Research Article

On the Performance of a Secure Storage Mechanism for Key Distribution Architectures in Wireless Sensor Networks

Juliano F. Kazienko,1 Paulo R. S. Silva Filho,2 Igor M. Moraes,2 and Célio V. N. Albuquerque2

1Computing Department, Federal University of Pampa, 97546-550 Alegrete, RS, Brazil
2Computing Institute, Fluminense Federal University, 24210-240 Niterói, RJ, Brazil

Correspondence should be addressed to Juliano F. Kazienko; kazienko@unipampa.edu.br

Received 10 October 2014; Accepted 24 March 2015

Security in wireless sensor networks (WSNs) demands efficient key management schemes. In particular, one of the main research challenges is to ensure secure key storage by sensors due to their constrained resources and high exposure to tampering attempts. To address this issue, we have proposed SENSORLock, a secure key storage mechanism which can be applied to different key distribution architectures for WSNs. In this work, we evaluate SENSORLock through different study cases considering three key distribution architectures: TinySec, SPINS, and NCD. Our goal is to demonstrate its feasibility in large-scale sensor networks.

Simulation results indicate that our mechanism introduces an average overhead of 1.9% in terms of end-to-end delay and provides a similar estimated power consumption compared to the aforementioned architectures. Hence, we argue that it is feasible to use SENSORLock (i) in large-scale sensor networks and (ii) under different key distribution schemes.

1. Introduction

Wireless sensor networks (WSNs) are a particular case of mobile ad hoc networks (MANETS). Sensor networks are composed of tiny nodes that collect data from the environment in which they are deployed, sending data through multiple hops toward sink nodes. These networks are applied with several purposes from weather forecasting to assisted living, contributing to the pervasive computing [1, 2].

Key management in WSNs certainly drives many issues. Indeed, it is frequently claimed that standard security mechanisms are prohibitive in WSNs because they demand extensive use of scarce resources, such as processing power, battery capacity, limited memory, and low bandwidth. However, since sensors are typically unattended, it is a challenge to protect sensitive data stored in sensors, as well as cryptographic keys. Key storage is a challenging issue in sensor networks which still demands suitable solutions [3–5].

In a previous work, we introduced SENSORLock [6]. It is a mechanism that assures secure key storage in sensors. In that work, its architecture is proposed and a first study case is presented where SENSORLock is applied. Besides, the evaluation was carried out so as to (i) show mathematically that the whole system and its keys are protected even if sensors are tampered with and (ii) determine the mechanism processing overhead using small-scale experiments. However, that work does not demonstrate the feasibility of using SENSORLock in different key distribution architectures and large-scale WSNs.

The main contribution of this work is to extend the application and supply a large-scale evaluation of SENSORLock to three distinct key distribution architectures: TinySec [7], SPINS [8], and network coding distribution (NCD) [9]. In this sense, we analyze SENSORLock through simulations using the TinyOS platform considering large-scale networks with 49, 100, 144, 225, and 400 nodes. For experimentation, we use the Collection Tree Protocol (CTP) as routing protocol and the Advanced Encryption Standard (AES) Algorithm. Simulation results reveal that SENSORLock scheme introduces an average end to end communication delay overhead varying from 0.5% to 4.4%. Moreover, the results indicate...
2. Related Works

According to Menezes et al. [10], key management deals with the following procedures: key generation, distribution, and deployment; key control and usage; key revocation and elimination; and key storage and backup. In our previous work, we focused on secure symmetric keys storage supported by a cryptographic module (CM) responsible for holding a secret Z used to encrypt all system keys and data stored within the sensor’s flash memory. We argue that CMs are affordable and, therefore, can be incorporated to off-the-shelf sensors, as discussed in Kazienko et al. [6].

In general sense, WSNs security architectures have been more concerned with key distribution, as the work of Alcaraz et al. [13]. Indeed, it consists in an important issue specially in wireless networks in which data is potentially subject to unauthorized access. However, it is also important to protect keys stored in unattended sensors. If a sensor is tampered with, it may compromise the network partially or even completely depending on the key distribution scheme adopted.

According to Hu and Sharma [14], there are two classes of tamper proof mechanisms: active and passive. The former involves tamper proof hardware circuits within the sensor nodes that enable an immediate response to tampering attempts, what makes them tamper-responsive. The latter includes mechanisms that do not require active circuits, such as protective coatings and tamper seals. In general, such material is tamper-evident, but some of them may damage the protected area if removed and, because of that, they are also called tamper-resistant mechanisms.

Additionally, some integrated circuit manufacturers offer affordable cryptographic modules (CMs), also known as trusted platform module (TPM). Table 1 shows CMs and their features, applications, and prices. As example of an off-the-shelf CM is AT97SC3204-U1M90 Atmel Crypto Controller—TPM [5]. Such CM costs USD 5.61 that corresponds to just 4% of the cost of a sensor TelosB or MicaZ. Therefore, we argue that it is an acceptable cost to aggregate security in sensor’s operation. In general, such CMs use active tamper proof mechanisms in order to protect sensitive data.

Traditionally, such modules are prepared to operate with asymmetric encryption algorithms, secure storage area, and true random number generator. However, public key cryptography is not suitable to sensor networks due to its processing costs and the need of an infrastructure that is difficult to provide in ad hoc networks, specially WSNs. On the other hand, symmetric encryption is computationally cheaper demanding low complexity infrastructure to such networks. Thus, sensor networks CMs could be simpler and should be customized to operate in this kind of network.

Martina et al. [15] proposed an open protocol targeted at secure private key management within hardware security modules (HSMs). The authors argued that such security modules protect the private key against attacks that involve logical and physical tampering or even the extraction of sensitive information from the protected area. As the protocol is designed to public key infrastructure environments, internal hardware and functions are prepared to operate with asymmetric encryption that implies higher computational cost and power consumption than symmetric encryption, as previously discussed.

One of the most popular architectures to sensor networks security is TinySec [7]. The authors argue that end to end security is unsuitable to WSNs due to data aggregation because intermediate nodes eventually may need to access, modify, and eliminate such information. As an alternative, they defend that security services such as authentication, integrity, and confidentiality should be provided between neighboring nodes, proposing their implementation at the link layer. The authors explain their architecture may support the development of a key distribution mechanism; however they do not define such mechanism. The TinySec architecture enables the authenticated message exchange among sensors through message authentication codes (MAC). Thus, the proposal uses a shared master key predistributed among all network nodes. For this reason, if an attacker captures the keys of just one node he has access to all network communications.

Perrig et al. [8] proposed security protocols for sensor network (SPINS) that defines a set of security protocols
targeted to sensor networks. It is composed of two blocks: SNEP, Secure Network Encryption Protocol, and μTESLA, in reference to the microversion of Timed, Efficient, Streaming, Loss-Tolerant Authentication Protocol. The former provides data confidentiality, authentication between nodes, and data freshness. The latter provides authenticated broadcast for networks with scarce resources. In SPINS, the security services are provided basically through symmetric encryption and MACs. The authors propose a symmetric protocol that relies on a sink node as a trusted third part responsible for pairwise key distribution. Each node shares a predistributed master secret key with the sink node. From this key, other keys are derived and used for secure communication between nodes. The work of Perrig et al. [8] does not assume secure storage.

Differently from previous proposals, the system of Oliveira and Barros [9], hereinafter referred to as network coding distribution (NCD), is based on key distribution, key predistribution, and mobility. In such scheme, the task of key distribution, which is carried out using a network coding technique [16], is accomplished by a mobile node (MN). This system, however, has a limitation: it allows an attacker to discover all keys used in the system when accessing the memory of any sensor along with the mobile node memory access.

On the other hand, SENSORLock deals with the problem of security in sensor networks from a broader perspective. It considers the need for secure key storage in sensor nodes quite important since they are great in number and their content is subject to capturing and tampering attempts. Thus, besides security in key distribution it is also intended, as explained in Section 3, to provide a larger resiliency and robustness to the system in such a way that the sensors’ capturing and tampering will not result in any key discovery.

3. SENSORLock Mechanism

First, we present SENSORLock and its fundamentals in Section 3.1. After, in Sections 3.2 and 3.3, we propose how to apply SENSORLock to two classical WSN key distribution schemes. Both are based on key predistribution. In Section 3.4, we propose the application of SENSORLock to a system based on key distribution that uses multiple keys. Thus, we demonstrate that SENSORLock is applicable to different key distribution schemes.

3.1. Fundamentals. SENSORLock is a secure key storage mechanism which can be applied to different key distribution architectures for WSNs. Essentially, SENSORLock mechanism addresses the key exposure problem by defining the following fundamentals:

(1) All keys stored in memory must be encrypted with a sensor private secret $Z_s$ from sensor $s$.

(2) The secret $Z_s$ must be stored in a cryptographic module (CM) with tamper proof technology.

(3) The CM must contain hardware modules to encrypt and decrypt keys and messages without even exposing the secret $Z_s$ or the keys used to encrypt and decrypt messages.

Based on such fundamentals, SENSORLock CM architecture is depicted in Figure 1. The secrets $Z_s$ and $Z'_s$ are stored in nonvolatile memory. The secret $Z_s$ is used to encrypt and decrypt all keys stored in sensor memory. On the other hand, $Z'_s$ is generated and used during the $Z_s$ key renewal process. Such keys are stored in tamper-responsive device that assures the destruction of them when a tampering attempt to CM is detected. A key generation (KG) block is used for generating and renewing of $Z_s$ secret.

The block $F_1$ is used to key initialization. For that, it receives a key from sensor’s memory and secret $Z_s$ stored within CM. The output of this block is the encrypted key which is stored in sensor’s memory. Moreover, $F_1$ block and exclusive or block $\oplus$ cooperate for receiving keys. That is, the block receives the distribution message and recovers a key that belongs to the other sensor, represented as $K_{(s2)}$ in Figure 1. Soon after that, such key is input of $F_1$ block, which encrypts it within the CM and stores it within the sensor’s memory.

The block $F_2$ performs key decryption. Its input is an encrypted key from the sensor’s memory and the secret $Z_s$. The outputs are the key $K_{(s)}$ in plain text from sensor $s$. Such key may be input of exclusive or block, as aforementioned, or input of the block $F_3$.

The block $F_3$ receives as input the key $K_{(s)}$ and a message $M$. The kind of processing that the block $F_3$ does over such message is defined through control information from the application. Thus, the block $F_3$ can process the message in different ways: encryption $E_{K_{(s)}}(M)$, decryption $D_{K_{(s)}}(M)$, and computation of message authentication codes $MAC_{K_{(s)}}(M)$. Moreover, the application selects the suitable key from sensor’s memory according to the security service demanded (confidentiality, authenticity, or integrity) and sends it to SENSORLock. As example of possible outputs from block $F_3$, we can cite the message $M$ encrypted and the message $M$ joined to the message authentication code, which are sent to the sensor communication system. On the other hand, when the sensor is receiving a message through sensor communication system, the message $M$ encrypted or the message $M$ joined to the message authentication code can be input of block $F_3$. In this case, the message $M$ is output.

![Figure 1: SENSORLock cryptographic module architecture.](image-url)
It is important to realize that even though the main motivation to use SENSORLock lies on the protection of keys stored in sensor’s memory, its design allows encrypting all data collected by sensor.

The CM’s generic design supports free choice of symmetrical algorithm, to encrypt and decrypt the secret data collected by sensor. The CM presented in Section 4.1, also used in SENSORLock of entropy. TinyRNG uses encryption algorithms, as CBC-MAC presented in Section 4.1, also used in SENSORLock what facilitates its implementation as KG within CM blocks allowing its customization. We can cite as symmetric ciphers the RC5 and AES [17–19].

As SENSORLock random number generator, an option of use is TinyRNG proposed by Francillon and Castelluccia [20]. It consists in a deterministic algorithm that generates pseudo-random numbers for nodes in WSNs. The authors proposed the use of bit errors in messages received by sensor as source of entropy. TinyRNG uses encryption algorithms, as CBC-MAC presented in Section 4.1, also used in SENSORLock what facilitates its implementation as KG within the proposed mechanism. Besides, a public implementation of TinyRNG is available which may contribute to reducing SENSORLock production costs.

In this paper, SENSORLock is applied to three symmetric key distribution architectures: TinySec, SPINS, and NCD. TinySec and SPINS are well-known security architectures for WSNs. They are based on key predistribution where a reduced number of keys are stored within sensors. Differently, NCD is based on predistribution and distribution. Moreover, it supports the storage of a huge number of keys in each sensor. It is important to highlight that SENSORLock is applicable to all architectures aforementioned, as detailed in the next sections.

3.2. Study Case I: SENSORLock Applied to TinySec. To solve the stored keys exposed problem observed in TinySec architecture and increase its robustness, we propose the application of SENSORLock in its context. TinySec provides security through shared master keys, called TinySec KEYS, among all network nodes: \( K_{\text{conf}} \) is used for confidentiality and the other, \( K_{\text{MAC}} \), is used to supply authentication and integrity of messages. Therefore, it is important to highlight that the capture of just one node by an attacker may lead to the impersonation of legitimate sensors and the access to network communications.

TinySec is depicted in Figure 2(a). In steps (1), (2), and (3), we can verify that the keys \( K_{\text{MAC}} \) and \( K_{\text{conf}} \) are distributed before the network operation. In step (4), the keys are used to encrypt a message. If confidentiality, integrity, and authentication are important, first \( K_{\text{conf}} \) is used to encrypt the packet payload field. Later on, the computation of the message MAC is done. Beyond the key \( K_{\text{MAC}} \), such computation receives as input whole packet. Finally, the MAC is attached to the packet. In step (5) the decryption of the message is carried out.

In Figure 2(b), SENSORLock is applied to TinySec. Initially, the keys are generated and predistributed among the sink and sensors. In the sensors A, B and sink, such keys are initialized, as shown in steps (1), (2), and (3). That is, \( K_{\text{MAC}} \) and \( K_{\text{conf}} \) are encrypted with the key \( Z_i \) from each sensor. Afterwards, the plain text versions of the keys are removed from the sensor memory, remaining only their encrypted version.

Suppose that Sensor A wishes to send a message with confidentiality to Sensor B; first we should accomplish the decryption operation \( D_{Z_i}(K_{\text{conf}}) \), as depicted in steps (4) and (5). Later on, the message \( M \) is encrypted \( E_{K_{\text{conf}}}(M) \). On Sensor B, \( M \) is received (7). \( K_{\text{conf}} \) is decrypted through steps (8) and (9). Finally, the message is decrypted in Sensor B through the function \( D_{K_{\text{conf}}}(M) \) (10).
Through the application of SENSORLock, $K_{\text{MAC}}$ and $K_{\text{conf}}$ are stored in a encrypted way in sensor memory. All encryption and decryption operation of such keys is accomplished within CM, avoiding plain text keys exposure in sensor memory. In summary, SENSORLock shields the keys used in TinySec architecture avoiding that attackers can read plain text keys from sensor memory, impersonating legitimate nodes or listen to the network communications.

For the SENSORLock applied to TinySec case, we argue that the access to the stored keys within sensors, sink included, simultaneously does not aggregate information to system keys discovery. That is, as each sensor encrypts keys with its own secret $Z$, the key discovery likelihood remains $1/2^n$ corresponding to brute-force attack.

### 3.3. Study Case II: SENSORLock Applied to SPINS

The second study case presents SENSORLock application to SPINS. In SPINS, in SNEP block, the authors pose an approach based on the use of pairwise shared master symmetric keys $X$ between sink and each sensor node. Perrig et al. [8] explain that other keys are generated from master key in order to provide security services. However, we point out that the secure storage of this key becomes critical. Since all master keys are stored as plain text, in sensors and in the sink, a capture and memory reading attack of a given Sensor $A$ compromise (i) the communications security between sensor and sink and also the communications security among sensor nodes.

Figure 3(a) depicts the SNEP key predistribution scheme. For the supplying of security services, SNEP uses symmetric cryptography and MAC. A project option is the use of different keys for encryption and MAC computation so as to avoid key reuse. In the network initialization, the shared key $X (X_{AS}, X_{BS}, \text{etc.})$ is loaded in regular sensor and sink memories, as depicted in steps (1), (2), and (4) of Figure 3(a). Such key is not used for message encryption. Nevertheless, the key $X$ is used as input of a derivation function [8] that yields, on its turn, the keys $K_{AS}, K_{SA}, K'_{AS},$ and $K'_{SA}$, steps (1) and (2), considering Sensor $A$ as example. Actually, such keys are employed in message encryption, as shown in steps (3) and (5). The keys $K$ are used for the confidentiality service and the keys $K'$ are used for the message authentication service. These keys are directional keys used according to the communication direction.

An example of key agreement using SNEP is presented in Perrig et al. [8]. Nevertheless, a lack of secure storage becomes possible that an attacker impersonates a given sensor and accomplishes key agreement with a legitimate sensor. In Figure 3(b), it presented SENSORLock application to SPINS encrypting the master key $X$ and derived keys $K$ as in sensors as in sink that mitigates the sensor impersonation problem by an attacker. In their proposal, the authors do not clarify how long such keys remain in memory. Also, what is the impact of frequently keys generation in terms of processing overhead and energy consumption is not addressed. We argue that such points are important in WSNs that typically have a huge amount of nodes. On the other hand, when the encryption of such keys is done, the keys $K$ can be stored for long periods in a safe way in sensor, avoiding generation, that is, derivations $X_{AS} \rightarrow K_{AS}, K_{SA}, K'_{AS}, K'_{SA}$ frequently.

With the encryption of the master key $X$ and derived keys $K$, SENSORLock application to SPINS prevents impersonation of legitimate sensors and, consequently, key agreement between an attacker node and a legitimate node.

For this study case, we highlight that the access to the stored keys within sensors, sink included, simultaneously does not aggregate information to system keys discovery. That is, as each sensor encrypts keys with its own secret $Z$, the key discovery likelihood remains $1/2^n$ corresponding to brute-force attack.

### 3.4. Study Case III: SENSORLock Applied to NCD

This study case is based on the Oliveira and Barros [9] scheme referred in this paper as network coding distribution (NCD). A previous
study of this system was carried out in Kazienko et al. [6]. In such work, it was presented a mathematical analysis and small-scale experiments in order to measure what is the processing overhead due to the application of SENSORLock in communications between sensors. On the other hand, in this work, we evaluate such scheme in large-scale WSNs.

The scheme in Figure 4(a) depicts pairwise keys distribution in WSNs. All system keys are stored as encrypted text in MN that accomplishes XOR operation between previously stored keys, one from Sensor A and other from Sensor B. These sensors want to communicate with each other (1). Through such operation, the key R is canceled out. In (2) it is possible to verify the pairwise key distribution encrypted broadcast transmission. In steps (3) and (4), the keys are received and stored as plain text in Sensors A and B. In this approach, if MN and a network regular sensor are captured [4], it is easy to derive the information R which is used to encrypt all system keys, compromising its security. On the other hand, we point out that SENSORLock can be applied to encrypt several keys as demanded by NCD architecture analyzed in this study.

In Figure 4(b), SENSORLock is applied. In the first place, it takes place the network initialization. In MN, all system keys are generated and stored in an encrypted way with the key R (1). In sensors, the initialization consists in encrypting predistributed keys to them, as shown in steps (2) and (3). After this procedure, just the encrypted versions of such keys remain in flash memory, and the other version, in plain text, is removed.

In operation phase, NM sends a broadcast message (5) that is decoded only by sensors A and B (6) (7). In (6), it takes place as the $K_{i(A)} \oplus K_{r(B)} \oplus K_{r(A)}$. As result of that operation, it is obtained as $K_{i(B)}$. Later on, B’s key is encrypted using the secret $Z_A$ and stored within sensor flash memory. When it is needed to send a encrypted message $M$, such key must be decrypted. This is done within of the CM as follows: the encrypted key $E_{Z_A}(K_{i(A)})$ (8) and the key $Z_A$ (9) are input parameters of the decryption function $D_{Z_A}$; thus $D_{Z_A}(E_{Z_A}(K_{i(A)})) = K_{i(A)}$. This key, $K_{i(A)}$, and the message to be sent $M(10)$ are input parameter to the encryption function yielding the encrypted text to be sent: $E_{K_{i(A)}}(M)$.

On the other hand, Sensor B receives the encrypted message $E_{K_{i(A)}}(M)$ (11). Later on, it decrypts A’s key (12) using its secret $Z_B$ (13) in the following way: $D_{Z_B}(E_{Z_B}(K_{i(A)})) = K_{i(A)}$. This key, $K_{i(A)}$, and the encrypted message serve as input parameter of another decryption function. Such function yields a plain text message (14) by the operation $D_{K_{i(A)}}(E_{K_{i(A)}}(M))$. Therefore, all messages are encrypted with the sender’s key.

The generic design of CM ensures freedom to choose the symmetric algorithm, as RC5 [17] and AES [19], for example, used by cryptographic functions running within CM. Additionally, we present a mathematical analysis to demonstrate SENSORLock’s security against capturing and tampering attacks.

Differently from the NCD’s proposal, where the access to the sensors memory allows an attacker to discover all the keys of the system, SENSORLock’s design assumes, by definition, that an attacker can exploit two attack points: the mobile node and the sensors. We specifically consider the following attacker model composed of threats caused by an attacker:

(i) The attacker is capable of listening to the wireless medium.

(ii) The attacker has free transit in the area in which the WSN is deployed.

(iii) The attacker can capture and access the memories of the sensors, the mobile node or both.
As demonstrated by NCD's authors, the access to the MN memory in an isolated way and consequently the knowledge of \(\{K_1 \oplus R, K_2 \oplus R, \ldots, K_m \oplus R\}\) does not increase the information that the attacker possesses regarding any key of the system. Mathematically, that verification can be described as:

\[
P(K_i = x \mid K_1 \oplus R = y_1, \ldots, K_m \oplus R = y_m) = P(K_i = x) . \tag{1}
\]

The result of (1) is given by the application of Bayes' theorem. After defining events \(A\) : \(K_i = x\) and \(B\) : \(K_1 \oplus R = y_1, \ldots, K_m \oplus R = y_m\) and calculating the probability \(P(A \mid B)\), the independence of both events is verified:

\[
P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)} = P(A) . \tag{2}
\]

As \(A\) and \(B\) are independent, \(P(A \mid B) = P(A)\), from (2), the occurrence of event \(B\) (the knowledge of the MN ciphered keys) does not influence the occurrence of event \(A\) (the discovery of a system key).

The same reasoning used for (1) is valid for secure key storage in the sensor side. In such case, the memory content knowledge of certain sensor \(s\) isolated, that is, \(\{K_1 \oplus Z_s, K_2 \oplus Z_s, \ldots, K_m \oplus Z_s\}\), does not supply more information to the attacker about the keys stored in the sensor. Hence, by defining event \(C\) : \(K_1 \oplus Z_s = w_1, \ldots, K_m \oplus Z_s = w_m\), it is verified that \(P(A \mid C) = P(A)\), where event \(C\) denotes the knowledge of the ciphered keys stored in the sensor.

Moreover, SENSORLock allows an additional level of security in relation to the approach wSS, which makes the system more robust, even if the attacker has access to the memories of the mobile node and of any sensor from the network, simultaneously. The resiliency against such event is demonstrated as follows.

**Theorem 1.** The knowledge of memory content of a sensor and the mobile node, simultaneously, does not increase the information concerning any key used by the system.

We have already seen in the previous paragraphs that \(P(A \mid B) = P(A)\) and \(P(A \mid C) = P(A)\). Additionally, we want to prove that \(P(A \mid B \cap C) = P(A)\). As events \(A, B,\) and \(C\), previously defined, are mutually independent, the multiplication theorem or compound probability is applied:

\[
P(A \cap B \cap C) = P(A \mid B \cap C) \cdot P(B \mid C) \cdot P(C) . \tag{3}
\]

By redrafting (3), the following is obtained:

\[
P(A \mid B \cap C) = \frac{P(A \cap B \cap C)}{P(B \mid C) \cdot P(C)} . \tag{4}
\]

However, on account of the independence among events, we can consider that \(P(A \cap B \cap C) = P(A) \cdot P(B) \cdot P(C)\) and \(P(B \mid C) = P(B)\). Then, it is possible to substitute the terms in (4) as follows:

\[
P(A \mid B \cap C) = \frac{P(A) \cdot P(B) \cdot P(C)}{P(B) \cdot P(C)} . \tag{5}
\]

From that we obtain:

\[
P(A \mid B \cap C) = P(A) . \tag{6}
\]

Hence, we demonstrate the system's security increases by the using of SENSORLock since the capturing of sensors and MN simultaneously does not increase the information concerning any key used by the system.

### 4. Evaluation

In this section, we present the experiments in order to evaluate SENSORLock mechanism. First, we present the methodology used. Second, the implemented architectures are described. Finally, we reveal our findings.

#### 4.1. Methodology

Firstly, we want to highlight that a security analysis was carried out in Kazienko and Albuquerque [21]. Additionally, real small-scale experiments were accomplished in Kazienko et al. [6] where a small scenario was deployed with TelosB and MicaZ sensors. Experimental results revealed the low footprint of code and SENSORLock low processing overhead. Another remark that facilitates experiments consists in the code developed for TinyOS: the same code is used for simulations and practical experiments.

In this work, for large-scale simulations, the following tools and platforms are used: TinyOS, nesC, TOSSIM, CTP, and security mechanisms. In the next paragraphs, we describe each of them.

TinyOS is a component-based operating system specifically developed for WSNs. TinyOS is written in nesC (network embedded systems C) language and it is free and open source. Such language is an event-driven programming language used to create sensor applications to run on TinyOS [22]. In our experiments, version 2.1.1 was used. TinyOS is operating system based in components. There are two kinds of components: module that contains the application implemented and configurations, responsible for connecting components.

For simulations, we used TinyOS Simulator (TOSSIM) [23]. TOSSIM is a discrete event simulator and works by translating hardware interrupts into discrete events. TOSSIM is a library of TinyOS. It is important to highlight that the TinyOS Simulator supports only the MicaZ architecture in its compilation process. Therefore, our experiments are performed over such architecture.

An important component available in TinyOS is the Collection Tree Protocol (CTP). It consists in a protocol responsible for routing and forwarding messages from regular network nodes to sink node. CTP organizes the network in tree structure with father and son nodes. The protocol builds routes of minimal cost to sink using the expected transmission count (ETX) metric. It is important to highlight that all implemented architectures in this work use such routing protocol. CTP was tested in different testbeds and was integrated with TinyOS 2.1.1, which supported our decision in its adoption. However, any WSN routing protocol could be considered for it.
Regarding security algorithms, we adapt the codes of [7, 24] according to our experimental purposes. All implemented and evaluated architectures in this paper employ AES encryption algorithm in cipher block chaining (CBC) operation mode. Also, we use CBC-MAC algorithm for MAC computations. Such MAC algorithm operates in CBC mode with AES as encryption algorithm. Its mechanism is quite similar to pure encryption, just for confidentiality. The main changing consists in filling with zeros the initialization vector. Thus, the last block of the message serves as authentication code [25]. It is important to mention that any block cipher symmetric algorithms could be used in our experiments. However, in order to carry out a fair comparison among evaluated architectures, security services are implemented using both AES, for confidentiality, and CBC-MAC, for authentication and integrity, in all architectures.

4.2. Simulated Architectures Description. We implemented specific applications so as to reproduce the working of each architecture. Thus, there are 7 simulation scenarios. Actually, there are 35 scenarios because for each scenario, the 7 ones said previously, are simulated networks with 49, 100, 144, 225, and 400 sensor nodes. The simulated area is 300 × 300 meters. The topology is relatively uniform, since the area is divided into the same amount of cells compared to the number of simulated sensors. Nevertheless, each sensor may be deployed in any part inside the cell. The 7 key distribution architectures implemented are described as follows:

1. Without Cryptography System. (wC) There is neither message nor key encryption in such architecture. Whole messages are sent as plain text.

2. TinySec. (TinySec) In such approach, the same key is predistributed among network sensors. The sensors generate a message and compute its MAC, sending both through the network. Intermediate sensors an sink receive such a message and compute MAC's received message, that is, MAC'. These nodes just forward messages where MAC = MAC'. TinySec is described in Section 3.2.

3. SENSORLock over TinySec. (SL/TinySec) In this scenario, SENSORLock is applied to TinySec. It was coded as a function that plays the role of cryptographic module where takes place encryption, decryption and secret storage. In this architecture, the predistributed key used to compute MACs is encrypted with Z, and stored within the sensor memory. Whenever the sensor sends a certain message, such a key is decrypted, with Z, and used to encrypt the message. In Section 3.2, it has shown more details about how to apply SENSORLock to TinySec architecture.

4. SPINS. (SPINS) In this architecture MACs are computed based on a shared secret X between sink and a regular sensor. For MACs encryption and computing, X is not used. But it is derived K-MAC from X. We implemented a MAC function to carry out such derivation, since the SPINS's authors use MAC in their proposal. During the sending of a message, each node receives each intermediate node receives and forwards it without MAC verification, as long as X key is not known by intermediate nodes. That is, in this approach, it is provides end to end encryption. It may be interesting or not according to the sensor network application requirements. Section 3.3 presents SPINS in detail.

5. SENSORLock over SPINS. (SL/SPINS) In this scenario, SENSORLock is applied to SPINS, protecting keys stored in sensors. Section 3.3 described the application of SENSORLock to SPINS.

6. Network Coding Distribution. (NCD) This scheme uses pairwise key distribution and key pre-distribution. Initially, the keys are loaded into the sensors. The MN knows and stores all network keys in a encrypted way. It is important to point out that a pair of sensors communicate between themselves only if they share a key. In this scenario we have two kinds of messages: distribution messages and data sending messages. Both are transmitted intermittently. Since the sensors recover pairwise keys, the communications take place. We point out that the communication takes place between node son and node father; that is, such nodes must share keys so that there is communication between them. Before forwarding a message, the node son verifies if there is a shared key with its father. In case it is true, the message is sent. The keys must be shared between sons and fathers due to the CTP son-father communication flows, which is used in our implementation. It is sufficient that just a pair of sensors in the end to end path do not have shared keys so that the message does not arrive to sink node. In such case, the message is discharged before to reach the sink. NCD architecture is discussed in Section 3.4.

7. SENSORLock over Network Coding Distribution. (SL/NCD) In this scenario, SENSORLock is applied to NCD. In SL/NCD, the preloaded keys into the sensors are encrypted with Z, Before the sensor sends a message, the key must be decrypted so as to encrypt the message. When the MN distributes keys, the sensors receive these keys and soon after they encrypt them with their secrets Z, Such new architecture, SENSORLock applied to NCD, is described in Section 3.4.

Among the studied architectures, NCD demands a MN for key distribution. Nevertheless, it is important to remark the authors do no specify its the mobility model in their proposal. Due to it, in NCD codification, it is used as a random mobility model. In this sense, during each simulation run, MN is deployed in 10 different positions within the 300 × 300 m terrain. Also, so as to simplify NCD architecture implementation, in each position the MN distributes all keys combination. It differs from NCD description; however it does not affect the results.
Table 2: Simulation parameters.

| Parameters       | Value                                                                 |
|------------------|----------------------------------------------------------------------|
| Number of events | 10,000,000                                                            |
| Number of runs   | 100                                                                  |
| Confidence level | 95%                                                                  |
| Number of sensors| 49, 100, 144, 225, and 400                                           |
| Simulated area   | 300 × 300 meters                                                      |
| Source of messages| Sensors                                                                |
| Target of sensors| Sink                                                                  |
| Size of messages | 16 bytes                                                              |

4.3. Results and Discussion. An experimental evaluation is presented in this section. Simulation was carried out so as to measure SENSORLock’s overhead when applied to key distribution architectures. For that, 7 architectures, described in Section 4.2, are simulated. Our goal is to verify the level of overhead caused by the additional encryption needed to solve the key exposure problem by using the SENSORLock scheme. For this evaluation, we consider the following metrics: end to end delay, power consumption estimation, number of sent/received packets, number of hops, access medium time, and scalability. Table 2 shows simulation parameters. Each simulation stop when the number of events generated described in Table 2 is reached. As examples of events, we can cite the transmission and receiving of messages, access to sensor’s memory, sensing, and so on [23].

To start with, we present the average end to end delay for a 49-node network in Figure 5. This average is obtained from 100 simulation runs. It is observed as a very low time difference among studied systems. The end to end delay difference obtained in SL/TinySec approach over TinySec is 2.057 ms. On the other hand, the results indicate an end to end delay difference from SL/SPINS approach over SPINS of 8.877 ms. Also, the end to end delay difference observed in SL/NCD over NCD is 13.580 ms. Hence, it is fair to state that SENSORLock introduces an average overhead of 8.171 ms (1.3%) and, on the other hand, it offers the benefit of ensuring confidentiality, authenticity, and integrity in network communications.

In general, the values oscillate between 453.482 ms and 1152.735 ms for wC, TinySec, SL/TinySec, SPINS, and SL/SPINS architectures. We observe that wC architecture presents a lower average delay in comparison to SPINS and TinySec architectures. On the other hand, the average delay in SPINS and SL/SPINS slightly exceeds wC architecture. Differently from wC system, SPINS accomplishes cryptographic procedures as key derivation and MAC computation by a regular sensor node. Additionally, sink node derives the key and computes MAC again so as to verify message authenticity and integrity. The average value of SL/SPINS denotes a negligible overhead over SPINS, as observed. However, the bar that represents the TinySec architecture overcomes the other approaches exposed until now. It is relevant to remember that in TinySec implementation a MAC is computed per hop in end to end path, causing major processing overhead and, consequently, increasing the end to end delay when compared to wC and SPINS approaches. As indicated through our experiments, SL/TinySec introduces a negligible overhead in terms of end to end delay increasing compared to TinySec.

On the other hand, the NCD and SL/NCD architectures obtained low delays compared to remaining approaches. Several factors contribute to such behavior, as the NCD architectures peculiarities and the CTP protocol operation. Implementation details and NCD peculiarities help to explain the obtained results regarding delay. In NCD approach, it is enough that just a pair of sensors (node son and node father) in end to end path does not have shared keys so that the message does not reach sink. Hence, messages from sensors placed near to sink, in terms of number of hops, tend to have more success probability in delivery of messages. Thus, on average, messages with 1-hop distance from sink are frequently delivered in NCD and SL/NCD architectures. This explains lower end to end delay and the abrupt reduction of number of delivered messages to sink node, when compared to other approaches.

We highlight that during simulations node message delivery was observed with, for example, 7 and 8 hops far from sink. However, the number of 1-hop delivered messages observed was too big that result in an average of 1.84 hops for NCD 49 architecture and 1.85 hops for SL/NCD 49 architecture. Such data is shown in Table 3.

Another NCD architecture peculiarity is the fact that there are two kinds of messages: distribution messages and sending data messages. Both are transmitted intermittently during all simulation time. It is believed that such aspect contributes to collision increment and, as a consequence, message loss increment, decreasing chances of message delivery of nodes far, number of hops, from sink node.

Additionally, the working of CTP protocol also can influence obtained result. In CTP, the relation son-father between nodes can change during the experiment (simulation run) that result in changing in relation son-father. Eventually, after
such changing, the son may not share keys with its new father and all message forwarded to such son node will not be forwarded toward sink node. Hence, as the bigger the end to end delay in terms of number of hops, the bigger the probability for a message to be dropped. On the other hand, those nodes placed near sink have major probability of delivery of their messages what contribute to the reduction of end to end delay in NCD and SL/NCD architectures.

For NCD codification, it is used as the current implementation of CTP available in TinyOS 2.1.1 where the beacon default initial value is 128 ms. The beacon is sent periodically and its value is incremented. It is used by CTP to link quality estimation. From that, a node can be associated with other father nodes in the new network tree established by CTP protocol.

Figure 6 shows the average end to end delay for a 100-node network. This average is obtained from 100 simulation runs. The delays oscillate between 684.828 ms and 1666.495 ms to wC, TinySec, SL/TinySec, SPINS, and SL/SPINS architectures. The results indicate a similar behavior to 49-node network where wC presents minor average delays in relation to SPINS and TinySec architectures; SPINS and SL/SPINS delay bars slightly overcome wC. SL/SPINS presents negligible overhead over SPINS architecture. However, TinySec and SL/TinySec bars result in higher delay than wC, SPINS, and SL/SPINS. One of the reasons is the major processing overhead in MACs computation. It is observed as a similar behavior in NCD and SL/NCD bars when compared to the 49-node networks. Their delays are remarkably small in relation to other architectures, due to the aforementioned reasons in the 49-node network.

In general, it is observed as a low end to end delay difference among systems. The difference of TinySec over SL/TinySec is 6.163 ms. The difference of SL/SPINS over SPINS is 24.531 ms. Finally, the difference of NCD over SL/NCD is 4.858 ms. In case of SL/TinySec, it is possible to notice in Figure 6 that the bar is slightly overcome by TinySec. However, it is not possible to state that an architecture is more efficient than others, because the error bars are statistically equivalent. Besides, we believe that simulation random component contributes to such result. Thus, we argue that SENSORLock mechanism introduces an average processing overhead of 7.742 ms (1.0%).

Figures 7, 8, and 9 present simulation results obtained from 100 simulation runs for 144-node, 225-node, and 400-node networks, respectively. For 144, 225, and 400 nodes, SENSORLock applications reveal 2.3%, 0.5%, and 4.4%, respectively, of overhead in terms of average end to end delay when compared to architectures that do not use such mechanism (TinySec, SPINS, and NCD).

The average number of transmitted (by sensors) and received messages (by sink) in all architectures is shown in Figure 10. This result reveals a decreasing behavior where the amount of delivered messages decreases according to the size of the network increases. It is important to notice that each experiment performs an amount of 10 million events. Thus, as the number of events is the same for all simulations, nodes in sparse network scenarios send more messages than nodes in dense network scenarios.

Besides, such behavior is also explained by major likelihood of packets collision occurrence in dense networks—the simulated area is kept the same for all scenarios: 300 m × 300 m—since the wireless medium, which is shared by nodes, is used for transmission. Another aspect that explains
such decreasing behavior is that more dense networks tend to have long end to end paths between nodes that increases the delivery time and the message loss probability.

It is observed that the architectures wC, TinySec, SL/TinySec, SPINS, and SL/SPINS present high absolute values of message sending, since the sending of messages is not conditioned to the existing of a shared key. That is, wC does not use any keys, whereas the keys are previously known by the sender and the receiver nodes, considering the remaining architectures aforementioned.

On the other hand, the NCD and SL/NCD architectures present smaller absolute values of sent messages. This is because in such architectures the sender node may not share keys with the receiver node. In this case, the message is not sent that results in reduction of effectively transmitted messages compared to other architectures. As aforementioned, the sender (node son) must share a key with receiver (node father) so that the message can be sent hop by hop safely in such architectures.

It is important to observe that SENSORLock is implemented over other architectures; the number of sent messages is very similar to the same scenario without its use (TinySec, SPINS, and NCD). Therefore, our mechanism does not introduce overhead in sent messages as well as significant variations in message delivery ratio. The delivery ratio is given by the ratio of the number of received messages and the number of transmitted messages. Figure 11 illustrates the message delivery ratio for all studied architectures.

Regarding the received messages, message loss is due to interferences and collisions present in wireless medium. Specially, it is observed that the number of received messages using NCD and SL/NCD is smaller than using other approaches, since NCD and SL/NCD demand hop by hop key sharing for a message to be successfully forwarded to sink.
Particularly, one aspect that can contribute to the small amount of received packets in TinySec and SL/TinySec consists in eventual packet drops due to MACs verification by intermediate sensors in end to end path. Such hop by hop verification does not take place in SPINS, for example, that verifies MACs only in end systems. In general, the scenarios present a delivery ratio with variation between 16% and 76%.

The other relevant issue in sensor networks is power consumption. Thus, measurements are done in order to estimate power consumption by nodes in several studied architectures. Figure 12 shows an average power consumption estimative for all network.

For estimation of network power consumption, the following events are considered: the sending, the forwarding, and the receiving of messages. We deem that processing consumption is less significant compared to power consumption caused by radio data transmission and receiving. The power consumption in radio transmission and receiving modes was obtained from MicaZ architecture specification because our simulation experiments are carried out over such sensor architecture [26].
medium competition than other architectures. As aforementioned, NCD and SL/NCD have the peculiarity of sending messages only when two nodes share keys that decreases the access medium competition and, consequently, the waiting time to obtain such access.

5. Conclusion

The main contribution of this work is to apply SENSORLock to three different key distribution architectures, TinySec, SPINS, and network coding distribution, and evaluate its performance. SENSORLock is a secure key storage mechanism which can be applied to different key distribution architectures tailored for WSNs. Its main goal is to solve the stored keys exposure problem. Additionally, SENSORLock may protect all data stored in sensor’s memory.

In this work, we focus on large-scale experiments relying on 49-node, 100-node, 144-node, 225-node, and 400-node networks. In this sense, we present an evaluation of SENSORLock through simulation using the TinyOS platform. We implement key distribution architectures using broadly known protocols and algorithms, as Collection Tree Protocol and Advanced Encryption Algorithm.

Simulation results reveal that SENSORLock is lightweight introducing low end to end communication delay and an estimated power consumption quite similar to existing approaches. We believe that SENSORLock can be produced with low cost. Public components may be used to reach a feasible cost, as TinyRNG. Besides, SENSORLock blocks operate with symmetric encryption, which is fast and implies in low computational cost—and, consequently, less power consumption—compared to asymmetric encryption. The mechanism assures the protection of data stored in sensors, using a tamper-protected secret $Z$, stored in a tamper-responsive device. Finally, the mechanism was applied to three key distribution architectures with different characteristics.

For this reason, we believe that SENSORLock is applicable to other sensor networks security architectures.

Even though our implementation does not fully model the original scheme in which the information $Z$, is protected within a CM area, we claim that such hardware implementation would make the encryption and decryption of keys even faster.

Hence, this work intends to demonstrate the feasibility of implementing SENSORLock in real large-scale sensor networks.

Conflict of Interests

There is no conflict of interests related to this paper.

Acknowledgments

This research effort is sponsored in part by Capes, CNPq, and Faperj.

References

[1] L. Atzori, A. Iera, and G. Morabito, “The Internet of things: a survey,” Computer Networks, vol. 54, no. 15, pp. 2787–2805, 2010.
[2] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, “Wireless sensor networks: a survey,” Computer Networks, vol. 38, no. 4, pp. 393–422, 2002.
[3] H. Kumar, D. Sarma, and A. Kar, “Security threats in wireless sensor networks,” IEEE Aerospace and Electronic Systems Magazine, vol. 23, no. 6, pp. 39–45, 2008.
[4] A. Becher, Z. Benenson, and M. Dornseif, “Tampering with motes: real-tampering physical attacks on wireless sensor networks,” in Proceedings of the 3rd International Conference on Security in Pervasive Computing (SPC ’06), pp. 1–5, April 2006.
[5] ATMEL, ATMEL Corporation, 2011, http://www.atmel.com.
[6] J. F. Kazienko, I. G. Ribeiro, I. M. Moraes, and C. V. N. Albuquerque, “SENSORLock: a lightweight key management scheme for wireless sensor networks,” Security and Communication Networks, vol. 6, no. 10, pp. 1198–1210, 2013.
[7] C. Karlof, N. Sastry, and D. Wagner, “TinySec: a link layer security architecture for wireless sensor networks,” in Proceedings of the 2nd ACM International Conference on Embedded Networked Sensor Systems, pp. 162–175, 2004.
[8] A. Perrig, R. Szewczyk, J. D. Tygar, V. Wen, and D. E. Culler, “SPINS: security protocols for sensor networks,” Wireless Networks, vol. 8, no. 5, pp. 521–534, 2002.
[9] P. F. Oliveira and J. Barros, “A network coding approach to secret key distribution,” IEEE Transactions on Information Forensics and Security, vol. 3, no. 3, pp. 414–423, 2008.
[10] A. J. Menezes, P. C. V. Oorschot, and S. A. Vanstone, “Key management techniques,” in Handbook of Applied Cryptography, chapter 13, pp. 543–590, CRC Press, Boca Raton, FL, USA, 1996.
[11] AVNET, AVNET Electronics Website, 2013, http://em.avnet.com/.
[12] WATCHDATA, WATCHDATA Website, 2013, http://www.watchdata.com/.
[13] C. Alcaraz, J. Lopez, R. Roman, and H.-H. Chen, “Selecting key management schemes for WSN applications,” Computers & Security, vol. 31, no. 8, pp. 956–966, 2012.
[14] F. Hu and N. K. Sharma, “Security considerations in ad hoc sensor networks,” Ad Hoc Networks, vol. 3, no. 1, pp. 69–89, 2005.

[15] J. E. Martina, T. C. S. de Souza, and R. F. Custodio, “OpenHSM: an open key life cycle protocol for public key infrastructure’s hardware security modules,” in Public Key Infrastructure: Proceedings of the 4th European PKI Workshop: Theory and Practice, EuroPKI 2007, Palma de Mallorca, Spain, June 28–30, 2007, vol. 4582 of Lecture Notes in Computer Science, pp. 220–235, Springer, Berlin, Germany, 2007.

[16] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Medard, and J. Crowcroft, “XORs in the air: practical wireless network coding,” IEEE/ACM Transactions on Networking, vol. 16, no. 3, pp. 497–510, 2008.

[17] R. L. Rivest, “The RC5 encryption algorithm,” in Proceedings of the Workshop on Fast Software Encryption, pp. 86–96, 1995.

[18] J. Lee, K. Kapitanova, and S. H. Son, “The price of security in wireless sensor networks,” Computer Networks, vol. 54, no. 17, pp. 2967–2978, 2010.

[19] NIST, “Announcing the advanced encryption standard (AES) FIPS,” Tech. Rep. 197, NIST, 2001.

[20] A. Francillon and C. Castelluccia, “TinyRNG: a cryptographic random number generator for wireless sensors network nodes,” in Proceedings of the 5th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt ‘07), pp. 1–7, IEEE, Limassol, Cyprus, April 2007.

[21] J. F. Kazienko and C. V. N. Albuquerque, “Secure secret key distribution and storage in wireless sensor networks,” in Proceedings of the 3rd IEEE International Symposium on Trust, Security and Privacy for Emerging Applications in Conjunction with 10th IEEE International Conference on Computer and Information Technology (TSP/CIT ‘10), pp. 890–895, 2010.

[22] P. Levis, S. Madden, J. Polastre et al., “TinyOS: an operating system for sensor networks,” in Ambient Intelligence, pp. 115–147, Springer, 2004.

[23] P. Levis, N. Lee, M. Welsh, and D. Culler, “TOSSIM: accurate and scalable simulation of entire TinyOS applications,” in Proceedings of the 1st International Conference on Embedded Networked Sensor Systems, pp. 126–137, November 2003.

[24] IIS, Indian Institute of Science—Cryptography Algorithms for TinyOS, 2012, http://bit.ly/rwzDJT.

[25] W. Stallings, Cryptography and Network Security: Principles and Practice, Prentice Hall, 5th edition, 2010.

[26] Crossbow, Micaz Datasheet, 2010, http://www.xbow.com.