Entropy as a Service: A Lightweight Random Number Generator for Decentralized IoT Applications

Ikram Ullah
University of Twente
Enschede, The Netherlands
i.ullah@utwente.nl

Nirvana Meratnia
University of Twente
Enschede, The Netherlands
n.meratnia@utwente.nl

Paul J. M. Havinga
University of Twente
Enschede, The Netherlands
p.j.m.havinga@utwente.nl

***Abstract*—Cryptographic Pseudorandom Number Generators (CPRNG) play a very crucial role in Internet of Things (IoT) security. Cryptographic protocols require random numbers for nonces, salts, and key generation. However, developing secure and lightweight CPRNG is strenuous. Insecure source of randomness can evolve in vulnerabilities and can jeopardize security mechanisms. As the number of IoT devices are expected to exceed over billions, the demand for distributed CPRNG increases. Manually configuring random numbers in large numbers of IoT devices is practically challenging and insecure. In this paper, we propose a cryptographically secure pseudorandom number generator based on sensor data as source of randomness. The appealing characteristic of a sensor data based random number generator is that sensor can possibly generate infinite data. Thus, having longer period and perhaps higher entropy. We also present proof-of-concept of potential usage of sensor data as a source of randomness. Furthermore, the mechanism is evaluated with the NIST statistical suite.**

***Index Terms*—Internet of Things, security, random number generator**

I. Introduction

The Internet of Things (IoT) is massively adopted by many industries for vast number of applications. In the last decade the number of IoT devices increased exponentially [1] and it is expected to increase even more. For instance in smart logistics, where pallets are embedded with smart sensors for analytical purposes (detect, predict, and prevent various events related to logistics). In smart logistics there is a complex chain of stakeholders, for which security and confidentiality are key issues. So the data generated by these smart sensors should be secure. Managing these sensors centrally is exceptionally challenging, so the need for distributed security mechanism increases [2]. In distributed mechanism every device is capable of generating their own random numbers without the need of any central party or manual configuration.

Sensor data can also be used for security purposes. For instance, intrusion detection [3] and random number generators [4] [5]. Random numbers are used as nonces in authentication protocols to avoid replay and reflection attacks. They can also be used to pad messages before their encryption or to combine with a password to prevent dictionary attacks [6]. Key establishment protocols can use random numbers to generate session keys [6] [7].

Important constituents for any randomization are the source, quality, and unpredictability of randomness [5]. Weak random number generators can be exploited [8] [9] [10]. Generating truly random numbers, which generally are based on non-deterministic physical phenomena are hard and rather slow [6] [11]. Therefore, pseudorandom numbers are generated instead. Generating pseudorandom numbers is comparatively faster. Various randomness sources as mentioned in [12] are used in real world applications, such as CPU temperature, positions of mouse movements, and physical features of the electric circuits. However, various existing sources of randomness have many limitations, such as limited generation rate, lack of randomness of data, requires an external device, and failures are difficult to detect [13]. In IoT, high entropy random numbers are hard to achieve due to lack of resources [14] [15]. Entropy in information theory is the measure of randomness of a random variable or outcome of a random process. Entropy of a discrete random variable (X) with probability p is defined as

$$H(X) = -\sum_{x \in X} p(x) \log p(x)$$

where x represents sample from X. Joint entropy $H(X, Y)$ of random variables X, Y is given as

$$H(X, Y) = -\sum_{x \in X} \sum_{y \in Y} p(x, y) \log p(x, y)$$

To attain high-quality random numbers, a high-entropy random source is necessary [16]. Therefore, in this paper we propose the usage of sensor data as the source of randomness. The ultimate goal of this research is to transform sensor data into high-quality source of randomness. We have demonstrated that it is possible to generate random sequences from sensor data. We have also compared the NIST statistical test [17] results of our mechanism with the *lmRNG* [18]. Based on our analysis, our proposed seed extraction methodology has shown dominance over *lmRNG* [18] in 5 out of 12 passed tests. The novelty of our approach is that, each sensor can independently generate random numbers without any need of manually embedding and configuring random numbers. Which is important in decentralized IoT environments.

II. Our Contributions

The main contributions of this paper are as follows:

- We propose a novel and practical mechanism to extract random seeds from sensor data. The extracted seeds pass NIST statistical tests.
- We propose slight modification in existing cryptographic pseudorandom number generator, namely TinyRNG [19], to make it more efficient and use it as cryptographic pseudorandom number generator.

III. Preliminaries

In this section, we provide a short introduction to Hamming distance, Fisher-Yates shuffle, randomness extraction, and NIST test suite. Since, these terminologies are used in the rest of the paper.

Hamming Distance: Hamming distance [20] between two sequences $x = (x_1, x_2...x_n)$ and $y = (y_1, y_2...y_n)$ denoted by $d_H(x, y)$, is the number of positions where the values x_i and y_i vary.

$$d_H(x, y) = \sum_{i=1}^{n} \delta(x_i, y_i)$$

where

$$\delta(x_i, y_i) = \begin{cases} 0 & x_i = y_i \quad (1) \\ 1 & x_i \neq y_i \quad (2) \end{cases}$$

If the two sequences are chosen at random (out of 2^n equally likely), then the Hamming distance is $n/2$, which demonstrates that the two sequences are significantly different.

Fisher-Yates Shuffle: Fisher-Yates shuffling algorithm produces a uniformly random permutation of an input array in linear time. It guarantees that each permutation is generated with equal probability. Given a finite sequence, it generates an unbiased pseudorandom permutation. Where every permutation is equally likely. The algorithm randomly draws an element from a sequence S of n elements, until no element remains. Pseudocode of this process is shown below.

```
for i from n-1 downto 1 do
   j ← random integer such that 0 ≤ j ≤ i
   exchange S[j] and S[i]
 }
```

Fisher-Yates shuffle is theoretically correct random permutation algorithm and uses $O(N)$ time and $O(1)$ space.

Randomness Extractor: Generating truly random numbers is computationally challenging. So, generally it is sufficient to derive a single secret key, which can be used as key for a pseudorandom function (generator) to generate further pseudorandom keys [21]. One way to derive secret key is using randomness extractor. Randomness extraction is a mechanism to derive uniform key from non-uniform randomness sources [21]. It distill unpredictability of source to indistinguishability of output from uniform distribution. The derived key can be used as an input to a pseudorandom number generator (PRNG), which potentially generates large stream of pseudorandom keys. Randomness extractor and Pseudorandom functions are related in their applications. Thus, it is beneficial to use the commonly used pseudorandom functions for randomness extraction [21]. Pseudorandom functions are convenient since they can also be used by other applications (such as, key exchange protocol) for key derivation [21]. Block ciphers related pseudorandomness mode, such as cipher block chaining message authentication code CBC-MAC is demonstrated as randomness extractor in [21].

NIST Test Suite: NIST statistical tests [17] are performed to ensure that the random numbers have has sufficient entropy. NIST test suite consists of a set of statistical tests to evaluate the randomness of a sequence. The output of these statistical tests is given as P-values. A given sequence passes the test when its P-value is greater than 0.01 [18].

IV. Related Work

Dinca & Hancke [1] proposed the concept of using IoT devices to gather data on human behaviour for creating randomness. They have used smartphone's sensors data as a seed for PRNG to generate secure encryption keys. Marghescu et al. [22] described a set of random number generators (RNG) within Android-based smartphones by utilizing different sensors. Song et al. [16] analyzed the process of scheduling information of Android systems and discovered that the rules of the PID calling order reflect the change of interruption times. The authors showed that random numbers can be generated by pre-processing the noise source information, quantization, and coding. Hong & Liu [4] proposed a seeding technique based on accelerometer data to generate seeds for random number generator. Predictable patterns from data were removed by calculating first-order derivative of data. Unpredictability was increased using Fast Fourier Transform (FFT) and the output was fetched into random number generator as seed. Krhovjak et al. [5] studied the usage of embedded and hands-free microphone and camera as sources of randomness. Based on their analysis, hands-free microphone is more sensitive and optical sensor noise is not suitable source of randomness. Other sources of randomness proposed in the literature are based on phone recording [7] and harvesting randomness directly from the network routing metadata [23]. However existing sources of randomness have various constraints for instance, they are computationally costly, have short period, generated very slowly, have high computation complexity, or require dedicated hardware. Therefore, we are proposing a decentralized seed derivation mechanism for IoT devices.

V. Proposed Pseudorandom Number Generator

We propose a lightweight approach to extract seed from sensor data. In order to extract seed, the proposed approach is applied to three different datasets. The datasets are collected using motion sensors, sound sensors, and channel state information. The randomness of the extracted seed is evaluated with NIST test suite. Finally, we describe how the seed is fetched into randomness extractor to generate cryptographic keys.

A. Random Seed Extraction from Sensor Data

Sensor data in the raw format (before processing) is highly correlated therefore, we extract the least significant bits (LSB), as these bits are more random and noisy. However, most significant bits (MSB) of channel state information (CSI) are used, since LSB of CSI are zeros. Furthermore, all computing devices have a finite precision representation, in which the integers are round-off. In the extraction of LSB, the bits are not round-off to maintain randomness. Our proposed seed extraction mechanism is shown in the Algorithm 1. The mechanism consists of two main operations, steps and random permutation. In steps operation, instead of extracting every consecutive data point (LSB), we extract every ith (i steps) data point (LSB), where i is the size of the step. The concept of step is similar to the step in a for loop in any programming language. Thus, with step operation we can skip repetitive data points and eliminate correlations among the data points. The data points (LSB) extracted after steps operation is of size $length((Data)/(step\ size))$. Then, the extracted data is converted into binary format. Again, step operation is performed on binary data. In random permutation operation, we use Fisher-Yates to uniformly extract certain number of binary bits. The number of bits to be extracted depend on the sizes of steps performed. In other words, larger the sizes of steps, fewer the number of bits can be extracted. For instance, smaller sizes of steps are possible if $0.01 * length(Data)$ number of bits are to be extracted. While larger sizes of steps are possible if $0.001 * length(Data)$ number of bits are to be extracted. The output is expected to be uniformly distributed sequence if $0.001 * length(Data)$ number of bits are extracted.

Algorithm 1: Derivation of Seed

```
Require: $Data$
Ensure: $RandomSeed$
  for $i = 1\ to\ length(Data)\ ,\ Step\ Parameter$ do
    $StepData \leftarrow Data(i)$
  end for
  for $j = 1\ to\ length(StepData)$ do
    $LSBData \leftarrow ExtractLSB(StepData)$
  end for
  for $k = 1\ to\ length(LSBData)$ do
    $BinaryData \leftarrow DecimalToBinary(LSBData)$
  end for
  for $m = 1\ to\ length(BinaryData)\ ,\ Step\ Parameter$ do
    $StepBinaryData \leftarrow BinaryData(m)$
  end for
  $RPStepBinaryData \leftarrow$
  $KnuthShuffle(length(StepBinaryData), 0.01 *$
  $length(Data))$
  for $n = 1\ to\ length(RPStepBinaryData)$ do
    $p = RPStepBinaryData(n)$
    $RandomSeed \leftarrow StepBinaryData(p)$
  end for
```

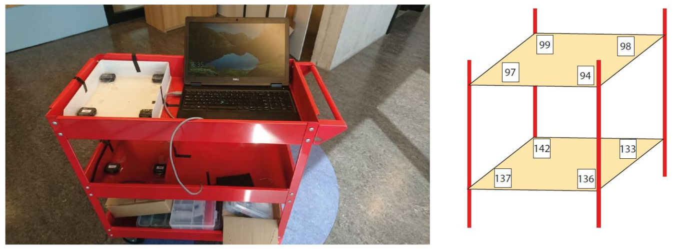

Figure 1. The experimental setup which mimics smart pallet is used to collect motion dataset.

B. Data Acquisition

1) Motion Sensor Dataset: We have setup a mock smart logistics environment that resembles pallets as shown in the Figure 1. The setup comprises of eight Promove [24] nodes placed on a trolley. This dataset includes accelerometer, compass, and high-g accelerometer. The trolley is locomoted to accumulate data. We collected the data three times, for approximately 10 minutes each time. To analyze different patterns for comparison and evaluation purposes. The data is collected with sampling rate of 500 Hz. Each type of sensor data (i.e. accelerometer) is analyzed individually.

2) Sound Sensor Dataset: The sound data is collected with a standalone microphone. In our analysis, sound data is stored in WAV file format. WAV file format is not compressed or manipulated. The sound data has the sampling frequency of 22000 Hz, pulse coded modulation (PCM) 32 bits, and the noise is not filtered. Sound data, mainly depends on the type of the device [5]. As different devices have different sensitivity. Every source of sound is not suitable for randomness such as, quiet room.

3) Channel State Information (CSI) Dataset: Channel state information (CSI) characterizes wireless signals propagation from the transmitter to the receiver at certain carrier frequencies [25]. Gigabyte Brix IoT (specifications shown in Table I) is used as transceiver node. It is connected to an access point (TP-LINK AC1750). The transmitter and receiver were approximately 25cm away from each other. WiFi network is used as connectivity. The CSI is collected over the received signals from the access point to the node. The sampling rate is the number of pings send from the node to the access point. CSI values are complex, so we extract the real and imaginary values individually, and then concatenate these values.

VI. Performance Evaluation & Discussion

Extensive analysis of the proposed seed extraction mechanism is performed. For each type of dataset, different steps sizes are analyzed till we found the optimal steps sizes that ensure sufficient seed entropy. Optimal steps sizes might be different for different datasets. As shown in the Figure 2, for our datasets step size 90 mostly give optimal probability to

TABLE I
HARDWARE SPECIFICATIONS GIGABYTE BRIX IOT

Component	Specifications
RAM	1x HyperX 8GB DRR3L-SO DIMM 1866 MHz
Processor	Intel Apollo Lake N34500
Hard drive	Transcend MTS800 SSD 128 GB (M.2 2280)
Graphics card	None
Wireless adapter	Intel N Ultimate Wi-Fi Link 5300
Size	165x105x27mm
Operating System	Ubuntu 14.04.4

pass NIST test. While for the binary data the optimal step size is 40. Our proposed seed derivation mechanism successfully extracted random seeds from all the three datasets mentioned above. The proportion of passed tests of the extracted seeds of size $0.001 * length(Data)$ is approximately 90%. We have observed that if smaller steps are performed and large number of bits are extracted, then it is highly probable that most of the NIST statistical tests will fail. Otherwise, if larger steps are performed and fewer bits are extracted, then there is very high probability (approximately 0.9) that the extracted seed will pass NIST statistical tests. We have tested our proposed seed extraction on the above mentioned three different data sources. Table II and III shows the results of the NIST statistical tests applied to the random bits sequence generated by our proposed seed extraction mechanism. 12 out of 15 tests are passed which demonstrates that the seed generated by our seed extraction mechanism is random. The NIST statistical tests passed by our seed extraction mechanism are compared with the similar tests passed by *lmRNG* [18]. Table IV shows the comparison of our proposed seed extraction mechanism described in section V-A with *lmRNG* [18]. The P-values of our mechanism is for sample of 20000 bits, while the P-values of *lmRNG* [18] is for sample of 1000000 bits. In this comparison, our methodology demonstrated dominance in certain tests. Since, in *lmRNG* [18] only P-values for 1000000 bits is given, so we do not know how much the *lmRNG* [18] P-values will change for sample of 20000 bits. Furthermore, we have used Hamming distance to compare the seeds generated by two different neighbouring sensors while both being in the same context, as shown in the Figure 1. As expected, the hamming distance was 50 for sequences of 100 bits. Which illustrates that the two seed are very different from each other.

A. PRNG

Similar to TinyRNG [19], we propose CBC-MAC as randomness extractor and use the same block cipher as the CPRNG. The CPRNG encrypts a counter using key generated by randomness extractor. Also, the extracted randomness is used to re-seed the key of the CPRNG. However, we suggest three possible changes in TinyRNG [19] that might improve it. First, in TinyRNG [19] two CBC-MAC are required, because of their source of randomness, while in our case we only need one CBC-MAC. Second, unlike TinyRNG [19] where the key for CPRNG is supposed to be provided at programming time and updated by reseeding, in our case we recommend to use

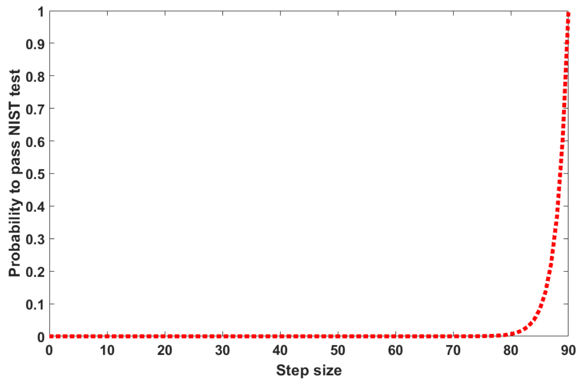

Figure 2. The probability to pass NIST test is minimal for step size below 90. While for step size above 90, the probability increases.

TABLE II
NIST STATISTICAL TESTS FOR THE THREE DATASETS FOR VARIOUS STEP SIZES AND RANDOM PERMUTATIONS.

Tests Passed	Tests not Applicable
Approximate entropy	Random excursions
Block frequency	Random excursions variant
Cumulative sums	Universal
FFT	
Frequency	
Linear complexity	
Longest run	
Non-overlapping template	
Overlapping template	
Rank	
Runs	
Serial	

TABLE III
NIST STATISTICAL TESTS P-VALUES FOR SEQUENCE OF 20000 BITS SEED.

Statistical Test	Distribution of P-Values
Approximate Entropy (block length 10 bits)	0.151871
Block frequency (block length 10 bits)	0.328051
Cumulative sums (Forward)	0.278876
Cumulative sums (Reverse)	0.176714
FFT	0.363646
Frequency	0.149162
Linear complexity	0.532336
Longest run	0.506970
Non-overlapping template	
template = 000110111	0.616752
template = 001001101	0.410845
template = 001011011	0.846972
template = 101111100	0.810269
template = 110111000	0.495427
template = 111010100	0.893308
template = 111101100	0.459074
Overlapping template	0.421596
Rank	0.181162
Runs	0.909460
Serial Test (block length 16)	0.383318
Serial Test (block length 16)	0.502142

TABLE IV
NIST statistical Tests P-Values of Our Proposed Seed Extraction Mechanism (20000 Bits) is Compared with P-Values of *lmRNG* [18] (1000000 Bits).

Statistical Test	**Proposed Methodology P-Values**	**lmRNG P-Values**
Approximate Entropy (block length 10 bits)	0.151871	0.429923
Block frequency (block length 10 bits)	0.328051	0.666245
Cumulative sums (Forward)	0.278876	0.668321
Cumulative sums (Reverse)	0.176714	0.340858
FFT	0.363646	0.254411 (DFT)
Frequency	0.149162	0.500279
Linear complexity	0.532336	0.452173
Longest run	0.506970	0.914025
Non-overlapping template		
template = 000110111	0.616752	0.435430
template = 001001101	0.410845	0.191687
template = 001011011	0.846972	0.595549
template = 101111100	0.810269	0.755819
template = 110111000	0.495427	0.395940
template = 111010100	0.893308	0.325206
template = 111101100	0.459074	0.204439
Overlapping template	0.421596	0.672470
Rank	0.181162	0.834308
Runs	0.909460	0.299736
Serial Test (block length 16)	0.383318	0.208837
Serial Test (block length 16)	0.502142	0.647530

the output of CBC-MAC as initial key for the block cipher and to reseed the CPRNG as shown in the Figure 3. The reason we are not recommending key at the programming level is that, we expect each individual sensor manage itself in a decentralized manner, by generating its random numbers without any need of preconfiguring keys. Also, in a situation where the number of sensors are over millions, configuring such large number of devices become challenging. Therefore, using the output of CBC-MAC as initial key for block cipher can prevent the unnecessary overhead. Third, in order to minimize the storage requirements, as sensor data is stateless, it is not necessary to have entropy pool. Entropy can be generated instantly and it can be fetched into the CBC-MAC. Also, if the adversary has access to the entropy pool, it is possible to extract the randomness from the pool. So we recommend to generate seeds on the fly. The choice of CBC-MAC as randomness extractor has many advantages, such as it utilizes low memory (re-using block cipher as CPRNG) [19], is a more practical, and a better randomness extractor [21].

VII. Security Analysis

There are many possible scenarios, when the attacker can exploit the sensor and consequently random number generators.

A. Exploiting Randomness Extractor

Block ciphers are cryptographically secure RNG. The extraction properties of CBC-MAC show that for any input distribution with sufficiently high entropy, CBC-MAC guarantee a uniform output for any family of permutations. Let f be a random permutation over $\{0, 1\}^k$, X be an input distribution with minimum entropy of at least $2k$ and $F(X)$ be the function f computed in CBC-MAC mode over L blocks. Then, the statistical distance between $F(X)$ and the uniform distribution on $\{0, 1\}^k$ is $L.2^{-K/2}$ [21]. Which proves that k-bits output from $F(X)$ is computationally indistinguishable from uniform distribution. Thus, it can be used as secure cryptographic keys.

B. Remotely

The attacker's aim is to predict the random number generated by observing pseudorandom numbers. Remote attack against our proposed seed derivation mechanism can be hard, since the attacker need to generate the same sensor data, perform the same steps, and extract the same randomly permuted bits.

C. Physically

When the attacker has physical access to the sensor, he can read the memory of the sensor and can change the code unless there are secondary protection such as program integrity verification techniques [19] [26]. However, if secure pseudorandom number generator is used to generate secret keys to encrypt the communication, then the attacker should not be able to decrypt the communication before or after the attack, instead can only decrypt the communication during the time of the compromise [19]. A third scenario can be that the attacker fit his own sensors into the network to eavesdrop and collect the data. However, the probability of generating correct seed is much less because, the random seed is extracted uniformly. We have compared the seeds of multiple sensors that are in the same network and at the same time using Hamming distance. And it turns out that the seeds generated by sensors are significantly different from each other.

VIII. Conclusion

We propose a seed extraction and PRNG mechanism with consideration of decentralization of IoT ecosystem. Where configuration and built-in keys is becoming a tedious job and technically impractical. Therefore, the reusability of sensor data for decentralized cryptographic purposes is very crucial. The results show that the proposed seed extraction mechanism derive random seed from sensor data and the seed is capable of passing 12 out of 15 NIST statistical tests. The results of the passed tests are compared with *lmRNG* [18]. In certain tests, our proposed methodology shows better results. The proposed CPRNG is seeded with the extracted seed. The results shown in [19], illustrate that the proposed CPRNG is secure, lightweight, and practical.

IX. Limitations

The proposed seed extraction mechanism is probabilistic. Since, sensitivity of sensors can impact the quality of the seed and also sensors behaviour is unpredictable. For instance, when the sensor is idle, there might be no randomness.

Acknowledgment

This work has been partially supported by the EFRO, OP Oost program in the context of Countdown project.

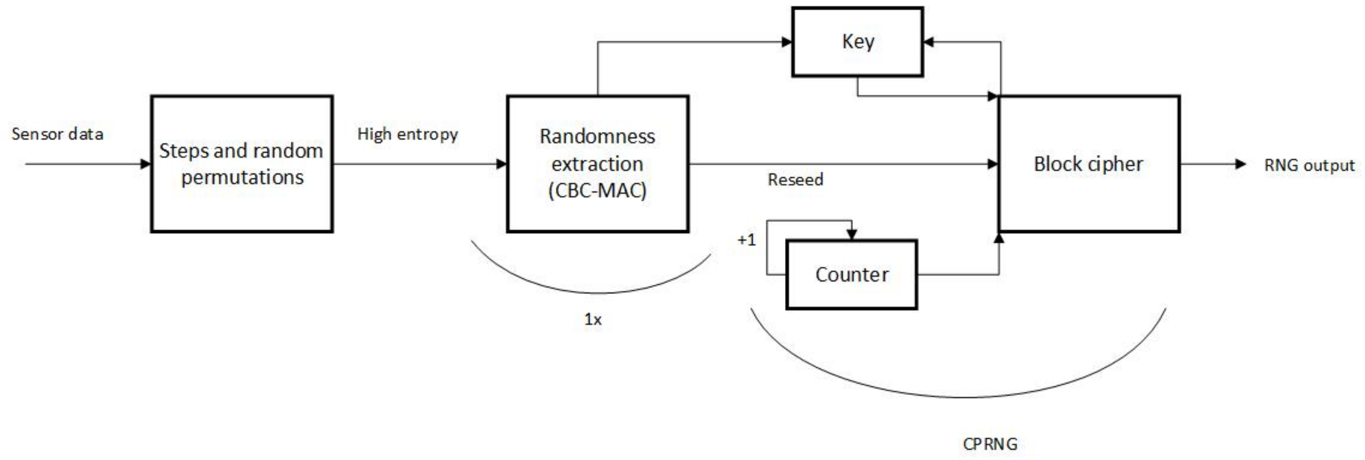

Figure 3. The proposed randomness extraction and pseudorandom number generation mechanism. CBC-MAC is used as randomness extractor and the same block cipher is used as the CPRNG. The CPRNG is a block cipher in counter mode encrypts a counter using key generated by randomness extractor.

References

[1] L. M. Dinca and G. Hancke, "Behavioural sensor data as randomness source for iot devices," in *2017 IEEE 26th International Symposium on Industrial Electronics (ISIE)*, June 2017, pp. 2038–2043.

[2] L. Francis, G. Hancke, K. Mayes, and K. Markantonakis, "Potential misuse of nfc enabled mobile phones with embedded security elements as contactless attack platforms," in *2009 International Conference for Internet Technology and Secured Transactions, (ICITST)*, Nov 2009, pp. 1–8.

[3] J. Kaur, P. Swain, and G. Tamizharasi, "Ship intrusion detection using accelerometer and temperature sensor," *International Journal of Research*, vol. 3, no. 9, pp. 461–466, 2016. [Online]. Available: https://journals.pen2print.org/index.php/ijr/article/view/4464.

[4] S. L. Hong and C. Liu, "Sensor-based random number generator seeding," *IEEE Access*, vol. 3, pp. 562–568, 2015. [Online]. Available: https://doi.org/10.1109/access.2015.2432140.

[5] J. Krhovják, V. Matyas, and P. Svenda, "The sources of randomness in mobile devices," *In Proceeding of NORDSEC*, pp. 73–84, 2007.

[6] J. Krhovják, "Analysis, demands, and properties of pseudorandom number generators," -, November 2019.

[7] F. Xiang, L. Zhang, Z. Zhang, and L. Zhang, "True random bit generator based on cell phone recording and chaotic encryption," in *2012 IEEE International Conference on Automation and Logistics*. IEEE, Aug. 2012. [Online]. Available: https://doi.org/10.1109/ical.2012.6308147.

[8] E. B. Barker and J. M. Kelsey, "Recommendation for random number generation using deterministic random bit generators," NIST, Tech. Rep., 2012. [Online]. Available: https://doi.org/10.6028/nist.sp.800-90a.

[9] Y. Dodis, D. Pointcheval, S. Ruhault, D. Vergniaud, and D. Wichs, "Security analysis of pseudo-random number generators with input," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security - CCS '13*. ACM Press, 2013. [Online]. Available: https://doi.org/10.1145/2508859.2516653.

[10] M. Dubal and A. Deshmukh, "On pseudo-random number generation using elliptic curve cryptography," in *Security in Computing and Communications*, S. M. Thampi, P. K. Atrey, C.-I. Fan, and G. M. Perez, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 77–89.

[11] J. C. Lagarias, "Pseudorandom numbers," *Statistical Science*, vol. 8, no. 1, pp. 31–39, 1993. [Online]. Available: http://www.jstor.org/stable/2246038.

[12] S. Ruhault, "Security analysis for pseudo-random number generators," Theses, Ecole normale supérieure - ENS PARIS, Jun. 2015. [Online]. Available: https://tel.archives-ouvertes.fr/tel-01236602.

[13] M. Herrero-Collantes and J. C. Garcia-Escartin, "Quantum random number generators," *Reviews of Modern Physics*, vol. 89, 04 2016.

[14] M. Pawlowski, A. J. Jara, and M. Ogorzalek, "Harvesting entropy for random number generation for internet of things constrained devices using on-board sensors," *Sensors*, vol. 15, pp. 26 838–26 865, 10 2015.

[15] M. L. Chaves, J. J. Márquez, H. Pérez, L. Sánchez, and A. Vizan, "Intelligent decision system based on fuzzy logic expert system to improve plastic injection molding process," in *International Joint Conference SOCO'17-CISIS'17-ICEUTE'17 León, Spain, September 6–8, 2017, Proceeding*. Springer International Publishing, Aug. 2017, pp. 57–67. [Online]. Available: https://doi.org/10.1007/978-3-319-67180-2_6.

[16] P. Song, Y. Zeng, Z. Liu, J. Ma, and H. Liu, "True random number generation using process scheduling of android systems," in *2018 International Conference on Networking and Network Applications (NaNA)*, Oct 2018, pp. 304–309.

[17] A. Rukhin. (2010, April) A statistical test suite for random and pseudorandom number generators for cryptographic applications. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf.

[18] A. Sojka and K. Piotrowski, "lmrng: A lightweight pseudorandom number generator for wireless sensor networks," *SECRYPT 2012 - Proceedings of the International Conference on Security and Cryptography*, pp. 358–363, 01 2012.

[19] A. Francillon and C. Castelluccia, "Tinyrng: A cryptographic random number generator for wireless sensors network nodes," in *2007 5th International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks and Workshops*, April 2007, pp. 1–7.

[20] T. Yamada, "Principles of error detection and correction," in *Essentials of Error-Control Coding Techniques*. Elsevier, 1990, pp. 11–37. [Online]. Available: https://doi.org/10.1016/b978-0-12-370720-8.50006-4.

[21] Y. Dodis, R. Gennaro, J. Håstad, H. Krawczyk, and T. Rabin, "Randomness extraction and key derivation using the cbc, cascade and hmac modes," in *Advances in Cryptology – CRYPTO 2004*, M. Franklin, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 494–510.

[22] A. Marghescu, G. Teseleanu, and P. Svasta, "Cryptographic key generator candidates based on smartphone built-in sensors," in *2014 IEEE 20th International Symposium for Design and Technology in Electronic Packaging (SIITME)*, Oct 2014, pp. 239–243.

[23] M. R. Khalili-Shoja, G. T. Amariucai, S. Wei, and J. Deng, "Secret common randomness from routing metadata in ad hoc networks," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 8, pp. 1674–1684, Aug 2016.

[24] Promove. [Online]. Available: http://inertia-technology.com/.

[25] Y. Ma, G. Zhou, and S. Wang, "WiFi sensing with channel state information," *ACM Computing Surveys*, vol. 52, no. 3, pp. 1–36, Jun. 2019. [Online]. Available: https://doi.org/10.1145/3310194.

[26] T. Park and K. G. Shin, "Soft tamper-proofing via program integrity verification in wireless sensor networks," *IEEE Transactions on Mobile Computing*, vol. 4, no. 3, pp. 297–309, May 2005.