IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 6, 2020, accepted January 30, 2020, date of publication February 10, 2020, date of current version February 18, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2972958

Random Number Generator Using Sensors for Drone

SEONG-MIN CHO[1], (Student Member, IEEE), EUNGI HONG[1], (Student Member, IEEE), AND SEUNG-HYUN SEO[2], (Member, IEEE)

[1]Department of Electrical Engineering, Graduate School, Hanyang University, Seoul 15588, South Korea
[2]Division of Electrical Engineering, Hanyang University ERICA, Ansan 15588, South Korea

Corresponding author: Seung-Hyun Seo (seosh77@hanyang.ac.kr)

This work was supported by the Research Fund of Hanyang University under Grant HY-2018-N.

ABSTRACT In recent years, drones have been widely used in many areas such as farming, movie making, surveillance and delivery. So, there is a need to protect these drones against security attacks including hijacking, spying and theft of stored data through the utilization of security mechanisms. Cryptographic keys are needed to operate these security mechanisms, and they must be generated by using random number generators which create unpredictable and non-regenerable random numbers. However, existing random number generators used in drones are not tailored for drones specifically as they use random sources generated on a desktop, not a drone. Recently, random number generators utilizing sensors in mobile phones and IoT (Internet of Things) devices have been studied, but are not appropriate for drones. In this paper, considering that drone sensors must be applied to flight and stationary modes, we proposed a drone specific random number generator called DroneRNG and implemented it. Then, we showed that our DroneRNG passed all of the NIST randomization tests and possesses better statistical properties and unpredictability than random number generators that are currently used in drones.

INDEX TERMS Drone, random number generator, sensors.

I. INTRODUCTION

Drones, which are also known as Unmanned Aerial Vehicles (UAVs), have emerged as an issue in the fourth industrial revolution, and in addition, many industries are using them in various ways [1]. For example, drones are no longer just used for shooting video in the sky but can also perform topographical analysis or rescue activities using filmed images. They also can be used for delivery services, climate forecast through sensing, and more.

Yet, at the same time there have been many security breaches against drones, such as the hacking of data stored in drones, eavesdropping on and manipulating communicated data, or injecting falsified sensor data into the drone [2]. So, it is imperative that we utilize some security mechanism for secure drone services. The security mechanism is built on cryptographic algorithms such as data encryption for protecting collected data, and digital signature schemes for device authentication in drones. The cryptographic key is essential for the cryptographic algorithm to work, and the security of the encrypted data with the key only relies on the secrecy of the key. Thus, the cryptographic key must be unknown to and unpredictable by an adversary [3]. These requirements for cryptographic keys may be satisfied by using good random numbers, which have unpredictable and non-regenerable properties, in the key.

For cryptographic purposes, the random number generator used for generating cryptographic keys should guarantee the output's good statistical properties and unpredictability. To achieve these properties, the random number generator requires non-deterministic random sources with high quality digital post-processing. Hence, the random number generator must utilize physical random sources, such as decay time of radioactive materials and electronic noise from semiconductors or registers, which are unpredictable and non-deterministic [4]. If physical random sources are not used, the cryptographic keys generated from that random number still can be easily predicted. So, in order to generate good random numbers, a source with physical randomness is very crucial. It is also needed to improve the statistical properties of digital post-processing, such as pseudo random number generators using cryptographic hash function. This is because

The associate editor coordinating the review of this manuscript and approving it for publication was Qiquan Qiao.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see http://creativecommons.org/licenses/by/4.0/

the random sources may still have insufficient randomness in statistical terms [5].

Currently, drones use open source cryptographic libraries such as OpenSSL or standard C random function to generate random numbers. The open source cryptographic libraries collect random sources from user input resources available on PCs, such as user/external peripherals like a desktop PC's mouse or keyboard, interrupt request time, and disk reading and writing time [6]. However, it is not suitable for drones because the drones have no such peripherals and some drones have no operating systems like Linux, which manage the interrupt request time and disk reading and writing time. So, the existing random number generators, that use physical random sources on PCs, cannot guarantee the output's unpredictability in drones. This is also applicable to mobile phones and IoT (Internet of Things) devices for the same reasons.

Therefore, to generate random numbers for drones, mobile phones, and IoT devices, it is necessary to utilize physical random sources available on these devices. Recently, many studies have worked on random number generators using sensors on IoT devices. According to the previous studies [7]–[13], sensors on smartphones or on-body IoT devices such as accelerometers, gyroscopes, and cameras can be used as physical random sources.

But the characteristics of the drone's sensor output differ when it is stationary and when it is flying. There is no motion when it remains stationary on the ground but it continues to move and vibrate while in flight.

For this reason, we cannot directly apply the previous random number generators using IoT sensors for use with drones. Accordingly, there is a need to design a random number generator which takes into account the characteristics of the drone.

In this paper, we propose a drone-specific random number generator, which considers the sensor characteristics that drones present in flight and in a stationary state, using accelerometer, gyroscope, and barometer signals captured by drones while flying and on the ground. The contributions of this paper are as follows:

- We evaluated the randomness of the output of the existing random number generators (RNGs) which are currently used for drones. In our experiments, we found that the output of the current drone RNG currently used in drones is not unpredictable enough to meet the statistical properties of a good random number.
- We differentiated the sensor characteristics between flying and stopped drones and assessed the randomness of these sensor values. Based on our analysis, we proposed the DroneRNG (Drone Random Number Generator) designed for drones, which collects physical random sources from drone sensors, and then post-processes the random sources for statistical randomness.
- We implemented the DroneRNG and provided it through Github.[1] We used another thread to collect entropy and to fill a random buffer to make it possible to generate random numbers and fill the random buffer at the same time. The random numbers generated from DroneRNG passed all NIST randomization tests. By comparing the DroneRNG with existing RNGs currently used in drones and IoT devices, we demonstrated that the output of DroneRNG has better statistical properties and more unpredictability.

[1] https://github.com/SeongminCho/DroneRNG

The rest of this paper is organized as follows: we introduce the concept of RNG and analyze RNG using IoT sensors in Section II. Section III evaluates the randomness of collected drone sensor data. And we propose DroneRNG in Section IV. Finally, Section V evaluates the performance of our DroneRNG and shows if it can pass all of the NIST randomization tests.

II. RELATED WORK

In this section, we introduce the concept of RNGs (random number generators) and analyze the current state of RNGs using IoT sensors. And we also analyze studies suggesting drone security solutions.

A. RANDOM NUMBER GENERATORS USING SENSORS

1) RANDOM NUMBER GENERATOR

The Random Number Generator (RNG) can be classified as either the True Random Number Generator (TRNG) or the Pseudo-Random Number Generator (PRNG).

A TRNG generates random numbers based on unpredictable natural phenomena and non-deterministic physical phenomena. Such phenomena include noise source signals such as decay time of radioactive materials and electronic noise from semiconductors or registers. A TRNG is designed based on hardware characteristics because of the need to measure and use noise source signals. However, the noise source signals may not be random because they vary depending on the environment. Therefore, a TRNG needs to use processing functions, such as the entropy distillation process, Fourier transform, and low (or high) pass filters, to overcome this weakness.

In comparison, a PRNG generates random numbers through deterministic algorithms, such as cryptographic hash functions or symmetric key encryption algorithms. As inputs for these algorithms, initial values, otherwise known as seeds, are required. Since the seed must have randomness and unpredictability itself, the outputs of a TRNG are used as the seeds of a PRNG. The random numbers generated by a PRNG have excellent statistical properties and may seem more random than those of a TRNG. However, in terms of security, it is important to make sure that the seed is not known in order to make the random number unpredictable. In a PRNG, the random number is reproducible from the seed, because the PRNG has a deterministic property. The general structure of the RNG consists of three steps as shown in Fig. 1.

The RNG collects entropy from the noise source signals in the first step (entropy collection step). It accumulates enough

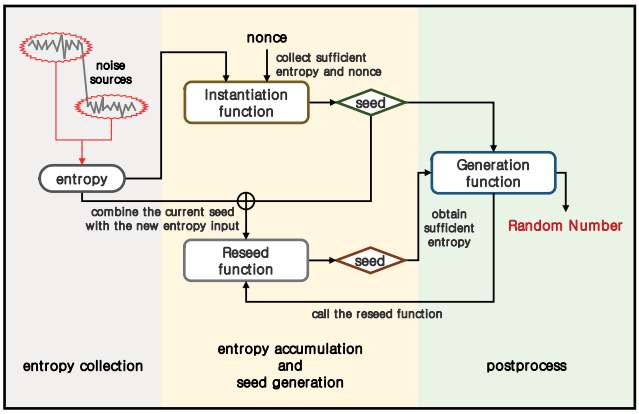


FIGURE 1. The structure of random number generator.

entropy and nonce and then produces a seed through an instantiation function in the next step (entropy accumulation and seed generation step). When a reseed function is called by a generation function, a new seed is produced by combining the seed from the instantiation function with a new entropy source collected from the first step. Finally, the generation function receives these seeds as input and generates random numbers in the last step (postprocess step).

PRNG uses all three steps and utilizes a deterministic algorithm in the third step. However, TRNG uses only the first and third steps. In the third step of a TRNG, the deterministic algorithm is replaced by a processing function.

2) RANDOM NUMBER GENERATORS USING SENSORS

Previous studies [7]–[13] used various sensors, such as accelerometers, gyroscopes, magnetometers, cameras, and microphones of smartphones or other IoT devices, to design an RNG. Among them, sensors which are available in drones include accelerometers and gyroscopes essential for flight.

Thus, in this section, we analyze the randomness of values output by accelerometers and gyroscopes, and also analyze the studies that have designed the RNG using their randomness.

An accelerometer measures acceleration through the reaction caused by inertia. It is used in a wide variety of devices, especially mobile or other wireless communication devices. A lot of studies have been conducted on how to generate random numbers using this accelerometer. In 2011, Voris *et al.* investigated the randomness of accelerometers for the TRNG [14]. They found that accelerometers can extract sufficient entropy even when stationary, and that it is resistant to various environmental changes. They also demonstrated the validity of accelerometer-based RNG in RFID tags.

In 2013, Hennebert *et al.* revealed that accelerometers provide sufficient entropy unlike temperature and barometric pressure sensors, which measure phenomena with high inertia [15]. The authors showed that the amount of min-entropy collected by accelerometers was overestimated at [14]. Loutfi *et al.* showed that resulting bits of three motion sensors, including accelerometers, are fit enough to be used as output of a TRNG [16]. They recorded new sensor values only when the sensors detected a change, so the continuous output bits of the sensors were not addressed.

Since the raw values from accelerometers have sufficient entropy, researchers have used them to study RNGs. In 2011, Suciu *et al.* proposed a TRNG that takes sensor values from hardware sensors such as accelerometers, magnetometers and orientation sensors in mobile phones and combines them to generate random numbers through XOR operations [10]. In 2014, Marghescu *et al.* proposed an RNG for Android smartphones using accelerometers [11]. They only utilized the Least Significant Bits (LSBs) of the accelerometer output to generate a random number sequence, and then applied the Von-Neuman PRNG. In 2016, Wallace *et al.* proposed a way to generate random numbers from sensors in mobile phones and IoT devices [12]. They collected data from 37 Android devices to analyze sensors, and determined that the output of the accelerometer had sufficient randomness. In addition, they designed a TRNG by combining the collected sensor values through a lightweight mixing algorithm. In 2017, Dinca *et al.* analyzed the randomness of biometric data collected from 6 smartphone sensors: accelerometer, gyroscope, and linear acceleration, gravity, rotation and sound sensors [17]. They showed that since human gait is predictable, it should not be used as a random source. However, in 2018, Sun *et al.* showed that the use of an accelerometer and a gyroscope together can generate random numbers even taking into account the predictability of human gait [13]. They re-indexed the values of the accelerometer and the gyroscope in descending order of absolute values of energy differences between a single gait cycle signal and the average signal of multiple gait cycles measured by each sensor. And then, the values of the accelerometer and the values of the gyroscope are bitwise-XOR operated. Using that method, Sun *et al.* proposed an RNG for on-body IoT devices based on time variation of signal output while walking.

A gyroscope measures or maintains directions using angular momentum principles. Like accelerometers, gyroscopes are used in a variety of devices, especially in devices that measure and calibrate posture, such as drones and virtual reality (VR) headsets. Along with accelerometers, Loutfi *et al.* showed that the gyroscope on the smartphone had enough randomness to allow all three-axis to be used as random numbers [16].

Since the gyroscope has sufficient entropy, like the accelerometer, Marghescu *et al.* proposed a PRNG using a gyroscope with the accelerometer [11]. Wallace *et al.* showed that the output of the gyroscope on Android devices has randomness [12]. In addition, they designed the TRNG by aggregating the outputs of the gyroscope as the seed(i.e. initial value), and performing folding technique and reduction function on the seed.

In Table 1, we compared RNG research using sensors in IoT or mobile devices. The table 1 shows the sensors used and identifies the kinds of RNGs and applications.

TABLE 1. Comparison of RNG researches using sensors.

Paper	Sensors used	Kinds of RNG	Randomness	Applications
[7]	camera	TRNG	0.61	Camera of smartphone
[8]	microphone	TRNG	N/A	microphone input of Desktop
[10]	accelerometer, magnetometer	TRNG	0.56	Smartphone
[12]	accelerometer, gyroscope, microphone	TRNG	0.48	Smartphone, IoT device
[13]	accelerometer, gyroscope	TRNG	0.41	on-body IoT device
[9]	microphone	PRNG	N/A	Smartphone
[11]	accelerometer, gyroscope, magnetometer	PRNG	N/A	Smartphone
proposed	accelerometer, gyroscope, barometer	TRNG	0.74	Drone

It also shows the randomness of each RNG. All the research shown in Table 1 utilizes the NIST randomization test for randomness verification. So, we compared the average of the p-values in the test. If the previous works [8], [9], [11] do not include the p-values of randomness in their papers, we marked the randomness of those RNGs as ''N/A''.

These existing random number generators, using sensors in IoT or mobile devices, are designed without considering differences in sensor values depending on the state of the devices. But the sensor value output when the drone is flying is very different compared to when it is stationary, because sensors equipped on the drones measure the motion of the devices. Our proposed random number generator considers these differences of sensor values between an in-flight state and a stationary state.

B. CRYPTOGRAPHY FOR SECURING DRONES

Drones have a number of security breaches, such as cyber attacks, physical capture, spoofing attacks, and WiFi attacks. Therefore, drones must be protected against these attacks with some drone-specific security mechanisms. To that end, a lot of studies have been conducted for drone security.

In 2015, J. Won *et al.* proposed a secure drone communication protocol that supports essential security functions such as key agreement, authentication, non-repudiation, and revocation in order to protect communication between drones and various smart objects [18]. In 2016, S. Seo *et al.* proposed a security framework for drone delivery service that uses white-box cryptography to protect data and cryptographic keys in delivery drones [19]. Their framework protects data stored in drones from cyber attacks and physical capture attacks, as drones may have to be operated in hostile areas. In 2017, A. Ivan *et al.* proposed a secure data transmission method between drones and a ground control station [20]. They applied an encryption method using one-time passwords to transmit data from drones with limited computational resources. J. Won *et al.* proposed cryptographic protocols for secure communication in drones for smart cities [21]. In 2018, S. Benzarti *et al.* proposed a drone authentication mechanism that can be applied to a drone-based IoT [22]. This mechanism relies on Identity-based Signcryption to provide authentication and tracking while drones are in flight. In 2019, A. Allouch *et al.* identified the security weaknesses of MAVLink and proposed MAVSec protocol, an enhanced version of MAVLink which uses a cryptographic algorithm for securing communications between UAVs and GCSs (Ground Control Stations) [23].

So far, research on drone security has been mainly focused on drone communication security and drone authentication. Most studies have designed lightweight cryptographic algorithms and authentication mechanisms which consider the limited computing power of drones for secure drone services. In order to apply cryptographic algorithms or authentication techniques to drones, cryptographic keys must be securely generated from drones. They must use good random numbers that should guarantee the randomness with good statistical properties and unpredictability. However, research on a drone random number generator capable of generating cryptographic keys in the drone has never been conducted, until now.

Therefore, we propose a drone-specific random number generator to securely generate cryptographic keys in a drone when equipped with a security protocol. If our DroneRNG is used in the drone security protocol, the cryptographic key can be generated securely, further improving the security of existing drone protocols.

III. EVALUATION OF DRONE SENSOR OUTPUT

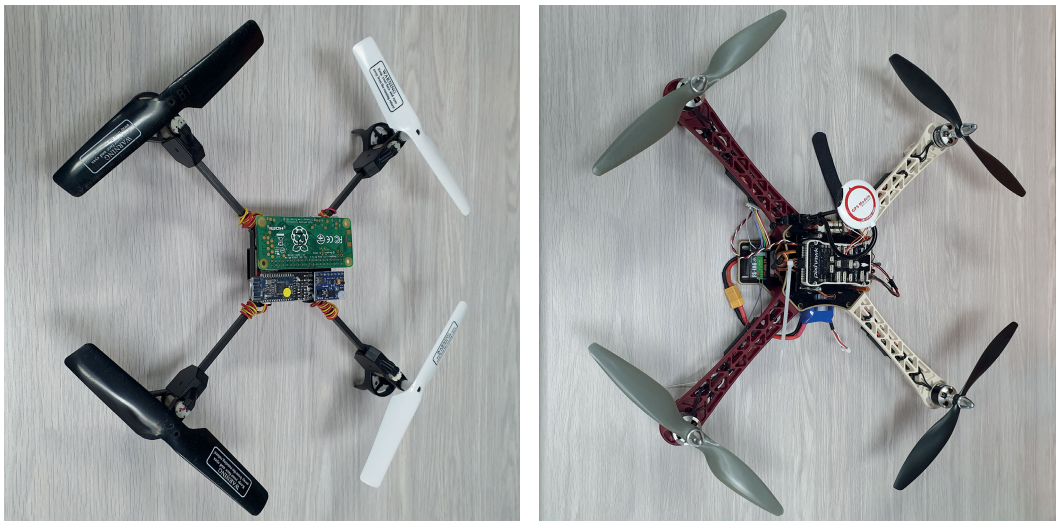
In this section, we will show how different the sensor outputs are when the drones are flying and when they are stationary. And then we will assess how random the raw sensor data output by drones is.

A. DATA COLLECTION

Like other IoT devices, drones are equipped with various sensors such as accelerometer, gyroscope and barometer. An accelerometer and gyroscope are essential for flight. The barometric pressure sensors are used to measure the altitude of drones. The barometer is not previously studied as a sensor for RNGs but it is essential for flying drones. We extract and collect values from sensors to assess randomness of raw data output by sensors equipped on drones. We analyze the randomness of accelerometers, gyroscopes, and altitude sensors, which are integral to the flight performance of the drones.

1) DRONES USED IN THE EXPERIMENT

For our experiments, we used a Raspberry Pi drone and a Pixhawk drone. The Raspberry Pi drone was made using the Raspberry Pi Zero board as a Flight Controller (FC), making



(a) Raspberry Pi Drone (b) Pixhawk Drone

FIGURE 2. Drones used in the experiment.

it more accessible and thus it is easier to perform various functions by mounting additional modules. The Pixhawk drone is friendly to the Unix/Linux programming environment and is used extensively by developers, providing powerful development capabilities using autopilot functions and multithreading. Fig. 2 shows the drones we used in the experiment, the Raspberry Pi drone on the left and the Pixhawk drone on the right.

The Raspberry Pi drone is equipped with an MPU6050 gyro & accelerometer and an MS5611 barometer for altitude measurement, while the Pixhawk drone utilizes an MPU6000 main gyro & accelerometer and an MS5611 barometer for altitude measurement.

2) COLLECTION METHOD

Voris *et al.* [14] showed that the raw sensor data has randomness since the sensor values include noise that are fall within the margin of error, and that sufficient entropy can be derived even when the accelerometer is in a stationary state. When the drone moves in flight, the values of the sensor fluctuate further due to changes in the values measured by the sensor and noise, thus giving better entropy than when it is stationary. Therefore, we use raw data from the sensors to enhance the randomness of the seed. The data collecting process was conducted in two situations to determine how the characteristics of the sensor values change when the drones are in a stationary state and in flight state.

1) In a stationary state, we collected sensor data after the drone was horizontally fixed indoors where no external influence could be exerted.
2) In a flight state, we collected sensor data on a drone flying in a straight line outdoors. We conducted the experiment by setting the flying altitude of the drone at 3 m. In practice, the drone had an average altitude error of 0.09 m and a maximum altitude error of 0.32 m.

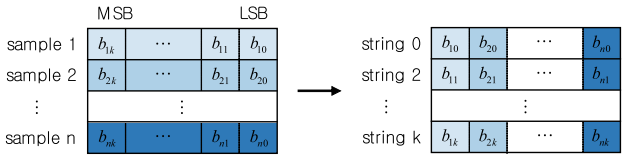


FIGURE 3. Bit rearranging.

B. EVALUATION OF RANDOMNESS BY SENSOR OUTPUT BIT

In order to evaluate randomness of drone sensors, we first characterize sensor values in both stationary and flight states. The oscillation of bits adjacent to the Least Significant Bit (LSB) has a small effect on the fluctuation in sensor values, and the oscillation of bits adjacent to the Most Significant Bit (MSB) has a large effect on the fluctuation in sensor values. Thus, in the stationary state, only the bits adjacent to the LSB fluctuate due to the influence of sensor noise. On the other hand, most bits fluctuate in a flight state.

In order to determine which sensor bits have randomness according to these characteristics, we conducted the randomness test by putting together the bits of the same bit position to one bit string, respectively, as shown in Fig. 3. This test classifies good bits that have randomness and bad bits that do not.

To evaluate the randomness of bit strings, we utilized the National Institute of Standards and Technology (NIST) statistical test suite for RNGs and PRNGs for cryptographic applications [5]. The NIST test suite provides statistical randomness information for a given bit stream. The NIST test suite also provides a total of 15 statistical tests. Each test is

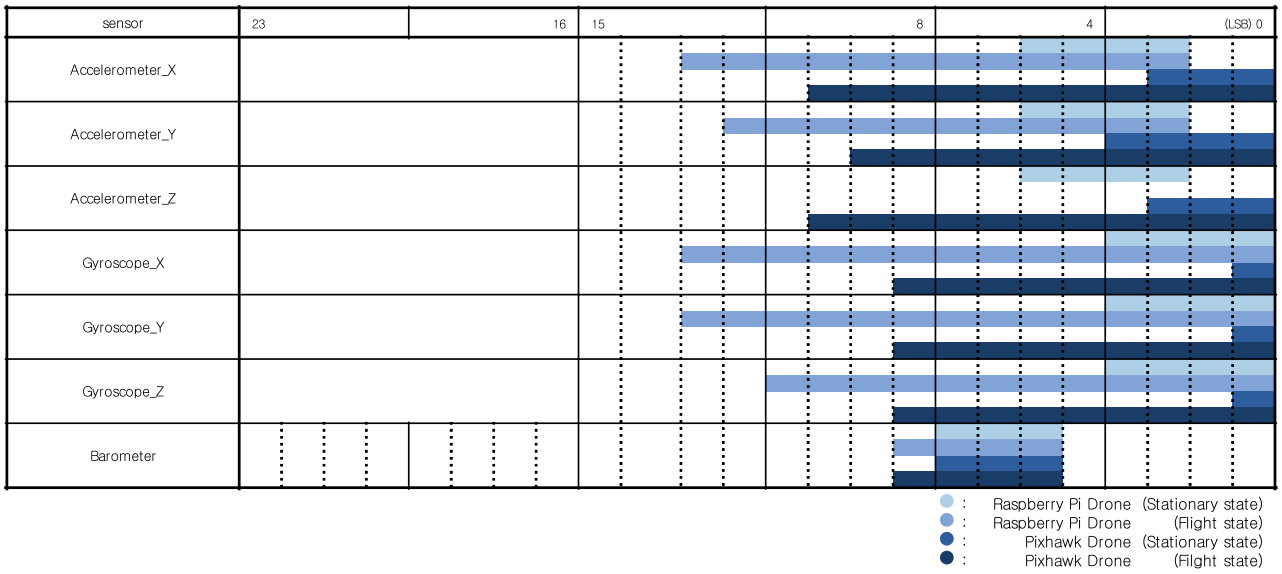


FIGURE 4. The randomness of each sensor by bit of the Raspberry Pi drone and the Pixhawk drone.

designed to assess whether a characteristic that appears in the uniform random stream appears at a given input stream, and its randomness can be assessed through p-value to indicate if each has been passed or not. If the p-value is greater than 0.01, then the bit stream has passed the test.

For the assessment of sensor data randomness by bit, we conducted seven tests out of the full NIST test suite, including the frequency test, a frequency test within a block, the runs test, the longest run of ones within a block, a binary matrix rank test, a DFT test, and an approximate entropy test. We classify the bits corresponding to the bit string as good bits if the bit string has passed at least three NIST tests in more than 75% of trials. These criteria are referred to as the analysis methodology in [12]. A bad bit is a bit that fails to pass the above criteria. We use only good bits among raw sensor data bits to provide enough entropy for the DroneRNG seed. The accelerometer and gyroscope of the Raspberry Pi drone output 16 bits each as sensor values for all three axes, while the barometer outputs 24 bits as sensor values. In the case of the Raspberry Pi drone in a stationary state, the accelerometer has a total of 12 bits (4 bits each) for all three axes that have randomness, and the gyroscope is the same. The barometer has three out of 24 bits with randomness. In a flight state, 23 bits from the accelerometer have randomness (12 bits for the x-axis, 11 bits for the y-axis) and the z-axis does not have a single bit of randomness. The gyroscope has a total of 40 bits of randomness, with 14 bits each on the x-axis and y-axis and 12 bits on the z-axis. And for the barometer, a total of 4 bits have randomness.

Like the Raspberry Pi drone, the accelerometer and gyroscope of the Pixhawk drone also output 16 bits each as sensor values for all three axes, while the barometer also outputs 24 bits as sensor values.

In the case of the Pixhawk drone in a stationary state, the accelerometer has a randomness of 10 bits in total, 3 bits each for the x-axis and z-axis, 4 bits for the y-axis. The gyroscope has a randomness of 3 bits in total for 1 bit each on all three axes, and the barometer, like the one on the Raspberry Pi drone, has a randomness with a total of 3 bits out of 24 bits. In particular, the number of bits with randomness in the gyroscope is only one bit per axis.

In a flight state, the accelerometer has a total of 32 bits with randomness (11 bits each on the x-axis and y-axis, 10 bits on the z-axis) and the gyroscope has a total of 27 bits with randomness (each has 9 bits on all three axes), and for the barometer, a total of 4 bits have randomness. Fig. 4 shows the randomness of each sensor by bit of the Raspberry Pi drone and the Pixhawk drone. Based on the experiment's results, we use 27 bits per one reading of sensor value with the Raspberry Pi drone in a stationary state and 67 bits in a flight state to collect the entropy of our RNG. In the Pixhawk drone, 16 bits per reading in a stationary state and 63 bits in a flight state are used to collect the entropy in our RNG.

IV. DroneRNG

We propose a DroneRNG that generates random numbers using drone sensors and we will explain the algorithm in this section. Fig. 5 shows the algorithm and the overall structure of our DroneRNG. The DroneRNG consists of five main components: Sensor Controller, Dividing, Shuffling, Byte Binding, and Mixing and Swap. The sensor controller controls whether the drone collects data from sensors, depending on the size of the random number pool. When we collect data, we divide it into good bits and bad bits according to our randomness test. After processing the good bits, we convert the bits to bytes. Then, we mix the data converted to bytes using

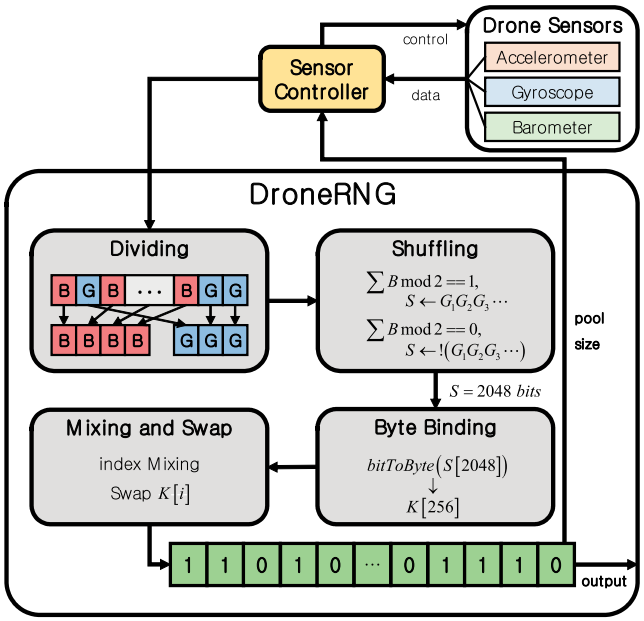


FIGURE 5. The algorithm and overall structure of DroneRNG.

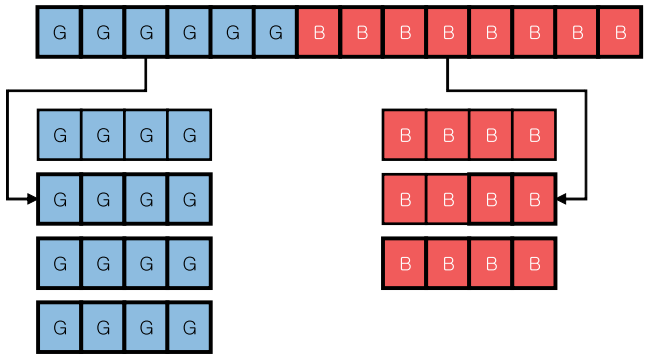


FIGURE 6. The dividing step.

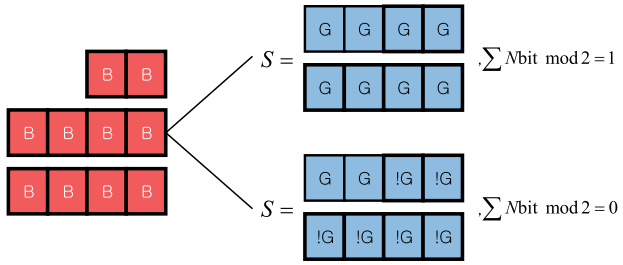


FIGURE 7. The shuffling step.

some algorithm. As our DroneRNG goes through these steps, it generates cryptographically secure true random numbers.

A. SENSOR CONTROLLER

The DroneRNG collects sensor data from drone accelerometers, gyroscopes, and barometers to generate random numbers. The sensor controller monitors the amount of data available in the random number pool, which has an internal buffer to maintain the minimum desired capacity. If the random numbers from the internal buffer are used and the amount falls below a certain level, the sensor controller collects the data from the sensor and sends it to the dividing step so that it can be used to generate random numbers. The sensor controller stops collecting data when the internal buffer is filled to a certain level.

B. DIVIDING

This step processes the raw data from the sensors. According to the randomness analysis by bit assessed in Section 3, the raw data of each sensor is divided into G bits (Good bits) with randomness and B bits (Bad bits) without randomness. And it is able to remove bits without randomness and make entropy have more randomness through postprocessing during the shuffling step. Since the G bits and B bits of each sensor are different depending on whether the drone is stationary or flying, we implement the dividing step differently depending on these states.

C. SHUFFLING

Using the G bits and B bits, we enhance the randomness of the seed used in the mixing and swap step. We take the parity of each B bit and process the G bits to be used as seeds according to the parity bit. If the parity bit is 1, save the G bit in a bit string S as it is. And if the parity bit is 0, use the bitwise NOT operation on each bit of G bits and save it in bit string S. When the bit string S, which is the size of 2048 bits, is filled, we carry out the next step.

We can say that maybe the bit string S obtained after the shuffling step already has some randomness because it consists of only bits with randomness that have enough entropy among the raw sensor data. However, when we assess the bit string S after the shuffling step using the NIST test suite, it does not pass the frequency test, the frequency test within a block, the runs test, as well as the longest run of ones within a block, but it does pass the binary matrix rank test, the DFT test, and the approximate entropy test. In order to improve these results, we designed the algorithm by adding two steps: The Byte Binding, and Mixing and Swap.

D. BYTE BINDING

A bit string S of 2,048 bits is converted to 256 Bytes and saved in array K of size 256 because the mixing and swap step, the next component of this process, generates random numbers by processing input in bytes.

E. MIXING AND SWAP

The Mixing and Swap is based on RC4 stream cipher. We create an array T of size 256 and initialize $T[i] = i$ where i is from 0 to 255. Then, using array K of size 256 as a seed, the arbitrary index j of array T is calculated using the following expression.

$$j = T[i] \oplus (j + K[i]) \mod 256, (i : 0 \sim 255)$$

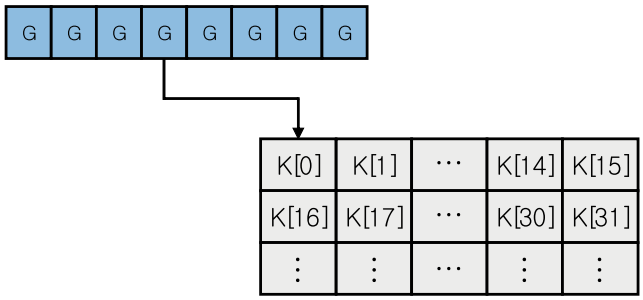


FIGURE 8. The byte binding step.

Algorithm : Mixing and Swap

input : K (256 Bytes)
output : Random Number

```
1 T[256]
2 j ← 0
3 for (i = 0 to 255){
      T[i] ← i
  }
4 for (i = 0 to 255){
      j ← {T[i] ⊕ (j + K[i])} mod 256
      swap (T[i], T[j])
  }
5 Random Number = T[256]
6 return Random Number
```

FIGURE 9. The mixing and swap algorithm.

Using the index j obtained from this calculation, we swap the T[i] and the T[j]. After iterating 0 to 255 for i, we save the last generated array T in bits in the random number pool, which is in the internal buffer. When designing the RNG up to this point, the output bit string of RNG passes the NIST randomness test suite, including not only the binary matrix rank test, the DFT test, and the approximate entropy test but the frequency test, the frequency test within a block, the runs test, the longest run of ones within a block. So, our RNG generates cryptographically secure true random numbers.

The SensoRNG [12], that we have referred to in the previous section, has weak p-values in the runs test and the rank test. Through this mixing and swap algorithm, we can improve the results of the runs test and the rank test. Also, DroneRNG is able to have much stronger p-values particularly when it comes to the cumulative sum test and the frequency test.

V. EVALUATION OF DroneRNG

We evaluated the performance of the DroneRNG we designed. We utilized the NIST test suite once again to evaluate the randomness of our DroneRNG output. We conducted a larger subset of tests, compared to section 3, for more rigorous evaluation of our outputs while only seven tests were conducted to evaluate the randomness by bit of raw sensor data in section 3. In addition to the seven tests previously conducted, we conducted a total of 10 tests including the cumulative sum test, serial test, and linear complexity test. We excluded the nonoverlapping template test, overlapping test, universal statistic test, and random excursion test due to the large number of potential parameters [12].

A. IMPLEMENTATION OF DroneRNG

We used the accelerometer and gyroscope with all three axes, and the barometer for the implementation of DroneRNG by referring to the results of the experiment in section 4. The Raspberry Pi drone in flight does not use the z-axis of the accelerometer because it does not have any bits with randomness in the output. Accordingly, we implemented our RNG using the accelerometer with only two axes (x and y), the gyroscope with three axes, and the barometer for the Raspberry Pi drone in flight. We utilized only bits with randomness, and the bits can be seen in Fig. 4. The Sensor Controller was implemented to work similarly to LinuxPRNG(LPRNG). The internal buffer is 4,096 bits like LPRNG [24]. When the internal buffer falls below 50 % capacity, the Sensor Controller activates the sensors to collect data and generates 2,048 bits of random numbers. Then, the buffer is filled with these 2,048 bits, and the Sensor Controller deactivates the sensors and stops collecting data to prevent waste of resources. And a random number generation is extracted from the buffer as needed. Extracting from the buffer and filling the buffer were implemented to operate on different threads. It enables the generation of random numbers when the random numbers, that are stored in the buffer, fall below the threshold while also filling the buffer at the same time.

B. ENTROPY POOL

We compare the random number pool of DroneRNG with the input pool of Linux PRNG using dev/random. The standard C random function is excluded from comparison because it does not have any entropy pool. Since LPRNG is not available on Pixhawk, the comparison is conducted on the Raspberry Pi drone. The input pool of the LPRNG on a desktop equipped with other user devices such as a mouse and keyboard is maintained at about the 2,048-bit level as 256 bits of entropy are dissipated and charged at about five-minute intervals as shown in Fig. 10.

However, for the Raspberry Pi drone without such user devices such as mouse or keyboard, the remaining input pool is maintained at the 256 and 512-bit level as shown in Fig. 11. The entropy is periodically used in Linux environments because Linux includes an Address Space Layout Randomization (ASLR). The ASLR is a memory-protection process for operating systems that changes the address of the data by randomly arranging the address space positions of key data areas of a process, including the base of the executable and the positions of the stack, heap, and libraries [25]. As a result, it prevents attacks on memory and uses random numbers to allocate addresses randomly.

TABLE 2. Randomness of sensor output comparison.

	RPi Drone						Pixhawk				
			Stationary state		Flight state			Stationary state		Flight state	
	CRAND	LPRNG	raw data	DroneRNG	raw data	DroneRNG	CRAND	raw data	DroneRNG	raw data	DroneRNG
Approx. Entropy	0.411264	0.545586	0.435102	0.584533	0.513895	0.645326	*0.000003*	0.223154	0.494678	0.376954	0.686802
Block frequency	0.471336	0.585751	0.535042	0.619709	0.586674	0.525563	0.179621	0.310125	0.644418	0.438739	0.715751
Cum. Sum (f)	0.610680	0.520274	*0.003860*	**0.999875**	0.457462	**0.999933**	**0.990002**	0.459841	**0.999367**	0.488048	**0.999531**
Cum. Sum (r)	0.534291	0.600174	*0.003320*	**0.999875**	0.502612	**0.999904**	**0.997211**	0.419488	**0.999367**	0.430660	**0.999498**
FFT	0.457250	0.302751	0.536937	0.561590	0.557101	0.238712	*0.000007*	0.516688	0.638170	0.602695	0.727055
Frequency	0.510056	0.429460	*0.002892*	**1.000000**	0.568635	**1.000000**	**0.878246**	0.585094	**1.000000**	0.601143	**1.000000**
Linear Complexity	0.538410	0.585204	0.495005	0.422657	0.514560	0.484159	0.405132	0.319488	0.512673	0.332008	0.445875
Longest Run	0.616710	0.578107	0.523295	0.697727	0.427797	0.698781	*0.000815*	0.359777	0.698781	0.420573	0.675553
Rank	0.605057	0.380930	0.485428	0.622145	0.519138	0.780703	0.519133	0.508538	0.587734	0.405280	0.780703
Runs	0.538800	0.545532	0.553537	0.765361	0.441601	**0.821530**	0.341266	0.344772	0.782889	0.318849	**0.890596**
Serial (f)	0.526890	0.257166	0.506249	0.607417	0.494206	0.578619	*0.000000*	0.416938	0.511314	0.302998	0.472524
Serial (r)	0.482851	0.325452	0.546516	0.621897	0.436888	0.614679	*0.000000*	0.448182	0.516503	0.409479	0.492133

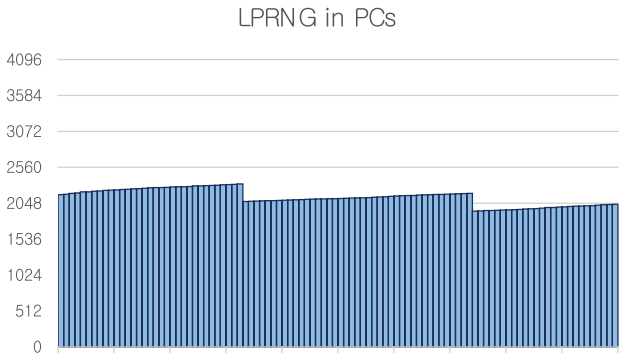


FIGURE 10. LPRNG entropy over time in desktop PCs.

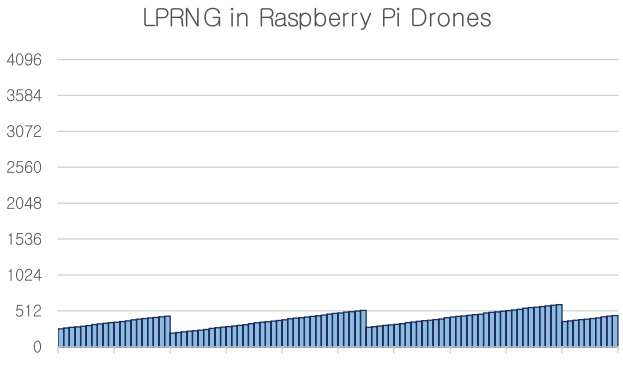


FIGURE 11. LPRNG entropy over time in Raspberry Pi Drones.

Fig. 11 shows the entropy trace of the LPRNG when users do not use random numbers at all. The remaining entropy, as seen in Fig. 11, is not sufficient, so it may run out when users use it. Since the entropy pool of Linux PRNG (LPRNG) is a blocking pool that uses dev/random, LPRNG does not generate random numbers when the entropy runs out.

We are able to fix such problems by using drone sensors which can rapidly output the entropy with sufficient randomness.

Accordingly, we design a DroneRNG that collects sensor data from drones and generates random numbers of 2,048 bits in a single process to fix such an entropy depletion problem.

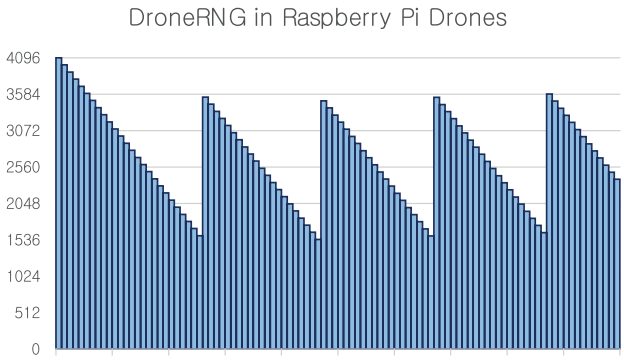


FIGURE 12. DroneRNG entropy over time in Raspberry Pi Drones.

The size of the entire entropy pool is set to 4,096 bits. When the amount of remaining entropy drops below 2,048 bits, we collect bits with randomness from sensor data and generate random numbers to fill the entropy pool. Fig. 12 shows the entropy trace of the random number pool in DroneRNG. The entropy is collected very fast and stored in the random number pool with sufficient randomness through DroneRNG.

C. RANDOMNESS OF SENSOR OUTPUT

We use accelerometers, gyroscopes, and barometers of drones as a seed of DroneRNG. These sensors are commonly equipped on all drones for flight. As we explored in section 3, the drone also outputs bits with randomness both when it is stationary and when it is in flight. Methods for generating random numbers with the Raspberry Pi drone are LPRNG and standard C random function (CRAND, rand()). The LPRNG collects noise signals from device drivers and other sources in the entropy pool through a character device called dev/random provided by the kernel and generates random numbers using these noise signals and interrupt time intervals that occur in the device driver.

Therefore, we compared the performance of our DroneRNG with the LPRNG and CRAND by quantifying randomness of each RNG through NIST randomization test. DroneRNG was also compared to the method of generating random numbers just by attaching only random bits of raw

sensor data in Table 2. The p-values of random numbers from DroneRNG are generally higher than the p-values of random numbers from existing RNGs shown in Table 2.

In the case of the Pixhawk drone, the LPRNG is not available because the OS is not Linux. Thus, with the Pixhawk drone we compare DroneRNG with the standard C random function and the method of attaching only raw sensor data. We run each RNG using sensors of both stationary and flying drones. Table 2 shows the comparison of the existing RNGs and DroneRNG in the Raspberry Pi drone and the Pixhawk drone.

The outputs of both CRAND and LPRNG, the existing RNGs of the Raspberry Pi drone, have sufficient randomness. However, the bit string generated by attaching only random bits of sensor data while the drone is stationary does not pass the cumulative sum test and frequency test. In the case of DroneRNG, all tests have been passed and the p-value of the frequency test is 1.0 so that bits 0 and 1 are output exactly half-and-half. In addition, the p-values are generally higher in all tests than those of the existing RNGs. In the NIST test suite, a higher p-value means that the output of an RNG is statistically more random [5]. In the flight state of the Raspberry Pi drone, the bit string generated by attaching only random bits of sensor data passes all of the tests. For DroneRNG, the p-value of the frequency test is also 1.0, and the p-values are generally higher in all of the tests compared to the existing RNGs.

The random numbers generated using CRAND in the Pixhawk drone fail the approximate entropy test, FFT test, and serial test. A bit string consisting only of random bits of sensor data - both stationary and flying - passes all of the tests. Using DroneRNG, the p-value of the frequency test is 1.0, and the randomness is greatly improved for all of the tests.

D. PERFORMANCE EVALUATION

In this section, we provide a performance evaluation of our algorithm. First, we analyzed the time complexity and space complexity.

The proposed algorithm consists of four steps. In the first step, the raw sensor values are divided into G bits and B bits in n operations, where n is the number of bits in a given input. The next step is to take the parity of each B bit and process the G bits according to the parity bit by performing another n operations. Then the third step requires additional n operations to convert bits into bytes. Finally, the last step is to perform one substitution operation and one byte-level XOR operation to generate the random number. Thus, the total number of operations required is $3n + 2$, with the time complexity of the DroneRNG being $O(n)$. In addition, since the algorithm does not use any recursive call and the memory required does not depend on the input value, the required memory space is given constant, resulting in a space complexity of $O(1)$.

Second, we implemented our algorithm and measured how long the CPU cycle took and how much RAM and virtual memory was used for the actions. We present the CPU cycle and memory required to generate random numbers of 2,048 bits each using CRAND, LPRNG, and DroneRNG.

TABLE 3. Performance comparison.

	Cycle (700 MHz)	Time (ms)	RAM (KB)	Virt. Mem. (KB)
DroneRNG(1)	173,600	0.248	1,396	
DroneRNG(2)	143,500	0.205	1,392	
DroneRNG(4)	127,400	0.182	1,428	
DroneRNG(8)	119,700	0.171	1,484	12,200
DroneRNG(16)	116,200	0.166	1,408	
DroneRNG(31)	114,100	0.163	1,360	
CRAND(1)	240,800	0.344	1,152	
CRAND(2)	126,700	0.181	1,092	
CRAND(4)	75,600	0.108	1,056	
CRAND(8)	46,900	0.067	1,056	1,796
CRAND(16)	35,700	0.051	1,092	
CRAND(31)	27,300	0.039	1,016	
LPRNG(8)	195,300	0.279	1,088	1,796

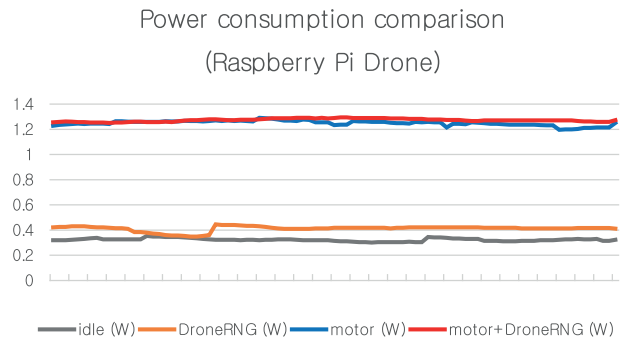
TABLE 4. Power consumption comparison.

Drone	Idle	With DroneRNG	With Motors	With Both
Raspberry Pi	0.32 W	0.41 W	1.25 W	1.27 W
Pixhawk	7.20 W	7.35 W	156.42 W	157.19 W

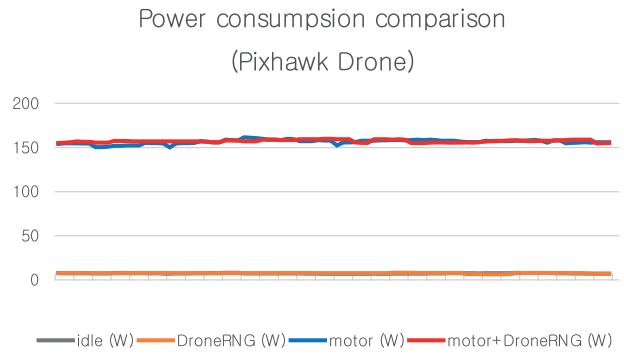
The number in parentheses on Table 3 is the bits of a random number generated. When we pass the number of bits as a parameter of DroneRNG, it extracts only the number of bits from the random number pool and outputs that as the random number. Our RNG is more efficient in this respect than the CRAND which always outputs a 31-bit random number. When a smaller random number than 31-bit is required, we use a random number divided by n that is a desired size. The size of a random number generated by LPRNG is fixed at 8-bit. As seen in Table 3, the cycle and time in the generation of random numbers of less than 2-bit in the Raspberry Pi drone are not significantly different from the CRAND and the DroneRNG, but the CRAND is faster in the generation of random numbers of larger size. Our DroneRNG generates random numbers, which are also cryptographically secure, in exchange for lower performance in terms of cycle and memory compared to CRAND, which is significantly less random. It also generates random numbers in drones that cannot use LPRNG, and it is faster than LPRNG.

E. POWER CONSUMPTION EVALUATION

A drone uses most of its power consumption for motors to run the propellers. We set up four situations and measured the power consumption in each situation to analyze how much power the DroneRNG needs. The four situations were an idle drone, a drone running only DroneRNG, a drone running only motors, and a drone running DroneRNG and motors together. Fig. 13 shows the power consumption comparison over time of the two drones in each of the four situations and Table 4 summarizes the average power consumption in four situations.



(a) Raspberry Pi Drone



(b) Pixhawk Drone

FIGURE 13. **Power consumption comparison in Drones.**

The idle Raspberry Pi drone used an average power of 0.32 W and the Raspberry Pi drone running only DroneRNG used an average power of 0.41 W. The Raspberry Pi drone running only motors used an average power of 1.25 W, and the Raspberry Pi drone running DroneRNG and motors together used an average power of 1.27 W. In the case of the Pixhawk drone, the idle drone used an average power of 7.2 W and the drone running only DroneRNG used an average power of 7.35 W. The drone running only motors used an average power of 156.42 W, and the drone running DroneRNG and motors together used an average power of 157.19 W.

When the Raspberry Pi runs DroneRNG in an idle state, the increase in power consumption is only 0.09 W. The difference between the Raspberry Pi running just the motors and running the motors with the DroneRNG is only 0.02 W. In the case of the Pixhawk, the increase in power when running DroneRNG in an idle state is 0.15 W. The difference then between using only motors and using motors with the DroneRNG is 0.77 W. The result was that power consumption for DroneRNG was miniscule compared to the amount of power the drone used to run the motors.

VI. CONCLUSION

In this paper, we showed how the sensor outputs are different when the drone is in flight and when it is stationary. We also confirmed that sensors in drones can be used as sources of randomness, and tested which output bits are random enough to be used as random sources for a random number generator.

Based on the results of our experiments, we designed a drone-specific random number generator, called DroneRNG, which can use random numbers in the program regardless of the depletion of entropy stored in the buffer by using thread, in order to collect sensor data and generate random numbers. Moreover, it is possible to use DroneRNG both when the drone is stationary and in flight.

The random numbers generated by our DroneRNG achieve enhanced statistical randomness. Particularly, the p-value is determined to be almost equal to 1 for the cumulative sum test and frequency test, which means that the random numbers are also cryptographically secure.

REFERENCES

[1] K. Schwab and N. Davis, *Shaping the Fourth Industrial Revolution*. 2018.
[2] R. Altawy and A. M. Youssef, "Security, privacy, and safety aspects of civilian drones: A survey," *ACM Trans. Cyber Phys. Syst.*, vol. 1, no. 2, p. 1–25, Nov. 2016.
[3] D. Eastlake, S. D. Crocker, and J. I. Schiller, *Randomness Requirements for Security*, document RFC 1750, Internet Engineering Task Force, Dec. 1994.
[4] R. Davies, "Hardware random number generators," in *Proc. 15th Austral. Statist. Conf.*, Jul. 2000.
[5] A. Rukhin, *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*, Standard 800-22, National Institute of Standards and Technology, 2001.
[6] F. Strenzke, "An analysis of OpenSSL's random number generator," in *Proc. EUROCRYPT*, 2016, pp. 644–669.
[7] X. Zhang, Z. Tang, Y. Zhang, and L. Qi, "Portable true random number generator for personal encryption application based on smartphone camera," *Electron. Lett.*, vol. 50, no. 24, pp. 1841–1843, Nov. 2014.
[8] R. Morrison, "Design of a true random number generator using audio input," *J. Craptol.* vol. 1, no. 1, pp. 1–4, 2001.
[9] P. P. A. Wibowo and F. W. Romadhon, "Generation of pseudorandom numbers from audio input in smart phone Android," in *Proc. 11th Int. Conf. Telecommun. Syst. Services Appl. (TSSA)*, Oct. 2017, pp. 1–5.
[10] A. Suciu, D. Lebu, and K. Marton, "Unpredictable random number generator based on mobile sensors," in *Proc. IEEE 7th Int. Conf. Intell. Comput. Commun. Process.*, Aug. 2011, pp. 445–448.
[11] A. Marghescu, G. Teseleanu, and P. Svasta, "Cryptographic key generator candidates based on smartphone built-in sensors," in *Proc. IEEE 20th Int. Symp. Design Technol. Electron. Packag. (SIITME)*, Oct. 2014, pp. 239–243.
[12] K. Wallace, K. Moran, E. Novak, G. Zhou, and K. Sun, "Toward sensor-based random number generation for mobile and IoT devices," *IEEE Internet Things J.*, vol. 3, no. 6, pp. 1189–1201, Dec. 2016.
[13] Y. Sun and B. Lo, "Random number generation using inertial measurement unit signals for on-body IoT devices," in *Proc. Living Internet Things, Cybersecur. IoT*, 2018, pp. 1–9.
[14] J. Voris, N. Saxena, and T. Halevi, "Accelerometers and randomness: Perfect together," in *Proc. 4th ACM Conf. Wireless Netw. Secur. (WiSec)*, 2011, pp. 115–126.
[15] C. Hennebert, H. Hossayni, and C. Lauradoux, "Entropy harvesting from physical sensors," in *Proc. 6th ACM Conf. Secur. Privacy Wireless Mobile Netw. (WiSec)*, 2013, pp. 149–154.
[16] J. Loutfi, A. Chehab, I. H. Elhajj, and A. Kayssi, "Smartphone sensors as random bit generators," in *Proc. IEEE/ACS 11th Int. Conf. Comput. Syst. Appl. (AICCSA)*, Nov. 2014, p. 773–780.
[17] L. M. Dinca and G. Hancke, "Behavioural sensor data as randomness source for IoT devices," in *Proc. IEEE 26th Int. Symp. Ind. Electron. (ISIE)*, Jun. 2017, pp. 2038–2043.
[18] J. Won, S.-H. Seo, and E. Bertino, "A secure communication protocol for drones and smart objects," in *Proc. 10th ACM Symp. Inf., Comput. Commun. Secur (ASIA CCS)*, Singapore, Apr. 2015, pp. 249–260.

[19] S.-H. Seo, J. Won, E. Bertino, Y. Kang, and D. Choi, "A security framework for a drone delivery service," in *Proc. 2nd Workshop Micro Aerial Vehicle Netw., Syst., Appl. Civilian Use (DroNet)*, 2016, pp. 29–34.
[20] I. Avdonin, M. Budko, M. Budko, V. Grozov, and A. Guirik, "A method of creating perfectly secure data transmission channel between unmanned aerial vehicle and ground control station based on One-Time pads," in *Proc. 9th Int. Congr. Ultra Modern Telecommun. Control Syst. Workshops (ICUMT)*, Berlin, Germany, Nov. 2017, pp. 410–413.
[21] J. Won, S.-H. Seo, and E. Bertino, "Certificateless cryptographic protocols for efficient drone-based smart city applications," *IEEE Access*, vol. 5, pp. 3721–3749, 2017.
[22] S. Benzarti, B. Triki, and O. Korbaa, "Privacy preservation and drone authentication using ID-based signcryption," in *Proc. SoMeT*, 2018, pp. 226–239, doi: 10.3233/978-1-61499-900-3-226.
[23] A. Allouch, O. Cheikhrouhou, A. Koubaa, M. Khalgui, and T. Abbes, "MAVSec: Securing the MAVLink protocol for ardupilot/PX4 unmanned aerial systems," 2019, *arXiv:1905.00265*. [Online]. Available: http://arxiv.org/abs/1905.00265
[24] P. Lacharme, A. Röck, V. Strubel, and M. Videau, "The linux pseudo-random number generator revisited," *Cryptol. ePrint Archive*, vol. 2012, no. 251, pp. 1–23, 2012.
[25] H. Shacham, M. Page, B. Pfaff, E.-J. Goh, N. Modadugu, and D. Boneh, "On the effectiveness of address-space randomization," in *Proc. 11th ACM Conf. Comput. Commun. Secur. (CCS)*, 2004, p. 298–307.

SEONG-MIN CHO (Student Member, IEEE) received the B.S. degree from the Division of Electrical Engineering, Hanyang University ERICA Campus, South Korea, in 2019, where he is currently pursuing the master's degree with the Department of Electrical Engineering.

His research interests include the IoT security, security of embedded systems, and post-quantum cryptography.

EUNGI HONG (Student Member, IEEE) received the B.S. degree from the Division of Electrical Engineering, Hanyang University ERICA Campus, South Korea, in 2018, where he is currently pursuing the master's degree with the Department of Electrical Engineering.

His research interests include the IoT security, security of embedded systems, and post-quantum cryptography.

SEUNG-HYUN SEO (Member, IEEE) received the B.S. degree from the Department of Mathematics, Ewha Womans University, Seoul, South Korea, in 2000, and the M.S. and Ph.D. degrees in computer science from Ewha Womans University, in 2002 and 2006, respectively.

She is currently an Associate Professor with Hanyang University. Before joining the faculty at Hanyang University, in 2017, she was an Assistant Professor with Korea University Sejong campus for two years. Before that, she was a Postdoctoral Researcher of computer science with Purdue University for two and half years, a Senior Researcher of Korea Internet and Security Agency (KISA) for two years, and a Researcher for three years in Financial Security Agency (FSA), South Korea. Her main research interests include cryptography, the IoT security, mobile security, blockchain, and post-quantum cryptography.