An Entropy Source based on the Bluetooth Received Signal Strength Indicator

Alexandre Giron[1,2], **Ricardo Custódio**[1]

[1]Laboratório de Segurança em Computação –
Universidade Federal de Santa Catarina (UFSC)
Florianópolis – SC – Brazil

[2]Universidade Tecnológica Federal do Paraná (UTFPR)
Toledo – PR – Brazil

alexandregiron@utfpr.edu.br, ricardo.custodio@ufsc.br

Abstract. *The Bluetooth Low Energy (BLE) is one of the popular communication technologies employed in the Internet of Things (IoT) context. IoT devices need random numbers to feed their security mechanisms, where the generation of random numbers presupposes the existence of entropy sources. However, there are few entropy sources available, due to the limited hardware resources of those devices. Given this scenario, this paper presents a scalable approach for gathering entropy, called Bluerandom. The approach is based on the Received Signal Strength Indicator (RSSI) within Bluetooth communications. The results show that Bluerandom can be used as an alternative source of entropy, improving the robustness of the cryptographic mechanisms for the IoT context.*

1. Introduction

The high popularity of the IoT and its use of sending user sensitive data imply the need to establish secure wireless communications. Examples of communication technologies in IoT include the traditional WiFi technology, 5G networks, and the Bluetooth Low Energy (BLE) technology [Böcker et al. 2017]. To address most of the security concerns about privacy, tampering, and authentication in IoT, cryptographic mechanisms are often required [Dinca and Hancke 2017]. These mechanisms depends on Random Number Generators (RNG) to feed their cryptographic primitives like nonces, Initialization Vectors (IVs), random keys, among others [Herrero-Collantes and Garcia-Escartin 2017].

In addition to its importance in a traditional cryptosystem, randomness is required, for example, in remote attestation protocols for IoT networks [Tan et al. 2019], and BLE devices need randomness for the random address technique [Cha et al. 2017, Collotta et al. 2018]. This technique for BLE is used in the device pairing process to prevent attackers from tracking devices. To perform such technique, each Bluetooth device has its Pseudorandom Number Generator (PRNG) that can be accessed by the LE Rand command of the protocol [SIG 2019]. Besides, the challenge-response authentication and encryption (if available) also need random numbers [SIG 2019]. Therefore, the PRNG in the device plays a key role in BLE security.

An entropy source is a resource needed by a True Random Number Generator (TRNG) to output random data. Generally, the TRNG collects randomness from one or more entropy sources. The TRNG output feeds a PRNG, which is deterministic, but the

latter's data throughput is predominantly higher than a TRNG [von zur Gathen 2015]. In this scheme, the quality level of the PRNG output depends on the entropy source used as seed. If an adversary can deduce the seed, the output can also be determined [Wallace et al. 2016].

A popular example of a PRNG is the Linux /dev/random device driver [Gutterman et al. 2006]. It is well known and prevalent in the cryptographic community, serving entropic data to a variety of security protocols. In a recent publication [Müller 2018], the concerns about the entropy sources available were taken into account in the development of a new approach for the /dev/random. Some of the challenges arise from the IoT scenario and small embedded systems, in which their limited resources imposes the need to achieve high efficiency in terms of performance and energy consumption.

Some of the standard entropy sources in a computer include data from the sound card, disk access times, the timing of interrupts, CPU Jitter, or user interaction data. However, in IoT, most of these sources are not available; the main source is in the microcontroller. Especially for the BLE devices, there is an RNG hardware component responsible for gathering entropy and generating random numbers. Microcontrollers like the Texas Instruments CC1312R [TI 2018], Nordic nRF52 Series [Nordic 2019], Cypress PSoC 4 [Cypress 2019], gather data from thermal noise, process and deliver it to seed the PRNG. For the PRNG implementation, a popular choice is often based on AES or SHA hardware implementations [Herrero-Collantes and Garcia-Escartin 2017] [Stallings 2017].

All of the hardware designs pointed above implement security mechanisms similarly. However, relying on only one or few entropy sources could be dangerous to the security of the system, for example, in case of hardware failures or security flaws. A famous incident is the EC_Dual PRG [Checkoway et al. 2014]. Besides, IoT devices do not have many alternatives for entropy sources available due to their reduced hardware capabilities. On the other hand, adding more hardware components in an IoT device has an impact in cost and energy consumption. So, it is essential to investigate and evaluate other possibilities to serve entropy for those devices.

In this paper it is proposed an alternative entropy source for the IoT context, called Bluerandom. The approach is based on the Bluetooth Received Signal Strength Indicator (RSSI). The purpose of the approach is to deliver entropic data as an additional source to the device, improving the robustness of the security mechanism against possible failures.

The paper is organized as follows. The Related Work in section 2 presents the research on the available entropy sources for the IoT context. In section 3 the Bluerandom approach is presented. Section 4 presents the experiments and section 5 discuss the results of the evaluation of Bluerandom. Lastly, section 6 presents the final considerations of this study.

2. Related Work

There are some alternatives for entropy sources in the literature, related to the IoT context. For example, researchers proposed a PRNG design for embedded microprocessors [Seo et al. 2014] and a TRNG with ring oscillators as a source of entropy, focusing on FPGA hardware [Kohlbrenner and Gaj 2004].

Wallace et al. [Wallace et al. 2016] proposed an RNG based on the sensors present in smartphones, called *SensoRNG*. The evaluation included various sensors found in smartphones, like gyroscope, accelerometer, magnetometer, GPS, microphone and camera. Gyroscopes used as an alternative entropy source were also evaluated in the work of Willers et al. [Willers et al. 2019].

Randomness can be obtained from the channel phase and from the Received Signal Strength Indicator (RSSI) in Wireless Networks [Shokri-Ghadikolaei et al. 2016, Wang et al. 2011]. Wang et al. [Wang et al. 2011] proposed a protocol, under a set of simulations, that use channel phase to generate cryptographic keys in wireless networks at the physical link layer. The main drawback of their approach is that it is required a common timing reference between the nodes, in order to generate the shared keys. On the other hand, the authors conclude that the extraction of randomness in Wireless Networks can be a good opportunity for IoT devices, when possible.

Some of the proposals presented above can increase hardware complexity and production cost, if components must be added to provide random numbers. Others were evaluated in a simulated environment, not in a real-world scenario. The proposal of this paper, detailed in the next section, aims to tackle these issues by exploring the environment for randomness.

3. Bluerandom Generator

The Bluerandom is a generator that reads RSSI values from BLE devices and then output bits based on a randomness extractor function. The center of Fig. 1 represents a device with Bluerandom. It sends commands to the Bluetooth controller to scan for nearby devices. Bluerandom keeps reading the signal strength information of each device. For each RSSI reading, an extraction method is employed to generate random numbers.

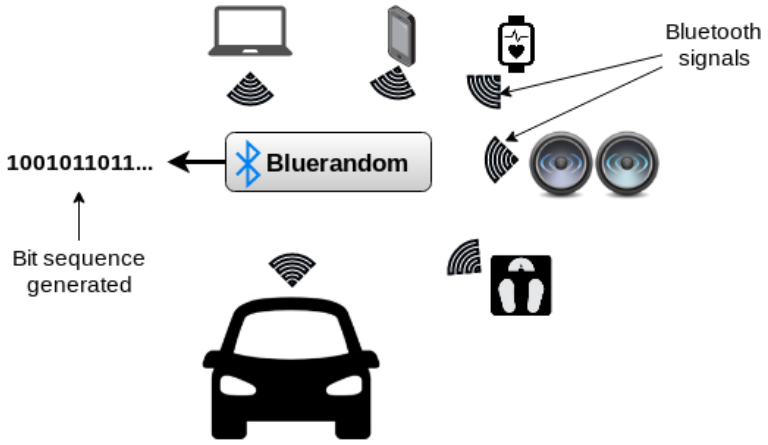

Figure 1. Bluerandom generator.

In the IoT context, the efficient use of the hardware is mandatory to minimize costs and energy consumption of the device. Bluerandom take advantage of the available hardware to collect entropy. Bluerandom obtains its data from nearby (other) Bluetooth devices. The only requisite is that there must be at least one Bluetooth device nearby advertising or in visible state.

The RSSI is defined as a measure of power of a device signal perceived on the receiving device [Wang et al. 2011]. In Bluetooth, one device starts advertising its beacons, and then the receiving device can calculate the RSSI. An application can read the RSSI value through the Host Controller Interface (HCI) from Bluez [BlueZ 2018]. The signal strength depends on distance and on the broadcasting power value measured [Huh et al. 2016].

A premiss of Bluerandom relies on the assumed unpredictability of the RSSI readings. Analyzing the RSSI readings from a single nearby Bluetooth device, there are small variations between the readings, presumably unpredictable, even without moving the device. This happens probably due to the precision of the sensor readings or due to the channel phase mentioned earlier. Besides, when considering an open space scenario with several devices, the variation of the readings would be on a larger scale. It is hard to predict if (and how) this variation occurs in the next reading. There could also be interference present and sensor movement (wearables, for example), increasing the unpredictability of the RSSI values.

3.1. Randomness Extraction functions

Having the RSSI values, Bluerandom needs a randomness extractor. Randomness extractors are post-processing functions in a TRNG. They transform the bits received from the entropy source into a uniform, usually smaller, random sequence [Herrero-Collantes and Garcia-Escartin 2017]. Depending on the extractor used, this means that some bits might be discarded if bias is detected.

A well-known extractor was proposed by von Neumann. For every pair of generated bits, discard the results 00 and 11 (to reduce bias) and assign a 0 to 01 and a 1 to 10 [Herrero-Collantes and Garcia-Escartin 2017]. In the work of Szczepanski et al. [Szczepanski et al. 2004], a method called "The last digit fluctuation" is used, which considers only the least significant bit gathered from the entropy source. Hashing functions and chaotic functions can also be used [Zhu et al. 2013]. We investigated three extractors in our proposal:

- Von Neumann extractor: in Bluerandom, this method is implemented together with the "Last digit fluctuation" [Szczepanski et al. 2004]. The von Neumann method is applied in the last bit of two consecutive RSSI readings.
- Odd or Even difference: this method also takes a pair of RSSI readings, and if the difference between them is odd, the method outputs a 1 and a 0 otherwise. When the difference is zero, there may be only one device nearby, which is providing readings without variation. In this case the method discards the readings. Besides, taking two RSSI readings at a time could decrease the throughput of this method.
- Odd or Even reading: this method outputs a 1 if the reading value is odd and a 0 if it is even, without discarding any value.

Although the extraction method has an impact on the throughput of Bluerandom, the frequency of the Bluetooth advertising beacons (or packets) is more important to that matter. It is from those beacons that the RSSI is calculated in the device which executes Bluerandom.

3.2. Test Environment

Mathematically proving that a stream of bits produced is truly random is effectively impossible [von zur Gathen 2015]. An alternative is to evaluate it through rigorous statistical testing to verify if a number sequence exhibits properties similar to what would be expected from a random sequence. For example, FIPS 140-2 and NIST SP 800-22 [Bassham et al. 2010] specify a set of statistical tests to verify uniformity and unpredictability of the number sequence from the RNG under evaluation.

All of the experiments were conducted by testing an implementation of Bluerandom in the Raspberry Pi 3 platform. The source code of the Bluerandom implementation is available at `https://github.com/AAGiron/Bluerandom`. The methodology chosen for the evaluation of Bluerandom was designed in four experiments:

- **Experiment 1** - Bluerandom test: the FIPS 140-2 statistical tests were applied to evaluate Bluerandom alone as an RNG. The throughput and byte entropy [von zur Gathen 2015] were also analyzed.
- **Experiment 2** - Bluerandom compared to a Bluetooth PRNG: as previously mentioned, Bluetooth devices must have a PRNG [SIG 2019]. This experiment aimed to compare the internal PRNG of a BLE device against Bluerandom seeding a PRNG. They were evaluated with NIST statistical tests.
- **Experiment 3** - Entropy Sources comparison: using the Dieharder test suite [Brown 2019], this experiment seeks the influence of Bluerandom as an entropy source compared to other entropy sources in a PRNG.
- **Experiment 4** - Providing Entropy: the *RNDADDENTROPY ioctl* system call allows the user to add some additional entropy to the Linux entropy pool. In this experiment, the entropy pool was measured when inserting Bluerandom numbers through the system call.

Regarding the test environment, the Experiment 1 was divided into two scenarios: one test was conducted in a mode called "Open scenario", and the other was conducted inside of a five-layer Faraday box. The open scenario test was conducted to verify the execution of Bluerandom in a real-world environment. The number of BLE devices present was varied to see if this variation has an impact on the Bluerandom output. The test inside a Faraday Box had the purpose of protecting the integrity of the experiment with only one BLE device, minimizing external influence or interference, and also to see if the RSSI variations – the premiss of Bluerandom – behaves differently in comparison with the open scenario environment.

It is worth to mention that the main purpose of proposing this approach is not to use Bluerandom as the only entropy source available in a system. Ideally, the output of Bluerandom could be used as input to a PRNG, which is generally more efficient [Stallings 2017]. For example, Chacha20 is a stream cipher that can be used as a PRNG for cryptographic applications [Procter 2014]. There is a proposal to use Chacha20 as part of the Linux RNG due to its throughput (in the order of a hundred MB/s, depending on the hardware) and low memory utilization [Müller 2018].

4. Experiment Results

In this section, the results are presented for each of the four experiments.

Table 1. Bluerandom evaluated with FIPS 140-2 tests

Number of Devices	von Neumann		Odd or Even Diff.		Just Odd or Even	
	FAIL	PASS	FAIL	PASS	FAIL	PASS
1 (Faraday)	1	0	2	0	4	0
1	4	0	7	0	9	8
2	7	0	12	0	13	16
4	9	0	17	0	10	28
8	19	0	31	6	16	63

4.1. Experiment 1

The first experiment aimed to evaluate Bluerandom as an RNG. Table 1 presents a summary of results from FIPS 140-2 statistical tests. The data sets for each execution of Bluerandom varies according to the extractor used and to the amount of BLE devices nearby. The "FAIL" and the "PASS" labels summarize the result of the statistical tests applied in chunks of 20 kbit of data. If f is the number of sets that fail and p is the number of sets that pass, this means that the generator produced $(p + f) * 20$ kbits of data to the test. For example, in the first row, the von Neumann extractor failed one time (20 kbits tested), and the "Odd or Even Difference" failed two times (40 kbits tested). This difference occurred because the throughput varies between the extractors.

The first row of Table 1 contains the results from the Faraday box test environment. The other rows are related to the Open scenario test environment. With only one device in the Faraday box, all extractors failed in FIPS tests. The table also shows that the "Just Odd or Even" extractor was able to produce more data from the devices, and the von Neumann extractor produced less data than the other extractors tested. For example, with 8 devices, the extractors were able to generate, respectively, 47.5 KiB, 92.5 KiB, and 197.5 KiB of data. This difference was expected because the "Just Odd or Even" extractor does not process two RSSI readings at a time and does not discard any reading, as the other extractors do.

It is worth mentioning that the von Neumann extractor, as implemented in this experiment, did not pass the tests because of the assumptions made on the RSSI readings. Instead of applying von Neumann extractor in the last bit of each of the two RSSI readings, it would be better to apply the von Neumann as a bias removal on the output of the "Just Odd or Even" or the "Odd or Even Difference" extractors. The drawback of this bias removal approach is that it would slow down the throughput of Bluerandom, which is already low. On average, Bluerandom generates 1 byte per second, depending mostly on the number of devices nearby and on the extractor used.

Another metric analyzed in Experiment 1 was the byte entropy, calculated with ENT software [Walker 2008]. The results are presented in Table 2. Interestingly, the byte entropy scales when the number of BLE devices is increased. This result supports the viability of using Bluerandom as an additional entropy source for the system.

4.2. Experiment 2

The second experiment was designed to evaluate the Bluerandom as an entropy source to a PRNG and comparing it to the internal PRNG from the Bluetooth controller of the Raspberry Pi 3.

Table 2. Entropy (bits per byte) of Bluerandom outputs

Number of Devices	von Neumann	Odd or Even Diff.	Just Odd or Even
1 (Faraday)	7.942342	7.808232	7.912642
1	7.418175	7.821669	7.987153
2	7.570067	7.905545	7.992010
4	7.669687	7.939813	7.995218
8	7.875071	7.983862	7.997471

The Bluetooth Specification states that Bluetooth devices must have a PRNG compliant with FIPS PUB 140-2 and NIST SP 800-22 tests [SIG 2019]. For BLE devices, the PRNG can be activated with the LE Rand command, returning a (pseudo) random number with 64-bit size [SIG 2019]. Also, the device shall use a seed from the entropy source with at least the minimum entropy required by the PRNG. It is left to the manufacturer to choose the entropy source and the PRNG implementation. Although it is not clear (or closed) in some documentation, reports from Bluez [BlueZ 2018] developers indicate that the PRNG used is often based on SHA or AES algorithms.

Therefore, a LE Rand application was created for generating numbers from the Bluetooth internal PRNG. Then the NIST tests were applied to evaluate 1 Gbit of data produced by this PRNG. To compare Bluerandom to the LE Rand generator, the output of Bluerandom seeded two implementations of PRNG: one is an AES-based PRNG, and the second is a SHA-based PRNG. The AES counter mode (CTR) implementation used is the one from the Mcrypt library [MCrypt], and the SHA-256 is from OpenSSL [OpenSSL].

The seeds obtained from Bluerandom's output were used for the AES PRNG key and Initialization Vector (IV), as recommended by NIST SP 800-22 [Bassham et al. 2010]. The reseeding process occurs when the PRNG reaches 1 MB of output. Since the LE Rand command outputs 64 bits at a time, each PRNG implementation was adapted to also output 64-bit numbers. The rightmost 64 bits were considered both for the AES-128 and SHA-256 in this experiment.

The execution of each of the two PRNG implementations produced 1 Gbit of data. Then, they were evaluated with the NIST tests. The results are summarized in Table 3. The best configuration for Bluerandom, obtained from Experiment 1, was selected for this experiment: the "Just Odd or Even" extractor and the test environment with 8 BLE devices.

Table 3. Bluetooth PRNG compared to Bluerandom plus PRNG

PRNG	Dataset	Entropy	NIST summary
LE Rand	1 Gbit	7.999998	All tests, but one, passed. 1 fail with proportion 979/1000
AES	1 Gbit	7.999998	All tests, but one, passed. 1 fail with proportion 979/1000
SHA256	1 Gbit	7.999998	All tests passed

Each PRNG evaluation with NIST tests results in a minimal proportion rate equal

to 980/1000. This proportion is defined as the number of binary sequences that passed over the total number of sequences [Bassham et al. 2010]. If the proportion for each test is higher than the minimal proportion rate, then it is considered that the PRNG passed on that test. The single test fails of LE Rand and AES PRNG had proportion 979/1000, which is very close to the pass threshold.

4.3. Experiment 3

Although the results from Experiment 2 are positive for Bluerandom as an entropy source, it was not possible to change the entropy source of the Bluetooth controller and replace it by Bluerandom. This experiment aims to compare the influence of changing the entropy source of the same PRNG.

Chacha20 PRNG has already been mentioned in this paper, and an interesting feature of its implementation is that it allows changing the seed source in three ways: from CPU Jitter RNG, from *getrandom()* system call or from /dev/random [Langley et al. 2016].

In this experiment, the Chacha20 was evaluated three times by the Dieharder test suite [Brown 2019]. This test suite has more statistical tests than the NIST suite. The first configuration of Chacha20 was defined only with /dev/random as the entropy source; the second configuration had only the CPU Jitter; and the third configuration had only Bluerandom as the entropy source. The Bluerandom configuration used was the same as the previous experiment.

Table 4 presents the results of all Dieharder tests. This suite classifies each test with "PASS", "WEAK" and "FAIL". It was found that the results are very similar and slightly better when Bluerandom was used. Again, this is not enough to conclude that Bluerandom is a better choice. On the contrary, this is another evidence that supports the use of Bluerandom as an additional entropy source, due to the similar behavior observed in the other entropy sources.

Table 4. Dieharder test results of Chacha20 using different entropy sources.

Entropy Source used	PASS	WEAK	FAIL
/dev/random	110	4	0
CPU Jitter	110	4	0
Bluerandom	111	3	0

4.4. Experiment 4

Fig. 2 presents the results obtained from the last experiment of this paper. Its main purpose was to see the influence of Bluerandom in the system entropy pool. The entropy pool is where the Linux RNG gathers its seeds (environmental noise from device drivers and other entropy sources) [Müller 2018].

The configuration used for measuring the system entropy count in this experiment was with few hardware support. No peripherals were attached to the Raspberry Pi board (such as keyboard and mouse) and also no network activity. The measurements occurred at intervals of one second, using the information located at /proc/sys/kernel/random/entropy_avail.

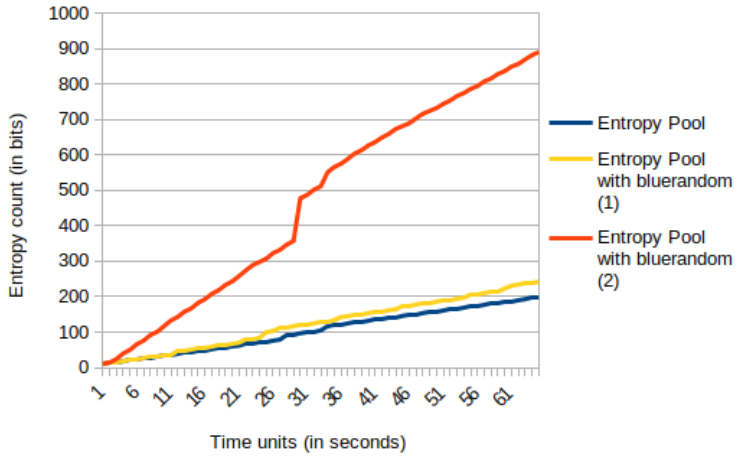

Figure 2. Effects of Bluerandom in the System Entropy Pool.

The first test measured in Fig. 2 (named "Entropy Pool"), without Bluerandom, shows slow linear growth. This was expected because the only entropy source available at the system was the CPU interrupts. The second test configuration measured the Entropy Pool with Bluerandom executing as an additional source of entropy. This test was performed with the Raspberry Pi 3 and with one BLE device advertising inside the Faraday box. A slight improvement was detected, as shown in Fig. 2, named as "Entropy Pool with bluerandom (1)". The third test was conducted in an open scenario, considering 4 BLE devices in the beginning. After 30 seconds, 6 BLE devices were in the range of Bluerandom. It was found that the growth in the entropy count is directly related to the quantity and advertising features of Bluetooth devices nearby.

5. Discussion

In this section, the results of the experiments are discussed and compared to the literature.

5.1. Scalability

The first suspect of scalability of Bluerandom is found in Table 1 when the amount of devices is increased, in Experiment 1. By increasing the number of devices, more data can be produced by Bluerandom, and the pass rate with FIPS 140-2 increases, considering the "Just Odd or Even" extractor. In Table 2, the byte entropy measured in Experiment 1 increased along with the number of nearby devices. This is an important feature of Bluerandom, because the quality of the output can be improved by when more BLE devices are nearby.

This scalability property could be related to the unpredictability level of which RSSI value will be read next. Besides the number of devices, other factors that could increase the unpredictability of the RSSI readings are: differences in the advertising interval of the readings, movement, interrupts and the use of blocking or non-blocking socket between the General Purpose Processor (GPP) and the Bluetooth microcontroller of some IoT platforms.

5.2. Security

For a secure RNG, a premiss is that it must pass all of the statistical tests. Therefore, the results presented in Table 1 indicate that Bluerandom is not prepared to be used as an

RNG by itself.

However, the results from the experiments supports the evidence that Bluerandom can be used as an additional entropy source for the system. In the experiments 2 and 3, Bluerandom used as seed to a PRNG achieved satisfactory results both when compared to the Bluetooth RNG (Table 3) and when compared to other entropy sources (Table 4).

Small differences were observed in those results, for example the pass rate in Table 4. They are not enough to claim an improvement or to affirm that Bluerandom performed better, but having similar results also indicate that Bluerandom can be used as an additional source of entropy.

The results of Bluerandom from Experiment 4 (Fig. 2) shows a growth in the available entropy in the system. This scenario allowed to evaluate the performance of Bluerandom in a limited device. The purpose of Bluerandom is not to replace entropy sources but serve as another alternative for the system.

5.3. Throughput

The advertising interval that allows Bluetooth devices to read the RSSI value can limit the throughput of Bluerandom. For example, the advertising interval can range from 20 milliseconds (ms) to 10.24 seconds [SIG 2019]. Considering only BLE devices, generally, they choose to advertise between 100 ms to 500 ms. On the other hand, if there are only Bluetooth Basic Rate or Enhanced Data Rate (BR/EDR) devices nearby, an inquiry scan must be performed. Then, the RSSI is measured from each inquiry response. Inquiry scans have a slower interval compared to the BLE devices [SIG 2019].

The implementation of Bluerandom was focused only on Bluetooth technology, mostly due to the popularity of the BLE devices. They are widely used to connect smartphones with low power sensors [Böcker et al. 2017], found in cars, laptops, tablets, stereo receivers, wearables, and others. BR/EDR devices were not considered due to the slower advertising interval.

5.4. Comparison with other generators

Table 5 presents the comparison of Bluerandom with other generators from the literature [von zur Gathen 2015]. The selected configuration for Bluerandom was with 4 devices and with the "Just Odd or Even" extractor. The metrics compared are the byte entropy (rounded to the fith decimal digit) and throughput. Bluerandom results are close to others in terms of byte entropy, but the throughput is low compared to other generators.

Table 5. Comparison of Bluerandom with other generators [von zur Gathen 2015]

Generator	Byte Entropy	Throughput
Bluerandom	7.99522	1 byte/s
Noisy Diode	7.99963	31.39486 kB/s
/dev/random (in the field)	7.99979	5.584 byte/s
/dev/random (in the lab)	7.99948	0.1917 byte/s
Linear Congruential	7.99969	193.64313 kB/s
Blum-Blum-Shub	7.99962	28.91291 kB/s

Regarding the results of Wang et al. [Wang et al. 2011], they achieve an average entropy equal to 0.9286 per bit (i.e. the more close to 1 is the better). It is important to highlight that their result is an average of 10 measurements, and it is obtained from a simulated environment [Wang et al. 2011], which turns difficult to compare. Nevertheless, Bluerandom achieved a normalized result of 0.9994025 per bit, computed from 7.99522 (Table 5) divided by 8.

6. Final Considerations

The results of the evaluation of Bluerandom allowed us to affirm that this approach could benefit IoT devices as an additional source of entropy, improving security. The scalability of Bluerandom was verified by increasing the number of BLE devices nearby. In addition, satisfactory results were obtained when Bluerandom was used as a seed for different PRNGs.

In summary, the contributions of this paper are listed below:

- An alternative entropy source, focusing on the IoT context.
- An evaluated implementation of Bluerandom with different extraction methods.
- A Proof-of-concept regarding the random number generation based on the Bluetooth *Received Signal Strength Indicator* (RSSI), and also its validation through statistical testing.

Even if we use a deterministic RNG, a single entropy source can be risky for many cryptographic protocols. There is where stands the main benefit of Bluerandom: to serve as an additional source of entropy, improving security and mitigating attacks on the RNG as a whole. Since RSSI information has variations and it is measured by the receiving device, it is hard to predict those values.

The security of IoT devices has the challenge to utilize its hardware resources efficiently to reduce costs and energy consumption. Exploring randomness in the environment using the available hardware instead of adding new components could help in this aspect. As long as the environment has devices nearby, Bluerandom can explore the randomness in it. Relying in additional entropy sources instead of only one can improve the robustness of security mechanisms that use cryptography.

6.1. Future Works

Although the experiments conducted showed practical scenarios, more evaluations of Bluerandom are being considered for future works. In addition, other metrics such as LQI (Link Quality Indicator) could be investigated and evaluated. It is worth mentioning that the RSSI information is a measure for any Wireless technology, not only for Bluetooth. This means that the same approach presented in this paper could be investigated and developed for other contexts as well. In addition, other sources can be investigated in the IoT context, like *wearables*, in order to verify if they can be used as entropy sources for cryptographic applications.

Acknowledgment

The authors would like to thank Emil Lenngren and the Bluez community for their feedback about the Bluetooth hardware and driver development. Also, the authors would like to thank the support of the Federal University of Technology (UTFPR).

References

Bassham, L. E., Rukhin, A. L., Soto, J., Nechvatal, J. R., Smid, M. E., Leigh, S. D., Levenson, M., Vangel, M., Heckert, N. A., and Banks, D. L. (2010). A statistical test suite for random and pseudorandom number generators for cryptographic applications. Technical report, National Institute of Standards and Technology (NIST).

BlueZ (2018). *BlueZ: Official Linux Bluetooth protocol stack*. BlueZ Project. Available at: `http://www.bluez.org/`.

Böcker, S., Arendt, C., and Wietfeld, C. (2017). On the suitability of bluetooth 5 for the internet of things: Performance and scalability analysis. In *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pages 1–7. IEEE.

Brown, R. G. (2019). Dieharder, a random number test suite. version 3.31.1. Available at: `http://webhome.phy.duke.edu/~rgb/General/dieharder.php`.

Cha, S.-C., Yeh, K.-H., and Chen, J.-F. (2017). Toward a robust security paradigm for bluetooth low energy-based smart objects in the internet-of-things. *Sensors*, 17(10):2348.

Checkoway, S., Niederhagen, R., Everspaugh, A., Green, M., Lange, T., Ristenpart, T., Bernstein, D. J., Maskiewicz, J., Shacham, H., and Fredrikson, M. (2014). On the practical exploitability of dual {EC} in {TLS} implementations. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, pages 319–335.

Collotta, M., Pau, G., Talty, T., and Tonguz, O. K. (2018). Bluetooth 5: A concrete step forward toward the iot. *IEEE Communications Magazine*, 56(7):125–131.

Cypress (2019). *CE221295 PSoC 6 MCU Cryptography: True Random Number Generation*. Cypress Semiconductor Corporation. Available at: `https://www.cypress.com/file/404176/download`.

Dinca, L. M. and Hancke, G. (2017). Behavioural sensor data as randomness source for iot devices. In *2017 IEEE 26th International Symposium on Industrial Electronics (ISIE)*, pages 2038–2043. IEEE.

Gutterman, Z., Pinkas, B., and Reinman, T. (2006). Analysis of the linux random number generator. In *2006 IEEE Symposium on Security and Privacy (S&P'06)*, pages 15–pp. IEEE.

Herrero-Collantes, M. and Garcia-Escartin, J. C. (2017). Quantum random number generators. *Reviews of Modern Physics*, 89(1):015004.

Huh, J.-H., Bu, Y., and Seo, K. (2016). Bluetooth-tracing rssi sampling method as basic technology of indoor localization for smart homes. *Int. J. Smart Home*, 10(10):1–14.

Kohlbrenner, P. and Gaj, K. (2004). An embedded true random number generator for fpgas. In *Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*, pages 71–78. ACM.

Langley, A., Chang, W., Mavrogiannopoulos, N., Strombergson, J., and Josefsson, S. (2016). Chacha20-poly1305 cipher suites for transport layer security (tls). RFC 7905, RFC Editor.

MCrypt. *Libmcrypt data encryption library*. Available at: `http://mcrypt.hellug.gr/lib/mcrypt.3.html`.

Müller, S. (2018). Linux random number generator—a new approach.

Nordic (2019). *nRF52811 Product Brief Version 1.0*. Nordic Semiconductor.

OpenSSL. *OpenSSL Cryptography and SSL/TLS Toolkit*. OpenSSL.org. Available at: `https://www.openssl.org/`.

Procter, G. (2014). A security analysis of the composition of chacha20 and poly1305. *IACR Cryptology ePrint Archive*, 2014:613.

Seo, H., Choi, J., Kim, H., Park, T., and Kim, H. (2014). Pseudo random number generator and hash function for embedded microprocessors. In *2014 IEEE World Forum on Internet of Things (WF-IoT)*, pages 37–40. IEEE.

Shokri-Ghadikolaei, H., Fischione, C., and Modiano, E. (2016). On the accuracy of interference models in wireless communications. In *2016 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE.

SIG (2019). *Bluetooth Core Specification Version 5.1*. Bluetooth Special Interest Group (SIG).

Stallings, W. (2017). *Cryptography and network security: principles and practice*. Pearson Upper Saddle River, 7 edition.

Szczepanski, J., Wajnryb, E., Amigó, J. M., Sanchez-Vives, M. V., and Slater, M. (2004). Biometric random number generators. *Computers & Security*, 23(1):77–84.

Tan, H., Tsudik, G., and Jha, S. (2019). Mtra: Multi-tier randomized remote attestation in iot networks. *Computers & Security*, 81:78–93.

TI (2018). *CC1312R SimpleLink High-Performance Sub-1 GHz Wireless MCU*. Texas Instruments Incorporated. Revised March 2019.

von zur Gathen, J. (2015). *Crypto School*. Springer-Verlag, 1 edition.

Walker, J. (2008). Ent: a pseudorandom number sequence test program. *Software and documentation available at/www. fourmilab. ch/random/S*.

Wallace, K., Moran, K., Novak, E., Zhou, G., and Sun, K. (2016). Toward sensor-based random number generation for mobile and iot devices. *IEEE Internet of Things Journal*, 3(6):1189–1201.

Wang, Q., Su, H., Ren, K., and Kim, K. (2011). Fast and scalable secret key generation exploiting channel phase randomness in wireless networks. In *2011 Proceedings IEEE INFOCOM*, pages 1422–1430. IEEE.

Willers, O., Huth, C., Guajardo, J., Seidel, H., and Deutsch, P. (2019). On the feasibility of deriving cryptographic keys from mems sensors. *Journal of Cryptographic Engineering*, pages 1–17.

Zhu, H., Zhao, C., Zhang, X., and Yang, L. (2013). A novel iris and chaos-based random number generator. *computers & security*, 36:40–48.