Sensor-Based PUF: A Lightweight Random Number Generator for Resource Constrained IoT Devices

Maaike Hillerström[1], Ikram Ullah[2(✉)], and Paul J. M. Havinga[2]

[1] University of Twente, Enschede, The Netherlands
m.a.m.hillerstrom@student.utwente.nl
[2] Pervasive Systems Group, Department of Computer Science, University of Twente, Enschede, The Netherlands
{i.ullah,p.j.m.havinga}@utwente.nl

Abstract. Internet of Things (IoT) prevalence is surging swiftly over the past years, and by 2050, the number of IoT devices are expected to exceed 50 billion. IoT has been deployed in many application domains such as smart health, smart logistics and smart manufacturing. IoT has significantly improved quality of our day-to-day life. However, IoT faces multiple challenges due to its lack of adequate computational and storage capabilities and consequently it is very strenuous to implement sophisticated cryptographic mechanisms for security, trust and privacy. The number of IoT devices are increasing drastically which potentially leads to additional challenges namely transparency, scalability and central point of failure. Furthermore, the growing number of IoT applications induces the need of decentralized and resource constrained mechanisms. Therefore, in this paper, we propose a decentralized Random Number Generator (RNG) based on sensor Physical Unclonable Functions (PUF) in smart logistics scenario. PUF is a secure and lightweight source of randomness and hence suitable for constrained devices. Data is collected from various sensors and processed to extract cryptographically secure seed. NIST tests are performed to appraise the aptness of the proposed mechanism. Moreover, the seed is fed into an Elliptic Curve Cryptographic (ECC) mechanism to generate pseudo-random numbers and keys which can potentially be used for authentication, encryption and decryption purposes.

Keywords: Internet of Things · Decentralization · Random number generator · NIST · ECC

1 Introduction

In this paper, we propose a lightweight sensor-based PUF random number generator. As per Gartner IoT definition, "IoT is the network of physical objects that contain embedded technology to communicate and sense or interact with their internal states or the external environment" [1]. Physical objects can be any device that can be connected to the Internet such as sensors, smartphones and tablets. Over the years, the freight transportation industry has undergone some significant changes, which introduce new challenges. Transportation companies have more vehicles to manage, their customers have

Published by Springer Nature Switzerland AG 2022
L. M. Camarinha-Matos et al. (Eds.): IFIPIoT 2022, IFIP AICT 665, pp. 89–105, 2022.
https://doi.org/10.1007/978-3-031-18872-5_6

higher delivery demands, and the transportation network has become more complex [4]. IoT has been playing a significant role to overcome these challenges. For instance, traditional logistics processes are mainly manual, thus error prone and time consuming. IoT has transformed the traditional logistics into smart logistics which is more dynamic, robust and efficient.

IoT devices generate and exchange enormous amount of data and these data are commonly called as "big data" [2]. Various IoT services use "big data" for monitoring, optimization, learning, automation [6] and ultimately impel eminent applications for our day-to-day life. In the pursuance of secure data communication and access, there is growing need for IoT data security. Although, many security schemes are proposed in the literature, however, most of them are developed for mobile devices, which have more power and computational resources than the resource constrained IoT devices and conventional schemes are not scalable. For smart logistics, the limited power, storage and computational resources of the IoT devices must be taken into account when developing IoT security schemes. Another important requirement for smart logistics security mechanisms is implicit security, where human interactions (manual configuration) are not required to set up and configure keys, since in smart logistics sensors are deployed remotely and are large in numbers. This means that the sensors are capable of generating their own cryptographic keys without the necessity of manual configuration or a central party [27]. This we refer to as an implicit and decentralized cryptographic scheme.

Furthermore, random number generators play a very crucial role in cryptographic mechanisms [33,36]. Insecure random number generators can imperil security algorithms and ultimately lead to vulnerabilities [27]. PUFs are a very good candidate for randomization, as they are very secure by relying on uncontrollable manufacturing variations and they are suitable for constrained devices. Due to the manufacturing variations for instance each accelerometer or gyroscope generates different data, even when they share the exact same movements. Therefore, this research focuses on the use of sensor-based PUF to generate random numbers for cryptographic mechanisms in smart logistics. In our proposed algorithms, we extract randomness from data based on the manufacturing variations of the sensors, which can be ultimately used in implicit security mechanisms. Furthermore, a sensor-based PUF uses the already existing sensors in the node without the need of any additional hardware. This would reduce the costs, since no additional sensory circuit is required. We have illustrated that the proposed algorithms are adept to extract randomness from sensor data. The randomness is validated through NIST tests. We have also compared our results with SRAM PUFs [32]. Furthermore, we have used an existing Elliptic Curve Cryptographic (ECC) mechanism [35] as pseudo-random number generator based on the extracted seed.

2 Our Contributions

In this research, various algorithms are proposed to extract secure random seed from sensor-based PUF for decentralized IoT application particularly for smart logistics. NIST tests are performed. And finally, an ECC mechanism is used to generate pseudo-random numbers and cryptographic keys from the extracted seed.

3 Background Knowledge

In this section, we provide a brief introduction of terminologies that are related to randomness in information security.

Entropy. Entropy is "the measure of randomness in data" [28]. In other words, it is "the amount of uncertainty an attacker faces to determine the value of a secret" [29]. A sequence which has n bit entropy has the same randomness of a uniformly distributed n bit sequence [29]. It is a key factor in information theory. As defined in [30], let us suppose a random sequence $\{x_1, ...x_n\}$ with probability $(p_1, ..., p_n)$ then the entropy of a discrete random variable X is given below.

$$H(X) \equiv H(p_1, ..., p_n) = -\sum_{i=1}^{n} p_i log p_i \tag{1}$$

Randomness Extractor. Randomness extraction is the primary phase of key generation [31]. It is a mechanism to transform a minimal entropy source into a shorter but maximal entropy (uniformly distributed). The output of randomness extractor is non-deterministic and thus suitable for cryptographic purposes. Not all sources of randomness in the raw format are random enough. Therefore, randomness extraction is used. Randomness extractor can be represented as below.

$$Ext : \{0,1\}^q \rightarrow \{0,1\}^p \quad where\, q > p \tag{2}$$

Fast Fourier Transform (FFT). Fast Fourier Transform (FFT) is one of the most important mathematical operation that is used to represent data in the frequency domain. It is fast mechanism to depict frequency components of the data (spectral analysis). It can be formulated as below.

$$X_k = \sum_{n=0}^{N-1} x_n e^{-2\pi i k n/N} where\, X_k \;\; is \;\; amplitude \;\; and \;\; phase \tag{3}$$

Shuffling Algorithm. Fisher-Yates Shuffle is a simple shuffling algorithm to obtain a random permutation of a finite array. In order to shuffle an array (Arr) with n elements, generate a random number between $(0, .., n-1)$, and swap the $n-1$ element of the array with the element at the index position of the random number, in the next iteration generate a random number between $(0, .., n-2)$ and swap the element at position $n-2$ with the element at the index position of the random number and so on. Pseudo-code of the Fisher-Yates algorithm is given below.

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

Hamming Distance. Hamming distance corresponds to the positions where two binary sequences (X, Y) differ. It is used to compare bit sequences of equal length. Hamming distance tests are performed to compare the output sequences of the randomness extractor for uniform distribution. It can be represented as below.

$$f_{HD}(X, Y) = \sum_{i=0}^{n} x_i \oplus y_i \tag{4}$$

Elliptic Curve. Elliptic Curve Cryptography (ECC) is a public-key cryptography. It is a collection of asymmetric key generation, digital signatures, encryption and decryption mechanisms. Elliptic Curve (EC) is illustrated by an equation below. The curve has two main features: horizontal symmetry and non-vertical lines on the curve can intersect the curve at no more than 3 places. Let E be an elliptic curve over a finite field F and a, b, x and y are elements on the field.

$$E : y^2 = x^3 + ax + b \tag{5}$$

NIST Test Suite. Presently, NIST is a standard state of the art randomness validation suite [32]. The NIST test suite [34] contains multiple statistical tests, designed for cryptographic purposes, that analyses a sequence for its randomness.

4 Related Work

PUFs are based on the natural variabilities that emerge from the manufacturing process, which make it impossible to create an identical device with the same circuit characteristics. These uncontrollable, device specific variations serve as a digital fingerprint to the device and can be used for various security applications such as device-identification, authentication and in encryption key generation. Many different PUF types have been designed in the last decade. The *optical* PUF consists of a transparent optical medium that is explicitly added to the device during manufacturing. The *coating* PUF is an explicit PUF based on a coating layer added on the chip. In case of a *magnetic* PUF [30] on a magnetic strip a ferromagnetic material is added, consisting of particles varying in size, shape and position. *Memory* PUFs are based on the preferred stable state of memory cells. The *threshold voltage (Vt)* PUF is based on the manufacturing variations of transistors. The *carbon nanotube* PUF exploits the manufacturing variations of the transistor. The *power distribution* PUF is based on the unique characteristics of power transfer lines in the power distribution grid in a circuit. The *acoustical* PUF uses acoustical delay lines of a circuit to characterize a system. The *super high information content (SHIC)* PUF uses nano-diodes in a matrix configuration, where each diode has an unique output. A *board* PUF is an explicit PUF consisting of a layer of capacitors implemented on a printed circuit board (PCB). A *delay based* PUF is an implicit PUF based on variations in delay of two identical paths in the chip circuit. In the *arbiter* PUF a comparator determines which path is the fastest and accordingly outputs a '0'

or a '1' as PUF response. The *clock* PUF is very similar to the *arbiter* PUF, as it determines the fastest path in the clock network of the circuit. A *Ring-Oscillator (RO)* PUF measures the delay of two identical circuit paths in a different manner as it is based on an oscillating frequency. A *Radio Frequency (RF)* based PUF uses the characteristics of a radio frequency wave to identify a system. The *sensor* PUF uses a sensor or a combination of multiple sensors to produce the PUF output. In Table 1 an overview of the comparison is given. As can be seen from Table 1 most PUFs are explicit and have extrinsic evaluation. This means that for most of these PUFs additional manufacturing steps are needed, which costs valuable time and money. In this research our aim is to propose PUFs mechanisms that are suitable for constrained devices and decentralized application without requiring dedicated hardware or architecture.

Table 1. PUFs comparison table. Implicit PUFs are inherent to the device, explicit PUFs need manufacturing variations explicitly added to the device. Extrinsic evaluation means the output is evaluated outside the PUF device, intrinsic evaluation happens on the PUF device. The Fractional Hamming Distance (FHD) inter is the similarity between the output from two different PUFs to the same input. The FHD intra is the similarity between the output from one input given to the same PUF twice. In modeling attacks, the PUF can be cloned when input-output pairs are known.

PUF	Reference	Parameter	Implicity	Evaluation	FHD inter (%)	FHD intra (%)	Tamper evident	Modeling attack
Optical	[7,8]	Light intensity	Explicit	Extrinsic	49.79	25.25	Yes	Not possible
Phosphor	[9,10]	UV light intensity	Explicit	Extrinsic	?	?	Yes	?
Coating	[7,11]	Capacitance	Explicit	Extrinsic	$\sim$50	$<$5	Yes	Possible
Magnetic	[10,12]	Magnetic field	Implicit	Extrinsic	?	?	Yes	?
SRAM	[7,13]	Transistor power-up state	Implicit	Intrinsic	49.97	3.57	?	Possible
Threshold Voltage	[10,14]	Transistor voltage	Implicit	Intrinsic	50	1.30	?	?
Carbon nanotube	[10,15]	Transistor current	Explicit	Extrinsic	49.67	1.90	?	Not possible
Power distribution	[7,16]	Resistance	Explicit	Extrinsic	?	?	Yes	?
Acoustical	[7,17]	Frequency spectrum	Implicit	Extrinsic	?	?	Yes	Possible
SHIC	[10,18]	Voltage/ current	Explicit	Extrinsic	?	?	?	Not possible
Board	[10,19]	Capacitance	Explicit	Extrinsic	47.21	3.63	Yes	Not possible
Arbiter	[7,20]	Signal delays	Implicit	Extrinsic	23	5	?	Possible
Clock	[10,21]	Clock signal	Implicit	Extrinsic	50.30	5.07	Yes	?
Ring-oscillator	[7,22]	Frequency	Implicit	Extrinsic	46	0.48	?	Possible
Radio frequency	[7,23]	Radio frequency scattering	Explicit	Extrinsic	?	?	Yes	?
MEMS	[10,24]	Accelerometer values	Explicit	Extrinsic	42.64	92.17	?	?
Sensor PUF	[25]	Characteristics photo diodes	Explicit	Extrinsic	?	?	No	?
Sensor PUF	[26]	Accelerometer values	Implicit	Extrinsic	?	?	No	?

5 Randomness Extraction

Sensor data in the raw format is mostly biased, correlated and not random enough to be used for key derivation. Therefore, randomness extraction mechanisms are used to transform weakly random (raw) sensor data into uniformly distributed sequence. We propose various algorithms aiming to extract uniformly distributed random seed from sensor-PUF data. As described earlier, the methods to obtain random sequences are designed for constrained devices. To randomize the sensor data, we have come up with four algorithms (Algorithm 1, 2, 3, 4).

Algorithm 0. Raw sensor data and combinations of various sensor data is tested for randomness without applying any randomness extraction mechanism.

Algorithm 1. This algorithm aims to randomize the sensor data by multiplying it with a set of three decimal digits of the constants e or π. For both constant up to trillion digits are known, therefore this algorithm does not have reuse the digits in a considerable time. Even if the algorithm would randomize a data stream of one million data points, the constants e or π would last at least, without reusing digits, 10 million and 16 million times, respectively. Five million digits of both constants e or π are loaded and for both constants their decimals are grouped per three decimals. Next, the data is multiplied with the constant's decimal values, from either e or π. Each data point is multiplied with one group of three digits. For example, multiplication of Acc_x, Acc_y, Acc_z with constant e is as follows: $Acc_{x1} \times e_{1-3}$, $Acc_{y1} \times e_{4-6}$, $Acc_{z1} \times e_{7-9}$, $Acc_{x2} \times e_{10-12}$, $Acc_{y2} \times e_{13-15}$, $Acc_{z2} \times e_{16-18}$, etc. After the multiplication the absolute value of the result is taken and the result is converted to binary and tested with the NIST test suite. The pseudo-code is shown in Algorithm 1.

Algorithm 1: Random Sequence generation by multiplication with π

```
Input: SensorData
Output: RandomSeed
for i in range(3000, length(SensorData)-3000) do
    ProcessedData ← SensorData
PiDecimals ← DecimalsPi
for i in range(length(PiDecimals) - 3) do
    GroupedPiDecimals ← ((PiDecimals(i) × 100) + (PiDecimals(i + 1) × 10) +
    PiDecimals(i + 2))
    i = i + 3
for i in range(length(ProcessedData)) do
    ProcessedData.YPR ← ExtractDecimals(ProcessedData.YPR)
    ProcessedData.Heading ← ExtractDecimals(ProcessedData.Heading)
for i in range(length(ProcessedData)) do
    MultipliedData ← ProcessedData(i) x GroupedPiDecimals(i)
for i in range(length(MultipliedData)) do
    AbsMultipliedData ← abs(MultipliedData)
RandomSeed ← AbsMultipliedData
```

Algorithm 2. In this algorithm, a bitwise XOR operation on various combinations of data samples is performed. Each data point is converted to a binary value to perform the bitwise XOR operation. Next, the XOR operation takes place in various combinations. In Table 3 the different XOR combinations are given. The bitwise XOR has been performed in the denoted order in the column combinations. For instance, $Acc_x \oplus Gyro_y \oplus Mag_z$ means that first the Acc_x data is XORed with the $Gyro_y$ data and the result is XORed with Mag_z. The pseudo-code is shown in Algorithm 2.

Algorithm 2: Random Sequence generation with XORing of sensor data

Input: SensorData
Output: RandomSeed
for *i in range(3000, length(SensorData)-3000)* **do**
  ProcessedData ← SensorData
for *i in range(length(ProcessedData))* **do**
  AbsDataToProcess ← abs(ProcessedData)
for *i in range(length(ProcessedData))* **do**
  ProcessedData.Yaw ← ExtractDecimal(ProcessedData.Yaw)
  ProcessedData.Pitch ← ExtractDecimal(ProcessedData.Pitch)
  ProcessedData.Roll ← ExtractDecimal(ProcessedData.Roll)
  ProcessedData.Heading ← ExtractDecimal(ProcessedData.Heading)
for *i in range(length(AbsDataToProcess))* **do**
  BinaryData ← Convert2Binary(AbsDataToProcess)
for *i in range(length(BinaryData))* **do**
  $Result_1 \leftarrow$ XOR($BinaryData_1 \oplus BinaryData_2$)
  $Result_2 \leftarrow$ XOR($Result_1 \oplus BinaryData_3$)
RandomSeed ← $Result_2$

Algorithm 3. The third algorithm is an extension of Algorithm 2. First, the Fast Fourier Transform (FFT) is applied to the data, after which the XOR operation from Algorithm 2 is performed. The FFT of each data stream is calculated separately. This means that no sensors are combined and that the multiple data streams (x, y, z) from one sensor are kept separated as well. The FFT is calculated on the data as one sequence. The result of a FFT consists of a real and an imaginary part. In this algorithm, only the real part is used and the imaginary part is discarded. Besides this, only the decimal number of the real part is used and the integer part is discarded as well. The next step is to convert every data stream to binary values, to be able to perform the bitwise logical XOR operation. Similar to Algorithm 2, the XOR operation is applied to various data combinations as shown in Table 3. The pseudo-code is shown in Algorithm 3.

Algorithm 4. Among all the algorithms mentioned previously, Algorithm 4 is an optimal and secure randomness extraction mechanism. This algorithm is an extension of Algorithm 3. In this algorithm, shuffling is applied to the output data stream of Algorithm 3. Fisher-Yates algorithm is used to perform shuffling and extract random permutations of various bits sizes (i.e. 100000, 512000). Fisher-Yates algorithm is being used since it is unbiased (every permutation is equally likely), linear in time and is fast. The pseudo-code is shown in Algorithm 4.

Algorithm 3: Random Sequence generation with XORing of FFT processed sensor data

Input: SensorData
Output: RandomSeed
for *i in range(3000, length(SensorData)-3000)* **do**
  ProcessedData ← SensorData
for *i in range(length(ProcessedData))* **do**
  AbsDataToProcess ← abs(ProcessedData)
for *i in range(length(ProcessedData))* **do**
  ProcessedData.YPR ← ExtractDecimal(ProcessedData.YPR)
  ProcessedData.Heading ← ExtractDecimal(ProcessedData.Heading)
for *i in range(length(ProcessedData))* **do**
  FFTDataRealImj ← FFT(ProcessedData)
  FFTDataReal ← abs(real(FFTDataRealImj))
  FFTData ← ExtractDecimal(FFTDataReal)
for *i in range(length(FFTData))* **do**
  FFTBinary ← Convert2Binary(FFTData)
for *i in range(length(FFTBinary))* **do**
  $FFTXOR_1 \leftarrow XOR(FFTBinary_1 \oplus FFTBinary_2)$
  $FFTXOR_f \leftarrow XOR\ (FFTXOR_1 \oplus FFTBinary_3\ )$
RandomSeed ← $FFTXOR_f$

Algorithm 4: Extraction of Random Seed

Input: SensorData
Output: RandomSeed
for *i in range(3000, length(SensorData)-3000)* **do**
  ProcessedData ← SensorData
for *i in range(length(ProcessedData))* **do**
  FFTDataRealImj ← FFT(ProcessedData)
  FFTDataReal ← abs(real(FFTDataRealImj))
  FFTData ← ExtractDecimal(FFTDataReal)
for *i in range(length(FFTData))* **do**
  FFTBinary ← Convert2Binary(FFTData)
for *i in range(length(FFTBinary))* **do**
  $FFTXOR_1 \leftarrow XOR(FFTBinary_1 \oplus FFTBinary_2)$
  $FFTXOR_f \leftarrow XOR\ (FFTXOR_1 \oplus FFTBinary_3\ )$
DataSize ← length ($FFTXOR_f$)
RandomPerm ← FisherYates(DataSize, 100000)
for *i in range(length(RandomPerm))* **do**
  DataPoint ← RandomPerm (i)
  RandomSeed ← $FFTXOR_f$(DataPoint)

6 Sensor-Based PUF Data Acquisition

This section aims to construct and implement a method with which datasets of sensor data are obtained. The sensing platform is integrated into pallets, and contain several

types of sensors, among which an IMU. We have used a 9DoF Inertial Measurement Unit (IMU) as shown in Fig. 1 to gather the movement data. It uses an accelerometer, gyroscope and magnetometer to determine with an on-board processor the linear and angular motion of the object it is attached to. It also uses the on-board processor to calculate the quaternions, yaw, pitch, roll and heading of the device. In this research, we have used both individual sensor data and combining multiple sensors data, namely: accelerometer $(x, y, z)\, axis$, gyroscope $(x, y, z)\, axis$, magnetometer $(x, y, z)\, axis$, quaternions $(w, x, y, z)\, axis$, yaw, pitch, roll and heading. The purpose behind combining multiple sensors data (i.e. accelerometer + gyroscope + magnetometer) is to analyse the impact of combination on randomness. The data is obtained by driving in a car, in trips ranging from 50 KM to 150 KM. The dataset contains subsets from different car trips, each with the same IMU configurations and positioning of the IMU in the car console. The data is sampled at 100 Hz. The first and last 30 s of data of each trip are removed. This is because during this time the car is assumed to be stationary, making it very unlikely random data would be created by the sensors. At a sampling rate of 100 Hz, 30 s of data amounts to 3000 samples. The absolute value of all data is taken. For some analyses, the data on the different axes of sensors with multiple axes are combined in one data stream per axis. For example, the accelerometer generates data on 3 axes;$(x, y, z)\, axis$ are combined in one accelerometer data stream. Before combining the yaw, pitch roll (YPR) data into one stream, the decimal values are extracted, to be used for the tests. The integer values are discarded, since visual inspection showed that these are not random values. Also, the decimal values of the heading data are used. Furthermore, for some analyses, the data from multiple sensors are combined into a single stream per axis (i.e. $Acc_{x1} + Gyro_{x1} + Mag_{x1}$).

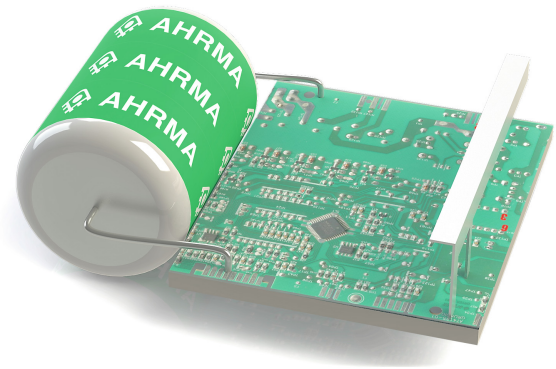

Fig. 1. Inertial Measurement Unit (IMU) is used to gather the movement data.

7 Results and Discussion

This section discusses the results of the randomness tests performed with the NIST test suite. For the NIST tests, the standard configuration is used, meaning all statistical tests are run. The data is input as ASCIIâĂŹs $0s$ and $1s$ and is tested in one sequence. The

algorithms are tested on various data sets and varying data points. The size of input bits tested with NIST are ranging from 130000 bits till 1100000. The average size of input bits is approximately 800000. The results of the tests are consistent.

Results of Algorithm 0. The results of Algorithm 0 is shown in Table 2. None of the tested data from sensors and combinations of sensors are random. All tested configurations failed almost all the NIST tests. Therefore we conclude that the gathered data in the its raw form is not random and thus as such not suitable for cryptographic usage.

Results of Algorithm 1. Table 2 shows that Algorithm 1 is not successful in randomizing the data. All tested sequences are not random and failed most of the tests. This is the case for both multiplication by e or π, as well as multiplication the full dataset or only half the dataset. For seven out of eighteen tests the 'rank' and 'linear complexity' tests passed, for both e or π, with the Yaw, Pitch, Roll (decimals) combination for π as exception. This combination did not pass the 'rank' test. The results show that even less tests are passed after manipulation by Algorithm 2, compared to Algorithm 0. Based on our analyses, multiplying sensor data by some constant does not make it random. For Algorithm 1, randomness results almost remained the same when multiplied by e or π. The results shown in the Table 2 is for e.

Results of Algorithm 2. Table 3 shows that results of Algorithm 2 are improved significantly compared to Algorithm 1. For Algorithm 2, based on XORing datapoints, most individual tests are passed and with that most tested sequences (combinations) are determined random. For some sequences the 'random excursions' and 'random excursions variants' tests are executed. In all cases this test is performed, the test passed.

Results of Algorithm 3. For the tests of Algorithm 3 the same sequences are used as for Algorithm 2, this time only the mantissa of Yaw, Pitch & Roll and Heading are used. The algorithm is tested both on the full dataset and half of the dataset. A large percentage of the tested sequences turned out to be random and passed the tests. When the fourier transform is performed on small data points, the sequence Mag_x, Mag_y, Mag_z and the sequence Yaw, Pitch, Roll, Heading are three out of four times not random. For all the sequences where all data points are used to calculate the fourier transform, the results are random. Table 3 shows the results of Algorithm 3.

Results of Algorithm 4. Table 4 NIST results of Algorithm 4 for a random permutation of bits size 100000. The results illustrate that Algorithm 4 has effectively passed all the NIST tests. Which shows that Algorithm 4 is capable of extracting secure and random seed. Furthermore, from a random sequence of bits size 500000 generated by Algorithm 4, 20 different sequences of bits sizes 5000 are extracted and pairwise hamming distance is calculated. Figure 2 shows the pairwise comparison of the 20 different permutations. The results shows that each sequence is completely different from the other sequences. Which clearly demonstrate that the sequences are uniformly distributed; the probability of occurrence of each sequence is almost equal. Algorithm 4 NIST results are compared with SRAM PUFs [32] for the same NIST settings and size of input bits

(512000). Table 5 shows comparison of Algorithm 4 with SRAM PUFs [32]. The results demonstrate the dominance of Algorithm 4 in randomization by passing all NIST tests and assures its feasibility for cryptographic applications.

Table 2. Results of randomness for Algorithm 0 and 1. The percentages depict percentage of sub tests passed. ✓ represents the NIST test is passed, while ✗ represents the NIST test is failed. Blue color depicts Algorithm 0 and green color depicts Algorithm 1. For Algorithm 1, combination of various sensor data is not performed. A – means either the test is not performed or NIST test suite gives no result. On average, the number of input bits to NIST test suite are 800000.

Data stream	Frequency	Block Frequency	Cumulative Sums	Runs	Longest Run	Rank	FFT	Non Overlapping	Overlapping	Universal	Approx Entropy	Random Excursions	Rand Excu Variant	Serial	Linear Complexity
Acc	✗✗	✗✗	✗✗	✗✗	✗✗	✓✗	✗✗	4% 0%	✗✗	✗✗	✗✗	– –	– –	✗✗	✓✗
Gyro	✗✗	✗✗	✗✗	✗✗	✗✗	✓✓	✗✗	5% 5%	✗✗	✗✗	✗✗	– –	– –	✗✗	✓✓
Mag	✗–	✗–	✗–	✗–	✗–	✓–	✗–	0% –	✗–	✗–	✗–	– –	– –	✗–	✓–
Quaternions	✗✗	✗✗	✗✗	✗✗	✗✗	✓✗	✗✗	6% 0%	✗✗	✗✗	✗✗	– –	– –	✗✗	✓✗
YPR	✗✗	✗✗	✗✗	✗✗	✗✗	✗✓	✗✗	0% 13%	✗✗	✗✗	✗✗	– –	– –	✗✗	✓✓
Heading	✗✗	✗✗	✗✗	✗✗	✗✗	✓✗	✗✗	7% 0%	✗✗	✗✗	✗✗	– –	– –	✗✗	✓✗
Acc+Gyro+Mag	✗–	✗–	✗–	✗–	✗–	✓–	✗–	1% –	✗–	✗–	✗–	– –	– –	✗–	✓–
Gyro+Mag	✗–	✗–	✗–	✗–	✗–	✓–	✗–	0% –	✗–	✗–	✗–	– –	– –	✗–	✓–
Mag+Quaternions	✗–	✗–	✗–	✗✗	✗✗	✓–	✗–	2% –	✗–	✗–	✗–	– –	– –	✗–	✓ –

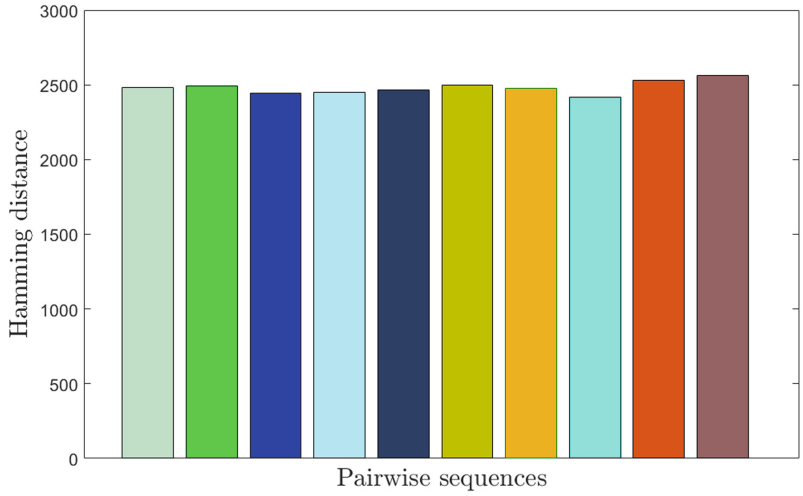

Fig. 2. Pairwise hamming distance between 20 sequences of bits size 5000 each.

8 Complexity

The unique features of the proposed randomness extraction algorithms are that we have used very simple and faster operations (functions) in order to be suitable for decentralized and implicit cryptographic schemes. For the Fisher-Yates algorithms, the time complexity is $O(N)$ where N is the size of the input array (sequence). For FFT, the time complexity is $O(NlogN)$ where N is the data size. The time complexity for XOR operation is $O(N)$, where N is the size of the binary sequences.

9 Pseudo-random Number Generator

Extracting secure seed from sensor-based PUF data might not always be desirable or feasible, so alternatively, we can use a pseudo-random number generator (PRNG) to generate long runs of pseudo-random numbers from the seed. PRNG feeds the seed into a deterministic algorithm to generate pseudo-random sequences in short time and the sequences are statistically pretty close to random. A standard PRNG has three characteristics: deterministic, efficient and periodic. The output of PRNG should be identical to uniformly distributed random variables [33]. Furthermore, PRNG generate uncorrelated sequences and has long period before repeating the cycle. We use an exiting ECC based random number generator [35] to generate pseudo-random numbers and the aim is to demonstrate the use case of the extracted sensor-based PUF seed for key derivation purposes. In comparison with other public key cryptography (RSA), ECC requires smaller key size but proffer similar security. A 256-bit ECC key provides approximately same security as 3072-bit RSA key [36]. Thus it is computationally efficient. Furthermore, ECC is scalable thus suitable for distributed IoT applications. Besides that, ECC is used in Bitcoin to generate public and private keys. The employed ECC mechanism [35] is premised on the addition of points on an EC over finite field. ECC requires random numbers to generate random curves [35] and secret parameters. We propose to use the extracted random sequences from the sensor-based PUF as seed to generate random curves and EC secret parameters. EC based pseudo-random generator as proposed in [35], can be integrated in a cryptographic system, is shown in Fig. 3 and it works as follow. Primarily, a finite field F, an elliptic curve E, a point on the curve P, and a seed k_1 are selected. The size of k_1 depends on the size of the finite field F. We presume that the seed k_1 and the initial point on the curve P can be generated from the our extracted sensor-based PUF random sequences. k_1 which is a seed in the first cycle is input into the k_nP module. The module performs the multiplication between k_n which is an integer and P which is point on the curve and subsequently generates a k_nP point. A sequence of pseudo-random bits x_n are obtained. The new seed k_{n+1} is formed by adding the x-coordinate of the point and the cycle number. As the authors [35] claim, the mechanism can be implemented without an additional hardware or software component which makes it suitable for constrained IoT devices. Furthermore, the statistical properties, period analysis, results and possible structural variations in the block diagram of EC based pseudo-random number generator are available at [35].

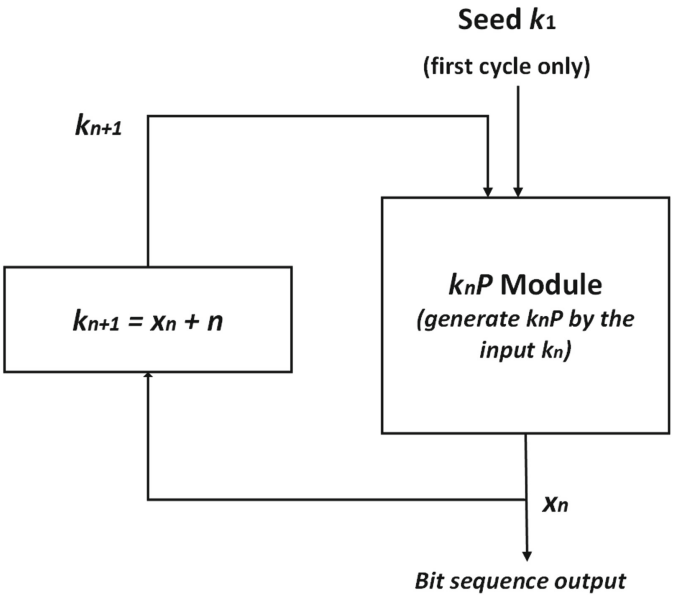

Fig. 3. Block diagram of ECC based pseudo-random number generator [35].

Table 3. Results of randomness for Algorithm 2 and 3. The percentages depict percentage of sub tests passed. ✓ represents the NIST test is passed, while ✗ represents the NIST test is failed. Blue color depicts Algorithm 2 and green color depicts Algorithm 3. A – means either the test is not performed or NIST test suite gives no result. On average, the number of input bits to NIST test suite are 800000.

Data stream	Frequency	Block Frequency	Cumulative Sums	Runs	Longest Run	Rank	FFT	Non Overlapping	Overlapping	Universal	Approx Entropy	Random Excursions	Rand Excu Variant	Serial	Linear Complexity
$Acc_x \oplus Gyro_y \oplus Mag_z$	✓✓	✓✓	✓✓	✗✓	✓✓	✓✓	✓✓	56% 97%	✗✓	✓✓	✗✓	– 100%	– 100%	50%✓	✓✓
$Q_w \oplus Q_x \oplus Q_y \oplus Q_z$	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓	99% 99%	✓✓	✓✓	✓✓	– 100%	– 100%	✓✓	✓✓
Yaw $\oplus$ Pitch $\oplus$ Roll	✓–	✓–	✓–	✗–	✗–	✓–	✓–	64% –	✓–	✓–	✗–	100% –	100% –	✗–	✓–
Yaw $\oplus$ Pitch $\oplus$ Roll $\oplus$ Heading	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓	100% 100%	✓✓	✓✓	✓✗	100% –	100% –	✓✓	✓✓
$Q_w \oplus$ Yaw $\oplus$ Heading	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓	97% 100%	✓✓	✓✓	✓✓	– 100%	– 100%	✓✓	✓✓
$Q_w \oplus$ Yaw $\oplus$ Heading $\oplus$ $Acc_x \oplus Gyro_y \oplus Mag_z$	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓	✓✓	99% 99%	✓✓	✓✓	✓✓	88% 100%	100% 100%	✓✓	✓✓
$Acc_x \oplus Acc_y \oplus Acc_z$	✗✓	✗✓	✗✓	✗✓	✗✓	✓✓	✗✓	25% 100%	✗✓	✗✓	✗✓	– 100%	– 100%	✗✓	✓✓
$Gyro_x \oplus Gyro_y \oplus Gyro_z$	✓✓	✓✓	✓✓	✓✗	✓✗	✓✓	✓✓	100% 95%	✗✗	✓✓	✓✗	100% 100%	100% 100%	✓✓	✓✓
$Mag_x \oplus Mag_y \oplus Mag_z$	✓✓	✓✗	✓✓	✗✗	✓✗	✓✓	✓✓	51% 99%	✓✓	✗✓	✗✗	100% 88%	100% 100%	50%✗	✓✓

Table 4. Results of randomness for Algorithm 4. The number of input bits to NIST test suite are 100000. The percentages depict percentage of sub tests passed. ✓ represents the NIST test is passed, while ✗ represents the NIST test is failed. A – means either the test is not performed or NIST test suite gives no result because the test is not applicable, since there are an insufficient number of cycles.

Data stream	Frequency	Block Frequency	Cumulative Sums	Runs	Longest Run	Rank	FFT	Non Overlapping	Overlapping	Universal	Approx Entropy	Random Excursions	Rand Excu Variant	Serial	Linear Complexity
$Acc_x \oplus Gyro_y \oplus Mag_z$	✓	✓	✓	✓	✓	✓	✓	100%	✓	–	✓	87.5%	100%	✓	✓
$Q_w \oplus Q_x \oplus Q_y \oplus Q_z$	✓	✓	✓	✓	✓	✓	✓	100%	✓	–	✓	–	–	✓	✓
Yaw⊕ Pitch⊕ Roll⊕ Heading	✓	✓	✓	✓	✓	✓	✓	100%	✓	–	✓	–	–	✓	✓
$Q_w \oplus$ Yaw $\oplus$ Heading	✓	✓	✓	✓	✓	✓	✓	100%	✓	–	–	–	–	✓	✓
$Q_w \oplus$ Yaw $\oplus$ Heading $\oplus$ $Acc_x \oplus Gyro_y \oplus Mag_z$	✓	✓	✓	✓	✓	✓	✓	100%	✓	–	✓	–	–	✓	✓
$Acc_x \oplus Acc_y \oplus Acc_z$	✓	✓	✓	✓	✓	✓	✓	98.6%	✓	–	✓	–	–	✓	✓
$Gyro_x \oplus Gyro_y \oplus Gyro_z$	✓	✓	✓	✓	✓	✓	✓	100%	✓	–	✓	–	–	✓	✓
$Mag_x \oplus Mag_y \oplus Mag_z$	✓	✓	✓	✓	✓	✓	✓	98.6%	✓	–	✓	100%	100%	✓	✓

Table 5. NIST test results of Algorithm 4 are compared with True random number generator based on SRAM PUFs [32]. The results of both the mechanisms are based on same NIST settings and input bits size of 512000.

Test	SRAM PUFs [32]	Our proposed Algorithm 4
Frequency	✓	✓
Cumulative sum	✓	✓
Runs	✓	✓
FFT	✓	✓
Longest run	✓	✓
Block frequency	✓	✓
Approximate entropy	✓	✓
Rank	✓	✓
Serial	✓	✓
Universal	✓	✓
Random excursions	n.a.	✓
Random exc. variant	n.a.	✓
Linear complexity	n.a.	✓
Overlapping template	n.a.	✓
Non-overlap. temp.	n.a.	98.6%

10 Conclusion

Secure access and sharing of data is very important in smart logistics. Random number generators play an important role in security mechanisms. However, generating random numbers is not necessarily straight forward or easy, especially for IoT devices and in IoT circumstances generating random numbers is more challenging. A sensor-based PUF that utilizes the sensors on an IoT device is a potential solution, as physical variations among the sensors could provide a good source of randomness. We proposed various lightweight randomness extraction mechanisms while taking into account the limited power, storage and computational resources of the IoT devices in smart logistics. Based on our analysis, sensor data in the raw form is not random, multiplying sensor data by some constant does not make the data random, XOR operation can somehow improve the randomness, and incorporation of XOR, FFT and random permutation significantly improve randomness and security of the seed. We can conclude that sensor data if processed accordingly can adequately be used to extract cryptographically secure random numbers.

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

References

1. Gartner Glossary. www.gartner.com/en/information-technology/glossary/internet-of-things. Accessed 11 Mar 2021
2. Hajjaji, Y., Boulila, W., Farah, I.R., Romdhani, I., Hussain, A.: Big data and IoT-based applications in smart environments: a systematic review. Comput. Sci. Rev. **39**, 100318 (2021). https://doi.org/10.1016/j.cosrev.2020.100318. ISSN 1574-0137
3. Eason, G., Noble, B., Sneddon, I.N.: On certain integrals of Lipschitz-Hankel type involving products of Bessel functions. Phil. Trans. Roy. Soc. London **A247**, 529–551 (1955)
4. Lee, S., Kang, Y., Prabhu, V.V.: Smart logistics: distributed control of green crowdsourced parcel services. Int. J. Prod. Res. **54**(23), 6956–6968 (2016)
5. UNECE. Terminology on combined transport. United Nations Economic Commission for Europe, New York and Geneva (2001)
6. Ullah, I., Meratnia, N., Havinga, P.: iMAC: implicit message authentication code for IoT devices. In: 2020 IEEE 6th World Forum on Internet of Things (WF-IoT), New Orleans, LA, USA, pp. 1–6 (2020). https://doi.org/10.1109/WF-IoT48130.2020.9221331
7. Maes, R., Verbauwhede, I.: Physically unclonable functions: a study on the state of the art and future research directions. In: Sadeghi, A.R., Naccache, D. (eds.) Towards Hardware-Intrinsic Security. Information Security and Cryptography, pp. 3–37. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14452-3_1
8. Pappu, R., Recht, B., Taylor, J., Gershenfeld, N.: Physical one-way functions. Science **297**(5589), 2026–2030 (2002)
9. Chong, C.N., Jiang, D., Zhang, J., Guo, L.: Anti-counterfeiting with a random pattern. In: 2008 Second International Conference on Emerging Security Information, Systems and Technologies, pp. 146–153. IEEE (2008)
10. McGrath, T., Bagci, I.E., Wang, Z.M., Roedig, U., Young, R.J.: A PUF taxonomy. Appl. Phys. Rev. **6**(1), 011303 (2019)

11. Tuyls, P., Schrijen, G.-J., Škorić, B., van Geloven, J., Verhaegh, N., Wolters, R.: Read-proof hardware from protective coatings. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 369–383. Springer, Heidelberg (2006). https://doi.org/10.1007/11894063_29
12. Indeck, R.S., Muller, M.W.: Method and apparatus for fingerprinting magnetic media. US Patent 5,365,586, 15 November 1994
13. Guajardo, J., Kumar, S.S., Schrijen, G.-J., Tuyls, P.: FPGA intrinsic PUFs and their use for IP protection. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 63–80. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74735-2_5
14. Lofstrom, K., Daasch, W.R., Taylor, D.: IC identification circuit using device mismatch. In: 2000 IEEE International Solid-State Circuits Conference. Digest of Technical Papers (Cat. No. 00CH37056), pp. 372–373. IEEE (2000)
15. Konigsmark, S.T.C., Hwang, L.K., Chen, D., Wong, M.D.F.: CNPUF: a carbon nanotube-based physically unclonable function for secure low-energy hardware design. In: 2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 73–78. IEEE (2014)
16. Helinski, R., Acharyya, D., Plusquellic, J.: A physical unclonable function defined using power distribution system equivalent resistance variations. In: 2009 46th ACM/IEEE Design Automation Conference, pp. 676–681. IEEE (2009)
17. Vrijaldenhoven, S., et al.: Acoustical physical uncloneable functions. Philips Internal Publication PR-TN-2004-300300 (2005)
18. Rührmair, U., Jaeger, C., Hilgers, C., Algasinger, M., Csaba, G., Stutzmann, M.: Security applications of diodes with unique current-voltage characteristics. In: Sion, R. (ed.) FC 2010. LNCS, vol. 6052, pp. 328–335. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14577-3_26
19. Wei, L., Song, C., Liu, Y., Zhang, J., Yuan, F., Xu, Q.: BoardPUF: physical unclonable functions for printed circuit board authentication. In: Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, pp. 152–158. IEEE Press (2015)
20. Lee, J.W., Lim, D., Gassend, B., Suh, G.E., Van Dijk, M., Devadas, S.: A technique to build a secret key in integrated circuits for identification and authentication applications. In: 2004 Symposium on VLSI Circuits. Digest of Technical Papers (IEEE Cat. No. 04CH37525), pp. 176–179. IEEE (2004)
21. Yao, Y., Kim, M.B., Li, J., Markov, I.L., Koushanfar, F.: ClockPUF: physical unclonable functions based on clock networks. In: Proceedings of the Conference on Design, Automation and Test in Europe, pp. 422–427. EDA Consortium (2013)
22. Gassend, B., Clarke, D., Van Dijk, M., Devadas, S.: Silicon physical random functions. In: Proceedings of the 9th ACM Conference on Computer and Communications Security, pp. 148–160. ACM (2002)
23. DeJean, G., Kirovski, D.: RF-DNA: radio-frequency certificates of authenticity. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 346–363. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74735-2_24
24. Aysu, A., Ghalaty, N.F., Franklin, Z., Yali, M.P., Schaumont, P.: Digital fingerprints for low-cost platforms using mems sensors. In: Proceedings of the Workshop on Embedded Systems Security, p. 2. ACM (2013)
25. Rosenfeld, K., Gavas, E., Karri, R.: Sensor physical unclonable functions. In: 2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), pp. 112–117. IEEE (2010)
26. Fukushima, K., Hidano, S., Kiyomoto, S.: Sensor-based wearable PUF. In: Secrypt, pp. 207–214 (2016)
27. Ullah, I., Meratnia, N., Havinga, P.: Entropy as a service: a lightweight random number generator for decentralized IoT applications. In: 2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), Austin, TX, USA, pp. 1–6 (2020). https://doi.org/10.1109/PerComWorkshops48775.2020.9156205

28. Johnston, A.M.: Comments on cryptographic entropy measurement. Juniper Networks. amj@juniper.net, 30 October 2019. https://eprint.iacr.org/2019/1263.pdf
29. Grassi, P.A., Garcia, M.E., Fenton, J.L.: NIST special publication 800–63-3. Digital Identity Guidelines. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-63-3.pdf
30. Simion, E.: Entropy and randomness: from analogic to quantum world. IEEE Access **8**, 74553–74561 (2020). https://doi.org/10.1109/ACCESS.2020.2988658
31. Fouque, P.-A., Pointcheval, D., Zimmer, S.: HMAC is a randomness extractor and applications to TLS. In: Proceedings of the 2008 ACM symposium on Information, computer and communications security (ASIACCS 2008), pp. 21–32. Association for Computing Machinery, New York (2008). https://doi.org/10.1145/1368310.1368317
32. Leest, V.V.D., Sluis, E.V.D., Schrijen, G.J., Tuyls, P., Handschuh, H.: Efficient implementation of true random number generator based on SRAM PUFs. Intrinsic-ID, Eindhoven (2012). https://www.intrinsic-id.com/wp-content/uploads/2017/05/True-Random-Number.pdf
33. Röck, A.: Pseudorandom number generators for cryptographic applications. Diplomarbeit zur Erlangung des Magistergrades an der Naturwissenschaftlichen Fakultat der Paris-Lodron-Universitat Salzburg Salzburg, March 2005
34. Rukhin, A.L., et al.: NIST Special Publication 800-22: a statistical test suite for random and pseudorandom number generators for cryptographic applications. NIST, April 2010
35. Lee, L.-P., Wong, K.-W.: A random number generator based on elliptic curve operations. Comput. Math. Appl. **47**(2–3), 217–226 (2004). https://doi.org/10.1016/S0898-1221(04)90018-1. ISSN 0898-1221
36. Suárez-Albela, M., Fernández-Caramés, T.M., Fraga-Lamas, P., Castedo, L.: A practical performance comparison of ECC and RSA for resource-constrained IoT devices. In: 2018 Global Internet of Things Summit (GIoTS), Bilbao, Spain, pp.1–6 (2018). https://doi.org/10.1109/GIOTS.2018.8534575