lmRNG: A Lightweight Pseudorandom Number Generator for Wireless Sensor Networks

Anna Sojka and Krzysztof Piotrowski
System Design Department, IHP, Frankfurt (Oder), Germany

Keywords: Wireless Sensor Network, Pseudorandom Number Generator, Lightweight Security.

Abstract: Wireless sensor networks (WSN) are often used in the areas where the data security is very important. The cryptographic protocols developed for WSN need to be as computationally inexpensive as possible due to the energy and computational constraints of WSN. The same requirements concern also the elements of these protocols, e.g. the random number generator. In this paper we present our work on a pseudorandom number generator for wireless sensor networks. It uses a modification of the Logistic Map, which is adapted to be used in the constrained environment of the WSN. In our approach we combine a non-deterministic seed source with the deterministic function to get the pseudorandom number generator. We present the results of the tests confirming that our approach fulfils the requirements of randomness and is a candidate to be used for cryptographic purposes.

1 INTRODUCTION

Applications using wireless sensor networks (WSNs) are becoming more and more popular. WSNs are composed of multiple tiny devices–sensor nodes–using radio communication. The most common WSN application is monitoring of some environment, collecting and processing the data and providing the results to a control center. WSNs are often used to monitor some phenomena requiring the processed data to be secured, e.g., in military or health applications. And since a sensor node is a microcomputer with constrained energy and computational resources, these constraints have to be taken into consideration, when developing or choosing the protocols for securing the exchanged data.

The cryptographic protocols for WSN often use random numbers, e.g., in the key exchange protocols or in the hashing algorithms. On the one hand, the random number generators have to be lightweight in terms of energy and time needed for computations and, on the other hand, according to the National Institute of Standards and Technology (NIST) the random numbers suitable for cryptographic purposes have to be unpredictable (NIST, 2010). This means that it shall be impossible to predict the next random number on the basis of the previously generated ones and it shall be also not feasible to determine the seed having some random numbers generated from it. NIST provides a test suite helping in evaluating a output of the random number generator. The test suite consists of 15 statistical tests examining the sequences and assessing their randomness.

There are two types of random number generators: Random Number Generators (RNGs) that use non-deterministic source for generating their outputs and Pseudorandom Number Generators (PRNGs) that use seeds to compute the random numbers. The results of PRNG are deterministic and depend on the seed–the input used to initialize the PRNG function. The sequence of pseudorandom numbers can be generated by anyone knowing the seed and the PRNG function. The proper design of the PRNG function ensures that the criteria for randomness are fulfilled.

Since the RNGs not always seem to fulfil the randomness requirements (e.g. time vector) and the seed for a PNRG has to be random and unpredictable, the output of a RNG can be used as a seed for the PNRG (NIST, 2010). Thus, in our *lmRNG* approach, we use a combination of both generator types, as suggested by NIST, but without developing any additional hardware. Our test platform was the IHPNode (K. Piotrowski, 2010) equipped with a MSP430F5438A microcontroller (TI, 2010) that includes an ADC converter. Thus, we use the ADC converter to produce the seeds and a software implementation of modified Logistic Map to generate the pseudorandom numbers.

Logistic Map as a chaotic map is very sensitive

Sojka A. and Piotrowski K..
lmRNG: A Lightweight Pseudorandom Number Generator for Wireless Sensor Networks.
DOI: 10.5220/0004016503580363
In *Proceedings of the International Conference on Security and Cryptography* (SECRYPT-2012), pages 358-363
ISBN: 978-989-8565-24-2

to initial conditions and can be used for generating pseudorandom numbers (T. Stojanovski, 2001). In (V. Patidar, 2009) the authors propose a pseudo random generator based on chaotic Logistic Map. But since the Logistic Map operates on real numbers from the interval $(0,1)$, the computations performed during the generation of random sequences are computationally too expensive for sensor nodes. Our modification of the Logistic Map ensures our approach to be suitable for WSN, since it is based on integer computations, and the statistical tests showed that the randomness criteria are fulfilled.

The design of a computationally inexpensive random number generator that does not require any additional hardware is a very important issue in the area of the wireless sensor networks. In (Y.H. Wang, 2006) the authors propose a random number generator based on single-electron phenomena. It is built of a single-electron transistor and a single-electron trap, thus, additional hardware components are needed. A simple logic operation is used in order to improve the statistical properties of the generated output. The logic is based on bit skipping and bit counting to reduce the information redundancy. Another approach designed for WSN is the tinyRNG (A. Francillon, 2007). The random number generator presented in this paper uses the transmission bit errors as the source of randomness. These bit errors are added to the cryptographic entropy accumulator, which is built using CBC-MAC function. The entropy is used to reseed the key of the Cryptographic Pseudo-Random Number Generator. It is a block cipher encrypting the counter using the key provided at the programming time and updated by reseeding with the value from the entropy accumulator. The output is available to applications through the TinyOS Random interface (TinyOS, 2011). This approach works properly only in case the radio transceiver is on. And one of the main trends in WSN is to keep the sensor nodes in sleep mode whenever possible, i.e., switching the radio off. Another solution is proposed in (G. Lo Re, 2011). This random number generator uses the sensing properties of the wireless sensor network. The main assumption here is that every sensor node in the network is able to perform sensing in order to generate random numbers and these are generated using the ADC converter, buffering and a CMAC algorithm. To avoid possible manipulation attacks, the sensing task is performed by randomly chosen node being the neighbour of the node requesting the random number. Thus, this approach requires additionally that nodes cooperate and communicate to obtain the random numbers, what significantly increases the costs.

In the rest of the paper we describe our approach in detail. We sketch the mathematical background and the requirements that have to be fulfilled by a random number generator. Then we prove the suitability of *lmRNG* for WSN and the results of the tests evaluating the randomness, performed using the NIST suite. Finally, we conclude the paper and draw on future improvements.

2 MATHEMATICAL BACKGROUND

2.1 Dynamical Systems and Mathematical Chaos

A dynamical system is a set of coupled ordinary differential equations which determine how the state of the system evolves over time (L. Kocarev, 2011). For some parameters the dynamical system can exhibit chaotic behaviour. Such a system is sensitive to initial conditions, i.e., small changes of the input parameter cause significant differences in the future values.

2.1.1 Logistic Map

Logistic Map is an example of chaotic map representing a population model and was introduced in (May, 1976). It is a simple recurrence relation of degree two mathematically written as:

$$x_{n+1} = r x_n (1 - x_n) \tag{1}$$

where the real number r is a control parameter and the initial element x_0 is to be defined at the beginning of computations. The elements $x_0, x_1, x_2, \ldots$ are real numbers lying in the interval $(0,1)$.

The Logistic Map is very sensitive to the parameter r, which has to be chosen carefully. There are some values of r resulting in non-chaotic behaviour of the Logistic Map, e.g., according to (L. Kocarev, 2011) for the numbers below 3.57 the chaotic map is in periodic regime.

2.1.2 The Lyapunov Exponent

The Lyapunov Exponent λ is a quantitative measure of the dependence on the initial conditions. We can observe that the system is chaotic, when the value of the Lyapunov Exponent is greater than 0. The Lyapunov Exponent can be estimated using the following expression:

$$\lambda = \ln(r) + \frac{1}{N+1} \sum_{i=1}^{N} \ln(1 - 2x_i) \tag{2}$$

where the real number r is the control parameter in equation 1 and the element x_i is a real number lying in the interval $(0,1)$ and is computed in the $i-th$ iteration of the Logistic Map.

2.2 NIST Test Suite

The NIST testing suite provides a set of statistical tests assessing the randomness of an evaluated sequence. The sequence consists of zeros and ones that represent the binary form of the numbers. The aim of a statistical test is to assess the probability that the null hypothesis (H0) is fulfilled, where H0 means that the sequence is random. Each test is focused on checking different properties of the sequence in order to test H0 and either accept or reject it. For this purpose in each test the randomness statistic is used. The randomness statistic has a distribution of possible values, and mathematical methods determine the reference distribution for it. The reference distribution allows computing a critical value, which determines the boundary between those samples resulting in a randomness statistic that leads to rejecting the H0 and those that lead to accepting the H0. For a given sequence the test statistic value is computed and when it is bigger than the critical value, the H0 is rejected. On the basis of the test statistic the $P-value$ is computed. The method for computing the $P-value$ is chosen for each test independently and for this purpose a special function is used. It can be for example the error function or the gamma function. It is the probability, that a perfect random number generator would have produced a sequence less random than the sequence that was tested. If the $P-value=1$ then the sequence appears to be random. For the NIST test suite the $P-value$ has to be larger than 0.01 to accept the null hypothesis. The number $\alpha=0.01$ is called the significance level.

2.2.1 The Testing Strategy

For a single generator a set of binary sequences is generated and evaluated by the set of tests. After the binary sequences have been tested, there are two approaches for interpreting the empirical test results. In the first approach the proportion of the sequences that passed the test, i.e. having $P-value>0.01$ is computed. The proportion is computed on the basis of the number of sequences to be tested and the sequences that passed the test. Let 1000 be the first number, and 998 the second one. Then the proportion is equal to 0.998 and the range of acceptable proportions is computed using the formula:

$$(1-\alpha)\pm 3\sqrt{\frac{(1-\alpha)\alpha}{m}} \qquad (3)$$

where m is a sample size and α is the significance level. Thus, for 1000 sequences the proportion should be larger than 0.9805607.

In the second approach the distribution of the $P-values$ is determined in order to check the uniformity. For that the $P_U-value$ of all the $P-values$ is computed. The $P_U-value$ is computed using the special function as defined in the NIST suite. If $P_U-value\leq 0.0001$, then the $P-values$ can be considered to be uniformly distributed.

3 OUR APPROACH

The *lmRNG* was developed for wireless sensor nodes to be used in lightweight cryptographic protocol: shortECC (A. Sojka, 2010), that operates on 32-bit long numbers.

3.1 Modification of the Logistic Map

Due to its simplicity and chaotic behaviour, the Logistic Map is a perfect candidate for a function generating random bits (May, 1976), (L. Kocarev, 2011). The main disadvantage of the Logistic Map is that it operates on real numbers, what is computationally expensive for resource constrained devices. Computations on integers are performed faster than those performed on real numbers (see Section 4). Additionally, on our microcontroller, the representation of real numbers is restricted to 7 digit precision, what can cause periodicity and lead to loss of chaotic behaviour of the Logistic Map. Thus, for our approach we used a modified Logistic Map with integers instead of real numbers. We proved that for this modification the Lyapunov Exponent is greater than 0. Our chaotic function is given by the following equation:

$$x_{n+1}=(x_n(1+x_n) \mod 2^{32})+1 \qquad (4)$$

And since the computations are performed on the integers greater than 0, the following special case has to be considered:

$$\text{if} \quad (x_n(1+x_n) \mod 2^{32})+1=0 \quad \text{then} \quad x_{n+1}=1. \qquad (5)$$

3.2 The Seed

According to the NIST recommendation (NIST, 2010) we decided to use a non-deterministic source for generating the seeds for the *lmRNG* and we have chosen the ADC module. The ADC module is fed with the on-chip temperature sensor and we use this integrated hardware combination to generate the seed.

The seed consists of 32-bits and each bit is generated using an independent measurement of the temperature sensor. To generate the seed we take the least significant bit of each measurement, because it is the most sensitive bit, i.e., even small oscillation of the temperature influences its value. We have also evaluated such a sequence of bits to check if it passes the NIST tests–not all the tests were passed. Thus, we decided to use the source of bits only for the non-deterministic seed. To improve the quality of the seed, the entropy distillation process can be performed (Pareschi, 2006), but this issue is out of scope of this work.

3.3 *lmRNG* Pseudorandom Number Generator

The *lmRNG* pseudorandom number generator uses two modified Logistic Maps that need to have different seeds as input.

$$x_{n+1} = x_n(1+x_n) \mod 2^{32}+1 \quad (6)$$

$$y_{n+1} = y_n(1+y_n) \mod 2^{32}+1 \quad (7)$$

In order to generate a n-bit long number z, n iterations of both maps are performed. In each iteration the outputs of both maps are compared and on the basis of this comparison the i-th bit of the number z is generated as follows:

$$\text{if} \quad (x_i > y_i) \text{ then} \quad z_i = 1, \text{ otherwise} \quad z_i = 0. \quad (8)$$

Our experiments have shown that using only a single modified Logistic Map to reduce computations and generating the bit streams by taking from the iteration results different combinations of bits, caused the generated bit sequences to have poor quality and, as a result, to fail the NIST tests. Additionally, the final method for determining the value of the i-th bit was proposed after a number of experiments. The combination of two maps with the decision based on the result of the comparison of their outputs causes the approach to have the desired properties, i.e., the output is random. Further, using two maps, each using a 32-bit seed as input increases the number of possible PRNG sequences. This reduces the chance of a brute force attack to recover the seed on the basis of the generated numbers.

4 EVALUATION

First, in order to evaluate our approach, we measured the time needed by the microcontroller for computing the iterations of the Logistic Map using different data types, i.e., the float numbers as it is used in the original form of the Logistic Map and for integers as we proposed in our modification. Table 1 presents the clock frequency independent results of our measurements and shows that even without the support of the hardware multiplier the computations on the integers need only the half of the clock cycles than those performed on floats. The hardware multiplier (HM) allows for six times faster integer computations than in case of floats. In comparison to the approach proposed in (V. Patidar, 2009), where the floats where used, our approach is more suitable for WSN. And the presented numbers of clock cycles required for integer multiplication can be optimized further, since the multiplication result is divided modulo 2^{32} and thus only a partial 32-bit multiplication is needed.

Table 1: Clock cycles needed for Logistic Map iterations performed on MSP430F5438A.

Data Type	1 Iteration	32 Iterations
float	522	16165
uint32	282	8485
uint32+HM	83	2117

4.1 Randomness Testing

For the randomness assessment of *lmRNG* we have generated 1000 binary sequences, each 1000000-bit long. For each sequence we have computed the Lyapunov Exponent, which was always greater than 0. It proves that our modified Logistic Map behaves chaotically as well. Further, all the sequences were tested using the NIST testing suite.

4.1.1 Uniform Distribution of P-values

The Tables 2 and 3 present the results of NIST tests performed on the random bit streams generated by the *lmRNG*. As mentioned in Section 2.2.1, for the 1000 tested sequences, the proportion of passed tests should be larger than 0.9805607. All the tests were passed, what means that the *lmRNG* generates pseudorandom numbers. The approach presented in (Y.H. Wang, 2006) was also tested using the NIST tests and it has been shown that some of the tests were not passed. Besides, the authors did not give the clear information about the number of sequences which were tested, what makes the two approaches hard to compare.

4.2 *lmRNG* Versus TinyRNG

The Table 4 presents the comparison of the *lmRNG* with TinyRNG. The *lmRNG* generator does not need any initialization, what in case of TinyRNG takes

Table 2: Results of Non-Parametrized Tests.

Statistical Test	**Distribution of P-Values**	**Proportion of passed tests**
Frequency (Monobit) Test	0.500279	0.9910
Cumulative Sums Test		
forward sums	0.668321	0.9920
reverse sums	0.340858	0.9920
Runs Test	0.299736	0.9880
Test for the Longest Run of Ones in a Block	0.914025	0.9850
The Binary Matrix Rank Test	0.834308	0.9950
The Discrete Fourier Transform (Spectral) Test	0.254411	0.9890
Random Excursions Test		
x = -4	0.137669	0.9869
x = -3	0.287113	0.9869
x = -2	0.306059	0.9918
x = -1	0.613238	0.9918
x = 1	0.770642	0.9886
x = 2	0.825651	0.9902
x = 3	0.901436	0.9837
x = 4	0.256024	0.9806
Random Excursions Variant Test		
x = -9	0.856548	0.9935
x = -8	0.899148	0.9951
x = -5	0.487035	0.9918
x = -2	0.552416	0.9821
x = 2	0.506324	0.9886
x = 5	0.275709	0.9951
x = 7	0.041549	0.9886
x = 9	0.773817	0.9886

Table 3: Results of Parametrized Tests.

Statistical Test	**Distribution of P-Values**	**Proportion of passed tests**
Linear Complexity Test (block length 500)	0.452173	0.9910
Serial Test (block length 16)	0.208837	0.9870
Serial Test (block length 16)	0.647530	0.9850
Non-Overlapping Template Matching Test		
template = 000110111	0.435430	0.9920
template = 001001101	0.191687	0.9940
template = 001011011	0.595549	0.9910
template = 101111100	0.755819	0.9850
template = 110111000	0.395940	0.9920
template = 111010100	0.325206	0.9930
template = 111101100	0.204439	0.9900
Overlapping Template Matching Test	0.672470	0.9900
Maurer's Universal Statistical Test	0.508172	0.9870
Frequency Test Within a Block (block size 10^4)	0.666245	0.9870
Approximate Entropy Test (block length 10 bits)	0.429923	0.9870

about 147 milliseconds. In case of TinyRNG the generation of seed is preceded by a number of entropy accumulations, each taking about 2 milliseconds. The generation of TinyRNG seed itself takes about 2 times more time than in case of *lmRNG*. And the generation of 64 random bits takes about 1 millisecond in

our approach compared to about 440 microseconds in TinyRNG. Altogether, the whole process of generation of the 64 random bits takes less time in case of *lmRNG*, even if the initialization phase is not taken into account. Additionally, the main advantage of *lmRNG* when compared to TinyRNG is that our approach allows for generation of random numbers also in case when the radio transceiver is turned off, e.g., the sensor node is in power saving mode. Keeping the radio to be turned off as often as possible is one of the main requirements for the wireless sensor networks.

Table 4: Comparison *lmRNG* with HM vs TinyRNG @ 8MHz.

Operation	***lmRNG***	**TinyRNG**
Initialization	0	146 ms
Seed Generation	530 μs	1.13 ms
Entropy Accumulation	0	2.16 ms
Generation of 64 bits	1 ms	440 μs

5 CONCLUSIONS

In this paper we presented a new cryptographic pseudo random generator for wireless sensor networks. The *lmRNG* is based on a modified logistic map equation and is adapted to the requirements of sensor nodes. It is computationally efficient and does not need any additional hardware to be built in on the sensor node. It combines the non-deterministic seed source with deterministic mathematical formulas to produce the pseudorandom outputs. The pseudorandom numbers obtained from *lmRNG* were tested using the NIST Test Suite containing 15 statistical tests and all the tests were passed. Thus, the *lmRNG* is a good candidate to be used in cryptographic protocols for wireless sensor networks and this aspect will be investigated in our future work.

REFERENCES

A. Francillon, C. C. (2007). TinyRNG: A Cryptographic Random Number Generator for Wireless Sensors Network Nodes. In *WIOPT'07, 5th Intl. Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks*, Limassol, Cyprus.

A. Sojka, K. Piotrowski, P. L. (2010). Shortecc: a lightweight security approach for wireless sensor networks. In *International Conference on Security and Cryptography, SECRYPT*. INSTICC.

G. Lo Re, F. Milazzo, M. O. (2011). Secure random number generation in wireless sensor networks. In *Proceedings of the 4th international conference on Security of information and networks*, pages 175–182, New York, NY, USA. ACM.

K. Piotrowski, A. Sojka, P. L. (2010). Body area network for first responders - a case study. In *The 5th International Conference on Body Area Networks, BodyNets*. ACM.

L. Kocarev, S. L. (2011). *Chaos-based Cryptography*, volume 354.

May, R. (1976). *Theoretical ecology : principles and applications / edited by Robert M. May*. Blackwell Scientific, Oxford :.

NIST (2010). A statistical test suite for random and pseudorandom number generators for cryptographic applications.

Pareschi, F. (2006). Chaos-based random number generators: Monolithic implementation, testing and applications. In *PHD. THESIS*.

T. Stojanovski, L. K. (2001). Chaos-based random number generators-part i: analysis [cryptography]. *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, 48(3):281 –288.

TI (2010). Msp430x5xxx/msp430x6xxx family. user's guide. http://www.ti.com/lit/ug/slau208i/slau208i.pdf.

TinyOS (2011). Tinyos web page. http://www.tinyos.net/.

V. Patidar, K. S. (2009). A pseudo random bit generator based on chaotic logistic map and its statistical testing. *Informatica*, 33:441 –452.

Y.H. Wang, H.G. Zhang, F. Z. B. W. (2006). An efficient random number generator for ad hoc sensor network. In *Wireless Communications, Networking and Mobile Computing, 2006. WiCOM 2006.International Conference on*, pages 1 –4.