Lightweight Distributed Attestation Scheme for Wireless Sensor Networks

Shinsaku Kiyomoto and Yutaka Miyake
KDDI R & D Laboratories Inc.,
2-1-15, Ohara, Fujimini, Saitama, 356-8502, Japan
kiyomoto@kddilabs.jp

Abstract. In this paper, we consider a simple, distributed remote attestation method for a WSN that does not require secret information, precise timing measurement, or a tamper-resistant device.

Key words: Sensor Network, Security, Remote Attestation

1 Introduction

In this paper, we present and analyze a new lightweight attestation scheme described below. The scheme is a distributed attestation scheme. Each node holds a value for the attestation of other nodes, and nodes use the values to check the validity of each other’s data. The scheme uses a simple attestation function that reduces the burden of each attestation by just sending a value, thus allowing frequent attestations. No precise timing measurements or secret keys are required. The attestation is executed on the basis of a check for any changes in the quantity of data. There is no empty in a local storage area after the initialization process, and all the data in the entire local storage area is the target of the attestation. A compromised node may try to send valid data instead of altered data to a node that checks the validity of the data, but the victim node cannot hold both the valid data and the altered data without alteration of data in the local storage area, due to the lack of the space on the local storage area. When a sensor node finds any unexpected changes in the data, it terminates to prevent the propagation of malicious code.

2 Model

2.1 Sensor Node

The sensor node has a CPU, a temporary cache for the CPU (primary memory), and local storage for programs and data. As in previous studies [4], we assume that malicious code is inserted into local storage and the target of the attestation scheme is data in local storage. All sensor nodes have a local storage area and we assume that the local storage area is divided into two parts: a program area $M_p$ for storing programs and related data and another memory area $M_A$.
for attestation. The program area $M_P$ and the memory area $M_A$ consist of $l$ registers $p_i$ ($0 \leq i \leq l - 1$) and $m$ registers $s_j$ ($0 \leq j \leq m - 1$), respectively. The total local storage area $M$ denotes $m_k$ ($0 \leq k \leq l + m - 1$). If the total size of the programs for the sensor node is smaller than the size of the $M_P$, the remaining space of $M_P$ is filled with random data. A sensor node can execute a checksum function $h(x)$, and can communicate with some other nodes in its vicinity. Sensor nodes construct an ad-hoc network with nearby sensor nodes, and data is transferred via the network. When a sensor node terminates, no communication of data can be received or transferred via the sensor node.

2.2 Attestation Scheme

An attestation scheme for the sensor node has to be able to discover malicious code (malware) to ensure the integrity of data in the program area. A sensor node that finds malicious code using the attestation scheme terminates itself to prevent further distribution of malicious code on the sensor network. The attestation scheme consists of the following two steps;

**Initialization:** A sensor node $a$ obtains a value $v_i^a$ in a register $p_i^a$ from a nearby sensor node $b$, and the sensor node calculates the checksum value $h(v_i^a)$ where the function $h(x)$ is a certain checksum function, and stores the checksum value in $M_A$. For simplicity, the checksum function is a bijective function that is defined as $h(x) : \{0, 1\}^n \rightarrow \{0, 1\}^n$, and it satisfies the second-preimage resistance. Sensor node $a$ repeats the calculation and stores the checksum value of the register on another sensor node until the memory area $M_A$ is full. All sensor nodes execute this operation in the initialization stage. Note that a register is chosen with the aim of avoiding duplication if possible, but some registers may not be chosen due to limited memory size $M_A$. To avoid such situation, we should set $M_P = M_A$. To avoid duplication, the sensor node sends a re-selection request to another sensor node, when the data is already chosen by another sensor node. When a sensor node receives such a request, the sensor node randomly selects a different register again. We assume that malicious code cannot be injected during the initialization process. After the initialization process, all registers of all sensor nodes have no remaining space for additional code injection.

**Attestation:** A sensor node $a$ randomly chooses a register $p_x^a$ in the program area $M_P$ and the memory area $M_A$. If the register belongs to $M_P$, sensor node $a$ obtains the corresponding checksum value $h_x = h(v_x^a)$ of the register $p_x^a$ from another sensor node $b$ that stores the checksum value $h_x$, computes the checksum value $h_x' = h(v_x^b)$ from the register $p_x^b$, and compares the value $h_x'$ with the checksum value $h_x$ from another sensor node $b$. Otherwise, the register $p_x^a$ holds the checksum value $h_x = h(v_x^a)$ of a register $p_x^b$ on another sensor node $b$. The sensor node $a$ obtains the register value $v_x^a$ of the register $p_x^a$ that corresponds to the checksum value $h_x$, calculates the checksum value $h_x' = h(v_x^b)$ from the register value $v_x^b$, and then compares the value $h_x'$ with the checksum value $h_x$ stored in register $p_x^a$. Malicious code stored in the memory area $M_A$ can be found by checking not only $M_P$ but also $M_A$, randomly.
If node $a$ finds a conflict between the values from node $b$, node $a$ executes an additional step named a termination process as follows:

**Termination:** To avoid the propagation of malicious code, sensor node $a$ requests termination from node $b$ and then terminates itself, when node $a$ finds node $b$ suspicious by reason of a conflict between the checksum and register values. Node $b$ also terminates when receiving a termination request from node $a$. Note that sensor node $a$ cannot distinguish two cases: checked data on sensor node $b$ is infected or its own data (in sensor node $a$) is infected. Thus, both sensor nodes should be terminated.

### 2.3 Adversary Model

We assume that the adversary is not able to either extend or replace the hardware of sensor nodes to increase memory size. The objective of an adversary is to control sensor nodes using malicious code. An adversary will try to install malicious code on sensor nodes, replacing any register(s) in $M_P$ or $M_A$ with the malicious code. For simplicity, we specify that the size of the malicious code is the same as the size of a register. We consider attacks described in existing papers [3, 1, 2]. There are nine different attacks that might be made on wireless sensor networks, and we will focus on an analysis of three types of attack, the data substitution attack, the memory copy attack, and the checksum value modification attack. As the other seven attacks are not required to be considered or just beyond the scope of this paper as explained below.

**Guessing Attack.** An adversary may guess the value that is requested by a checking node. However, it can be assumed that the probability of success of such a guess is negligibly small.

**Data Substitution Attack.** An adversary may attempt to change some location in the memory region. The adversary modifies selected portions of the program area and maintains a copy of the unmodified portions elsewhere in the program area or the memory area. The attestation request is forced to refer to a location where the corresponding unmodified portion exists. We should consider this attack in our experiments.

**Memory Copy Attack.** An adversary maintains a copy of the unmodified values of a program area or a memory area while injecting malicious code. Either the correct code is copied to another location in memory and the malicious code is injected at the location of correct code, or the attacker simply injects the malicious code in some other location in the area. We should consider this attack in our experiments.

**Computation Attack.** This attack includes on-the-fly execution of a correct checksum value of the register and replacing an input of the attestation algorithm. To compute a correct checksum value (or just input a correct checksum value to the attestation algorithm), the attacker has to hold a correct register value or a correct checksum value. Thus, this attack is identical to the memory copy attack.

**Parallelization/Re-Ordering Attack.** This type of attack is for timing-based attestation schemes. The adversary speeds up the self-checksum computation by
executing operations in parallel or by re-ordering the operations. This speedup allows the adversary to compute the correct checksum faster than expected. Our attestation scheme is not based on a timing measurement. Thus, this type of attack need not be considered.

**Replay Attack.** An infected node may send a correct value that was obtained in a previous attestation; however, to send the value, the node must store the value in the local storage area. This attack is not realistic due to the limited size of the local storage area. Thus, an attacker will try to use a data substitution attack or a memory copy attack.

**Impersonation Attack.** An adversary could interrupt the execution of the attestation process. There is no way to carry out an impersonation attack in our scheme. A sensor node is checked by the nearest nodes, and it is assumed that no node is able to impersonate another and in this way come between an attester and a subject.

**Compression Attack.** A compression attack [1] is an extension of the memory copy attack. An adversary compresses data in the local storage area in order to obtain enough memory space to store malicious code. Techniques [5, 4] for avoiding compression attacks have been presented. Thus, we will not consider a protection mechanism against a compression attack here.

**Checksum Value Modification Attack.** The attacker injects a malicious code to $p^i_a$ in sensor node $a$, and then replace the checksum value $h_j = H^j(a)$ stored in sensor node $b$ to the checksum value of the malicious code. When both register and checksum values are altered, the attestation step is correctly executed despite the presence of malicious code. However, it would be difficult for an adversary to find the correct pair of register and checksum values. We will consider this attack in our experiments.

## 3 Analysis

To construct a network topology for experiments, we used a *random graph* that has $N$ nodes connected to $n$ edges that are chosen randomly from $N(N - 1)/2$, where $N$ is the number of nodes. The random network is generated based on the probability $P_L$ that a sensor node selects a register of another sensor node to compute a checksum value in the initialization process. That is, the selection of a register is equivalent to the edge between two nodes; thus, a random network is generated to select the register of other sensor nodes with probability $P_L$ until the registers of $M_A$ are full. The probability $P_L$ controls the density of the random network. That is, larger $P_L$ generates more contrasting dense networks like clustered networks. The infection process of malicious code will be treated as a probabilistic behavior and simulated as follows, similarly to existing work:

1. The register of a sensor node $a$ that has malicious code is selected randomly.
2. A register of sensor nodes connected to sensor node $a$ is randomly selected.
3. The selected register is infected by malicious code. If both a register value and its checksum value are compromised, no further attestation is executed between them.
We do not assume a “smart” adversary that understands the network topology and efficiently injects a malicious code into sensor nodes. Such a strong adversary model will be considered in a future study.

Using the above simulation, experiments were executed under the following conditions: (a) When all sensor nodes are infected by a malicious program, the experiment is terminated. (b) When all sensor nodes are terminated, the experiment is terminated. (c) When no sensor node that includes malicious code exists, the experiment is terminated. (d) At each step, either attestation process execution or execution of malicious code infection is randomly selected; malicious code infection is selected when $-\log(p_x)/rN_m < -\log(p_y)$. The parameter $r$ is the fraction of infection/attestation, and infection or attestation is selected for each simulation step according to $r$. The parameter $N_m$ denotes the number of registers having malicious code at that point. The random numbers $p_x$, and $p_y$, $(0 < p_x, p_y \leq 1)$ are used to generate a probabilistic event based on exponential random numbers.

We used a parameter set for the experiments as follows: $N=100$, $M_P=4$, and $M_A=4$. We evaluated the number of steps to find a compromised register and the number of steps to stop the experiments by four values of $P_L$ and values of $r$ ($0.00001 \leq r \leq 1$). We also evaluated the fraction of surviving nodes, i.e. those that had not been invaded and terminated, when any experiments were stopped. To evaluate the effect of the termination process, the experiments include two cases: experiments with the termination process, and experiments without it.

We executed a maximum of 400,000 steps in each experiment, and all results are the average values of 100 experiments. The experiments include two cases: (a) the termination case in which a sensor node executed the termination process when it discovered malicious code and (b) the no-termination case in which a sensor node executed no additional operation, even when malicious code was found. The termination case removed all malicious code from the sensor network after 76 steps, making the fraction of surviving nodes 0.92 over 76 steps. Figure 1 shows the fraction of surviving nodes when the experiment was terminated. Where $P_L = 0.01$, the results of the no-termination case were similar to those of the termination case because malicious code efficiently propagated to other nodes due to the sparseness of the network, and the propagation speed was slower than in other cases. The survival fraction in the no-termination cases with $P_L = 0.02, 0.03, 0.04$ was dramatically reduced as $r$ increased. However, in cases that used the termination operation, the survival fraction remained high, reaching $r = 0.03$. Thus, the attestation scheme, including the termination process, was effective in protecting sensor networks against the propagation of malicious code when we selected an appropriate $r$, such as $r \leq 0.03$. We should select an appropriate $r$ when configuring a sensor network, taking into account the network structure, and the local storage size of sensor nodes, as well as the expected rate of attestation for an assumed propagation model of malicious code.
4 Conclusion

We evaluated the scheme using random networks of sensor nodes and presented optimal settings for the frequency of attestation. The results suggested that the attestation scheme was effective with certain parameter settings.

References