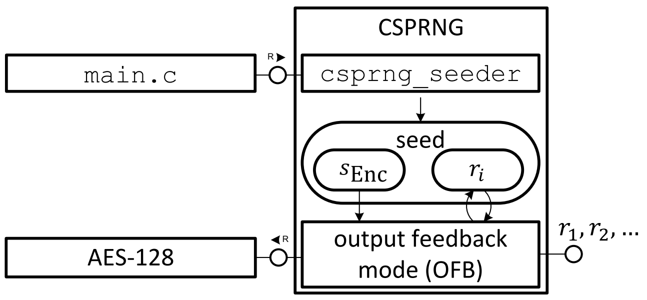


Fig. 3: Design of our CSPRNG

SRAM cell, whether its value has flipped from 0 to 1, from 1 to 0, or stayed unchanged. In the case of a flip from 0 to 1, our method appends a 1 to the resulting seed. In the case of a flip from 1 to 0, our method appends a 0 to the resulting seed. In the case of no flip, our method appends nothing to the resulting seed. If all comparisons are done and the resulting seed is still too short, our method repeats from Step 1.

Observe that our method exploits that the probability that an SRAM cell flips from 0 to 1 is the same as that the SRAM cell flips from 1 to 0. This is because each SRAM cell has a stable probability of settling to a value of 1. In other words, we apply the von Neumann extractor to consecutively read power-up values of an SRAM cell [12].

Compared to current methods for extracting close-to-uniformly distributed seeds from power-up SRAM states [8]–[10], ours has three advantages. First, our method extracts uniformly distributed seeds, rather than just close-to-uniformly distributed seeds. Second, if many SRAM cells become 0- or 1-skewed due to a freezing attack, our method will simply draw more samples. Finally, our method is based on the information-theoretically secure and seedless von Neumann extractor.

V. Implementation

We implemented our method as part of a complete CSPRNG for the Contiki operating system [26]. Fig. 3 gives an overview of our CSPRNG. The exchangeable `csprng_seeder` gets invoked at boot time and has the task to provide our CSPRNG with a seed $s_{\text{Enc}} \| r_1$, where $s_{\text{Enc}}, r_1 \in \{0,1\}^{128}$. Once the plugged-in `csprng_seeder` set up $s_{\text{Enc}} \| r_1$, our CSPRNG serves cryptographic random numbers by invoking the possibly hardware-accelerated Advanced Encryption Standard (AES)-128 block cipher in OFB, as described in Section II-C.

Currently, we provide three `csprng_seeders`, namely the `sram_seeder`, the `iq_seeder`, and the `mix_seeder`. All of them have CC2538 system on chips (SoCs) as their target platform [18]. A CC2538 SoC comprises a Cortex M3 microprocessor, up to 512KB of program memory, up to 32KB of SRAM, and an IEEE 802.15.4 transceiver. CC2538 SoCs are built into various IoT prototyping platforms, such as OpenMotes and Re-Motes [27].

The `sram_seeder` follows our proposed method for extracting uniformly distributed seeds from power-up SRAM states. For resetting the SRAM, the `sram_seeder` leverages that CC2538 SoCs do not retain data stored in the second half of their SRAM in power mode (PM) 2. Hence, in order to reset the SRAM, the `sram_seeder` shortly switches to PM 2 and back again. Notably, this trick enables using power-up SRAM states for reseeding at runtime, too. In fact, the `sram_seeder` can also be invoked at runtime, which will reseed our CSPRNG. Also, by using an SRAM region that is not retained in PM 2, we prevent NBTI from taking effect since no data is kept in this SRAM region for long.

The `iq_seeder` extracts seeds from I/Q LSBs. For randomness extraction, we assume that an attacker cannot modify more than 64 boolean properties of the distribution of I/Q LSBs, and that 200 I/Q LSBs (i.e. 200 I LSBs and 200 Q LSBs) always have a min-entropy of at least 256. These assumptions allow us to use Barak et al.'s Toeplitz matrix-based extractor $\text{Ext}_{\text{Toeplitz}} : \{0,1\}^n \times \{0,1\}^d \rightarrow \{0,1\}^m$ [13]. $\text{Ext}_{\text{Toeplitz}}$ multiplies an $\mathcal{F}^{m' \times n'}$ Toeplitz matrix with the value $x \in \mathcal{F}^{n' \times 1}$ of an entropy source, where $\mathcal{F}$ is a finite field. The first column and first row of the Toeplitz matrix are defined by a fixed seed $s_{\text{Ext}} \in \mathcal{F}^{m'+n'-1}$. Our `iq_seeder` sets $\mathcal{F} = GF(2^8)$, $m = 8m' = 128$, $n = 8n' = 400$, and $d = 8(m' + n' - 1) = 520$. If invoked, the `iq_seeder` samples 400 I/Q LSBs and calls $\text{Ext}_{\text{Toeplitz}}$ twice: the first 200 I/Q LSBs yield s_{Enc} and the other 200 I/Q LSBs yield r_1. According to results of Skorski, our `iq_seeder` achieves with probability $1 - 10^{-9}$ over the choice of s_{Ext} that the statistical distance between s_{Enc} and the uniform distribution U_{128} over $\{0,1\}^{128}$ is at most 10^{-9} [14]. Likewise, the statistical distance between r_1 and U_{128} is at most 10^{-9} with probability $1 - 10^{-9}$ over the choice of s_{Ext} [14]. Of course, the `iq_seeder` can also be invoked at runtime for reseeding.

Lastly, the `mix_seeder` combines the `sram_seeder` and the `iq_seeder` by XORing their generated seeds. Additionally, the `mix_seeder` XORs this seed with the lastly generated seed so as to amplify the security of reseeding at runtime. Altogether, as long as one of the generated seeds remains secret, the `mix_seeder` keeps up security.

VI. Evaluation

In the following, we assess the min-entropy of power-up SRAM states and I/Q LSBs. Subsequently, we compare the energy consumption of seeding with I/Q LSBs, power-up SRAM states, or both.

A. Entropy Assessment

To estimate the min-entropy of power-up SRAM states, the following experiment was conducted. An OpenMote dumped 512 bytes of its SRAM. Then, the SRAM was reset by shortly switching to PM 2 and back again. Next, the same 512-byte SRAM region was dumped again. These steps were repeated until 1000 samples were obtained. Eventually, the empirical min-entropy of each SRAM cell was calculated based on its empirical probability of settling to 1.

Fig. 4 shows, per SRAM cell, the empirical probability of settling to a value of 1. As expected, a pattern emerged since many SRAM cells are 1-skewed or 0-skewed. Also, there are plenty of neutral-skewed SRAM cells. The mean empirical min-entropy of an SRAM cell value turned out to be 0.07089.

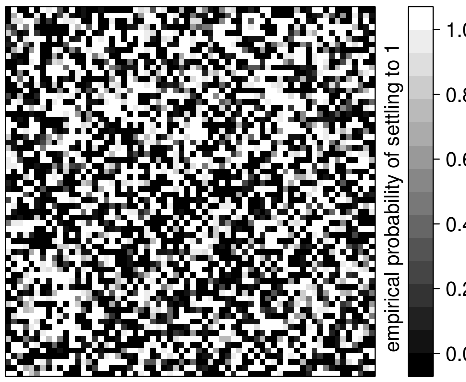
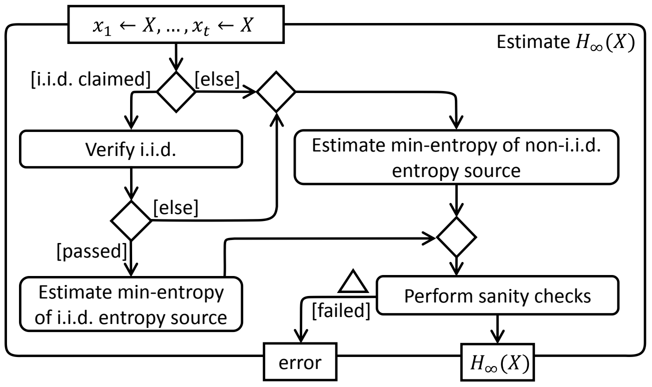


Fig. 4: Pattern of power-up SRAM states of a CC2538 SoC


$x_1 \leftarrow X, \ldots, x_t \leftarrow X$
Estimate $H_\infty(X)$
[i.i.d. claimed]
[else]
Verify i.i.d.
Estimate min-entropy of non-i.i.d. entropy source
[else]
[passed]
Estimate min-entropy of i.i.d. entropy source
[failed]
Perform sanity checks
error
$H_\infty(X)$
Fig. 5: Workflow of NIST 800-90B

The min-entropy of I/Q LSBs was estimated using an open-source implementation of NIST 800-90B [28]. Basically, if X is the entropy source under test, NIST 800-90B takes values $x_1 \leftarrow X, \ldots, x_t \leftarrow X$ as input and outputs either an error or an estimation of $H_\infty(X)$. Internally, NIST 800-90B treats i.i.d. and non-i.i.d. entropy sources differently, as depicted in Figure 5. The min-entropy of i.i.d. entropy sources is estimated via a bins test. Yet, prior to running the bins test, NIST 800-90B verifies that $x_1, \ldots, x_t$ are i.i.d. by calculating various statistics, such as an over/under runs score, an excursion score, a directional runs score, a covariance score, and a collision score. The min-entropy of non-i.i.d. entropy sources is estimated by taking the minimum over five min-entropy estimators, namely a collision test, a partial collection test, a Markov test, a compression test, and a frequency test. Finally, if the estimated min-entropy $H_\infty(X)$ passes two additional sanity checks, NIST 800-90B outputs $H_\infty(X)$.

We sampled 1,000,000 pairs of I/Q LSBs and passed

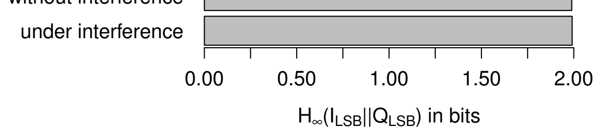


Fig. 6: Min-entropy of pairs of I/Q LSBs

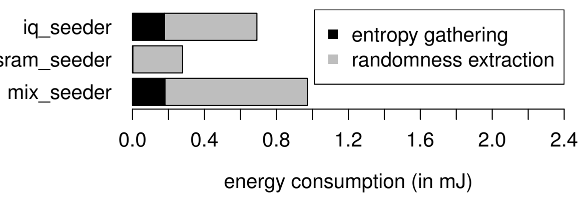


Fig. 7: Mean energy consumption of self-seeding

them to NIST 800-90B. As a result, NIST 800-90B reported that I/Q LSBs are i.i.d., as well as that they are almost perfectly random, as shown in Figure 6. We also repeated this experiment with the difference being that a nearby OpenMote continuously transmitted while sampling I/Q LSBs. The results did not change significantly, as shown in Figure 6. NIST 800-90B does, however, not report that I/Q LSBs are truly random, which is why our `iq_seeder` employs an extractor function. Another cautionary note about these results is that they do not exclude that the min-entropy of I/Q LSBs reduces under other sorts of interference or eavesdropping.

B. Energy Consumption

To measure the energy consumption of our `csprng_seeders`, an OpenMote, a μCurrent Gold, and a Rigol DS1000E oscilloscope were connected in series. This enables capturing the OpenMote's current draw over time, as is further detailed in [27]. In three subsequent runs, the current draw while seeding with the `iq_seeder`, the `sram_seeder`, and the `mix_seeder` was measured. Per run, 100 samples were obtained. The `sram_seeder` was configured to use a 1024-byte SRAM region so that the `sram_seeder` never had to obtain more than two power-up SRAM states. For converting current draw into energy consumption, a constant supply voltage of 3V was assumed like in [29]. Throughout, the energy consumed in PM 2 was subtracted.

Fig. 7 shows the mean energy consumption of each of our `csprng_seeders`. The `iq_seeder` is relatively energy consuming since it needs to enable the energy-consuming receive mode of the CC2538 SoC for sampling I/Q LSBs. Moreover, the `iq_seeder` executes Barak et al.'s Toeplitz matrix-based extractor, which constitutes the bulk of its energy consumption. By contrast, the `sram_seeder` consumes much less energy since the von Neumann extractor finishes quicker and because the `sram_seeder` gathers entropy by reading the SRAM, which costs very little in terms of energy.

VII. Conclusions and Future Work

Current methods for extracting seeds from power-up SRAM states are either impractical or not information-theoretically secure, and, in any case, vulnerable to freezing attacks. We have proposed an information-theoretically secure, practical, as well as freezing attack-resistant method for extracting seeds from power-up SRAM states. However, since attackers may predictably influence power-up SRAM states by tampering with the supply voltage, we recommend mixing this entropy source with others. In our implementation, we mix power-up SRAM states with I/Q LSBs. For future work, we suggest revisiting some of the mentioned entropy sources on IoT devices with a focus on the often-neglected aspect of information-theoretically secure randomness extraction.

References

[1] J. Deng, C. Hartung, R. Han, and S. Mishra, "A practical study of transitory master key establishment for wireless sensor networks," in *Proceedings of the First IEEE/CreateNet Conference on Security and Privacy for Emerging Areas in Communication Networks (SecureComm 2005)*. IEEE, 2005, pp. 289 – 302.

[2] H. Seo, J. Choi, H. Kim, T. Park, and H. Kim, "Pseudo random number generator and hash function for embedded microprocessors," in *Proceedings of the 2014 IEEE World Forum on Internet of Things (WF-IoT)*. IEEE, 2014, pp. 37–40.

[3] D. Chen, M. Nixon, T. Lin, S. Han, X. Zhu, A. Mok, R. Xu, J. Deng, and A. Liu, "Over the air provisioning of industrial wireless devices using elliptic curve cryptography," in *Proceedings of the 2011 IEEE International Conference on Computer Science and Automation Engineering (CSAE)*, vol. 2. IEEE, 2011, pp. 594–600.

[4] N. Saxena and M. Uddin, "Blink 'em all: scalable, user-friendly and secure initialization of wireless sensor nodes," in *Proceedings of the 8th International Conference on Cryptology and Network Security (CANS 2009)*. Springer, 2009, pp. 154–173.

[5] K.-F. Krentz and G. Wunder, "6doku: towards secure over-the-air preloading of 6LoWPAN nodes using PHY key generation," in *Proceedings of the European Conference on Smart Objects, Systems and Technologies (Smart SysTech 2015)*. VDE, 2015.

[6] E. Barker, J. Kelsey *et al.*, "Recommendation for random number generation using deterministic random bit generators," 2012, NIST special publication 800-90A.

[7] Y. Dodis, D. Pointcheval, S. Ruhault, D. Vergniaud, and D. Wichs, "Security analysis of pseudo-random number generators with input: /dev/random is not robust," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security (CCS '13)*. ACM, 2013, pp. 647–658.

[8] D. Holcomb, W. 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] V. van der Leest, E. van der Sluis, G.-J. Schrijen, P. Tuyls, and H. Handschuh, "Efficient implementation of true random number generator based on SRAM PUFs," in *Cryptography and Security: From Theory to Applications*, D. Naccache, Ed. Springer, 2012, pp. 300–318.

[10] A. Van Herrewege, V. van der Leest, A. Schaller, S. Katzenbeisser, and I. Verbauwhede, "Secure PRNG seeding on commercial off-the-shelf microcontrollers," in *Proceedings of the 3rd International Workshop on Trustworthy Embedded Devices (TrustED '13)*. ACM, 2013, pp. 55–64.

[11] Y. Dodis, R. Gennaro, J. Håstad, H. Krawczyk, and T. Rabin, "Randomness extraction and key derivation using the CBC, Cascade and HMAC modes," in *Proceedings of the 24th Annual International Cryptology Conference (CRYPTO 2004)*. Springer, 2004, pp. 494–510.

[12] J. von Neumann, "Various techniques used in connection with random digits," 1951.

[13] B. Barak, R. Shaltiel, and E. Tromer, "True random number generators secure in a changing environment," in *Proceedings of Cryptographic Hardware and Embedded Systems (CHES 2003)*. Springer, 2003, pp. 166–180.

[14] M. Skorski, "True random number generators secure in a changing environment: Improved security bounds," in *Proceedings of the 41st International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2015)*. Springer, 2015, pp. 590–602.

[15] D. Eastlake, J. Schiller, and S. Crocker, "Randomness Requirements for Security," RFC 4086, Internet Engineering Task Force, 2005.

[16] K. Mowery, M. Wei, D. Kohlbrenner, H. Shacham, and S. Swanson, "Welcome to the entropics: boot-time entropy in embedded devices," in *Proceedings of the 2013 IEEE Symposium on Security and Privacy (SP)*, 2013, pp. 589–603.

[17] *2.4 GHz IEEE 802.15.4 / ZigBee-Ready RF Transceiver (Rev. C)*, Texas Instruments, http://www.ti.com/lit/ds/symlink/cc2420.pdf.

[18] *CC2538 SoC for 2.4-GHz IEEE 802.15.4 & ZigBee/ZigBee IP Applications User's Guide (Rev. C)*, Texas Instruments, http://www.ti.com/lit/ug/swru319c/swru319c.pdf.

[19] R. Latif and M. Hussain, "Hardware-based random number generation in wireless sensor networks(wsns)," in *Proceedings of the Third International Conference on Advances in Information Security and Assurance (ISA 2009)*. Springer, 2009, pp. 732–740.

[20] C. Hennebert, H. Hossayni, and C. Lauradoux, "The entropy of wireless statistics," in *Proceedings of the 2014 European Conference on Networks and Communications (EuCNC)*. IEEE, 2014, pp. 1–5.

[21] A. Francillon and C. Castelluccia, "TinyRNG: a cryptographic random number generator for wireless sensors network nodes," in *Proceedings of the 5th International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt 2007)*. IEEE, 2007, pp. 1–7.

[22] C. Hennebert, H. Hossayni, and C. Lauradoux, "Entropy harvesting from physical sensors," in *Proceedings of the Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '13)*. ACM, 2013, pp. 149–154.

[23] J. Voris, N. Saxena, and T. Halevi, "Accelerometers and randomness: Perfect together," in *Proceedings of the Fourth ACM Conference on Wireless Network Security (WiSec '11)*. ACM, 2011, pp. 115–126.

[24] N. Bardis, A. Markovskyi, N. Doukas, and N. Karadimas, "True random number generation based on environmental noise measurements for military applications," in *Proceedings of the 8th WSEAS International Conference on Signal Processing, Robotics and Automation*, 2009, pp. 68–73.

[25] *Random number generation using the MSP430*, Texas Instruments, http://www.ti.com/lit/an/slaa338/slaa338.pdf.

[26] A. Dunkels, B. Grönvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks (LCN 2004)*. IEEE, 2004, pp. 455–462.

[27] X. Vilajosana, P. Tuset, T. Watteyne, and K. Pister, "OpenMote: open-source prototyping platform for the industrial IoT." in *Ad Hoc Networks*, vol. 155. Springer, 2015, pp. 211–222.

[28] E. Barker and J. Kelsey, "Recommendation for the Entropy Sources Used for Random Bit Generation," 2012, NIST DRAFT Special Publication 800-90B.

[29] K.-F. Krentz, Ch. Meinel, and H. Graupner, "Countering three denial-of-sleep attacks on ContikiMAC," in *Proceedings of the International Conference on Embedded Wireless Systems and Networks (EWSN 2017)*. Junction, 2017.