# *Secure random number generation in wireless sensor networks*



G. Lo Re, F. Milazzo, M. Ortolani

In Proceedings of the 4th international conference on Security of information and networks, 2011, pp. 175-182

NDS Research Group - Networking and Distributed Systems
University of Palermo, ITALY
http://www.dicgim.unipa.it/networks

# Secure Random Number Generation in Wireless Sensor Networks

**Giuseppe Lo Re** **Fabrizio Milazzo** **Marco Ortolani**
{giuseppe.lore,fabrizio.milazzo,marco.ortolani}@unipa.it

**Abstract**

Reliable random number generation is crucial for many available security algorithms, and some of the methods presented in literature proposed to generate them based on measurements collected from the physical environment, in order to ensure true randomness. However the effectiveness of such methods can be compromised if an attacker is able to gain access to the measurements thus inferring the generated random number. In our paper, we present an algorithm that guarantees security for the generation process, in a real world scenario using wireless sensor nodes as the sources of the physical measurements. The proposed method uses distributed leader election for selecting a random source of data. We prove the robustness of the algorithm by discussing common security attacks, and we present theoretical and experimental evaluation regarding its complexity in terms of time and exchanged messages.



## 1 Introduction

Information security algorithms aim to provide protection against unauthorized access, to ensure that data sources can be trusted, or to check if any modification occurred on transmitted data along the path to the end user.

They typically rely on the availability of random numbers in order to perform their operations; for instance, key exchange algorithms make use of the so called "nonce" [16] to perform authentication, and the "counter mode" [4] of DES block cipher uses a random number to start the encryption process. The main challenge when using random number sequences is the characterization of their statistical properties; typical requirements for such sequences are that they presents no order and no coherence, i.e. no regularity patterns are to be discovered in a sequence.

We can regard a random number generator (RNG) as a black box whose outcome consists of symbols from a given alphabet. Such generator is typically implemented either by relying on a specialized *algorithm*, or on a *physical* process. The former approach involves the use of mathematical functions to generate a number sequence with random features, and the resulting generator is named Pseudo-Random Number Generator (PRNG), since its randomness is only apparent and, in fact, the sequence is deterministically predictable by knowing all the parameters of the algorithm. The latter approach generates numbers starting from measurements related to a physical process; in this case the generator is named a True Random Number Generator (TRNG), because the sequence is actually non deterministic and unpredictable; in mathematical terms such generator may be represented by the following formula:

$$r = f(m_1, m_2, ...., m_n), \tag{1}$$

where $r$ is the generated random number depending on a function $f$ of the environmental measurements $m_i$.

Regardless of the type of generator, the process itself must be *secure* in its own implementation. For instance, if an encryption system relies on a randomly generated secret key, the lack of security within the random number generation process would compromise the entire system as long as an attacker could easily infer the key.

Our paper describes a complete security infrastructure built upon our TRNG previously described in [7], which aimed to produce truly random numbers exploiting measurements collected by a Wireless Sensor Network (WSN) [5], i.e. a network composed of a possibly large number of sensor nodes collecting physical measurements from the surrounding environment. We assume that each sensor node is able to generate a random number using its readings,

i.e. it effectively implements a generator of the form described by Eq. 1; nonetheless, if a node merely relied on its own measurements, the generation process could be easily broken by an attacker. In fact, information security systems always assume that the security algorithm (i.e. function $f(\cdot)$) is known to the attacker; moreover, in the case of sensor nodes, a malicious user could attempt to obtain the measurements by violating the analog-digital converter (ADC) of a node, thus gaining complete knowledge of the random number generation process. The vulnerability is due to the complete *locality* of the process; however, if a node plays the role of *measurements requester* with respect to other nodes that act as *measurements generators*, the overall robustness of the system against security attacks would be improved as the attacker would be additionally required to know the source of the measurements. Clearly, if the network is composed of $N$ nodes, we expect that the probability for the attacker to guess the generator node, is $P_{guess} \geq 1/N$; in the following, we will prove via theoretical and experimental assessments that our security infrastructure ensures that the probability $P_{guess}$ approaches the theoretical minimum value of $1/N$.

In literature, the problem of selecting a node in a graph according to some underlying criteria is known as *Leader Election*, where in our terminology the *leader* is represented by the *measurements generator*. In order to make our algorithm of practical use, we limit the number of nodes that may be chosen as generator; namely, since the time complexity of the leader election, and therefore of the random number generation process directly depends on the dimension of the network, we limit the scope of a requester to a $k$-neighborhood. We will prove that this requires linear time complexity for the random number generation process, whereas it imposes exponential decay for the $P_{guess}$ probability.

The remainder of the paper is structured as follows: Section 2 presents a brief survey on leader election and secure leader election algorithms; Section 3 depicts the TRNG previously implemented and how it is possible to violate its security when nodes use only the locally sensed measurements; Section 4 describes the main phases constituting our distributed approach to secure the TRNG using randomized leader election. Sections 5, and 6 discuss the theoretical and experimental assessment of our algorithm, and finally Section 7 presents our conclusions.

## 2 Related Work

Our algorithm relies on the selection of one network node according to some underlying criterion; this problem often occurs in distributed algorithm design, and is known as *leader election*. It was initially addressed for *ring* networks by G. Le Lann [12] who proposed a synchronous algorithm with $O(n)$ time complexity and $O(n^2)$ communication complexity. Further improvements were introduced by Chang *et al.* [3], and Hirschberg-Sinclair [9], which reduced communication complexity to $O(n \log n)$. Afek *et al.* [2] studied the problem of leader election for *fully connected* networks, providing lower bounds for synchronous and asynchronous networks of $O(n \log n)$ communication complexity, and $O(\log n)$ time complexity. Finally, Kanevsky *et al.* [1] provided lower bounds for leader election algorithm for an arbitrary network topology with $O(diam)$ time complexity (where $diam$ is the diameter of the network), and $O(|E| + n \log n)$ message complexity; they additionally showed that such two lower bounds cannot be simultaneously achieved.

Our leader election algorithm specifically considers asynchronous and arbitrary topologies, since imposing more stringent requirements to the topology or to the timing, of a wireless sensor network, appears too restrictive.

Algorithms developed for asynchronous networks are usually based on tree construction or on search algorithms [13]; for instance asynchronous broadcast and convergecast based on breadth first search (BFS) can be used to elect a leader in an arbitrary network, with an optimal time complexity of $O(diam)$ and a communication complexity of $O(n|E|)$. An alternative approach by Gallagher *et al.* [8] builds a minimum spanning tree with optimal communication complexity $O(|E| + n \log n)$, but with $O(n \log n)$ time complexity.

One of the first works investigating issues related to leader election in WSNs [14] is based on a distributed routing protocol named TORA [15], which allows to manage highly dynamic topologies, but does not address security issues. In particular, given that such leader election method is comparison-based, it is prone to violation by a malicious node which can tamper with the values used to select the leader, increasing its chances to be elected.

The Secure Extrema Finding Algorithm (SEFA) [19] developed for synchronous networks requires a shared evaluation function for computing a unique value for the metric chosen to elect the leader; however it assumes that asymmetric key encryption is computed on nodes, despite its heavy computational requirements. Although our work requires encrypted transmissions as well, in order to prevent an attacker from performing traffic analysis, we chose to use the less expensive symmetric key encryption, as provided by the IEEE 802.15.4 communication standard for WSNs.

A leader election algorithm for mobile ad hoc networks considering topology changes such as node disconnection, partition merging and splitting is proposed in [20]. The method is assessed by evaluating the fraction of time spent by the network without a leader; unfortunately such metric is not useful for our approach, where the leader election

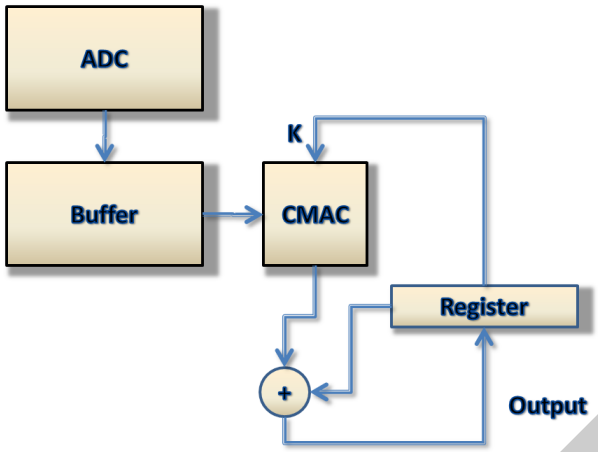


Figure 1: Block diagram for the TRNG implemented on wireless sensor nodes

algorithm is only run on demand. Nevertheless such work shows how an approximate method for leader election using tree construction is more suitable for networks with dynamic topologies as compared to exact methods like shortest path tree (SPT) or minimum spanning tree (MST).

As regards the specific issue of true random number generation (TRNG), well-known examples are [11] where the thermal noise present in resistors is used as source of randomness, and [10] which presents a TRNG based on the latency of readings from a hard drive, and also [18], where a TRNG is designed by making use of the sampling phase jitter in oscillator rings. Finally [6] presents a random number generator for WSNs that makes use of transmission error bits as source of randomness; however such work lacks a robust testing on number randomness unlike [7] where extensive testing was conducted using the NIST Test Suite, and which forms the basis for the RNG used here.

## 3 The proposed RNG

The RNG we implement for each wireless sensor node is detailed in [7] and takes the form shown in Figure 1.

The ADC of a sensor node is used to sense the environment and its measurements are sent to the Buffer block. Such measurements are then encrypted by a CMAC algorithm based on Data Encryption Standard (DES). The result of the CMAC is XOR-ed with the Register block to produce the random number denoted as output of the entire process. The register block acts also as a key $k$ for CMAC, so the same set of measurements will in general produce different outputs.

The generation process is not secure because an attacker could read the measurements by physically violating the ADC block of the wireless sensor node. In addition, suppose that the attacker is able to know the value of one output at a given time $t$. In this case, given that the value of the output is used as key for CMAC block, and that ADC could be easily violated, any random number generated after time $t$ will be easily inferred by the attacker.

To protect the generation process we choose to move the sensing task toward a randomly chosen generator node within a $k$-neighborhood of the requester node. The result is that the attacker must additionally know the source of the measurements to infer the random number; however such attack becomes impracticable for an high number of neighbor nodes. The next section provides the details for the randomized leader election, detailing its two phases: tree construction and convergecast.

## 4 Randomized Leader Election

Our algorithm is composed by two main phases, according to the guidelines provided by [13]: $k$-neighborhood identification, and leader messages convergecast.

An efficient way to address the first step is the construction of a tree rooted at the requester node with depth $k$, so that exactly one path exists between the requester and any other node; the availability of a tree-based network allows to perform the second step of leader discovery by using a simple convergecast method where network nodes filter out potential leaders, until the unique leader will be elected at root node.

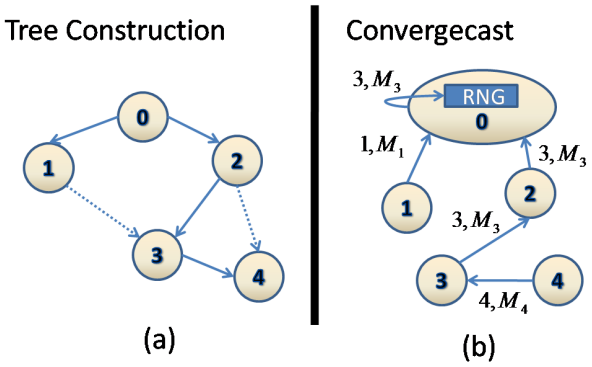


Figure 2: The two phases of the leader election algorithm.

A $k$-neighborhood of a requester node $r$ is a subgraph $G_k(r) = (V, E)$, where $V = \{v_i : d(r, v_i) \leq k\}$ contains all the vertices (nodes) with distance no more than $k$ from $r$ ($|V| = n$), and $E$ contains their connecting edges; for simplicity's sake we will consider only undirected edges (bi-directional links). The diameter *diam* of the subgraph is the longest shortest path between the requester node and each other node within the considered $k$-neighborhood; in our case $diam = k$.

Figure 2(a) depicts the algorithm in action for the two phases of tree construction and convergecast, in a network composed by five nodes. The requester node (Id 0) tries to construct a 3-neighborhood involving nodes having Id 1 to 4. Note that some edges are removed during tree construction phase since each node must set only one node as its parent. Additionally note that node 4 is not reached by the shortest path, since it is possible that messages in the network travel with different delays. In this case the message from node 2 to node 4 reaches it destination after a parent node was set (node 3).

After completion of the tree construction, nodes start sensing. During the convergecast phase, each of them performs a local leader election, forwarding the Id and the set of measurements of the chosen leader, to the parent node; in order to avoid network traffic monitoring transmissions are encrypted with symmetric encryption (AES-128) provided by the IEEE 802.15.4 standard.

Each node waits for a predefined amount of time in order to collect the set of measurements by its children nodes; only one among the received measurements and the self-generated one will be chosen via random selection, thus implicitly identifying the corresponding node as the current candidate for leadership; such selection process terminates at a requester, which will finally be able to elect its leader (i.e. the node chosen as generator for the measurements to be used by that requester).

In figure 2(b) such process is depicted by the label associated to each edge, containing the pair composed by the Id and the set of measurements of the chosen Leader. In particular given that node 1 and 4 are leaves, they elect itself as leader for their subtree. Node 3 chooses among node 4 and itself for the leadership, and the outcome is the node 3. Node 2 repeat the same process, choosing still the node 3 as leader for its subtree. Node 0 chooses among itself, node 1 and node 3, obtaining the node 3 as leader for the entire tree. After choosing the leader, the requester decrypts the relative measurement set, and processes the content through the random number generator described in Section 3. As depicted in the figure, the requester node will use the measurements set generated at node 3 as seed for the RNG.

## 4.1 Tree Construction Phase

In order to select a tree on a graph, we use a simple flooding algorithm with limited depth, similar to Asynchronous Spanning Tree algorithm described in [13]. Let us define the *Parent Proposal* message as a triple: $< Id_{req}, Id_{send}, TTL >$, where $Id_{req}$ represents the Id of the requester node, $Id_{send}$ the sender node, and $TTL$ is used as stopping criterion for the flooding process. The process of tree construction is started at requester node, initializing the value of $TTL$ to $k$. Each node receiving *Parent Proposal* messages sets $Id_{send}$ as its parent node; any subsequent received messages of this type will be discarded. Note that the receiver node should also reply to the immediate sender with an ACK message advertising its own children Id, but we will show later that this can be avoided by implementing a timeout mechanism on the forwarder nodes, which also avoids potential deadlocks. The only known drawback of such algorithm is related to its non deterministic behavior, as it does not ensure that a subgraph $G_k(r)$ will produce the same tree across different runs. Furthermore, it is possible that nodes having

distance $k$ from the requester will be not included in the tree, due to the fact that the algorithm determines one of $O(n^n)$ possible spanning trees depending on the (variable) delay associated to the *Parent Proposal* messages traveling within the network. However (as we will see later) such non optimal behavior does not significantly affect the performance of our security system, so we decided to trade determinism for lower complexity (determinism could have been provided by the significantly more complex shortest path tree or minimum spanning tree algorithm).

### 4.2 Convergecast Phase

When the tree construction phase ends, leaves of the tree starts a convergecast process. Such process ends at the requester node that chooses only one message, belonging to the leader node, containing the set of measurements that will be used to generate the random number.

Let us define the convergecast messages as *Leader Proposal*, a tuple of four values: $< Id_{send}, Id_{lead}, \{m_j\}, n >$, where $Id_{send}$ is the Id of the sender node, $Id_{lead}$ is the Id of the leader for the subtree rooted at the sender node, $n$ is the number of nodes of the subtree rooted at sender node, and $\{m_j\}$ is the set of measurements perceived from the leader node.

Leader Proposal messages are generated by leaf and non leaf nodes setting the value of $n$ to 1. After collecting the Leader Proposal messages from its children, a node generates also a Leader Proposal for itself and performs a *local* leader election selecting randomly the leader proposal message that will be forwarded to its parent node. However, before sending the outcome to parent node, sender node updates the value of $n$ within the chosen leader proposal message with the summation of the values of $n$ over the entire set of Leader Proposal messages. The process terminates at the requester, with the election of a global leader.

The random selection process must weights the selection probability of each Leader Proposal for the number of nodes among that message was chosen as leader (the $n$ value), to guarantee that the leader is chosen with uniform probability. Mathematically speaking each Leader Proposal $LP_j$ is chosen to be forwarded to the parent with a probability equal to:

$$p_{sel}(LP(j)) = \frac{n(j)}{\sum_i n(i)}, \tag{2}$$

where $n(.)$ is the fourth component of the Leader Proposal and $i$ ranges among the set of the Leader Proposal messages. Such selection rule ensures that any Leader Proposal is chosen with the same probability of $1/N$ as Leader at root node if $N$ is the number of nodes of the entire tree; mathematical proof is provided in Section 5.1.

In order to take into account link failures, we additionally consider the case where a message is lost. When no coordination mechanism is enabled a deadlock may occur on intermediate nodes, as they are required to wait for the reception of convergecast messages from all of their children before performing local leader election. A simple solution consists in introducing a timeout forcing local leader election; assuming that reliable estimates for the node-to-node communication delay $d$, and for the processing delay $l$ are available, the timeout can be set during the tree construction phase, by considering the TTL of the received Parent Proposal message:

$$timeout = 2 \cdot TTL \cdot (l + d). \tag{3}$$

## 5 Theoretical Assessment

In order to prove the validity of our approach, and its feasibility in a real scenario including resource-constrained distributed nodes, we provide a theoretical evaluation of the robustness of our algorithm against common security attacks, as well as of its complexity in terms of the required number of exchanged messages and of the overall running time.

### 5.1 Random Selection

The non determinism of the tree construction algorithm implies that the number $n$ of nodes within the $k$-neighborhood is not known a priori, therefore it is impractical to steer the leader selection in accordance to a predefined probability distribution, so we choose to employ the on-the-fly rule depicted in Eq. 2. Now we show via inductive proof, how such rule ensures that each node within the k-neighboorhood is chosen with the same probability to become the leader.

- Suppose that node $j$ candidates itself as leader for the subtree rooted by itself; it generates a Leader Proposal message $LP(j)$ that will be chosen with probability equal to $1/n(j)$ where $n(j)$ is the total number of the nodes within the subtree rooted at $j$.

- Suppose that $LP(j)$ travels among levels of the tree, and that becomes the leader of a subtree rooted at node $i$. Assume that it is true that the selection rule in Eq. 2 ensures a uniform probability of $1/n(i)$, if the tree rooted at $i$, has exactly $n(i)$ nodes.
- Now suppose that the requester node $r$ apply the depicted selection rule; the overall selection probability for an LP message is:
  - $1/n(r)$, the probability that requester chooses itself as leader
  - $[1/n(i)] \cdot [n(i)/n(r)] = 1/n(r)$ the probability for which the node j is selected as leader

This is sufficient to prove that any node $j$ (including the requester node) is chosen as leader with equal probability of $1/n(r)$ if the node rooted at the requester has $n(r)$ nodes. The chosen rule for computing the selection probability distribution finally needs to be coupled with a sampling method in order to effectively perform leader election; in particular our implementation relies on *importance sampling* [17], which uses the local clock of each sensor node as random seed.

## 5.2 Security Analysis

Security aspects of our algorithm are assessed by providing the set of countermeasures that can be adopted against the possible attacks that a malicious user can perform to break the security of the random generation method. As previously mentioned, messages are encrypted by using symmetric key algorithms, where the key is supposed to be shared among nodes; such level of security does not add any burden to our system, as it is already provided by the commonly available communication standard for wireless sensor nodes.

**Masquerading**. Let us suppose that the intruder uses a malicious sensor node. A possible attack consists in attempting to maximize the probability to become the generator node by inferring the generated random number. The attacker might act as follows: a malicious sensor node $j$ is deployed close to the requester node, in order to be accepted as its neighbor; the sensor node would thus generate a Leader Proposal with an high fictitious value for $n_j$, which would be received by the requester which in turn would be fooled choosing the attacker node as the leader. In this case the symmetric key encryption, combined with the use of a nonce, allows to avoid such attack. The requester node inserts within its tree construction message a nonce value $g$. The message is encrypted with the secret key; whenever a node needs to send a leader proposal message, it computes a function of such value (e.g. $f(g) \equiv g + 1$) to prove that the sender node shares the secret key. In this case, the malicious node can only send meaningless messages which would be immediately discarded by the requester. If the nonce has $b$ bits, the attacker could become the leader only guessing the correct value of $f(g)$, with a probability of $1/2^b$. We chose to employ a 128 bit nonce.

**Guessing the leader**. The attacker might attempt to discover the potential leader, in order to steal the measurements to be sent to the requester, so that it could easily forge the random number of the requester. The robustness against such attack is related to the probability that the attacker successfully guesses the leader, which is $1/n(r)$ as shown in Section 5.1. Considering a $k$-neighborhood, and assuming an average branching factor $b$, the number of nodes $N$ that the attacker should examine grows exponentially as $O(b^{k+1})$, thus making such attack impracticable for reasonable values of $k$.

**Brute force on the key**. If the length of secret key is $b$ bits, a brute force attack to the key requires, on average, $2^{b-1}$ attempts. Practical implementations of symmetric key encryption for wireless sensor networks (AES-128), employ 128 bit keys, which may be considered computationally secure at present.

**Traffic Analysis**. A possible way to infer the leader node is to monitor network communications. This kind of attack aims to infer some information by observing the structure of the transmitted messages. The best way to counter such attack is to keep fixed the length of every encrypted message. This is the reason why we included the measurements within the Leader Proposal messages instead of sending them after the leader is elected. If measurements were sent after convergecast phase, it would be possible for the attacker to follow the requester message toward the leader, and violate its ADC reading the generated measurements (as in the Guessing the Leader attack).

## 5.3 Complexity Analysis

Complexity of algorithm can be analyzed by considering its two main constituting phases: tree building and messages convergecast.

The tree building phase is analogous to the Asynchronous Spanning Tree algorithm described in [13] except for the fact that we impose a limit to the depth of the request. Asynchronous Spanning Tree has time complexity of

Table 1: Variance of the election probability for different values of Depth and Branching factor.

| | | Depth | | |
|---|---|---|---|---|
| | | 2 | 3 | 4 |
| **Branching Factor** | 1 | 6.2 | 3.9 | 4.3 |
| | 2 | 2.5 | 1.9 | 1.8 |
| | 3 | 6.3 | 2.2 | 1.3 |
| | 4 | 4.5 | 2.4 | 1.2 |
| | 5 | 2.1 | 1.8 | 1.6 |

$O(diam \cdot (l+d))$, where $diam$ is the diameter of the network, $l$ is the processing delay, and $d$ is the node-to-node communication delay; its message complexity is $O(|E|)$. However it is possible that the longest path of the generated tree has a length greater than $diam$, up to $n$ in the worst case, which affects the subsequent convergecast algorithm imposing a time complexity of $O(n \cdot (l+d))$.
This is one of the major reasons why we chose to modify it by imposing a limit of $k$ to the distance between root node and leaf nodes; the longest path is thus limited to $k$ hops, resulting in a time complexity of $O(k \cdot (l+d))$, while message complexity is left unchanged to $O(|E|)$; the complexity of the convergecast process however is lowered to $O(k \cdot (l+d))$, due to the fixed length of the longest path, while the message complexity is equal to $O(n)$.

Considering both phases, we obtain an overall time complexity of $O(k \cdot (l+d)) = 2k \cdot (l+d)$, and a message complexity equal to $O(n+|E|)$. Although time complexity is optimal and message complexity falls below the lower bound reported in [1], this is not surprising considering that our solution is not optimal. This may be easily checked by observing that, although Asynchronous Spanning Tree (AST) generates a tree, it does not ensure that every node $j$ is reachable from the root node $i$ through the shortest path with distance $d^*(i,j)$; this will in general result in $d^*(i,j) \leq d(i,j)_{AST}$. It may occur that $d^*(i,j) \leq k < d(i,j)_{AST}$, so that some nodes, that would be considered as $k$-neighbors according to a shortest path tree algorithm, will instead be discarded during the election phase, thus increasing the probability that the attacker may successfully break the system by guessing the leader.



# 6 Experimental Evaluation

In order to verify the statistical properties of the distribution probability for leader election, and to provide a comparison between our method and other optimal ones, we devised some test scenarios; in particular we considered a real set of 60 TelosB nodes deployed within an area of 25x15 $m^2$ at our Department.

## 6.1 Uniform Distribution for Leader Election

The purpose of this set of experiments is to show the effective distribution probability for the leaders, when using importance sampling based on the internal clock of TelosB nodes (see Section 5.1); in particular, the random seed was computed as a function of the milliseconds from the time the node was powered on.

Our tests considered the TelosB nodes arranged according to 15 different tree topologies for the network. The branching factors $b$ for each of the configurations ranged from 1 to 5, and the depth $k$ from 2 to 4. Each configuration was tested over 1,000 runs; results were collected at the base station physically connected to a PC.

Table 1 shows the values for the variance on the election probability for each network arrangement; note that all values are scaled by a factor of $10^{-4}$. The number of nodes for each configuration is given by:

$$N = min(60, b^k - 1/b - 1),$$

so clearly the mean selection probability is $1/N$. The variance we obtained is small and falls within the range $1.6 \cdot 10^{-4}$ to $6.2 \cdot 10^{-4}$ , where the mean selection probability ranges from 1/2 to 1/60.

## 6.2 Comparison with Optimal Complexity Algorithms

This set of experiments is aimed to show how our leader election method and other optimal methods differ in terms of reached nodes (for a fixed depth $k$), and in terms of communication and time complexity. In particular we consider our algorithm in its entirety (tree building, and convergecast) and compare its complexity to the lower bounds on

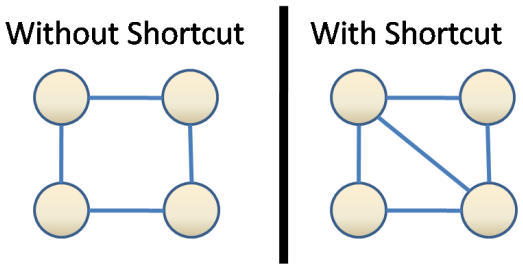


Figure 3: The different topology patterns used to compare our algorithm and the optimal one.

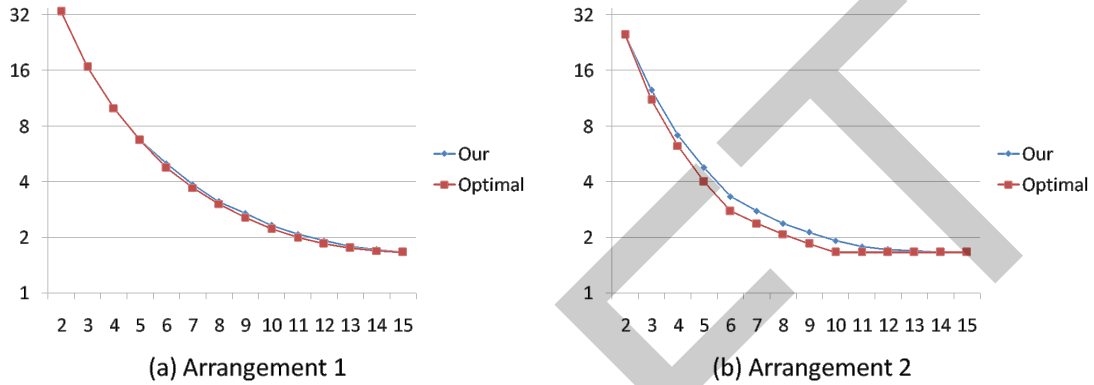


Figure 4: Comparison for the guessing probability normalized in the interval [0;100] (Y-Axis logarithmic scale) between our algorithm and the optimal one for the two arrangements. The X-axis indicates the used values of $k$.



leader election provided in [1]. It is worth stressing the fact that no algorithm can achieve optimality for both measures simultaneously.

Given that our method is not deterministic, due to the use of Asynchronous Spanning Tree, we try to evaluate its behavior by executing it on two kinds of topologies. We considered a grid topology of $6 \times 10$ nodes reproducing the patterns shown in Figure 3 where each link is bidirectional. For the second topology arrangement we included additional (diagonal) links creating shortcuts that will be consistently preferred by optimal algorithms, and sometimes also used by our algorithm. This allowed us to compute the average number of reached nodes in trees generated by our implementation as compared to the optimal algorithms; in practice, this is equivalent to compare the $P_{guess}$ probabilities in both kinds of algorithms.

The algorithm we developed was executed 1,000 times for each value of $k$ ranging from 2 to 15, and for the two kinds of chosen topologies; in addition we set the estimate of node-to-node communication delay $d$ to 1s, while processing delay $d$ was set to 250ms.

Figures 4 and 5 show the experimental results for the selected topologies, with values of $k$ ranging from 2 to 15. Results are evaluated in terms of the probability for an attacker to guess the leader (i.e. the inverse of the average number of reached nodes) and the number of exchanged messages.

For the arrangement 1, $P_{guess}$ is better for the optimal method only for a 0.07% in average; however message complexity of our algorithm outperforms the optimal method by a factor 2 on average. For the arrangement 2, the optimal method reaches all network nodes using the shortcuts at depth $k$ equal to 10, obtaining in average a $P_{guess}$ probability lower than 0.2% from our method; Our algorithm reaches the same performance of the optimal one at depth $k = 11$, although the number of exchanged messages is lower by a factor 1.5 on average.

The experimental results relative to the guess probability are obviously better for optimal methods, since they are able to reach the maximum number of nodes for a given value of $k$. However it can be noted that, from a security point of view, $P_{guess}$ is quasi equal for both methods.

From a complexity point of view our implementation is also equal to time optimal method, reaching the lower bound of $\Omega(k) = 2k$; we omit to explicitly include time complexity in the figures because it is equal for both our and time optimal algorithm. The time needed to perform out a leader election, given the value of $k$, can be computed using Eq. 3 that is linear in $k$. As regards message complexity, the lower bound for optimal algorithms of $\Omega(|E| + n\log n)$

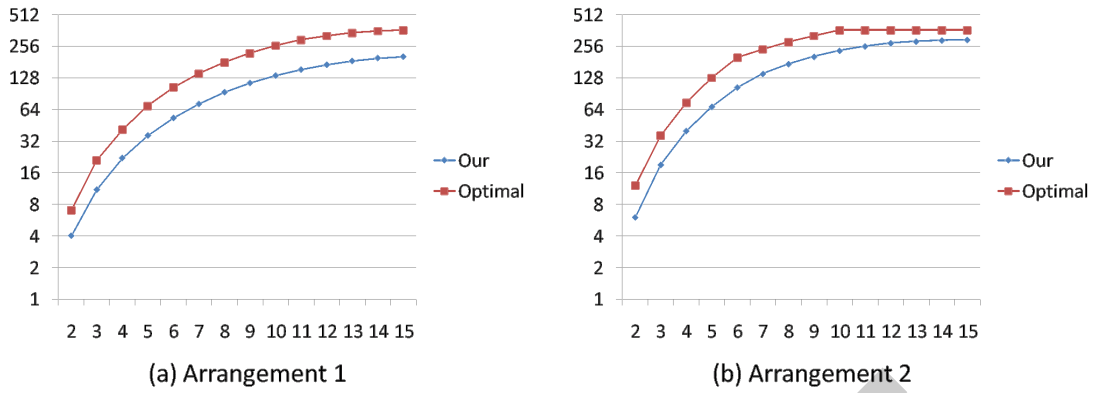


Figure 5: Comparison for the number of messages (Y-Axis logarithmic scale) sent by our algorithm and the optimal one for the two arrangements. The X-axis indicates the used values of $k$.

was used to compute the total number of exchanged messages for optimal leader election algorithm.

By observing experimental results can be concluded that optimal methods offer little improvement to the security (in terms of $P_{guess}$) with respect to the devised approximate method. However, message complexity of our approximate method is substantially lower than the optimal one, offering a great improvement in terms of network traffic.

# 7 Conclusions

This paper discussed an approach to secure random number generation using leader election in a real world scenario involving wireless sensor networks, as the source for physical measurements to be used to provide true randomness. The proposed algorithm involves two subsequent phases, for tree construction and for electing a leader among the sensor nodes, respectively. Both phases are analyzed in detail, with respect to security and complexity issues, and compared to theoretically optimal approaches. We used $P_{guess}$, message and time complexity as discriminant to choose which method is more suitable for random number generation.
From a security perspective our method is slightly worse than the optimal one; however the difference can be considered negligible.
Time complexity of our method and optimal ones are equal, and can not be used to decide which method is better than the other.
Finally message complexity of our method $O(|E|+n)$, is better than the optimal one of $O(|E|+n\log n)$, and can be also observed by numerical results provided in the experimental section.
Considering that the security achieved from both approaches is quite similar, while message complexity is substantially different, we conclude that our approximate security algorithm is more suitable and efficient for the purpose of Secure Random Number Generation with respect to exact optimal methods.

# References

[1] H. Abu-Amara and A. Kanevsky. On the complexities of leader election algorithms. In *Computing and Information, 1993. Proceedings ICCI'93., Fifth International Conference on*, pages 202–206. IEEE, 1993.

[2] Y. Afek and E. Gafni. Time and message bounds for election in synchronous and asynchronous complete networks. In *Proceedings of the fourth annual ACM symposium on Principles of distributed computing*, pages 186–195. ACM, 1985.

[3] E. Chang and R. Roberts. An improved algorithm for decentralized extrema-finding in circular configurations of processes. *Communications of the ACM*, 22(5):281–283, 1979.

[4] W. Diffie and M. Hellman. Privacy and authentication: An introduction to cryptography. *Proceedings of the IEEE*, 67(3):397–427, 1979.

[5] D. Estrin, L. Girod, G. Pottie, and M. Srivastava. Instrumenting the world with wireless sensor networks. In *Acoustics, Speech, and Signal Processing, 2001. Proceedings.(ICASSP'01). 2001 IEEE International Conference on*, volume 4, pages 2033–2036. IEEE, 2001.

[6] A. Francillon and C. Castelluccia. 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. IEEE, 2007.

[7] V. Gaglio, A. De Paola, M. Ortolani, and G. Lo Re. A TRNG exploiting multi-source physical data. In *Proceedings of the 6th ACM workshop on QoS and security for wireless and mobile networks*, pages 82–89. ACM, 2010.

[8] R. Gallager, P. Humblet, and P. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Transactions on Programming Languages and systems (TOPLAS)*, 5(1):66–77, 1983.

[9] D. Hirschberg and J. Sinclair. Decentralized extrema-finding in circular configurations of processors. *Communications of the ACM*, 23(11):627–628, 1980.

[10] 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*, pages 103–111. ACM, 1998.

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

[12] G. Le Lann. Distributed systems, towards a formal approach. *Information processing*, 77:155–160, 1977.

[13] N. Lynch. *Distributed algorithms*. Morgan Kaufmann, 1996.

[14] N. Malpani, J. Welch, and N. Vaidya. Leader election algorithms for mobile ad hoc networks. In *Proceedings of the 4th international workshop on Discrete algorithms and methods for mobile computing and communications*, pages 96–103. ACM, 2000.

[15] V. Park and M. Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *infocom*, page 1405. Published by the IEEE Computer Society, 1997.

[16] G. Popek and C. Kline. Encryption and secure computer networks. *ACM Computing Surveys (CSUR)*, 11(4):331–356, 1979.

[17] B. Ripley and E. Corporation. *Stochastic simulation*, volume 21. Wiley Online Library, 1987.

[18] B. Sunar, W. Martin, and D. Stinson. A provably secure true random number generator with built-in tolerance to active attacks. *IEEE Transactions on computers*, pages 109–119, 2007.

[19] S. Vasudevan, B. DeCleene, N. Immerman, J. Kurose, and D. Towsley. Leader election algorithms for wireless ad hoc networks. 2003.

[20] S. Vasudevan, J. Kurose, and D. Towsley. Design and analysis of a leader election algorithm for mobile ad hoc networks. 2004.