A TRNG exploiting multi-source physical data

V. Gaglio, A. De Paola, M. Ortolani, G. Lo Re

In Proceedings of the 6th ACM workshop on QoS and security for wireless and mobile networks, 2010, pp. 82-89

A TRNG Exploiting Multi-Source Physical Data

Vincenzo Gaglio
vincenzo.gaglio@gmail.com

Alessandra De Paola
depaola@unipa.it

Marco Ortolani
ortolani@unipa.it

Giuseppe Lo Re
lore@unipa.it

Department of Computer Engineering
University of Palermo
Viale delle Scienze, ed 6. - 90128 Palermo, Italy

ABSTRACT

In recent years, the considerable progress of miniaturization and the consequent increase of the efficiency of digital circuits has allowed a great diffusion of the wireless sensor network technology. This has led to the growth of applications and protocols for applying these networks to several scenarios, such as the military one, where it is essential to deploy security protocols in order to prevent opponents from accessing the information exchanged among sensor nodes. This paper analyzes security issues of data processed by the WSN and describes a system able to generate sequences of random numbers, which can be used by security algorithms and protocols. The proposed True Random Number Generator (TRNG) exploits measurements obtained from sensor nodes, in order to allow every node to produce random data upon request, without involving a trusted third party. The proposed TRNG behavior has been tested by carrying out the NIST tests, and the obtained experimental results indicate the high degree of randomness of the produced numbers.

Categories and Subject Descriptors

G.3 [**Mathematics of Computing**]: Probability and Statistics—*Random number generation*

General Terms

Algorithms, Experimentation, Measurement, Security, Verification

Keywords

Wireless Sensor Networks, Random Number Generator, Network Security

1. INTRODUCTION

One of the most challenging issues in wireless sensor networks research field concerns security. Securing applications based on sensor networks may boost their use in several areas, first of all the military one. Different technologies have been developed to achieve this target, but these attempts have been hindered by the difficulty in generating and distributing session keys or public and private key pairs necessary for encryption algorithms.

The Identity-Based Encryption (IBE) [7] is a possible solution for this issue. Such system makes use of the node identifier (node ID) as public key, thus avoiding the expensive operations needed for key generation and distribution. In this case, the key exchange protocol requires the generation of one or more session keys needed for two entities that intend to communicate, and uses completely random "nonce" values during the distribution phase. The keys/nonce could be precomputed and sent to the nodes involved in message exchange; otherwise nodes could compute the necessary information when needed, thus avoiding security issues about key management and distribution. Authors of [4] propose TinySec, the first link layer security architecture for wireless sensor networks. This security system is based on the use of a block cipher, RC5 or Skipjack, with a Cipher Block Chaining encryption scheme. In this case the manufacturer could store the key and the initialization vector on the sensor node; this would lead to remarkable risks since an opponent could obtain this information and decrypt all data exchanged between sensor nodes, as it is sufficient to know the information inserted by the manufacturer.

Another possible solution is to use one of the publicly available sequences of random numbers. However they represent a very limited source as compared to the potential requirements of an application. An opponent can also easly obtain these numbers by knowing the data set being used; this represents a great risk for the security of applications based on it.

A valuable alternative consists in allowing each sensor to use a "secret" information obtained through a true random number generator. Most existing generators are devices based on microscopic phenomena such as thermal noise, photoelectric effect or other quantum phenomena. Intel developed a chip for sampling thermal noise by amplifying the voltage measured on a resistor [3]. A Bell Labs group produced a technique based on response time variations of the requests read from a hard drive sector [2]. These techniques ensure security of a large number of applications, which often use random bit sequences as keys for the symmetric encryption algorithms. Instead of computing the sequence of random numbers and sending results to the sensor nodes, an alternative approach is to allow each node to autonomously compute the necessary security information.

In [1] Francillon et al. present TinyRNG, a Cryptographic Pseudo-Random Number Generator for wireless sensor networks. Their generator uses the transmission bit errors as the source of randomness. Due to their unpredictability, these errors are difficult to observe and to be manipulated by an opponent. The generator output depends on its initial state, assigned through an initial secret key, whose value is set in order to tune the behavior of all remote nodes constituting the WSN. This seed is generated by a central workstation, which exposes the whole system to possible attacks by an opponent trying to guess the secret key.

We propose a True Random Number Generator which uses a non-deterministic source to produce randomness, namely the physical quantities measured by the sensor nodes. This system produces sequences of random numbers using the information about temperature, humidity and light exposure detected by the sensor nodes. This mechanism exploits common off-the-shelf sensor network technology and does not require any dedicated hardware. Moreover, the absence of any secret seed, generated in a centralized way, decreases potential system vulnerabilities.

We evaluated our system performance by using the tests presented in *NIST Special Publication 800-22rev1* [8], in order to detect possible non-randomness in the produced sequences. Test execution provides encouraging results, allowing us to conclude that, with high probability, the generated sequences are truly random.

The rest of this paper is organized as follows. Section 2 introduces the concepts related to the random number generation and techniques used for testing and correcting an existing generator. Sections 3 and 4 give a detailed description of the proposed random number generator and its integration with a tool previously developed at our Department for hybrid WSN simulation [5]. Section 5 presents the statistical tests used for system validation, and analyzes the obtained results in comparison with another well-established TRNG. Finally, in Section 6, we discuss future developments of our approach.

2. RANDOM NUMBER GENERATION

Security of many cryptographic systems is based on the generation of unpredictable quantities. These random numbers play a key role in both symmetric encryption, and public key cryptography. For instance, the value produced by a random number generator can be used as keystream for the One-Time Pad cipher, or as secret key of DES algorithm or else as "nonce" value used for mutual authentication schemes. In all these cases, the exploited values should be random so that the probability of selecting a given value is small enough to prevent opponents from guessing the number. For example, suppose that an encryption algorithm uses 2^{56} keys; if a secret key k is selected using a true random number generator, an opponent will be forced to try on average 2^{55} keys before finding the correct key [10]. Instead, if the key k is selected by choosing 16 random secret bits, expanded to 56 using a well known function f, then it will be sufficient for the opponent to try only 2^{15} values.

Typically, instead of truly random numbers, applications use sequences of pseudo-random numbers that are in fact obtained through a deterministic procedure using a small set of initial values, called "seeds". These numbers resemble those generated by truly random processes with respect to their statistical characteristics, although they can be easily reproduced starting from the knowledge of the initial seeds.

2.1 True Random Number Generators

A True Random Number Generator (TRNG) uses a non-deterministic source for producing the necessary randomness. These generators are based on stochastic sources influenced by unpredictable natural processes. However, the devices used for the random numbers production are often prone to errors or failures. Thus, it is necessary to periodically perform statistical tests in order to verify the goodness of the generator.

Numbers produced by a TRNG can be used straightaway or they can be fed into a pseudo-random number generator; for direct use, it is necessary to meet specific criteria of randomness, measured by applying statistical tests, in order to determine whether the source is truly random or not [8]. Some physical processes can produce correlated and biased bits; for this reason a *de-skewing* technique could be used, in order to correct erroneous bits produced by a "faulty" generator. For instance, let us consider a sequence of bits which are uncorrelated but biased, where the probability of producing a bit `0` is p. It is possible to combine group of bits in order to reduce the bias effect; in this example it is enough to group bits in pairs, thus obtaining the symbols `00`, `01`, `10`, `11`, associated to probability values respectively equal to p^2, $p(1-p)$, $(1-p)p$, $(1-p)^2$. In order to obtain a source of information in which symbols are equiprobable, it is sufficient to discard `00` and `11` and to transform the `10` and `01` respectively into `1` and `0`. The resulting sequence will contain unbiased and uncorrelated bits. In general, a practical method for de-skewing is to process the bit stream with a cryptographic hash function like MD5 or SHA-1.

We list below some physical processes used for random number generation:

- Time interval between the emission of two particles during radioactive decay or the emanation of photons in a semiconductor;
- Frequency instability of an oscillator;
- Air turbulence inside the disk's enclosure causing fluctuations in time required for readings;
- Thermal noise produced by a semiconductor;
- Atmospheric noise, detected by a radio receiver attached to a workstation;
- Sound detected from a microphone;
- System clock;
- Time spent during keystrokes or mouse movements;
- System load and/or network statistics.

It should be noted that generators based on the phenomena listed above may be observed or manipulated by an opponent.

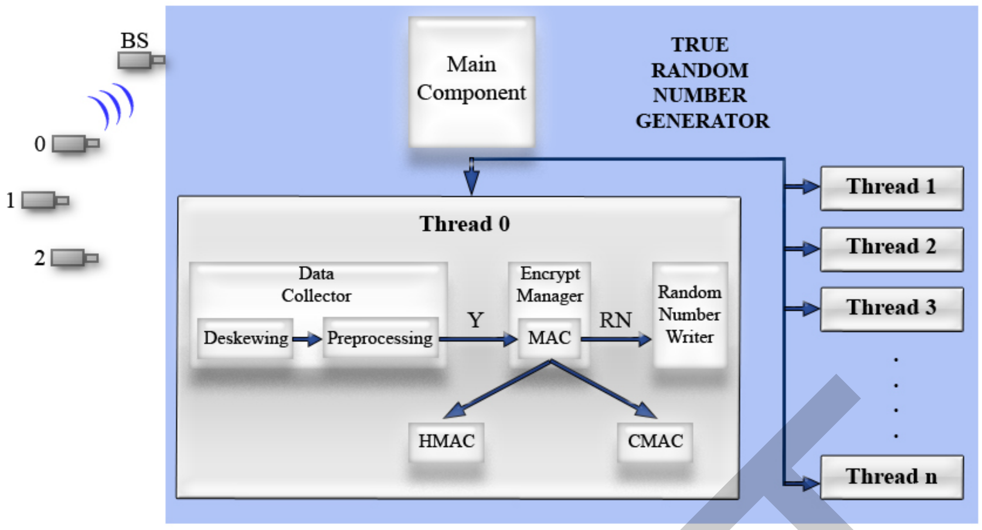

Figure 2: Block diagram of the proposed system.

3. THE PROPOSED TRNG

The proposed true random number generator exploits measurements gathered by nodes of a wireless sensor network in order to produce random numbers on board and on demand. These features allow to avoid storing and transmitting sensitive information, thus reducing vulnerabilities in the network. The system can also coexist with other applications running within the WSN, which use the information about temperature, humidity and light exposure in order to perform their task. Sensory measurements derive from a physical process that can be considered as a stochastic source and that is characterized by a suitable entropy level. In particular, the considered information sources, namely temperature, humidity and light exposure sensors, present a value of entropy per bit equal to 0.54, 0.71 and 0.66. Sensory measurements are the seeds of a process whose goal is to perform de-skewing and to add *diffusion* and *confusion*, in order to improve the statistical profile of the produced number sequence.

This process is based on hash functions and Message Authentication Code (MAC) algorithms. The former scatter the statistical structure of the input into large-range statistics of the output; this goal is achieved by increasing the number of input digits that affect a single output value. Just as an example, let us consider the following naïve hash function:

$$z_t = \sum_{i=0}^{t-1} (x_i - z_i); \quad (1)$$

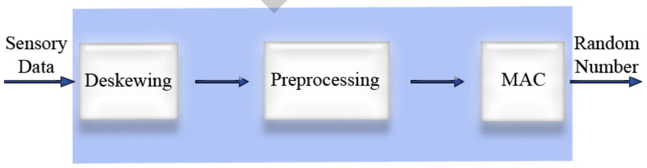

Figure 1: Process flow: from sensory data to random numbers.

where the resulting value z_t at time t is produced by exploiting all the previous received values x_i with $i = 0, 1, 2, ...t-1$. MAC algorithms, on the other hand, allow to increase *confusion* in the produced output, by making the relationship between the input and output statistics as complex as possible; this goal is achieved by applying a strong substitution algorithm [9].

We developed and tested this system using a wireless sensor network formed by TelosB motes. The only necessary requirement for this WSN is the presence of at least two sensor nodes: one for sampling sensory data and forwarding them to the other, and the second node acting as *Base Station*, and processing gathered data in order to produce the random number sequence. The Base Station processes received data by using a de-skewing technique, and applying a pre-processing phase whose purpose is to remove redundant information and obtain the randomness degree required for the generator; the results thus obtained are then passed to a MAC algorithm, whose task is to further shuffle bits for increasing *diffusion* and *confusion* over the data. This process flow is shown in Figure 1.

The Base Station core is designed according to the multi-thread paradigm, where a main thread is responsible for coordinating a thread pool, in which each one is responsible for measurements processing related to a specific mote of the network (see Figure 2). The *DataCollector* module addresses the gathering, de-skewing, and pre-processing of raw data received from sensor nodes. The result produced by the preprocessing phase is then inserted within a window buffer, containing all the sensory information that will be processed by the MAC algorithm. The *EncryptManager* module manages data encryption using the MAC *Encoder*, with the required key and initialization vector. Once the generation process is completed, the *RandomNumberWriter* module passes the produced random number to the opportune application.

The proposed system allows users to customize the TRNG behavior by tuning the preprocessing and MAC phases.

The main goal of the preprocessing phase is the improvement of the statistical profile of the sequence of values, by increasing the conditional independence of a value with respect to the previous ones. To achieve this goal we do not consider the direct output of the entropy source, rather during this phase we extract the small and unpredictable variations among consecutive sensory readings.

This preprocessing phase can be tuned by selecting one out of three available techniques: XOR, MSE or Residuals. The first method consists of performing an exclusive disjunction between "unique" data present within each packet. Every data block consists of M sensory measurements, each occurring multiple times within the same packet. In our implementation the value of M is set to 10. Each measurement is identified by a string of 8 or 16 bits depending on the type of sensor used: temperature sensors produce binary sequences of 16 bits, whereas light sensors whose output is of 8 bits. If the message contains M distinct sensory measurements x_i with $0 \leq i \leq M-1$, then the system computes the resulting bit string Y_{XOR} as:

$$Y_{XOR} = X_0 \oplus X_1 \oplus X_2 ...... \oplus X_{M-1}. \qquad (2)$$

The second preprocessing method is based on the computation of the Mean Square Error (MSE) on the average of received sensory measurements. The global mean at time t_N, can be obtained by dividing the sum of the average values computed on each packet ($mean_{t_k}$ with $0 \leq k \leq N-1$), by the number of messages received so far:

$$global\_mean_{t_N} = \frac{\Sigma_{k=0}^{N-1} mean_{t_i}}{N}, \qquad (3)$$

with

$$mean_{t_k} = \frac{\Sigma_{i=0}^{M-1} X_i}{M}, \qquad (4)$$

where X_i is the i-th measurement present within the packet received at time t_k and $mean_{t_k}$ is the mean computed on the packet. Computing the global mean does not introduce a significant delay in the random number generation, but only a small initial latency due to filling the buffer of $mean_{t_i}$ values.

After determining the global mean, the system will estimate a Mean Square Error, using the following equation:

$$Y_{MSE} = \frac{\Sigma_{i=0}^{M-1}(X_i - global\_mean_{t_N})^2}{M}. \qquad (5)$$

The third and last preprocessing technique relies on the computing of residuals associated with each measurement X_i, according to the following equation:

$$Y_{RES} = (X_i - global\_mean_{t_N}), \qquad (6)$$

where the $global\_mean_{t_N}$ is computed as in equation 3.

Numbers produced by the preprocessing phase are stored in a window buffer used to keep track of past values. The *EncryptManager* component of each thread computes a MAC over data inside the buffer, and the resulting output represents the random value produced by the system. The size of the buffer is one of the system parameters which can be set by users. It is worth noting that a large buffer allows users to keep track of a larger amount of past measurements and consequently it could contain a greater variety of values. This increases the randomness of the system at the cost of an increment of the system load. Data input in the window buffer is performed according to a first-in-first-out policy: a new value is put on the last free position obtained after a left shift of the buffer, thus removing the oldest measurement.

In order to customize the MAC phase, the proposed system allows users to select one out of two different MAC algorithms based on hash functions and block ciphers, respectively:

- **HMAC** (keyed-Hash Message Authentication Code) with either SHA-512 (Secure Hash Algorithm) or MD5 (Message Digest Algorithm 5);

- **CMAC** (Cipher-based Message Authentication Code), choosing between three symmetric encryption algorithms, DES (Data Encryption Standard), 3DES (Triple Data Encryption Standard), and AES (Advanced Encryption Standard), as block cipher.

The MAC is computed on the WindowBuffer using a different key for each received message. The method used for computing the key varies according to the different types of MAC and the relative encryption algorithm. The key used for CMAC with DES changes for each iteration and its value is the sum modulo 2^{64} of the previously computed MAC codes. The current value is stored within a register which is updated after receiving each packet (see Figure 3).

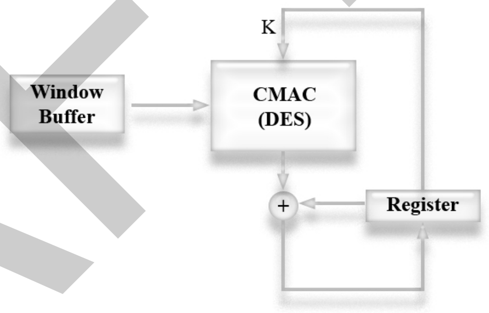

Figure 3: CMAC computation on the WindowBuffer using the register value as key.

Triple DES is another mode of DES operation. It takes three 64-bit keys, for an overall key length of 192 bits. Our CMAC uses 3DES with two 64-bit keys, exploiting the same key for the first and third 3DES keys. The keys used during each time step t_i are given by the values of the register at the two previous times, $k_{t_{i-1}}$ and $k_{t_{i-2}}$:

$$MAC_{t_i} = E(k_{t_{i-2}}, D(k_{t_{i-1}}, E(k_{t_{i-2}}, buffer))), \qquad (7)$$

where $buffer$ is the content of the window buffer at current time step t_i.

When the selected MAC algorithm is HMAC or CMAC with AES, the key has exactly the same value of the MAC code computed in the previous step.

4. HYBRID SIMULATION FOR WSN

In order to assess the statistical properties of the proposed TRNG, a set of simulations were performed using the hybrid simulator for WSN [5], developed at our lab as an extension of the well known TOSSIM [6], with the aim of supporting the development of a wide range of applications whose outcome is highly dependent on the peculiarities of the considered deployment environment. The hybrid simulation allows the interaction between real and simulated nodes, and thus the setting-up of simulation scenarios in which a small number of real nodes are interspersed into a larger virtual network.

The use of simulated nodes allows for the generation of easily scalable scenarios, while nodes deployed in the real sensor field are used as a complement, in order to generate realistic data models and to steer the behavior of their virtual counterpart. The hybrid simulator exploits realistic models of the data obtained from real nodes, in order to manage the behavior of simulated ones; thus experimental evaluation of the proposed TRNG achieves more plausible results than those given by exploiting only purely mathematical models.

In order to integrate the random number generator with our simulator, we expanded it with a specific plugin for TRNG offering the following functionalities:

- Selection of the motes used for generating random sequences (TelosB, MicaZ or other sensor nodes);
- Choice of the sensor type offered by the hardware platform selected in the previous step (temperature, humidity or light exposure for a TelosB mote);
- Parameter setting for the random number generator.

Furthermore, users can specify the following input parameters, in order to tune the TRNG behavior:

- *Collector Type*: the preprocessing technique to be used for collecting data (XOR, MSE or Residuals);
- *Window size*: the size of the window buffer;
- *MAC Type*: the MAC algorithm used for random sequences generation (HMAC or CMAC);
- *Encoding Type*: the encryption technique adopted by the MAC algorithm specified at the previous step (DES, 3DES or AES with CMAC, SHA-512 or MD5 with HMAC);
- *Num of bits*: maximum number of bits produced by the system.

5. EXPERIMENTAL RESULTS

Several tests can be applied to a number sequence in order to verify the randomness of its terms. These tests check for the presence of a specific pattern within the sequence and produce a statistical value which is then compared to a critical threshold; if the result exceeds this threshold, then the sequence may rightfully be considered random. In order to validate our random number generator we used the NIST (National Institute of Standards and Technology) suite [8] containing 15 statistical tests, each of which uses a different technique for identifying a possible case of non-randomness in the input sequences. These tests were developed in order to evaluate randomness of binary sequences having arbitrary length, produced by both pseudo-random and true-random number generators, based on software and hardware implementation, respectively.

The NIST suite contains the following tests:

- *Frequency (Monobit)*, whose purpose is to check whether the number of ones and zeros in a sequence are approximately the same as would be expected for a truly random sequence;
- *Frequency within a block*, identical to the previous except for the definition of a fixed block size, M, limiting the computation;
- *Runs*, determines whether the number of "runs" of ones and zeros of various lengths is as expected for a random sequence, where a "run" is an uninterrupted sequence of identical bits;
- *Longest Run of ones in a block*, checks whether the length of the longest run of ones within the tested sequence is consistent with the length of the longest run of ones that would be expected in a random sequence;
- *Binary Matrix Rank Test*, computes the rank of disjoint sub-matrices of the entire sequence in order to check for linear dependence among fixed length substrings belonging to the original sequence;
- *Discrete Fourier Transform*, uses peak heights in the Discrete Fourier Transform of the sequence in order to detect periodic features in processed data;
- *Non-overlapping template Matching*, determines whether a generator produce too many occurrences of a given aperiodic pattern;
- *Overlapping Template Matching*, identical to the previous except for the pattern detection phase where the window slides only one bit before resuming the search;
- *Maurer's "Universal Statistical"*, checks whether or not the sequence can be significantly compressed without loss of information, so determining an eventual non-randomness for high compression ratio;
- *Linear Complexity*, uses a linear feedback shift register (LFSR) in order to check for randomness in a data set: random sequences are characterized by longer LFSRs;
- *Serial*, determine whether the number of occurrences of m-bit overlapping patterns is approximately the same as would be expected for a random sequence;
- *Approximate Entropy*, as for the previous test, the purpose is to compare the frequency of overlapping blocks of two consecutive lengths (m and m+1) against the expected result for a random sequence;
- *Cumulative Sums*, computes cumulative sum of adjusted (-1, +1) digits in the sequence and compares obtained result with that of a random sequence (which should be near zero);
- *Random Excursions Test*, computes several cumulative sums of input sequence and determines if the number of visits to a particular state (-4,-3,-2,-1 and +1,+2, +3,+4) deviates from what one would expect for a random sequence;
- *Random Excursions Variant*, identical to the previous except for the number of states used, eighteen instead of nine.

5.1 Randomness Evaluation

NIST tests are defined on the basis of some general assumptions that can be made about random binary sequences:

1. Bits are generated according to a uniform distribution; that is, at any point of a sequence the probability of a 0 or 1 is equal to $\frac{1}{2}$. This means that the expected number of 0 values within the sequence of length n is equal to $\frac{n}{2}$;
2. Test results remain consistent while varying the scale; that is, a test can be applied to any number of subsequences extracted randomly from the main sequence; if such a sequence is random, then any extracted subsequence will also be random;
3. Test results remain consistent while varying the seeds: the generator must produce random sequences even if starting values change.

NIST statistical tests use the incomplete gamma function, *igamc*, and the complementary error function, *erfc*, for computing reference values on data set:

$$erfc(z) = \frac{2}{\sqrt{\pi}} \int_{z}^{\infty} e^{-u^2} du; \tag{8}$$

$$igamc(\alpha, x) = \frac{1}{\Gamma(a)} \int_{0}^{x} e^{-t} t^{a-1} dt; \tag{9}$$

where $\Gamma(a) = \int_{0}^{\infty} e^{-t} t^{a-1} dt$.

Each test processes binary data and estimates input parameters of the functions listed above. The output value (also called P-value), is then compared to a threshold value, and if this P-value falls below this threshold then the sequence fails the randomicity test; in our experimental setting the threshold value is set to 0.01. Otherwise, for the chosen test, a sequence satisfies the required property of randomness. Analysis of test results is performed through a double check. First of all it is required to compute the ratio of sequences satisfying a statistical test; then the uniformity of the P-values distribution is checked. If both these approaches fail in producing conclusive results, further tests should be conducted on different samples of the generator, in order to distinguish a "statistical anomaly" from a clear evidence of non-randomness.

The ratio of sequences passing a test can be computed by dividing the number of trusted bit strings by the total amount of processed sequences:

$$ratio = \frac{num\_of\_trusted\_sequences}{num\_of\_sequences}. \tag{10}$$

The acceptable range for ratio values is computed according to the following formula:

$$ratio > p - 3\sqrt{\frac{p(1-p)}{m}}, \tag{11}$$

where $p = (1 - threshold) = 0.99$ and m is the number of processed sequences. For instance, if $m = 10$, then the portion of sequences passing a test should be greater than 0.895607.

Uniformity of P-values distribution may be determined via the application of an incomplete gamma function to the P-values (i.e. a P-value of the P-values). The interval between 0 and 1 is initially divided into 10 sub-intervals, then system counts the number of P-values lying within each subinterval, thus obtaining a histogram of P-values. The following equations are then used in order to compute the resulting overall P-value:

$$X^2 = \sum_{i=1}^{10} \frac{(F_i - \frac{m}{10})^2}{\frac{m}{10}}, \tag{12}$$

$$P\text{-}value = igamc(9/2, X^2/2), \tag{13}$$

where F_i is the number of P-values lying in sub-interval i and m is the sequence size. If the obtained value is greater then 0.0001, then the set of P-values can be considered uniformly distributed.

5.2 Test Execution

Tests from the NIST suite were executed on sequences produced by our TRNG; for the sake of this set of experiments we actually deployed a limited set of TelosB nodes, and integrated them with additional simulated nodes in order to form a larger hybrid WSN. Actual nodes will provide the users with a choice of on-board sensors to be used for the generation process; this choice will affect the performances of the generator depending on the entropy of the selected information sources. Tested sequences were obtained by varying the input parameters, such as the preprocessing technique and the encryption methods. Experimental results show that the proposed TRNG passes all submitted tests, both with respect to uniformity and to ratio, with values much higher than the thresholds, that in our experimental setting are set to 0.0001 and 0.895607 values respectively.

In order to verify system robustness when using light exposure sensors, we executed several tests at regular intervals. In particular, the system was set to perform a test, wait for completion and restart its execution, every hour; in this way the behavior of the proposed TRGN has been tested with different ambient light intensity values and across multiple days. Figure 4 shows the trend of P-values obtained from the Frequency Test during a whole day, and shows that the proposed generator exhibits good properties of randomness also during the night, using the "noise" received from light sensors, as each P-value is greater than 0.0001.

In order to evaluate the behavior of the proposed TRNG, we propose a comparative analysis with respect to the Quantum Random Bit Generator (QRBG) [11], a well known TRNG, based on the photonic emission in semiconductors and on the subsequent detection by a photoelectric effect. In this process photons are casually observed, independently from each other. Timing information of detected photons is used for generating random data.

In order to compare the performance of the proposed TRNG with QRBG, we produced 100 sequences, each composed by 1,000,000 bits. According to equation 11, a test is rightly considered passed if the ratio of trusted sequences is greater than 0.960150. As shown in Table 1, experimental results are similar to those produced by QRBG, with the two notable exceptions where our TNRG outperforms QRBG. In particular, we observe that the sequences produced by our generator pass all the statistical tests, unlike

Table 1: Comparison between the proposed TRNG and the Quantum Random Bit Generator.

STATISTICAL TEST	Proposed System		QRBG	
	P-VALUE	PROPORTION	P-VALUE	PROPORTION
Frequency	0.616305	0.9900	0.816537	0.9600
BlockFrequency	0.058984	1.0000	0.637119	0.9900
CumulativeSums	0.739918	0.9700	0.897763	**0.9500 ***
Runs	0.474986	0.9700	0.366918	0.9900
LongestRun	0.045675	0.9900	0.816537	1.0000
Rank	0.455937	0.9900	0.595549	0.9900
FFT	0.030806	1.0000	0.080519	1.0000
NonOverlapT	0.935716	1.0000	0.494392	**0.9500 ***
OverlappingT	0.224821	0.9800	0.171867	1.0000
Universal	0.816537	0.9900	0.030806	0.9900
ApproximateEntr	0.162606	0.9800	0.437274	0.9800
RandomExcursions	0.637119	1.0000	0.334538	0.9992
RandomExcurVar	0.706149	0.9844	0.080519	0.9831
Serial	0.851383	0.9700	0.595549	0.9800
LinearComplexity	0.897763	0.9900	0.213309	0.9800

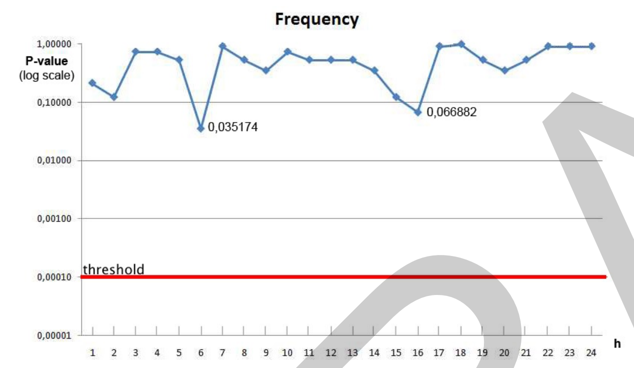

Figure 4: Trend of P-values over a day, computed by the Frequency Test, for a TRNG that uses only light exposure sensors.

QRBG whose output is not suitable for the *CumulativeSums* and *NonOverlappingTemplate* tests.

Based on these results, we can conclude that the behavior of the proposed system completely satisfies the statistical properties required by a True Random Number Generator.

6. CONCLUSION AND FUTURE WORK

This paper addressed the issue of producing true random number sequences to be used for security applications for wireless sensor networks. The proposed approach exploits readings of physical quantities performed by on-board sensors, as sources of randomness. In order to assess the goodness of the proposed method we tested the generated sequences through well-established test suites, such as those provided by NIST. Reported experiments show encouraging results about the statistical profile of the generated random numbers.

We are currently working on the development of a proof-of-concept security application for WSN in order to show that no additional computational overhead is caused by our TRNG that may interfere with pre-existing applications.

7. REFERENCES

[1] A. Francillon, C. Castelluccia, and P. INRIA. TinyRNG: A cryptographic random number generator for wireless sensors network nodes. In *Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks and Workshops, 2007. WiOpt 2007. 5th International Symposium on*, pages 1–7. Citeseer, 2007.

[2] M. Jakobsson, E. Shriver, B. Hillyer, and A. Juels. A practical secure physical random bit generator. In *Proceedings of the 5th ACM Conference on Computer and Communications Security*, page 111. ACM, November 1998.

[3] B. Jun and P. Kocher. The Intel random number generator. *Cryptography Research Inc. white paper*, April 1999.

[4] C. Karlof, N. Sastry, and D. Wagner. Tinysec: A link layer security architecture for wireless sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 162–175. ACM, November 2004.

[5] A. Lalomia, G. L. Re, and M. Ortolani. A hybrid framework for soft real-time wsn simulation. In *13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications*, October 2009.

[6] P. Levis, N. Lee, M. Welsh, and D. Culler. Tossim: Accurate and scalable simulation of entire tinyos applications. In *the First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003)*. ACM, November 2003.

[7] L. B. Oliveira, D. Aranha, E. Morais, F. Daguano, J. Lòpez, and R. Dahab. Tinytate: Identity-based

encryption for sensor networks. website:
http://eprint.iacr.org/2007/020.pdf, 2007.

[8] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, and S. Vo. A statistical test suite for random and pseudorandom number generators for cryptographic applications. *Storming Media*, 2001.

[9] C. Shannon. Communication theory of secrecy systems. *MD Computing*, 15(1):57–64, 1998.

[10] W. Stallings. *Cryptography and Network Security (4th Edition)*. Prentice Hall, 2006.

[11] M. Stipčević and B. Rogina. Quantum random number generator based on photonic emission in semiconductors. *Review of Scientific Instruments*, 78:045104, 2007.