Pamukkale Üniversitesi Mühendislik Bilimleri Dergisi

Pamukkale University Journal of Engineering Sciences

Xorshiftul+: A novel hybrid random number generator for internet of things and wireless sensor network applications

Xorshıftul+: Nesnelerin interneti ve kablosuz duyarga ağı uygulamaları için yeni bir melez rastgele sayı üreteci

Ömer AYDIN[1], Cem KÖSEMEN[2]*

[1]Dokuz Eylul University, Faculty of Economics and Administrative Sciences, Izmir, Turkey.
omer.aydin@deu.edu.tr

[2]Dokuz Eylul University, The Graduate School of Natural and Applied Sciences, Computer Engineering, Izmir, Turkey.
cem.kosemen@bakircay.edu.tr

Received/Geliş Tarihi: 27.09.2019 Revision/Düzeltme Tarihi: 07.12.2019 doi: 10.5505/pajes.2020.00344
Accepted/Kabul Tarihi: 09.01.2020 Research Article/Araştırma Makalesi

Abstract

There are an increasing number of lightweight devices such as smart cards, radio frequency identification (RFID) tags, wireless sensor nodes and devices associated with the Internet of Things (IoT) concept, all of which need effective and lightweight security structures. One of the basic elements used in authentication protocols is the nonce values that are generated by PRNGs. Also random numbers are used for encryption process in secure communication. It has demonstrated great suitability for devices with limited resources in terms of performance, resource usage and randomness. Proposed PRNG is tested with NIST statistical test suite (NIST STS), which is one of the most comprehensive randomness test tools. It is also implemented, tested on wireless identification and sensing platform (WISP) passive RFID tag and compared with well-known PRNGs. As a result of the comparisons, it has better results than its rivals.

Keywords: xorshift, lightweight, pseudo random number generator, hardware random number generator, hybrid random number generator, WISP, security, xorshiftL+, xorshiftUL+.

Öz

Akıllı kartlar, radyo frekansı tanımlama (RFID) etiketleri, kablosuz sensör düğümleri ve Nesnelerin İnterneti (IoT) konsepti ile ilişkili düşük kaynaklı cihazların sayısında artış olmaktadır. Kimlik doğrulama protokollerinde kullanılan temel öğelerden biri SRSÜ'ler tarafından üretilen nonce değerleridir. Ayrıca güvenli iletişimde şifreleme işlemi için rasgele sayılar kullanılır. Sınırlı kaynaklara sahip cihazlarda performans, kaynak kullanımı ve rastlantısallık açısından uygunluk göstermiştir. Önerilen SRSÜ, en kapsamlı rastgele sayı test araçlarından biri olan NIST istatistiksel test takımı (NIST STS) ile test edilmiştir. Ayrıca, kablosuz tanıma ve algılama platformu (WISP) pasif RFID etiketi üzerinde test edilmiş ve uygulanmış ve iyi bilinen SRSÜ'lerle karşılaştırılmıştır. Karşılaştırmalar sonucunda rakiplerinden daha iyi sonuçlar vermiştir.

Anahtar kelimeler: xorshift, hafif, sözde rastgele sayı üreteci, donanım rastgele sayı üreteci, melez rastgele sayı üreteci, WISP, güvenlik, xorshiftL+, xorshiftUL+.

1 Introduction

In recent years, emerging technologies, such as smart cards, radio frequency identification tags (RFID), wireless sensor network (WSN) nodes and the concept of Internet of things (IoT) brought not only new solutions but also challenges in their scope of application. The proliferation of devices manipulating or transmitting sensitive and critical information requires more attention to security issues, because classical security algorithms cannot offer effective and feasible security solutions for these groups of devices. Thus, many lightweight cryptographic algorithms have been suggested in literature, including block ciphers [1]-[7], and hash functions [8]-[10]. The aim of lightweight security algorithms is to find a balanced solution for performance, speed and security needs taking into account limitations such as storage, and processing power.

Random number generation has an important role in cryptography and authentication, and therefore, in security. These numbers can be used as the key or the seed for key generation in cryptographic algorithms and as nonce values in authentication protocols. For example, random numbers can be used as the secret keys for symmetric encryption algorithms such as advanced encryption standard (AES) and data encryption standard (DES). All these encryption algorithms and authentication protocols need random number sequences generated with levels of high randomness to prevent attackers infiltrating the system. Generating these number sequences, requires a source called a random number generator (RNG). There are two types of RNGs based on their sources. True random number generators (TRNGs) which use hardware sources; and pseudorandom number generators (PRNGs), that use algorithms for generating random numbers or bit sequences [11]. Most PRNGs create random numbers more rapidly than TRNGs, and are suitable for stream ciphers. Moreover, well-designed PRNG algorithms can be easily implemented in lightweight devices. However, the generated number sequence can be predicted in PRNGs if both the algorithm and initial seed are known [12]. For this reason, PRNGs must be computationally secure and seeded by an unpredictable source. Thus, the two RNG types should be used together. A random number generated by a TRNG is used as the initial seed of a PRNG function. This type of random number generator can be called as hybrid random number generator (HRNG). HRNGs make a combination of PRNG (fast generation and high quality random numbers) seeded repeatedly by TRNG (high unpredictability but slow generation). HRNG design

*Corresponding author/Yazışılan Yazar

should resolve the challenge of the balance between speed and predictability [12].

Ultra-light random number generators have been developed specifically for ultra-lightweight devices. These number generators have simple mathematical and bitwise operations (AND, OR, XOR and +). In this paper, a new, effective and ultra-lightweight hybrid random number generator is proposed. The generator was implemented on wireless identification and sensing platform (WISP). This newly designed HRNG combines a TRNG that uses the temperature sensor with the newly designed PRNG, a light version of Marsaglia's xorshift algorithm [13]. The PRNG is called the xorshiftUL+, with "UL" denoting "ultra-lightweight". xorshiftUL+ is suitable for ultra-lightweight and lightweight devices such as IoT devices and RFID tags due to its randomness, performance and resource usage.

WISP has 16-bit programmable microcontroller. It also has accelerometer, light sensor and temperature sensor as built-in physical sensors [14]-[16]. Unlike most of the other RFID tags, WISP family of devices are programmable. WISP 5, which is developed at the University of Washington laboratories, is used in this work for testing the performance and runtime of the hybrid generator [17].

Our contribution is an HRNG that uses a PRNG initialized by a chosen TRNG. This HRNG is ultra-lightweight, it is proposed and tested on WISP passive RFID tag. All random numbers created by this HRNG were tested by the NIST STS.

The rest of this article is organized as follows. Section 2 discusses previous and related works. In Section 3, details of material and methods used for proposed HRNG are given. Section 4 gives experimental results of performance, test, evaluation and discussions. Finally, the paper is concluded in Section 5.

2 Related works

The literature contains many solutions for random number generations, which are the main issues in the security of the lightweight and ultra-lightweight devices. Studies are currently focusing on ways to overcome these challenges using innovative solutions. In this study, we examined previous RNG implementations and their statistical and security properties.

Avaroğlu et al. [18] proposed a new hybrid PRNG by means of an additional input introduced to transition and output functions used in a raw PRNG system in order to eliminate failure to meet the R4 security requirement. R4: following random numbers cannot be calculated if the internal state value is known or if it is possible to predict the internal state value even when it is not known [19]. The random number generator developed in this study uses AES and similar algorithms that are complex and resource intensive. Therefore, it will not be suitable for all lightweight devices.

Avaroğlu [20] proposed a PRNG that uses two Arnold [21] cat map outputs. The output of this generator was tested with NIST STS and some other analytical tests. Author admitted that bit rate decreased after sampling. In the manuscript there is no test or information about if the generator is suitable for lightweight devices.

Koyuncu and Özcerit proposed a study that presents a Sundarapandian–Pehlivan chaotic system's Xilinx Virtex-6 FPGA implementation for TRNG. TRNG has a speed of 58.76 Mbit/s. It was verified by NIST 800-22 standards and FIPS 140-1 [22]. In the article there is no specific information about if it is suitable for lightweight devices such as RFID or IoT devices.

Çabuk et al. [23] proposed a new PRNG by modifying well known xorshift algorithm called xorshiftR+, after developing many versions of the original xorshift128plus by changing parameters. Finally, three final versions were developed and compared. WISP passive RFID tag was used to implement these algorithms which were checked against electronic product code generation 2 (EPCGen2) standards, and the ENT, NIST statistical test suite (NIST STS) and TestU01 tests. After the tests, the authors selected the best of the three versions based on the test results, resource usage and performance.

Kösemen at el. [24] developed a pseudorandom number generator by using genetic programming method. Genetic programming method uses Shannon entropy calculation for the fitness function. Mathematical and logical operators were used to generate a PRNG satisfying NIST STS tests and EPCGen2 standards.

Lawnik [25] used adequate chaotic transformation with uniform distribution and recommended a pseudo-random number generation method. This method changes continuous distributions into uniform distributions by flattening, allowing generation of pseudo-random numbers by continuous distribution. For the flattening process, it uses the frequency of the occurrence of successive chaotic transformation branches. Standard normal distribution example is used to analyze this method. In this paper, recommended PRNG was not tested with NIST STS.

Rose made an [26] analysis about cryptographic quality of the KISS ('Keep it Simple Stupid') PRNG. Marsaglia and Zaman first specified KISS in 1993 [27] and Marsaglia published C code in 1998. Some authors argued that KISS PRNG is cryptographically secure, although Marsaglia himself never claimed this. Rose showed that KISS PRNG does not meet certain cryptographically secure PRNG criteria, demonstrating that the initial state of the KISS PRNG can be recovered with 70 output words, and takes about 2 hours depending on the computer hardware. Rose also pointed out that Marsaglia's 2011 version of KISS is vulnerable to divide-and-conquer attack, so KISS is not suitable for applications needing cryptographically secure generator.

Alcin et al. proposed a high-speed chaotic true random number generator based on artificial neural network. They claimed that it generates random numbers that pass all randomness tests and this TRNG can be used in cryptographic and communication applications [28].

Rahmat et al. developed a hybrid pseudorandom generator using Vector algebra for a traditional game called Kuaci that was recently developed for Android systems. Milliseconds of the system clock were used as seeds [29]. The generator recommended in this application has not been tested by any statistical test such as Diehard or NIST.

A circuit is implemented to generate random numbers on a highly efficient FPGA card that generates 32-bit random numbers operating at a frequency of 125 MHz by Devi et al. [30]. These random numbers were tested with Diehard and NIST. In this study there is no comparison against well-known RNGs according to performance and resource usage.

3 Material and methods

To develop a random number generator that is effective, secure and lightweight, it is important to be very familiar with the characteristics of both the hardware and the environment, and also, to create an effective algorithm that produces random bit sequences and seeds from the hardware sources. To achieve this, the temperature sensor on the WISP passive RFID tag has been used to produce the initial seed. The production of this random number generator was made possible by exploiting the xorshift+ algorithm, which is lightweight in terms of resource utilization, and experiments were carried out until suitable results were obtained.

Our proposed HRNG has a PRNG that is a member of "Mersenne Twister" (MT) [31], well-known feedback shift register superclass, and a TRNG that uses temperature sensor as the hardware source.

3.1 xorshift+

Using addition operation instead of using multiplication makes non-linear transformation faster. Saito and Matsumoto proposed this idea in their XSadd generator. This generator adds two consecutive outputs of an underlying xorshift generator based on 32-bit shifts [32].

XSadd fails several BigCrush tests so Vigna introdued xorshift+ family. This family is based on 64-bit logical shift operations. For example, the code shown in Figure 1 belongs to xorshift128plus generator. It uses 128 bits of state. It is one of the well-known member of the xorshift+ family.

```
struct xorshift128plus{   uint64_t a, b;  };

/* The state must be seeded so that it is not all zero */

uint64_t xorshift128plus (struct xorshift128plus state *state) {
            uint64_t t = state->a;
            uint64_t const s = state->b;
            state->a = s;
            t ^= t << 23;                         // a
            t ^= t >> 17;                         // b
            t ^= s ^ (s >> 26);                   // c
            state->b = t;
            return t + s;
}
```

Figure 1. xorshift128plus C implementation.

3.2 WISP

Wireless identification and sensing platform (WISP) is a passive RFID device with a microcontroller and sensors. WISP was first developed by Intel Research Seattle. Then studies continued at the Sensor Systems Laboratory of the University of Washington. It is a passive tag so doesn't have any built-in battery. Energy that is required for both powering the sensors and sending the response to the RFID reader was harvested from the radio signals sent by the RFID Reader. WISP can be the EPC Gen1 or Gen2 tag [33]. There is a 16-bit lightweight microcontroller on it. For example, Wisp 5.0 has Texas Instruments MSP430FR5969 microcontroller. This tag also includes 1 accelerometer, 1 temperature sensor, analog digital converter (ADC) and 1 light emitting diode (LED). It has system clocks running on different frequencies.

3.3 Methods

Figure 2 shows the 16-bit true random number generation flow chart with a temperature sensor, and an ADC. The ADC produces 12-bit values. The least significant bit (LSB) of the 12-bit values is set as the random number's first bit, and the random number is shifted logically to the left.

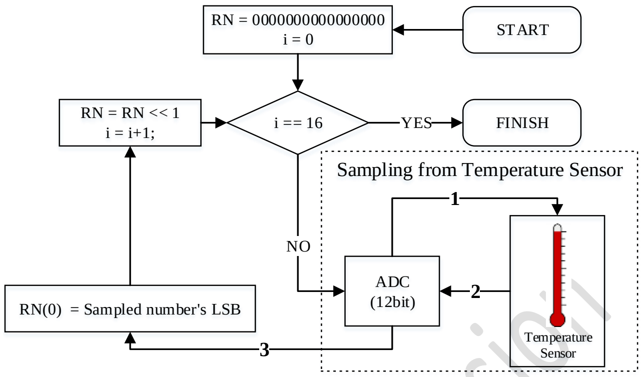

Figure 2: True random number generator flow chart [34].

The process given in Figure 2 continues for 16 samplings, thus generating a 16-bit true random number. 16-bit true random number generation steps can be seen in Table 1.

Table 1: A sample 16-bit true random number generation steps.

i	LSB of the sampled value from ADC	RN16
0	1	0000000000000001
1	0	0000000000000010
2	1	0000000000000101
3	1	0000000000001011
4	0	0000000000010110
5	0	0000000000101100
6	0	0000000001011000
7	1	0000000010110001
8	0	0000000101100010
9	1	0000001011000101
10	1	0000010110001011
11	1	0000101100010111
12	0	0001011000101110
13	1	0010110001011101
14	1	0101100010111011
15	0	1011000101110110

In the scope of our study, we changed the original xorshift128+ by random scramblings made by our predictions. We generated many PRNGs with fewer shift operations and shorter seeds compared with the original xorshift128+ and selected the one having the best NIST STS results. xorshiftL+ algorithm was presented in 2018. In that study, a lightweight HRNG called xorshiftL+ is mentioned. The authors claimed that this HRNG passes NIST STS tests [35]. We extended this study and made tests, comparisons. This newly created HRNG produces 32-bit random numbers. C programming language implementation of the proposed HRNG tested on WISP RFID is shown in Figure 3.

```
int32_t x = TRNG (); // Uses WISP's built-in temperature sensor
int32_t xorshiftULplus (int32_t s) {
            int32_t y = s + x;
            x = x ^ (x << 3);
            x = x ^ (s) ^ (x >> 5) ^ (x >> 2);
            return y;
}
```

Figure 3: xorshiftUL+ C implementation.

Firstly, a 16-bit random number is generated by sampling from WISP's built-in temperature sensor. This number is given to PRNG as a seed. PRNG adds x and s values with mathematical addition operation and stores the result to y. Afterwards, x value is shifted to the left 3 times and this shifted value is taken into XOR operation with its previous status. The overall result is stored on x again. Then, the value of x is shifted 5 times to the right and 2 times to the right separately. s, x and these two new shifted values are all taken into XOR operation and the result is stored on x. As a result, the y = s + x value in the first step is returned. We want operations within the algorithm to change the y value in the next cycle so we return y value which is calculated in this first step. In this way, it will ensure randomness at every step.

4 Experimental results and discussion

In this section, the randomness of the numbers generated by the proposed random number generator is evaluated. NIST, and ENT tests results and EPCGen2 security requirements were examined and evaluated. At the same time, run time comparison on WISP passive RFID and the known rival algorithms were shared and evaluated.

4.1 NIST Results

The new HRNG was tested using the well-known test suite, NIST STS, which has a battery of statistical tests, such as Rank, Runs, Serial, Frequency, FFT, etc. It makes 188 runs for 15 different tests by running certain tests with different parameters [36]. In each round 64 million bits (2 million 32-bit numbers) are generated by the HRNG and tested with NIST STS. This is repeated tens of times. The output of the random number generator is well distributed statistically if it's serial correlation results are near zero and the output has high entropy. This and other properties ensure that the generator passes the NIST STS tests. NIST STS results show that the new HRNG is successful according to test results, and also that HRNG produces statistically random number sequences. NIST STS results of the one of the generated bit sequences are given in Table 2. Detailed NIST test results and generated random numbers can be seen on http://srg.cs.deu.edu.tr/publications/2019/xorshiftULplus/index.htm.

Table 2: NIST STS results for xorshiftUL+.

STATISTICAL TEST	P-Value	Proportion
Frequency	0.253551	1.000000
BlockFrequency	0.011250	1.000000
CumulativeSums(Average)	0.236986	1.000000
Runs	0.148094	0.937500
LongestRun	0.213309	1.000000
Rank	0.671779	1.000000
FFT	0.739918	1.000000
NonOverlappingTemplate (Average)	0.427527	0.987542
OverlappingTemplate	0.468595	1.000000
Universal	0.407091	1.000000
ApproximateEntropy	0.407091	0.906250
RandomExcursions (Average)	0.178357	1.000000
RandomExcursionsVariant (Average)	0.254208	1.000000
Serial (Average)	0.264708	0.984375
LinearComplexity	0.100508	1.000000

The NIST test package contains 15 different tests based on hypothesis testing. These tests are designed to measure a specific Null hypothesis. It attempts to determine whether a sequence of zeros and ones is random. As a result of these tests, a probability value called p-value is calculated for each test. P-value is compared with a level of significance (a) value. If p-value > a, the generated sequence is considered statistically random. If p-value < a, the test will fail. This "a" value is selected in the range [0.001, 0.01]. So we can see that all tests are successful according to p-values. On the other hand, except for the random excursion (variant) test, proportion value for all tests should be at least 29 out of 32 (0.875). For random excursion (variant) test, this value should be at least 23 in 25 (0.92). All proportion values are also successful for xorshiftUL+.

4.2 Time and operator comparison results

xorshiftUL+ was tested with NIST STS test suite and compared with well-known algorithms listed in Table 3. All of the algorithms ran on WISP and personal computer (PC). Table 3 shows that xorshiftUL+ is the fastest on both WISP and PC. In the time evaluation of the xorshiftUL+ algorithm, only the PRNG time value is calculated and TRNG is excluded because it works once at the beginning of the algorithm to produce a seed, and the time required is negligible. xorshiftUL+ generates random numbers about 16.24% faster than the closest rival on WISP environment and about 15.05% faster on PC environment. It also outperforms the rival algorithms in terms of number of operators used and variable lengths. Compared with the xorshift128+ algorithm, it is seen that the operations in the xorshift128+ algorithm are performed on 64-bit length numbers. It is also observed that there are 1 mathematical addition operation, 23 logical left shifts, 23 logical right shifts and 4 XOR operations in this algorithm. Likewise, xorshift128+, all operations in the xorshiftR+ algorithm are 64 bits length variables. xorshiftR+ has 1 mathematical addition, 23 logical left shift, 17 logical right shift and 2 XOR operations. In our new proposed xorshiftUL+ algorithm, there are 1 mathematical addition, 3 logical left shifts, 7 logical right shifts and 4 XOR operations. This new algorithm has been developed to ensure the faster generation of numbers by applying less processing with 32 bit variables.

Table 3: Time comparison with the known PRNG algorithms.

PRNG algorithm	Elapsed time on WISP (ms)	Elapsed time on PC (ms)
Uprng (1000 numbers generated)	35695	0.437100
AKARI 1(1000 numbers generated)	39873	0.227833
AKARI 2(1000 number generated)	25022	0.171398
xorshiftUL+(10000 number generated)	1934	0.023285
xorshiftR+(10000 numbers generated)	2309	0.027410
xorshift128+(10000 numbers generated)	3183	0.033485
xorshift1024+(10000 numbers generated)	3307	0.043265
xorshift64*(10000 numbers generated)	3868	0.040750
xorshift1024*(10000 numbers generated)	4001	0.043925
well1024(10000 numbers generated)	5242	0.041230
LFSR113(10000 numbers generated)	6489	0.058850
LFSR258(10000 numbers generated)	10545	0.134425

4.3 EPCGen2 security requirements

There are three conditions to satisfy security level that are specified by the EPCGen2 standard. These are:

- Probability of a single RN16 shall be bounded by [37]:

  $0.8 / 2^{16} < P\ (RN16=j)) < 1.25 / 2^{16}$

- This condition is met when maximum 10000 tags are considered and the condition is not dependent to the revival time of the tags. Generating same 16 bits random numbers shall have a probability less than 0.1% for two or much more tags.

- To meet this condition a 16-bit random number is predicted with a probability not bigger than 25×10^{-3} % If the RNG's outcomes of the prior draws, performed under identical conditions, are known.

XorshiftUL+ was also checked for EPCGen2 standards that are given above. Three conditions of EPCGen2 standard as pointed out in the EPC™ Gen-2 Class 1 document [38] were examined, to ensure that it meets the RFID tags' security standards.

Firstly, we know that each 16-bit random number selected (RN16 16-bit random number) from generated 2^{30} numbers should have the probability of $0.8/2^{16} < P(RN16) < 1.25/2^{16}$. Proposed HRNG should satisfy this condition. We generated numbers and checked the result. xorshiftUL+ satisfies this condition with the probability of $0.917/2^{16} < P(RN16) < 1.048/2^{16}$ for 2^{30} numbers.

Secondly, simultaneously identical sequences' probability for 10000 tags should be less than 0.1%. xorshiftUL+ has two inputs, and these two seeds are 32-bit integers, so calculating the probability as $(1/2^{32}) \times (1/2^{32}) = 1/2^{64}$, the result is $[10000 \times (1/2^{64})] * 100 = 5.42\% \times 10^{-18} < 0.1\%$, this condition is also satisfied.

Table 4: ENT test results for xorshiftUL+

Test	**(1st run seeds)**		**(2nd run seeds)**		**(3rd run seeds)**	
	x (int32_t)	**s (int32_t)**	**x (int32_t)**	**s (int32_t)**	**x (int32_t)**	**s (int32_t)**
	1,976,529,755	-1,089,211,351	-2,112,299,155	438,506,025	154,5271,002	269,685,289
Entropy	1.000000 bits/bit		1.000000 bits/bit		1.000000 bits/bit	
Compression Rate	%0		%0		%0	
X^2 Statistic	0.23 %62.84		1.56 %21.18		0.08 %78.22	
Arithmetic Mean	0.5001		0.4998		0.5000	
Monte Carlo π	3.158355158 (0.53% error)		3.160407160 (0.60% error)		3.157515158 (0.51% error)	
Serial Correlation Coefficient	-0.005478		-0.005967		-0.005634	

The final condition is that an RN16 drawn from a tag's RNG shall not be predictable with a probability greater than 0.025%. This was proved using an ENT test suite detailed and defined on www.fourmilab.ch/random [39]. The detailed results and test parameters can be seen in Table 4.

The ENT test package performs some tests using the file including the numbers generated by the random number generators. The results of files produced with different seeds by xorshiftUL+ can be seen in Table 4. The entropy value of this test is maximum 1. This is because each character is represented by a single bit. Since the entropy value is close to 1, the compression ratio is also close to zero. These two values mean that the result is good.

The result of the chi-square test is expected to be between 10% and 90%. As it shown in the table, these test results are also satisfactory.

The perfect value for the arithmetic mean is 0.5. For xorshiftUL+, these values are very close to 0.5. Monte Carlo value should be close to Pi value. As we see in the table, the values are very close to the Pi value and the error percentage is low.

The Serial correlation coefficient value that queries the relationship of a byte in a sequence with the previous bytes should be close to zero. It is not important whether this value is negative or positive. xorshiftUL+ produced close to zero output based on this test result.

5 Conclusion

This study presents a proposal for a solution for existing and pending security challenges on lightweight and ultra-lightweight devices in consideration of resource and time constraints. New solutions are offered for random number generation which are the key points of security for lightweight devices. The WISP passive RFID tag confirming to EPC Gen2 standard was used as the ultra-lightweight device to conduct tests and experiments. WISP has built-in sensors, 256-bit AES encryption and can be programmed, and so was selected for its high usability and applicability in the scope of future technologies. Marsaglia's well-known Mersenne Twister based xorshift random number generator was modified to produce a new PRNG. To initialize the PRNG, WISP RFID tag's temperature sensor was used to obtain a true random number by performing 16 temperature samplings in a cycle. Our newly created HRNG was proposed as a combination of a PRNG and a TRNG. The time required for random number generation using the HRNG was estimated and compared with some of the previous well-known random number generators shown in Table 3 in the previous section. A similar approach was taken to investigate whether the random number generator satisfies 3 conditions for EPC™ Gen-2 Class 1 standards. Finally, the quality of the random number series generated by the HRNG was examined using the NIST STS. These tests and evaluations revealed that the new HRNG satisfies 3 conditions for EPC™ Gen-2 Class 1 standards, passes all NIST STS tests and generates random numbers approximately 16% faster than the closest rival.

For future works, xorshiftUL+ can be implemented on different IoT devices and the results investigated in terms of time, resource and performance requirements.

6 References

[1] Beaulieu R, Shors D, Smith J, Treatman-Clark S, Weeks B, Wingers L. "The SIMON and SPECK families of lightweight block ciphers". 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), IEEE, San Francisco, CA, USA, 7-11 June 2015. Doi: 10.1145/2744769.2747946.

[2] Canniere C D, Dunkelman O, Kneˇzevi´c M. "KATAN and KTANTAN – a family of small and efficient hardware-oriented block ciphers". International Workshop on Cryptographic Hardware and Embedded Systems CHES 2009: Cryptographic Hardware and Embedded Systems, Springer, Lausanne, Switzerland. 272-288, 2009. Doi: 10.1007/978-3-642-04138-9_20.

[3] Guo J, Peyrin T, Poschmann A, Robshaw M. "The led block cipher". 13th International Workshop Cryptographic Hardware and Embedded Systems–CHES 2011, Nara, Japan. 326-341, 2011a. Doi:10.1007/978-3-642-23951-9_22.

[4] Gong Z, Nikova S, Law Y W. “A new family of lightweight block ciphers”. Juels A., Paar C. (eds) RFID. Security and Privacy. RFIDSec 2011. Lecture Notes in Computer Science, vol 7055. Springer, Berlin, Heidelberg. 1-18, 2012. Doi:10.1007/978-3-642-25286-0_1.

[5] Knudsen L, Leander G, Poschmann A, Robshaw MJB. “PRINTcipher: a block cipher for ic-printing”. 12th International Workshop Cryptographic Hardware and Embedded Systems, CHES 2010, Santa Barbara, USA. 16–32, 2010. Doi:10.1007/978-3-642-15031-9_2.

[6] Shibutani K, Isobe T, Hiwatari H, Mitsuda A, Akishita T, Shirai T. “Piccolo:an ultra-lightweight blockcipher”. International Workshop on Cryptographic Hardware and Embedded Systems CHES 2011: Cryptographic Hardware and Embedded Systems, Springer, Nara, Japan. 342–357, 2011. doi : 10.1007/978-3-642-23951-9_23.

[7] Wu W, Zhang L. “LBlock: a lightweight block cipher”. International Conference on Applied Cryptography and Network Security ACNS 2011: Applied Cryptography and Network Security, Springer, Nerja, Spain. 327–344, 2011. Doi: 10.1007/978-3-642-21554-4_19.

[8] Aumasson JP, Henzen L, Meier W, Naya-Plasencia M. “Quark: a lightweight hash”. Journal of Cryptology, 26(2), 313–339, 2013. Doi: 10.1007/s00145-012-9125-6.

[9] Bogdanov A, Kneˇzevi´c M, Leander G, Toz D, Varıcı K, Verbauwhede I. “SPON-GENT: a lightweight hash function”. International Workshop on Cryptographic Hardware and Embedded Systems CHES 2011: Cryptographic Hardware and Embedded Systems, Springer, Nara, Japan. 312–325, 2011. Doi: 10.1007/978-3-642-23951-9_21.

[10] Guo J, Peyrin T, Poschmann A. “The PHOTON family of lightweight hash functions”. Annual Cryptology Conference CRYPTO 2011: Advances in Cryptology, Springer, Santa Barbara, CA, USA. 222–239, 2011b. Doi: 10.1007/978-3-642-22792-9_13.

[11] Turner N. “Software vs. hardware RNG’s”. iGaming Business Magazine 2pp. issue 55, http://www.igamingbusiness.com/sites/default/files/file/March_%20April%202009/28-29_Mar_Apr09.pdf, 2009. (30.09.2019).

[12] Fischer V, Bernard F. “True random number generators in FPGAs”. Badrignans, B., Danger, J., Fischer, V., Gogniat, G., Torres, L. (eds) Security Trends for FPGAS From Secured to Secure Reconfigurable Systems, Dordrecht. 101-135, 2011. Doi: 10.1007/978-94-007-1338-3_5.

[13] Marsaglia G. “Xorshift RNGs”. Journal of Statistical Software 8(14), 2003. Doi: 10.18637/jss.v008.i14.

[14] Chae H, Salajegheh M, Yeager D, Smith J R, Fu K. “Maximalist cryptography and computation on the WISP UHF RFID tag”, Wirelessly Powered Sensor Networks and Computational RFID, 175–187, 2013. January 2013.

[15] Sample A, Yeager D, Powledge P, Smith J. “Design of a passively-powered, programmable sensing platform for UHF RFID systems,” IEEE International Conference on RFID, 149–156, March 2007.

[16] Smith J R, Sample A P, Powledge P S, Roy S, Mamishev A. “A wirelessly-powered platform for sensing and computation” UbiComp 2006 Ubiquitous Computing Lecture Notes in Computer Science, vol. 4206, 495–506, September 2006.

[17] Wisp5, Retrieved from http://wisp5.wispsensor.net/, (20.09.2019).

[18] Avaroğlu E, Koyuncu I, Özer A B, Türk M. “Hybrid pseudo-random number generator for cryptographic systems”, Nonlinear Dynamics 82(1-2), 239-248, 2015.

[19] Koç Ç. Cryptographic Engineering. Springer, New York, 2009.

[20] Avaroğlu E. “Pseudorandom number generator based on Arnold cat map and statistical analysis”, Turkish Journal of Electrical Engineering & Computer Sciences, 25(1), 633-643, 2017.

[21] Arnold VI Avez A “Problemes Ergodiques de la Mecanique Classique”. Science 1968(159), 1344-1344. (in French), 1968.

[22] Koyuncu İ, Özcerit AT. “The design and realization of a new high speed FPGA-based chaotic true random number generator”. Computers & Electrical Engineering, 58, 203-214, 2017.

[23] Çabuk U C, Aydın Ö, Dalkılıç G. “A random number generator for lightweight authentication protocols: xorshiftR+”. Turkish Journal of Electrıcal Engıneerıng & Computer Sciences 25(6), 4818-4828, 2017. Doi:10.3906/elk-1703-361.

[24] Kösemen C, Dalkılıç G, Aydın Ö, “Genetic programming based pseudorandom number generator for wireless identification and sensing platform”. Turkish Journal of Electrıcal Engıneerıng & Computer Sciences, 26(5); 2500-2511, 2018. Doi: 10.3906/elk-1710-155.

[25] Lawnik M. “Generation of pseudo-random numbers with the use of inverse chaotic transformation”. Open Mathematics formerly Central European Journal of Mathematics 16(1); 16-22, 2018. Doi: 10.1515/math-2018-0004.

[26] Rose GG. “KISS: a bit too simple”. Cryptography and Communications 10(1), 123-137, 2018. Doi: 10.1007/s12095-017-0225-x.

[27] Marsaglia G, Zaman A. “The KISS generator”. Technical Report, Department of Statistics, Florida State University, Tallahassee, FL, USA, 1993.

[28] Alcin M, Koyuncu I, Tuna M, Varan M, Pehlivan I. “A novel high speed Artificial Neural Network–based chaotic True Random Number Generator on Field Programmable Gate Array”. International Journal of Circuit Theory and Applications 47(3), 365-378, 2019.

[29] Rahmat R F, Ramadhana S, Faza S, Fawwaz I, Nababan E B. “Implementation of Vector Algebra and The Hybrid Pseudo Random Number Generator in Android Game of Kuaci”. Journal of Physics: Conference Series, 1235(1), 012089, IOP Publishing, 2019.

[30] Devi DI, Chithra S, Sethumadhavan M. “Hardware Random Number Generator Using FPGA”. Journal of Cyber Security and Mobility 8(4), 409-418, 2019.

[31] Matsumoto M, Nishimura T. “Mersenne twister: a 623-dimensionally equi-distributed uniform pseudo-random number generator”. ACM Transactions on Modeling and Computer Simulation (TOMACS) 8(1), 3-30, 1998. Doi: 10.1145/272991.272995.

[32] Saito M, Matsumoto M. "XORSHIFT-ADD (XSadd): A variant of XORSHIFT". http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/XSADD/, 2014.

[33] Smith JR. “History of the WISP Program”. Wirelessly Powered Sensor Networks and Computational RFID, 13-29. Springer, New York, NY, 2013.

[34] Dalkılıç G. “Radyo frekansı ile tanımlama etiketleri için gerçek rastgele sayı tabanlı üreteç”. Dokuz Eylül Üniversitesi Fen ve Mühendislik Dergisi 18(54/3), 640-651, 2016. Doi: 10.21205/deufmd.2016185427.

[35] Aydın Ö, Dalkılıç G. “A Hybrid Random Number Generator for Lightweight Cryptosystems: Xorshiftlplus”. 3rd International Conference on Engineering Technology and Applied Sciences (ICETAS), Skopje, Macedonia, 2018.
[36] Rukhin A, Soto J, Nechvatal J, Smid M, Barker E, Leigh S, Levenson M, Vangel M, Banks D, Heckert A, Dray J, Vo S. “A statistical test suite for random and pseudorandom number generators for cryptographic applications”. NIST Special Publication NIST SP 800-22 Rev 1a, 1-131, 2010. https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecial publication800-22r1a.pdf, (30.09.2019).
[37] Peris-Lopez P, Hernandez-Castro J C, Estevez-Tapiador J M, Ribagorda A. “LAMED—a PRNG for EPC class-1 generation-2 RFID specification”. Computer Standards & Interfaces, 31(1), 88-97, 2009.
[38] EPCGlobal GS1. “EPC™ radio-frequency identity protocols generation-2 UHF RFID specification for RFID air interface protocol for communications at 860 MHz – 960 MHz version 2.0.1 ratified”, 1–152, April 2015, https://www.gs1.org/sites/default/files/docs/epc/Gen2 _Protocol_Standard.pdf, (30.09.2019).
[39] Walker J.” ENT - a pseudorandom number sequence test program”, Fourmilab. http://www.fourmilab.ch/random /, (30.09.2019).