Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2018) 26: 2500 – 2511

doi:10.3906/elk-1710-155

Genetic programming-based pseudorandom number generator for wireless identification and sensing platform

Cem KÖSEMEN, Gökhan DALKILIÇ, Ömer AYDIN*
Department of Computer Engineering, Faculty of Engineering, Dokuz Eylül University, İzmir, Turkey

Received: 16.10.2017 • **Accepted/Published Online:** 29.04.2018 • **Final Version:** 28.09.2018

Abstract: The need for security in lightweight devices such as radio frequency identification tags is increasing and a pseudorandom number generator (PRNG) constitutes an essential part of the authentication protocols that provide security. The main aim of this research is to produce a lightweight PRNG for cryptographic applications in wireless identification and sensing platform family devices, and other related lightweight devices. This PRNG is produced with genetic programming methods using entropy calculation as the fitness function, and it is tested with the NIST statistical test suite. Moreover, it satisfies the requirements of the EPCGen2 standards.

Key words: Pseudorandom number generation, genetic programming, wireless identification and sensing platform, radio frequency identification, security

1. Introduction

Pseudorandom number generators (PRNGs) are required in many cryptographic applications for fast and effective generation of secret keys and nonce values. Secure communication protocols use random nonce values for authentication purposes, which are very critical parts of these protocols. In this research, a lightweight PRNG is produced to provide a secure end-to-end communication between the wireless identification and sensing platform (WISP) family of devices and their readers. Since this PRNG's main target is lightweight devices, only the mathematical addition operator and the bitwise logical operators are used. Besides WISPs, this new PRNG can be used on any other lightweight device, like various Internet of Things (IoT) devices, which transfer sensitive information that needs to be protected.

A WISP is basically an RFID device with a general-purpose, fully programmable 16-bit microcontroller and customizable external physical sensors like an accelerometer, light sensor, and temperature sensor [1–3]. The WISP uses the power of the reader's emitted radio signals to work, which allows it to run without a battery. The WISP has a processor and it can execute software programs preloaded on it, unlike most of the other passive RFID tags. That feature makes WISPs suitable for implementing security protocols and allows running these protocols easier. The PRNG produced in this work is also programmed onto a WISP and runs on its processor flawlessly.

WISP 5 is the latest version of the WISP device family, developed at the University of Washington laboratories. A WISP 5 device obtained from the University of Washington is used for testing purposes in this work. WISP 5, presented on the website wisp5.wikispaces.com, conforms to the EPC Class 1 Generation 2 (EPC C1G2) standard for ultrahigh frequency (UHF) RFID tags.

*Correspondence: omer.aydin@deu.edu.tr

There are many different ways to generate random numbers. True random number generators (TRNGs) use nondeterministic sources, for example physical sources like temperature or magnetic fields, as their entropy sources and thus generate true random numbers [4]. TRNGs generally have a slow rate of number production. PRNGs generate random numbers with deterministic functions that are basically mathematical formulas [5]. The same seed given to these functions always outputs the same sequence as the result. PRNGs have higher rates of number production, but it is required that PRNGs be seeded with true random numbers.

PRNGs should be tested with appropriate tools to determine their statistical quality (randomness). Currently, there are several types of test software that use different methods. In this work, the NIST statistical test suite (STS) is selected for validating the quality of the generated random numbers, because NIST STS is one of the well-known statistical test suites and has 15 different tests [6–9]. NIST STS applies a set of statistical tests to the binary sequence that is generated by the given PRNG. These tests evaluate the statistical properties, like uniformity of zeroes and ones, repeating patterns, entropy, etc.

Genetic programming (GP) is a method for problem-solving that provides a full or an approximate solution among a set of initial computer programs. This initial set (called the population) of programs or expressions may change depending on the problem domain. For example, if the solution is a good PRNG, the possible population consists of PRNG functions that are formed by numbers, inputs, and mathematical or logical operations. Each individual of the population has a fitness value based on how well it solves the problem.

In the initial generation, the randomly generated set of individuals has low fitness scores as expected. After generations pass by applying selection, crossover, and mutation operations to the individuals, better generations with higher fitness values are obtained and so the population is improved. If any of the individuals reach the aimed fitness score in any specific generation, that individual is considered to be the solution to the problem [10,11].

2. Related works

There are some previous studies in which the GP methods were used to generate random numbers or PRNGs. One of the first works for producing a PRNG with a genetic algorithm (GA) was Koza's work that produced a randomizer [12]. That randomizer converts a sequence of consecutive numbers to random numbers; hence, it does not rely on a recursive seed. As its fitness function, it uses a bit entropy calculation method, where binary sequence outputs with higher entropy are considered better. Bit entropy calculation as the fitness function is also used in this work. Koza's work includes some simple statistical test results, like frequency and gap tests, but not a complete suite like NIST STS. It also uses costly arithmetic operations like modulus and division, which can be a problem in lightweight devices of the IoT.

Tomassini et al.'s work generated pseudorandom sequences directly by using a cellular automaton (CA) instead of generating a deterministic PRNG [13]. The GA in this work is used for producing a suitable CA. This GA evolves the cells of a single, nonuniform CA, which has cell rules that are initialized randomly. This CA generator achieved good results within the Diehard test suite.

In Chlumecky et al.'s study, methodology and software using a GA and a newly created hydro random number generator (HRNG) for the optimization of a rainfall-runoff model were introduced [14]. Their new HRNG generates random numbers based on hydrological data and gives better results than PRNGs, as they claimed. Nevertheless, this paper does not include structural or statistical analysis of the HRNG's security.

Çabuk et al. in their work [15], proposed a new xorshift-based lightweight PRNG, called xorshiftR+, which is specially designed for the constrained devices of the IoT and tested on a WISP. The authors also

obtained very satisfying results in terms of speed and randomness from NIST STS and a more comprehensive suite, TestU01.

Ibrahim and Dalkılıç focused on the security of healthcare systems, and especially authentication protocols, in their article [16]. They developed a new mutual authentication protocol using elliptic curve cryptography and the advanced encryption standard (AES) for RFID tags. This protocol was tested and coded on WISP passive RFID tags. Although they claimed that it overcomes the security problem for healthcare systems using RFID tags, they used a WISP 5 built-in random number generator for their authentication protocol. RNGs should be examined structurally and statistically to say that they are secure, and they did not give any information about the security status of that PRNG.

Knežević, in his study [17], stated that evolutionary computation could be used to create PRNGs, which has an important role against side channel attacks. Knežević concluded that the GP method can speed up exponential computation or can be used to break the Merkle–Hellman cryptosystem in the asymmetric key cryptography and also can create Boolean functions or good quality S-boxes for symmetric key cryptography. In this article, the claims were not directly supported by experimental results, and only articles that support the assertion were reviewed.

Challa et al. investigated the latest authentication protocols proposed for implantable medical devices (IMDs) and WISPs in their study [18]. In this paper, they gave comparative details considering IMDs' computation and communication costs and functionality features among the existing authentication schemes. They called attention to some difficulties that need to be mitigated for authentication schemes to be designed for IMDs. Although they mentioned the importance of random number generators in their paper, there was no evaluation about the resource usage and security analysis of those RNGs that have one of the key roles for authentication protocols.

Hernandez-Castro et al.'s work used the avalanche effect and the strict avalanche criterion to measure the fitness of PRNGs [19]. It generates an initial set of random numbers with the Mersenne Twister generator and randomly changes a single bit. After that, it calculates the Hamming distance between these values and the output of their PRNG.

Hernandez-Castro and Tomassini's studies produced good results with Diehard, but they did not contain the NIST STS results, which is a more comprehensive package than the Diehard test package [20]. Moreover, it was mentioned that only successful results were obtained with statistical tests and the results were not given for the measurements and tests performed on any equipment.

LAMED is another project that also uses the strict avalanche criterion as its fitness function [21]. As in our project, a PRNG for low-cost, low-power RFID tags is produced considering the EPC standard, and a hardware implementation has been designed to test that PRNG, too. However, instead of running the algorithm on real hardware, the authors of LAMED preferred to create only a conceptual design.

Khan and Bhatia's work applied a GA on a binary sequence that was already generated and selected a key from that sequence [22]. The coefficient of correlation calculation was used in the fitness function. Even though it has encouraging results, no known statistical tests were provided along with those results.

Leonard and Jackson's high entropy RNG production with single-node genetic programming (SNGP) work was also based on Koza's bit-entropy calculation method [12,23]. It also uses the same operator set that consists of arithmetical operators as in Koza's study. SNGP is a form of GP, which uses graphs and dynamic programming methods that improve efficiency. Although the resulting RNG has some good results, it could not pass all of the NIST STS tests.

Picek et al. generated a deterministic RNG function for lightweight devices with the Cartesian genetic programming method [24]. In this method, the expressions are represented as an indexed graph. The avalanche criterion and approximate entropy tests are used in its fitness function. This work uses a different GA approach for generating the PRNG, but does not provide any testing results.

3. Materials and methods

Producing a PRNG with GP is tricky, because there is no certain testing method that ensures that a PRNG has high quality. There are a vast number of statistical tests that determine different means of quality of the generated sequences from a particular PRNG. Because of that, different fitness calculation methods according to these tests were tested in previous works.

Shannon entropy calculation [25] used in this work is one of these suitable statistical testing methods. Binary sequences with higher entropy are statistically well distributed and tend to appear more random.

The GA is a suitable method for generating a PRNG since bunches of mathematical/logical expressions that can be used as PRNG functions are essentially elements of a set with infinite members. Selecting a suitable PRNG function from that set can be done using a heuristic method such as the GA.

3.1. Genetic algorithm initial population

The GA starts with an initial population consisting of randomly generated individuals (chromosomes). In this work, each individual is a mathematical prefix expression and each token of these prefix expressions is a gene of the GA. The prefix form is used because it is easier to store and to make operations on.

The optimal value for the population size was determined as 50 because different population sizes bigger than 50 were tested in the GA, but no positive or negative effect was observed on the results. Using a bigger population size also increases the running time of the GA.

For generating the initial population, an operator set must be determined. The operator set used is given in Eq. (1):

$$\mathtt{O} = \{ +,\ \&,\ |,\ \wedge,\ >,\ !\}. \tag{1}$$

The operators are as follows: the addition, bitwise and, bitwise or, bitwise exclusive or, left rotation (left circular shift), and unary bitwise not operators. Other mathematical operators like multiplication, division, and modulus have higher costs and are not implemented for lightweight devices like WISPs, so these operators are not appropriate to be used.

There are also integer numbers and input parameters that are used in the expressions. Input is indicated with the letter J. Integer numbers are selected between 1 and 16 to add variety to the generated PRNGs. We selected these numbers after testing a bigger range of numbers. Increasing the range of the numbers does not affect the results significantly.

3.2. Genetic algorithm steps

The first step in the genetic algorithm is tournament selection [26]. The tournament selection randomly picks a predefined number of individuals from the population and selects the fittest of the picked individuals. In this work, 5 individuals are picked for the tournament selection. Since the population size is 50, 5 is an appropriate number for selecting one individual from that relatively small population. After two tournament selection operations are applied to the population consecutively, two individuals are gathered as the result.

These two individuals picked by the tournament selection are the inputs of the single-point crossover operation. For the single-point crossover, the tree representations of two expressions from Figure 1 exchange parts from randomly selected operator nodes, one from each of them. Basically, they exchange a subtree. The fitter of the new two individuals, which has the higher entropy, is selected as the result. Only the selected operator nodes are swapped between expressions, so the total token count does not change. On this basis, two expressions, Eq. (2) and Eq. (3), are given to the crossover operation as inputs and the tree representations of these expressions given in Figure 1 are formed.

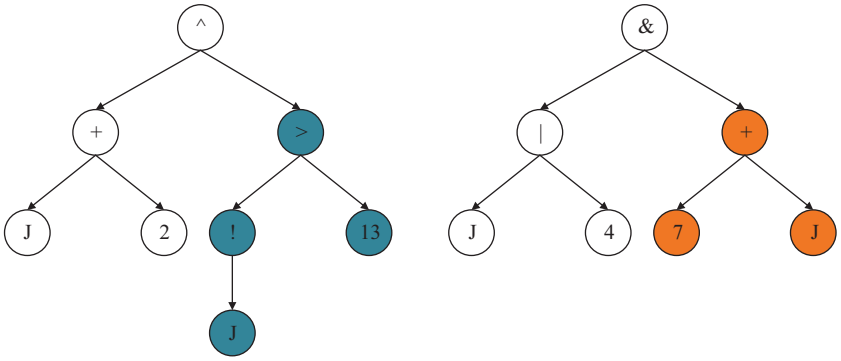

Figure 1. Tree representations of the expressions before the crossover operation.

$$\wedge \ + \ \texttt{J 2} \ > \texttt{! J 13} \tag{2}$$

$$\& \ | \ \texttt{J 4} \ + \ \texttt{7 J} \tag{3}$$

If the "$>$" node of the first tree and "$+$" node of the second tree are selected as the crossover points, after the crossover operation, these two expressions become like in Eq. (4) and Eq. (5). The new trees are given in Figure 2.

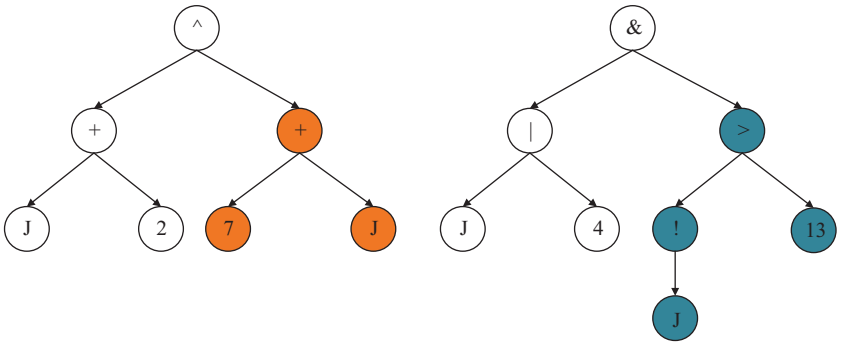

Figure 2. Tree representations of the expressions after the crossover operation.

$$\wedge \ + \ \texttt{J 2} \ + \ \texttt{7 J} \tag{4}$$

$$\& \ | \ \texttt{J 4} \ > \texttt{! J 13} \tag{5}$$

The maximum token count of the expression produced by the crossover operation is limited to 80. That means that there cannot be an expression that has more than 80 tokens as a result of a crossover operation.

After the crossover operation returns a new individual, the mutation operation is applied. The mutation operation only changes one operator of the expression with a probability of 5%. The rate 5% is selected experimentally, but in GAs, the mutation probability is generally low and it is set just to add some variety to the population. For example, if a binary operator is selected as a target of the mutation operation, it is replaced with a new randomly selected binary operator. Finally, this individual becomes a new member for the next population.

Also, the fittest individual is preserved and remains unchanged in the next generation. This method is known as elitism in GP. The pseudocode of the genetic algorithm steps is given in Figure 3.

```
do
  initialize newPopulation
  for i <- 0 to populationSize do
    Elitism(population, newPopulation)
    firstSelection <- TournamentSelection(population)
    secondSelection <- TournamentSelection(population)
    newIndividual <- Crossover(firstSelection, secondSelection)
    Mutate(newIndividual)
    newPopulation.Add(newIndividual)
  end for
  fittest <- newPopulation.fittest
while fittest.fitness <= 18
```

Figure 3. The GP method algorithm.

3.3. Fitness function

The fitness function determines how good the solution is. In our case, the individual with the best entropy value is considered as the fittest. The Shannon entropy method is used for calculating the entropy of the generated binary sequences [25]. It basically calculates the minimum number of symbols to encode the data as in data compression. The formula of Shannon entropy is shown in Eq. (6).

$$H = -\sum_{i=0}^{N} p_i \log_2 p_i \tag{6}$$

The H value is separately calculated for every n-bit entropy value. The logarithmic base is 2, because the data are encoded into binary sequences. p_i is the frequency of the specific character in the binary sequence. Assuming that we want to calculate the 2-bit entropy, first, all possible 2-bit characters' (00, 01, 10, 11) frequencies in the binary sequence are to be calculated. Likewise, 1-bit entropy calculates the frequencies of 0s and 1s. After these $p$values (frequencies) are obtained, $H$is calculated as per the formula. Since the frequencies vary between 0 and 1, we add a minus sign to the formula to negate the negative result.

In our GA's fitness function, 1-bit to 8-bit and 16-bit entropies are calculated, divided by their n-bit entropy values, and then summed. For instance, the 8-bit entropy value is divided by 8, which results in 1 at maximum. The purpose of this division is that every n-bit entropy value should weight the same in the final fitness value. When they are between 0 and 1, each n-bit entropy value has the same effect on the final fitness value. Thus, the maximum entropy value that an individual can get is 9, because every n-bit entropy test returns 1 at maximum. The fitness function calculates the score over two sequences from two different seeds, so the maximum fitness value of an individual becomes 18.

For calculating the fitness of an individual, a predefined seed value is to be given and it generates 2^{18} values. The input and output values of the expressions are 64-bit signed integers. Each output is used as the next step's input, which is similar to other PRNGs. The 2^{18} integers' entropy is calculated for two different seed values and the fitness of the individual is finally determined.

4. Experimental results and discussion

The fittest expression produced with this experiment is shown in Eq. (7) in the prefix form.

$$\begin{aligned} &+\ \mathtt{J}\ +\ +\ \mathtt{J}\ +\ >\ +\ \mathtt{J}\ +\ \wedge\ !\ \mathtt{J}\ 16\ !\ +\ \mathtt{J}\ >\ \mathtt{J}\ 6\ 11\ !\ +\ +\ +\ +\ \mathtt{J}\ +\ \mathtt{J}\ +\ \mathtt{J}\ + \\ &+\ \mathtt{J}\ +\ \mathtt{J}\ +\ >\ \mathtt{J}\ 6\ !\ !\ +\ \mathtt{J}\ >\ \mathtt{J}\ 6\ !\ \wedge\ !\ \mathtt{J}\ 16\ !\ >\ \mathtt{J}\ 4\ +\ \mathtt{J}\ +\ +\ \mathtt{J}\ +\ \mathtt{J}\ +\ > \\ &+\ \wedge\ >\ \mathtt{J}\ 6\ 14\ \wedge\ !\ \mathtt{J}\ 16\ 11\ !\ +\ \mathtt{J}\ >\ \mathtt{J}\ 6\ \wedge\ !\ \mathtt{J}\ 16\ !\ +\ \mathtt{J}\ >\ \mathtt{J}\ 6\ !\ \wedge\ >\ \mathtt{J}\ 6\ 16 \end{aligned} \tag{7}$$

In Figure 4, C implementation of the PRNG algorithm is given.

```
int64_t PRNG(int64_t J)

{

  int64_t a1 = (J >> 6) | (J << 58);

  int64_t a2 = (J >> 4) | (J << 60);

  int64_t a3 = J + J;

  int64_t a4 = ~J ^ 16;

  int64_t a5 = J + a1;

  int64_t a6 = J + a4 + ~a5;

  int64_t a7 = (a6 >> 11) | (a6 << 53);

  int64_t a8 = (a1 ^ 14) + a4;

  int64_t a9 = (a8 >> 11) | (a8 << 53);

  return ~(a1 ^ 16) + a3 + ~(~a2 + a3 + ~a5 + a3 + a3 + ~a4 + a5

 + a5 + a6 + a9) + a7;

}
```

Figure 4. C implementation of the proposed PRNG.

This expression has 98 tokens, where 55 of them are operators, and it has a fitness value of 17.99429. It takes 11 generations in the GA to obtain this solution. However, it is found that the expressions with much

higher entropy do not necessarily have good NIST STS results when tested. Because of that, even if the GA produces expressions with entropy values higher than 17.999, they are not used.

4.1. NIST STS results

NIST STS results for a sequence of 16 million bits generated by the newly obtained expression are given in Table 1. As shown in the table, the produced PRNG successfully passes all of the NIST STS tests. This indicates that the resulting sequence is statistically well distributed, so it provides the feeling and functionality of a random sequence to some extent [6]. The bit-stream count property of NIST STS is given as 32 in this application. The proportion values mean the ratio of the bits in the 32 bit-streams that passed the test. NIST STS evaluates 29 bits and above as successful. That means proportions that are bigger than 29/32 (0.90625) are considered as random by the test suite.

Table 1. NIST STS results for the proposed PRNG.

Test	P-value	Proportion
Frequency	0.804337	1.00000
BlockFrequency	0.299251	0.96875
CumulativeSums	0.671779	0.96875
CumulativeSums	0.949602	1.00000
Runs	0.253551	1.00000
Longest Run	0.804337	1.00000
Rank	0.534146	1.00000
FFT	0.739918	1.00000
NonOverlappingTemplate (Avg.)	0.417842	0.98649
OverlappingTemplate	0.054199	1.00000
Universal	0.949602	1.00000
ApproximateEntropy	0.602458	1.00000
RandomExcursions (Avg.)	0.017591	0.97917
RandomExcursionsVariant (Avg.)	0.032163	0.97839
Serial	0.739918	1.00000
Serial	0.671779	1.00000
LinearComplexity	0.468595	1.00000

4.2. ENT results

Another pseudorandomness testing software is the ENT Test Suite that is defined and detailed at www.fourmilab.ch/random. ENT applies a set of statistical tests to an output file generated by the subject PRNG. In Table 2, the three ENT test results for 128 MB output files that are generated from randomly selected seed values are given.

The entropy result can be 8 at maximum, because each character is encoded in 1 byte that stores 8 bits. Since the calculated entropy values are high, as seen in Table 2, the compression rate is 0%, which is a good result. The ENT Test Suite gives the entropy results with a maximum precision of 6. Since all of the results are very close to 8, they are represented as 7.999999 in Table 2. The ENT Test Suite calculates the chi-square

Table 2. ENT test results for the proposed PRNG.

Test	2,121,363,896 (1st seed)	897,893,119 (2nd seed)	1,732,455,681 (3rd seed)
Entropy	7.999999 bits/byte	7.999999 bits/byte	7.999999 bits/byte
Compression rate	0%	0%	0%
χ^2 statistic	276.17 (17.31%)	255.47 (48%)	255.1035 (48.64%)
Arithmetic mean	127.4947	127.5085	127.5012
Monte Carlo π	3.141592609012 (0.00% error)	3.141444551072 (0.00% error)	3.141423987469 (0.01% error)
Serial correlation coefficient	–0.000100633592	–0.000123993062	–0.00000661128

distribution of the binary sequence, too. The chi-square rate indicates how frequently a random binary sequence would exceed the calculated value. The calculated rate is expected to be in the range of 10% to 90%, which is considered successful. The arithmetic mean is calculated by dividing the sum of all the bytes generated by file length. The perfect score is 127.5, so the generated PRNG has successful results for that value. The Monte Carlo value for the Pi test returns a result that converges to a Pi value. The resulting values are close to Pi with a very small error rate between 0% and 0.01%, which is a good result.

The serial correlation coefficient shows how much a byte in the sequence depends on the previous bytes. It is expected that this value should be close to zero, while it can be either negative or positive. As in Table 2, the generated PRNG has serial correlation coefficient values that are close to zero.

4.3. EPCGen2 randomness requirements

While EPCGen2 does not specifically instruct how a PRNG should be implemented, it expects that any suitable PRNG will pass three statistical tests [27]:

Test 1: The probability of a 16-bit random number selected from the generated sequence of the PRNG must be bounded by $0.8/2^{16}$ and $1.25/2^{16}$ as shown in Eq. (8) and Eq. (9).

$$0.8/2^{16} < P < 1.25/2^{16} \tag{8}$$

$$0.000012 < P < 0.000019 \tag{9}$$

This requirement is basically a 16-bit entropy calculation. Since the generated PRNG outputs are 64-bit numbers, they are divided into four parts to calculate the frequency.

After generating a 512 MB output file, the probability (P) for each number is found between 0.000013 and 0.000017. Since these numbers are within the boundaries, this requirement is satisfied.

Test 2: For 10,000 tags, the probability that more than one tag produces the same sequence must be below 0.1% whenever the tags are energized.

For this requirement, we have to calculate how many different seed values the PRNG can accept. It is calculated as in Eq. (10):

$$\left[10000 \times \left(\frac{1}{2^{64}}\right)\right] \times 100 \cong 5x10^{-14} < 0.1\% \tag{10}$$

Considering that the same sequence cannot be generated with different seeds, this requirement is satisfied by our PRNG as shown in Eq. (10).

Test 3: A 16-bit number selected from the PRNG must not be predictable with a probability better than 0.025%.

This requirement can be checked with serial correlation calculation of the generated sequence by using the ENT Test Suite. It is tested with 128 MB files generated with the PRNG and all results pass the test with success. The obtained results from different seeds are listed in Table 3. The requirement is satisfied since all the sample values are below 0.00025 (0.025%).

Table 3. Serial correlation test results of the proposed PRNG.

Seed value	Serial correlation coefficient
1,824,581,738	0.000023774318
845,522,866	0.000042487019
290,056,763	0.000071459114
685,445,128	0.000041220041
2,115,072,129	0.000010820175

4.4. Time constraint of EPCGen2

The EPCGen2 standard also has a time constraint for generating random numbers. Each tag has 18 ms for encryption [28,29]. With our benchmark, for the test with the WISP 5 device, ten thousand 64-bit numbers are generated in between 7200 and 7300 ms, which means the PRNG is executed 10,000 times. A 64-bit number is thus generated in approximately 0.72 to 0.73 ms, which is clearly acceptable considering the total time slot of 18 ms.

5. Conclusion

There are previous works that generated PRNGs with different GP and fitness methods, but very few of them can practically be used in lightweight devices, like WISP passive RFID tags. In this work, a lightweight PRNG that successfully passes all the tests of the NIST STS is produced with GP methods. This brand-new PRNG consists of performance-efficient operators, like logical bitwise operators and the arithmetical add.

This study shows that a statistically good PRNG can be generated with GP methods using the entropy calculation as its fitness function. With this entropy calculation fitness method, a good PRNG can be produced in a small number of generations.

Unlike most of the other PRNGs generated in the past, the PRNG generated in this work is tested on real hardware and analyzed in terms of resource and time consumptions. It is also tested with EPCGen2 randomness conditions.

For future works, other statistical properties (like in NIST STS) can be combined with bit entropy calculation to improve the fitness function and generate better PRNGs.

Acknowledgment

This study was supported by the Scientific and Technological Research Council of Turkey (TÜBİTAK) with project number 215E225.

References

[1] Smith JR, Sample AP, Powledge PS, Roy S, Mamishev A. A wirelessly-powered platform for sensing and computation. Lect Notes Comp Sci 2006; 495-506.

[2] Sample A, Yeager D., Powledge P, Smith J. Design of a passively-powered, programmable sensing platform for UHF RFID systems. In: IEEE International Conference on RFID; 26–28 March 2007; Grapevine, TX, USA. New York, NY, USA: IEEE. pp. 149-156.

[3] Chae HJ, Salajegheh M, Yeager DJ, Smith JR, Fu K. Maximalist cryptography and computation on the WISP UHF RFID tag. In: Smith JR, editor. Wirelessly Powered Sensor Networks and Computational RFID. New York, NY, USA: Springer, 2013. pp. 175-187.

[4] Stipčević M, Koç ÇK. True random number generators. In: Koç ÇK, editor. Open Problems in Mathematics and Computational Science. Berlin, Germany: Springer, 2014. pp. 275-315.

[5] Park SK, Miller KW. Random number generators: good ones are hard to find. Commun ACM 1988; 31: 1192-1201.

[6] Bassham LE, Rukhin AL, Soto J, Nechvatal JR, Smid ME, Leigh SD, Levenson M, Vangel M, Heckert NA, Banks DL. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. Gaithersburg, MD, USA: National Institute of Standards and Technology, 2010.

[7] Barker E, Kelsey J. Recommendation for Random Number Generation Using Deterministic Random Bit Generators. NIST Special Publication 800-90A. Gaithersburg, MD, USA: NIST, 2014.

[8] Turan MS, Barker E, Kelsey J, McKay KA, Baish ML, Boyle M. Recommendation for the entropy Sources Used for Random Bit Generation. NIST Special Publication 800-90B Second Draft. Gaithersburg, MD, USA: NIST, 2016.

[9] Barker E, Kelsey J. Recommendation for Random Bit Generator (RBG) Constructions. NIST Special Publication 800-90C Second Draft. Gaithersburg, MD, USA: NIST, 2016.

[10] Koza JR. Genetic programming as a means for programming computers by natural selection. Stat Comput 1994; 4: 87-112.

[11] Koza JR. Genetically breeding populations of computer programs to solve problems in artificial intelligence. Proc Int C Tools Art 1990; 1: 819-827.

[12] Koza JR. Evolving a computer program to generate random numbers using the genetic programming paradigm. In: Fourth International Conference on Genetic Algorithms; July 1991; San Diego, CA, USA. pp. 37-44.

[13] Tomassini M, Sipper M, Zolla M, Perrenoud M. Generating high-quality random numbers in parallel by cellular automata. Future Gener Comp Sy 1999; 16: 291-305.

[14] Chlumecky M, Buchtele J, Richta K. Application of random number generators in genetic algorithms to improve rainfall-runoff modelling. J Hydro 2017; 553: 350-355.

[15] Çabuk UC, Aydın Ö, Dalkılıç G. A random number generator for lightweight authentication protocols: xorshiftR+. Turk J Electr Eng Co 2017; 25: 4818-4828.

[16] Ibrahim A, Dalkılıç G. An advanced encryption standard powered mutual authentication protocol based on elliptic curve cryptography for RFID, proven on WISP. J Sensors 2017; 2017: 2367312.

[17] Knežević K. Combinatorial optimization in cryptography. In: 40th International Convention on Information and Communication Technology, Electronics and Microelectronics; 22–26 May 2017; Opatija, Croatia. pp. 1324-1330.

[18] Challa S, Wazid M, Das AK, Khan MK. Authentication protocols for implantable medical devices: taxonomy, analysis and future directions. IEEE Consum Electr Mag. 2018; 7: 57-65.

[19] Hernández-Castro JC, Isasi P, Seznec A. On the design of state-of-the-art pseudorandom number generators by means of genetic programming. In: Congress on Evolutionary Computation; 19–23 June 2004; Portland, OR, USA. pp. 1510-1516.

[20] Sıs M, Ríha Z. Faster randomness testing with the NIST statistical test suite. In: Fourth International Conference on Security, Privacy, and Applied Cryptography Engineering; 18–22 October 2014; Pune, India. pp. 272-284.

[21] Peris-Lopez P, Hernandez-Castro JC, Estevez-Tapiador JM, Ribagorda A. LAMED - A PRNG for EPC Class-1 Generation-2 RFID specification. Comp Stand Inter 2009; 31: 88-97.

[22] Khan FU, Bhatia S. A novel approach to genetic algorithm based cryptography. Int J Res Comp Sci 2012; 2: 2249-8265.

[23] Leonard P, Jackson D. Efficient evolution of high entropy RNGs using single node genetic programming. In: Annual Conference on Genetic and Evolutionary Computation; 11–15 July 2015; Madrid, Spain. pp. 1071-1078.

[24] Picek S, Sisejkovic D, Rozic V, Yang B, Jakobovic D, Mentens N. Evolving cryptographic pseudorandom number generators. Lect Notes Comp Sci 2016; 9921: 613-622.

[25] Shannon CE. A mathematical theory of communication. Bell Syst Tech J 1948; 27: 396-399.

[26] Miller BL, Goldberg DE. Genetic algorithms, tournament selection, and the effects of noise. P Soc Photo-Opt Ins 1995; 9: 193-212.

[27] GS1. EPC^{TM} Radio-Frequency Identity Protocols Generation-2 UHF RFID Specification for RFID Air Interface. Lawrenceville, NJ, USA: GS1 EPCglobal Inc., 2013.

[28] Feldhofer M, Dominikus S, Wolkerstorfer J. Strong authentication for RFID systems using the AES algorithm. In: International Workshop on Cryptographic Hardware and Embedded Systems; 11–13 August 2004; Cambridge, MA, USA. pp. 357-370.

[29] Özcanhan MH, Dalkılıç G. Mersenne twister-based RFID authentication protocol. Turk J Electr Eng Co 2015; 23: 231–254.