A Software-Based Group Attestation for Wireless Sensor Networks

Tamer AbuHmed

Graduate School of IT and Telecommunication,
INHA University,
Incheon 402-751, South Korea
Fax: +82-32-8650480
E-mail: tamer@isrl.kr

Jeonil Kang

Graduate School of IT and Telecommunication,
INHA University,
Incheon 402-751, South Korea
Fax: +82-32-8650480
E-mail: dreamx@isrl.kr

DaeHun Nyang*

School of Information and Computer Engineering,
INHA University,
Incheon 402-751, South Korea
Fax: +82-32-8650480
E-mail: nyang@inha.ac.kr

KyungHee Lee

Department of Electrical Engineering,
University of Suwon,
Suwon 445-743, South Korea
Tel: +82-31-2298164
E-mail: khlee@suwon.ac.kr

Abstract: Sensor nodes are vulnerable to compromise due to their unattended deployment. The low cost requirement of the sensor node precludes the use of expensive tamper resistant hardware for sensor physical protection. Therefore, the adversary can easily reprogram the compromised sensors and deviate their functionality. We address this problem by proposing software-based remote code attestation mechanisms for different WSN scenarios, which can limit the misuse introduced by malicious attackers. An attestation scheme of node-to-base station is proposed and generalized to two distributed group based attestation schemes. Our group based attestation schemes localize the attestation process and substitute the need for running multi hops attestation for each node out of the base station range. The simulation results show that proposed schemes are feasible and well-suited for wireless sensors. Moreover, we analyze the security of the proposed protocols and show that, unlike the previously related proposed work in literature, the proposed schemes loosely depend on the accurate measurement of the execution time.

Keywords: wireless sensor networks; software attestation; validation; memory traversal.

1 Introduction

Security of Wireless Sensor Network (WSN) is an important issue ever since several WSN applications were developed. The constraints on memory, computation, and communication in sensor nodes make security a challenging task. Moreover, deploying sensors in a hostile and unattended environment can lead to several attacks, including node capturing, physical tampering, and nodes reprogramming. Therefore, sensor node program

integrity and consistency are considered to be crucial for WSNs' functionality and security. The consequences of node compromise do not only affect the WSNs' functionality, but also the security services of WSN (i.e., data confidentiality, integrity, and authenticity). Moreover, by sensor node compromise, the attacker becomes capable of launching several attacks such as data forgery, selective packet forwarding, and Denial of Service (DoS). The direct solution to countermeasure these security concerns is applying a memory integrity check mechanism with an efficient and secure attestation protocol (19) so that the attacker, who captures a sensor node and tries either to modify the original program or upload a new code, will be detected with high probability by the WSN attestor. To overcome these security problems, we propose two different attestation mechanisms for one-hop base station-to-node attestation and distributed group based attestation. Both mechanisms have the same memory check notion for the attestation's internal process but differ in the communication protocols. In the attestation's internal process, the attestor will verify the node's program integrity and the content of the whole memory. Attesting the whole memory prevents the attacker from using the extra memory space of the sensor to save the original program. In the case of node compromise, the attestor can use different mechanisms which are out of the paper's scope - such as, simply broadcasting the sensor ID in WSN as a compromised node, or reprogramming the node. Moreover, code attestation is considered to be an important technique for detecting node compromise in WSN when the attestor detects anomalous behavior of a specific sensor node. In this paper, we proceed our primitive works in (5) and (2) related to the software-based attestation for WSN. Our original contributions are as follows:

- We propose two distributed group-based attestation schemes for different WSN scenarios. Our proposed group-based attestation schemes localize the attestation process inside the WSN and avoid the use of running multi-hop attestation for each node out of the base station range.
- We further analyze using simulation both the performance and security of the basic (i.e., base station to node attestation) and group-based attestation schemes. Simulation results demonstrate the security robustness and applicability of our proposed group-based attestation scheme in terms of computations and communication overhead.

The rest of the paper is organized as follows. In Section 2, we survey the related work in secure attestation for WSN. In Section 3, the threat model and assumptions of software-based attestation are presented. In Section 4, we introduce the basic (base station to node attestation) scheme that includes the node preparation (i.e., memory filling techniques), the memory traversal techniques, and the secure attestation protocols. We present in Section 5 the group-based attestation protocols for a portion or the whole WSN. We discuss the security attacks for the software-based attestation and our schemes immunity against them in Section 6, then we present the evaluation of our attestation techniques and protocols in Section 7. Finally, we conclude the paper with a summary of contributions in Section 8.

2 Related Work

The prior work on code attestation can be categorized into two work streams: Hardware-based and software-based approaches. All hardware approaches for the sensor node attestation are based on using the Trusted Platform Module (TPM) which is specified by Trusted Computing Group (TCG) (9). In (22; 25), trust anchors (sensor nodes) are deployed in WSN for attestation. The anchors report the hardware and software configuration status to the remote attestor so that the attestor individually checks each sensor's received status and decides whether the nodes are compromised or not. However, it is infeasible to install a TPM in each individual sensor node due to the cost consideration. To mitigate the WSN cost, Krauß *et al.* adopted TPM in hybrid sensor networks (13). In (13), the sensor nodes were organized in clusters and the cluster heads are non-resource constrained equipped with TPM. This approach has several drawbacks, including the requirement of cluster heads to performing heavy cryptographic operations and keeping the memory configurations consistent during the sensor runtime.

Unlike the previous approach, software-based attestation is considered to be promising for WSN due to the size, cost, and power limitation of sensor nodes. Hence, a reliable hardware is not required. In (5), Choi *et al.* introduced a protocol where the attested memory uses a random seed provided by the verifier to produce a pseudo-random bit stream and uses it to fill the empty memory of the sensor. Then, a full hash of the attested memory is sent back to the verifier. More details about their scheme are shown in Section 4. In (28), Yang *et al.* introduced distributed software-based attestation that performed pseudo-random memory traversal over memory that empty space filled with incompressible random noise before the deployment of the sensor nodes. The advantages of this scheme are: (i) the computation efficiency of the block-based traversal, which will be mentioned later, over cell-based traversal (21) and (ii) the distribution manner of the attestation protocol, which could be effective in the case where the trusted verifier (e.g., base station) cannot enter the transmission range of the sensor node directly. However, secret share distribution, univariate polynomial evaluation for the attested node, and extensive communication overhead are the main drawbacks of their scheme. Moreover, the scheme is vulnerable to a compromised

node cheating after one attestation round. AbuHmed et al. (2) proposed two memory filling techniques for filling the empty space of the program memory. Also, a dynamic block-based pseudo-random memory traversal algorithm in order to enhance the original block-based pseudo-random memory traversal algorithm is proposed. More details about their work are introduced in Section 4.

In (19; 21; 20), the authors introduced another remote software-based attestation to verify the integrity of a code running on an embedded device. The main idea behind their techniques of software-based attestation is to check the correctness of the response value, which is computed pseudo-randomly by cryptographic hashing over the memory content of the device. Meanwhile, those proposed schemes use attestation routine to hash the whole sensor memory in such a way so that the routine cannot be optimized further (i.e, compressed). Therefore, the adversary cannot exploit the size difference in order to inject a malicious code without detection. Also, these schemes assume direct sensor node attestation (i.e., single hop attestation). This is not always possible since the trusted verifier cannot always enter the transmission range of all the nodes. Moreover, these schemes assume a precise time measurement by the attestor before the attested sensor sends its response back. Thus, if the response of the attested sensor is delayed because of any reason related to network, the attestor will assume it is a compromised node. Thus, these mechanisms are sensitive to any external factor, such as network channel collision and network delay, especially when we have a wireless environment where problems with the communication latency of even a one-hop cannot be determined. Thus, it is impossible to design a reliable one-hop and/or multi-hop attestation scheme without designing a complete timing-independent attestation protocol, or at least reducing time dependency to a fair level. Thus, the presence of an attacker modifying the sensor's memory and controlling network latency is easily detectable. Nevertheless, considering resource constrained environment and the lack of strong cryptographic primitives in WSN, the design of a fully independent attestation scheme becomes infeasible. While design of our attestation schemes, we focused on making the attestation process less time sensitive compared to the existing schemes, rather than designing time independent schemes (i.e., the attestor is tolerant to network related delay without any security concerns).

3 Threat Model and Assumptions

3.1 Threat Model

Sensor networks usually operate in unattended and hostile environments. Also, sensor nodes lack expensive tamper-resistant hardware due to the low-cost requirement. An attacker who physically compromises the sensor node has full control over the memory's contents and the secret keys of the node. Then, the attacker can either modify the running sensor code or upload a malicious code. The attacker can also launch various kinds of insider attacks, including passive attacks, like eavesdropping, and active attacks such as fake data injection, selective forwarding, DoS, and routing protocol misbehavior. Furthermore, in order to bypass the attestation process, the compromised nodes can collaborate by sharing their resources.

3.2 Assumptions

In this section, we show the general assumptions of our attestation schemes. Specific assumptions related to protocols are mentioned at the time of the describing the protocols in Section 4.3. We assume WSN sensors are homogeneous in the hardware and software contents. Any key predistribution scheme, such as EG or polynomial-based key predistribution scheme, is used to establish pair-wise keys for either among neighboring nodes, or node and BS (27). The attestation code is pre-depolyed to each node and does the attestation's functionality based on the attestation scheme applied. Like the previous schemes (21), (24)-(17), once the sensor node is compromised, the sensor node becomes under the full control of the attacker without any hardware modification of the sensors such as enlarging the sensor memory, supporting the virtual memory (15; 14), and increasing the processor's speed. Moreover, we do not assume any tamper-resistance hardware to be installed on the sensor. For the purpose and measurements of time and power consumption of our proposed attestation schemes, we assumed a specific sensor node type. However, our attestation schemes are independent of the underlying sensor's hardware.

Sensor Node Hardware Description: There are various types of sensor node hardware construction. For memory layout, these types of construction are compromised of mainly two kinds of architecture: Harvard architecture and Von Neumann architecture. The Atmel AVR sensors, such as Mica2, have Harvard memory architecture, where the data memory is physically separated from the program memory. In contrast, Von Neumann memory architecture has the data memory and program memory in the same physical memory. Consequently, the codes of the data memory of the latter architecture are executable as the program counter of the program jumps to the data memory addresses. In this paper, Harvard architecture was used because of its popularity. For example, MICA2 mote has the data memory of 4kB RAM, which has the program stack and variables, and a 128kB flash program memory, which has both the sensor's program and a bootloader for the program update. Also, each sensor usually has 512 kB external flash memory (14) in order to hold the collected data from the surrounding environment. Broadly speaking, most of the sensor node constructions are similar to MICA mote's.

4 Node Attestation Schemes

Mainly, the attestation in WSN is a technique used to protect the WSN from an insider attacker who injects a malicious code into the sensor memory. An illustration of this is shown on Figure 1. By compromising a node, an attacker can run various kind of attacks in the WSN that may not be detected by other security detection techniques. Thus, the attestation process is supposed to prove that the sensor memory has specific contents. A normal sensor node usually has memory that is bigger than the deployed application, as illustrated in Figure 1(a). Thus, the attacker can easily compromise a node and put his code in the empty space, as illustrated in Figure 1(b). This shows that proving the correctness of the original code is not enough because the attacker can construct a correct proof over the original memory contents without being detected. Consequently, if the attestation process succeeds, the sensor node will have the intended memory contents without any attacker code. Therefore, our idea is to fill the empty space in the memory before constructing the proof of memory correctness. To this end, we can perceive that the attestation concept is an interaction between the prover and the verifier (i.e., sensor node and attestor, respectively) in order to prove the correctness of the whole memory's contents and to detect a compromised sensor node. Thus, the abstract view of our proposed mechanisms is divided into two main parts: (*1*) the communication protocol, which is in the form of a challenge from the attestor and a response from the attested node, and (*2*) the attestation process inside the node which is mainly divided into two phases: (*i*) memory filling phase, where we introduce two different filling techniques to fill the empty space in the program memory and one to fill the data memory of the sensor as a protection strategy, and (*ii*) memory traversing phase, where two secure memory traversing techniques compute the response that will be sent to the verifier by the sensor nod. However, there is trade-off between the performance and security of these two schemes. In the following subsections, we introduce the memory filling techniques, memory traversing techniques, and the communication protocol for the BS and sensor node attestation setting. The notations used throughout this paper are given in Table 1.

Table 1 Notations.

Term	Declaration
N_i	ID of sensor node i
CH_i	ID of Cluster Head i
$MAC_{k_{a,b}}()$	AES-based Cipher Block Chaining Message Authentication Code between a and b parties
K_{BS,N_i}	Symmetric key between BS and sensor node N_i
S_i	Seed for noise generation of a sensor node N_i
C_i	Challenge sent to the sensor node N_i for memory traversal.
CS_i	Computed checksum value of sensor node N_i
TS_i	Timestamp of the synchronized WSN which is tightly equal in both of the BS and WSN's nodes
t_{min}	The minimum time required to compromise the sensor node by the attacker after the deployment (7)
b	Computed Block size used in the memory traversing algorithms

4.1 Memory Filling Techniques

The memory filling is the first component of the attestation process inside the attested node. Here, memory filling is applied to fill the remaining memory that is not used by the sensor application with random data in order to prevent an attacker from exploiting this space by injecting a malicious code. Thus, the filling algorithm targets the data memory and program memory, and it will not be applied on the external EEPROM because of its slow access time, which can be detected easily if the attacker uses it.

4.1.1 Program Memory Filling

The program memory is the place where the sensor codes reside during the running of the sensor application. The sensor node is manufactured for various types of applications. The program size of the sensor cannot be estimated in advance, so the sensor node is manufactured with a reasonable program memory size. Consequently, the extra empty space remains after sensor application is deployed and can be a target for accommodating malicious code. Our motivation is to fill this empty space with incompressible noise in the program memory. Below, we show two kinds of program memory filling techniques.

Pre-deployment Noise Filling: In this technique, the WSN's administrator simply fills the fixed free memory with random noise before the sensors are deployed. Hence, the program memory is as shown in Figure 1(c). The pre-deployment noise filling is summarized as follows (2):

1. For each sensor node, the trusted attestor generates and stores a table whose entries are $< N_i, K_{BS,N_i}, S_i >$, where node ID N_i, pairwised key K_{BS,N_i}, and seed S_i are used for the incompressible random noise generation. Attestor keeps this information for the attestation stage in order to emulate and evaluate the sensor node memory.

2. The trusted attestor loads sensor node N_i with K_{BS,N_i} and then fills the empty space of the memory with random noise generated from S_i.

After these steps, the nodes are ready to be deployed, and they contain an application code, an

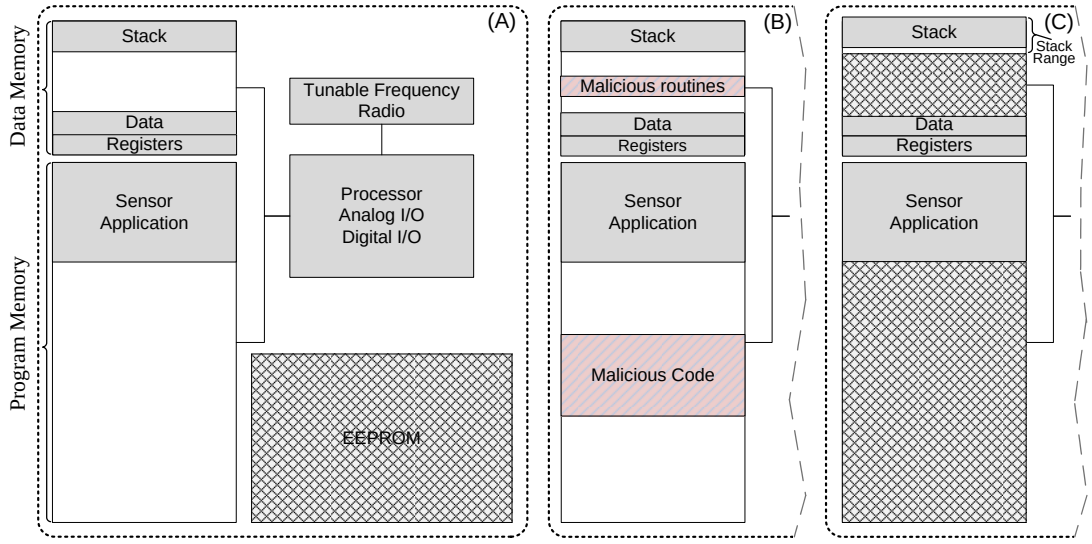


Figure 1 Program and data memory of (a) a normal sensor node, (b) a sensor node with malicious code injection in either the program and data memory or both, and (c) a sensor node filled with incompressible random noise.

attestation function, and cryptographic primitives. The memory layout of the node is shown in Figure 1(c). Although filling this free space in the sensor node with incompressible random noise before the deployment is simple and effective, this technique is not applicable for all WSN applications because, the application needs to collect routing information and geometry-related information from the environment before operating and expanding the application code size. Thus, a post-deployment noise filling technique is required.

Post-deployment Noise Filling: The post-deployment noise filling technique is proposed for scenarios where the attestor cannot exactly estimate the program memory needed for sensor applications before deployment. In this case, the sensor node is loaded with an application code, an attestation function, and cryptographic primitives, and is then deployed without filling the empty memory. In the safe time (i.e., $< t_{min}$), and without any attestation request, each node generates incompressible random noise by using a pre-deployed secret seed, S_i which will be removed after filling process (2).

Since our filling techniques depend on an expansion of small seed, the memory overhead in our attestation schemes is negligible. The attestor does not have to keep the entire memory of each sensor node. The random noise in the program memory that was filled by either the pre-/post-deployment filling technique can be recovered by storing only the seed S_i. Afterwards, at the attestation time, BS generates the random noise of the sensor node N_i on the fly from S_i. Hence, the verifier's permanent memory of 10,000 nodes is computed as: (10,000 (nodes) $\times$ 2B (node's ID) $\times$ 8B (node's key)$\times$ 8B (Seed)) $\approx$175 KB.

4.1.2 Data Memory Filling

Since the application installed to the program memory consumes a fixed memory size, it can be filled in advance. However, the data memory has dynamic contents, such as temporary data and stack, which the application needs in order to run. The space occupied by these temporary data is dynamic, and thus the aforementioned filling techniques cannot be applied for the data memory. However, since the RAM usage depends on the application, the administrator (i.e., attestor) can easily estimate the upper bound of the dynamic memory usage and tighten unfilled area used by stack (i.e., stack range) in advance as illustrated in Figure 1(c). By doing so, the unfilled memory area will be with few bytes and any attempt to use this tight area by attacker will make the application of sensor node crash. In the following, we use the filling technique of (5) with more comprehensive description and options for data memory filling. During the memory filling and starting with the initial address given to the filling algorithm, the attested node starts filling the empty space in the data memory immediately after it receives the challenge C_i from the attestor. According to (5), this challenge is used as an initial seed to generate the first block of filling random data as shown on the left of Figure 2. Assume that the size of empty space is S_E, then the algorithm finishes the first filling round after arriving $Mem[S_E]$. Thereafter, a new round starts where the random stream generated from hash values of the predecessor memory block is XORed with the previous filled memory to refill the same memory location in the reverse direction. For example, in the first round, the memory filling function initializes cache(e.g., 512 bit in SHA-1 case) with the attestation challenge C_i plus zeros. Afterwards, the memory filling function consecutively generates hash blocks (Mem[i]) until it reaches the end of the memory. If the random

generation arrives at the end of the empty memory, the memory filling function proceeds in the opposite direction and fills the XOR-ed value between the existing random number and newly generated random number as shown in Figure 2. This process continues until the XOR operation reaches to the beginning of the free memory space. We described an example of two rounds and more general form is illustrated in Figure 2, where R_j denotes the j-th memory block Mem[j] generated by the hash function. Applying the XOR operation after the first filling round is crucial for preventing the attack of online computation of memory blocks, and more details about this attack will be introduced in section 6. In Figure 3 and 4, one can see the algorithmic description of the data memory filling in the case of the SHA-1 hash and $RC5$ functions, respectively. For SHA-1 hash function, initially a temporary cache having the size equal to the hash function's input is filled with the seed and zeros. Then, it is hashed and the output block is written to the first memory block. The next 160-bit free memory block will be filled with the hash of the previous 512-bits of the memory contents. In the case of $RC5$ block cipher function, the filling essence is same, but the only difference is that $RC5$ function takes initially the zero vector and the seed and outputs one block. Since the input and output block size of $RC5$ function is equal, the block of the previous $RC5$ execution is fed to the $RC5$ with the same seed in order to generate the next block. This technique is securing the attestation process because the data memory is filled with a fresh random content at the attestation time. However, freshness of the filled random contents is not enough for trusting the attestation process because a attacker can compute these random blocks online at the attestation time. Thus, we need to apply special traversing technique to harden this attack as we will see in section 4.2.1

Input : n, seed
i, initial address
, *assigned_round* number of rounds
Output: null

```
1  /*Initialization */
2  round = 0;
3  tmp ← h^512(n ‖ 0 ··· 0); //Generate the
   initial blocks
4  Mem[i] ← h^512(tmp);
5  S_E = |Memory| - i + 1;
6  for j = i to S_E do
7      tmp ← Right(tmp)^384 ‖ Mem[j];
       //previous 512-bits;
8      Mem[j] ← h^512(tmp);
9  round++;
10 while round != assigned_round do
11     /* XOR operation with hashing after the
       first round*/
12     /*if round is even: go right to left, else go
       left to right */
13     if round mod 2 > 0 then
14         for j=S_E to i do
15             tmp ← Right(tmp)^384 ‖ Mem[j];
               //previous 512-bits;
16             Mem[j] ← Mem[j] ⊕ h^512(tmp);
17         round++;
18     else
19         for j=i to S_E do
20             tmp ← Right(tmp)^384 ‖ Mem[j];
               //previous 512-bits;
21             Mem[j] ← Mem[j] ⊕ h^512(tmp);
22         round++;
```

Figure 3 Algorithmic description of the data memory filling using the SHA-1 hash function.

4.2 Memory Traversal Techniques

For the second component of the attestation mechanism, we will discuss how the proof that is sent out by the sensor node is computed in order to convince the attestor about the memory correctness. The memory traversal algorithm runs over the sensor's program and data memories except the stack range in order to generate a checksum value that informs the attestor about the sensor memory status, i.e., whether it is modified or not. Reminding that both memories are already filled using the memory filling techniques of section 4.1. Also, the memory traversal algorithm runs after the sensor node receives the challenge from the attestor. Mainly, the traversal algorithm must be hard to manipulate for the attacker who tries to modify sensor code and generate the correct proof at the same time. Hence, the algorithm must traversal the memory in such a way that the probability of the attacker's undetectability is negligible. For memory traversal, a cryptographic one-way hash function is used in several existing code attestation schemes, including the Spinellis scheme (24) and the Program Integrity Verification (PIV) scheme (17) where a new hash algorithm is generated for each attestation round and sent to a sensor node, along with the attestation request. In the following subsection, we introduce two traversing techniques that can compute a secure checksum of the memory content.

4.2.1 Full Memory Traversal Technique

The Full Memory Traversal Technique (FMT) was proposed in (5) and it is a simple full memory traversal for program memory and filled data memory (RAM). In this technique, the whole memory is traversed except the stack range shown in Figure 1(c) in order to compute the response checksum to the verifier. In the FMT algorithm, the traversing will be initiated after the attested node receives the challenge C_i from the BS and the memory is filled with the aforementioned filling techniques. The attested node simply starts computing the hash value over the program and data memory after considering the

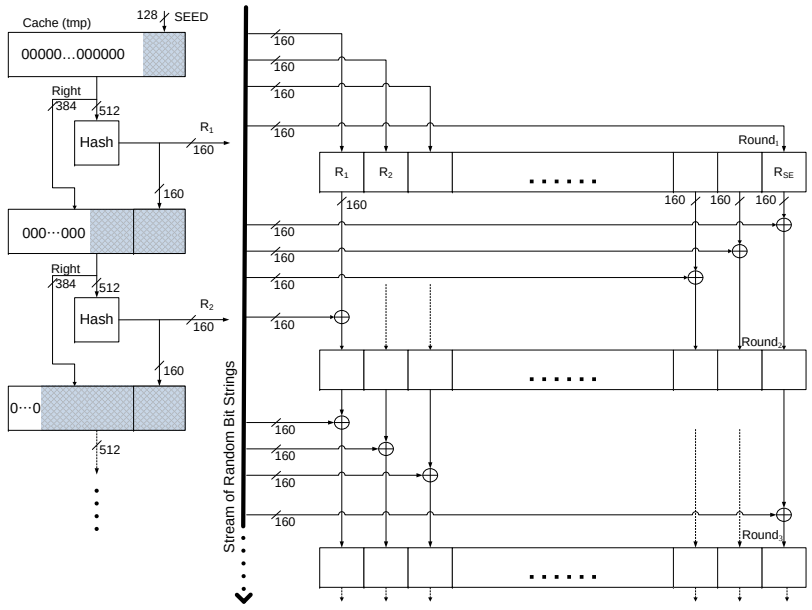


Figure 2 The filling process of the empty memory space with random using SHA-1 and XOR operations, where the filling function is shown on the left and the memory rounds on the right.

memory as a matrix structure, as shown in Figure 6. The checksum computation is shown in Figure 5. The reason for not computing the checksum in a sequential way is to prevent parallel computing by other compromised nodes' help, i.e., the proxy attack. Details about this attack are shown in Section 6.

4.2.2 Pseudo-random Traversal Technique

In SWATT (21), another technique known as a pseudo-random memory traversal is used to compute the checksum over the memory cells of the sensor node. Though SWATT scheme is sensitive to the attestation response time and computation overhead, the pseudo-random memory traversal used in the scheme is immune against attacker misbehavior. Yang *et al.* (28) proposed a distributed software based attestation for WSN, where a more efficient pseudo-random traversal is implemented, compared to SWATT. In Yi Yang *et al.*'s scheme, a block-based pseudo-random memory traversal was proposed to improve the cell-based traversal of SWATT in the sense that a lower number of traverses are required to verify the same amount of memory space. The pseudo-random memory traversal approaches depend on the Coupon Collector's Problem (3). The result of the Coupon Collector's Problem states that the cell-based memory traversal approach like the SWATT scheme iterates more than $O(m\ln m)$ times to access each memory cell at least once with a high probability, where m is the memory size. However, for the same memory space, the Block-based Pseudo-random Memory Traversal (BPMT) approach requires $O(\frac{m\ln m}{b})$ iterations, where b is the block size, in order to obtain an equal probability as in the cell-based memory traversal(28). In our previous work, we modified the original BPMT to make it more efficient and secure (2). Hence, we proposed two algorithms: (i) fixed block size BPMT, where the block size is not predetermined as the original algorithm. However, the attestor sets the block size in each attestation round either explicitly or implicitly. This step makes the traversed blocks unpredictable to the attacker. Also, (ii) a dynamic block size BPMT algorithm is proposed. The goal of the dynamic BPMT is to enhance the security and computation complexity by decreasing the average access per cell compared to the fixed BPMT, as we will see in Section 7. Our algorithm will use the $RC5$ block cipher function, which is the most efficient and adequate cryptographic algorithm that can be used in WSN due to its high performance and low memory space requirements (12). Thus, $RC5$ (18) with a 64 bit block size is used and is recommended to be used as Pseudo-Random Number Generator (PRNG) inside the BPMT. In dynamic BPMT, $RC5$ takes the attestor value C_i as a seed to produce 8-byte values treated as four memory address, i.e., memory address is 2-byte (14), namely t_0, t_1, t_2 and t_3. For each memory address, 1-byte of checksum CS_i is updated based on the results of the bitwise XOR of b number of consecutive memory cells. The implementation of $RC5$ is already available in TinySec (12). $RC5$ running in counter mode (CTR) is used due to the fact that each memory cell is individually accessible by encrypting the right counter (28). Though $RC5$ is patented by RSA Security, $RC5$ could be easily replaced by other block ciphers schemes, such as MISTY1, AES, or Skipjack (13). In fact, the b parameter of the BPMT algorithm is crucial for security and performance purposes. For example, if the $b = m$, the algorithms have the lowest security level and the highest performance, where the checksum is the

Input: *seed*, *iaddr*= starting address of RAM, *no_round*= number of rounds, $wSize$ = word size in byte (8, 16, 32), $bSize$ = RC5 block size in byte (32, 64, 128)
Output: void

```
1  begin
2      /* Initialization */
3      mSize ← |RAM|; /* memory size in byte */
4      for (i ← iaddr; i < mSize; i++) do
       Mem[i] ← 0;
5      for (i ← 1; i ≤ bSize/wSize; i++) do
       tmp[i] = 0;
6      /* Filling */
7      for (j ← 0; j < no_round; j++) do
8          if (j mod 2 = 0) then
9              for (i ← iaddr; i < mSize; i ←
               i + bSize) do
10                 tmp ← RC5_seed^bSize(tmp);
11                 for
                   (k ← 1; k ≤ bSize/wSize; k++)
                   do
12                     Mem[i + k − 1] ←
                       tmp[k] ⊕ Mem[i + k − 1];
13         else
14             for (i ← mSize − bSize; i ≥
               iaddr; i ← i − bSize) do
15                 tmp ← RC5_seed^bSize(tmp);
16                 for
                   (k ← 1; k ≤ bSize/wSize; k++)
                   do
17                     Mem[i + k − 1] ←
                       tmp[k] ⊕ Mem[i + k − 1];
18 end
```

Figure 4 Algorithmic description of the data memory filling using the RC5 hash function.

Input : n, seed
i, initial address
b, block size
Output: CS_i

```
1  counter← 0;
2  while end of memory do
3      for j=counter to 512/b do
4          tmp ← tmp || Mem[index][counter]; //
           512-bits;
5          counter ← counter + 1;
6          if counter=S_E then
7              counter← 0;
8              Index ← Index+1  mod  20; // (20)
               is the number of rows
9      Mem[j] ← h^512(tmp);
10     if counter=S_E & Index = 20 then
11         break end of memory ;
12         return CS_i;
```

Figure 5 Algorithmic description of FMT memory traversal algorithm using SHA-1.

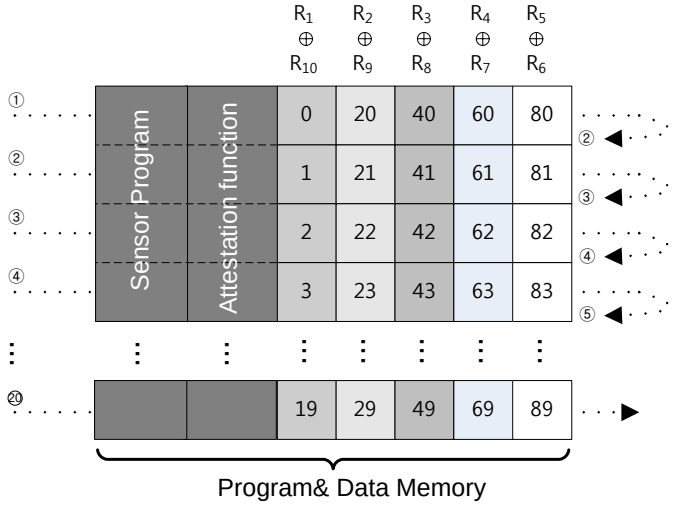


Figure 6 FMT algorithm checksum computation over the filled memory using SHA-1 hash function and XOR operation.

XOR of the whole memory, and vice versa. Thus, it is easy for the attacker to save the checksum value or to modify the code and find collision. However, if the block size is of reasonable size(e.g., one Mica2 read of 16 byte), the security will depend on three factors:(i) the unpredictable block size and traversed location, (ii) the overlap among the traversed blocks, (iii) the dynamic block expansion and shrinking of the traversed blocks in the dynamic BPMT. The dynamic BPMT algorithm is shown in Figure 7.

4.3 Attestation Protocol

The last component of the attestation mechanism is the communication protocol for the attestation request and response by the attestor and the attested node, respectively. Two secure attestation protocols were constructed in (2), and any of these could be used according to the time synchronization of the underlying WSN. The protocols were executed in a scenario where the Base Station (BS) is the attestor and sensor node (N_i) is the attested node. Moreover, the protocols follow the conventional challenge and response protocol in a secure way. Generally, once the attestation of the sensor node is needed, the attestation protocols can be executed. For example, the BS schedules the attestation protocol to be periodically executed in order to guarantee the correctness of the WSN's functionality and to exclude any compromised node. Besides, the attestation protocol can be executed as a response action to the intrusion detection system alarm where a

Input : n, number of iteration
C_i, seed
Output: C $(C_0, C_1, \ldots, C_7)$, checksum value

```
1  C is initialized to C_i;
2  tmp is 1 byte initialized to 0;
3  j is initialized to point to C_0, the first byte of C;
4  b is given as input for fixed-block algorithm or computed as function in dynamic algorithm;
5  for i=1 to n/4 do
6      (t_0, t_1, t_2, t_3) = RC_seed(C_i);
7      b = function of (t_0, t_1, t_2, t_3)  mod m;
8      for k=0 to 3 do
9          for r=0 to b-1 do
10             tmp = tmp ⊕ Mem[t_k + r  mod m] ;
11         C_j = C_j + tmp;
12         tmp = 0;
13         j = (j + 1)  mod 8;
```

Figure 7 Algorithmic description of the dynamic BPMT algorithm.

misbehavior is detected by a WSN administrator. At the attestation time, the attestor sends a random challenge securely to the attestation function. Afterwards, the attestation function can execute the data memory filling, followed by any of the memory traversing algorithms over the already filled memory. The sensor node sends the checksum back to the BS which decides the correctness of the memory contents. The decision merely depends on the validity of the received checksum along with the resultant checksum of the emulated memory of the sensor node. In the following subsections, we intrduce the protocols used for the attestation scenario in more details.

The proposed protocol is used in the absence of time synchronization. Thus, the attested sensor node authenticates the BS attestation request by using the challenge response protocol before starting its attestation function. By doing so, the sensor nodes are immune to the DoS attack, which could be launched by replaying the attestation request and exhausting the sensor nodes' power. This attestation protocol consists of four steps: attestor request, node checksum computing, node response, and attestor verification. The steps are described as follows:

1. **Attestor request**: In this phase, the attestor, or BS, sends an attestation request to initiate the verification of memory content of the targeted sensor node. The request includes three communication steps for sending a challenge value C_i which would be used in the next step by the traversing algorithms. These steps are described as follows:

   BS $\rightarrow N_i$:

   $$Req_i, ID_{BS}, N_i, C_i, MAC_{K_{BS,N_i}}(ID_{BS}, C_i)$$

   $N_i \rightarrow$ BS:

   $$ID_{BS}, N_i, r_i, MAC_{K_{BS,N_i}}(ID_{BS}, C_i, r_i)$$

   BS $\rightarrow N_i$:

   $$ID_{BS}, N_i, MAC_{K_{BS,N_i}}(ID_{BS}, r_i + 1)$$

   In the protocol, a secure request is firstly initiated from the BS, including a random challenge C_i to the sensor node N_i. Afterwards, a sensor node N_i authenticates the request and verifies the ID_{BS}. Nonetheless, this step is not sufficient to approve the BS's request freshness. Hence, the node verifies the freshness of the request by sending back random challenge and another MAC that includes C_i, r_i. Finally, BS verifies the MAC and sends back MAC of $r_i + 1$, which can be verified by N_i. If these steps are correctly executed, the sensor node assumes this request as a valid attestation attempt and starts the next step (i.e., the attestation procedure). Otherwise, the sensor assumes this request as an illegal or replay request.

2. **Node checksum computing**: The sensor node memory is divided into two parts, i.e., data memory and program memory, which have a small and large size, respectively, and the free space in the program memory is already filled. Thus, after N_i successfully authenticates the BS request, N_i fills its data memory and computes the checksum CS_i over the whole memory by using one of the two memory traversal techniques.

3. **Node response**: Therein, the target node N_i sends the computed checksum as a response value CS_i to the attestor. The response message of the sensor node is different in each attestation round and depends on C_i. Node N_i sends the checksum with $MAC_{K_{BS,N_i}}(ID_{BS}, C_i, CS_i)$ in order to prevent replay attack. This step is depicted as follows:

   $N_i \rightarrow$ BS:

   $$Res_i, ID_{BS}, CS_i, MAC_{K_{BS,N_i}}(N_i, C_i, CS_i)$$

4. **Attestor verification**: Finally, the attestor authenticates the node's response by verifying its MAC value. Since the attestor knows the memory content of the targeted node N_i, including the incompressible random noise loaded before or after the sensor node deployment, the attestor internally recomputes the checksum CS'_{N_i} over N_i's emulated memory layout by using the same

traversal algorithm. In the case of $CS'_{N_i} \neq CS_i$, the attestor either puts the sensor in a blacklist by broadcasting its identity as a compromised node or undoes the changes of the attacker by reprogramming the node.

However, there are many WSN applications that deal with real time monitoring of physical phenomena. For such monitoring applications, physical time often plays a crucial role. Thus, time synchronization service among sensor nodes on one side, and between BS and sensors node on the other side, was already applied and could be exploited for our attestation scheme (26). Hence, in the presence of time synchronization service, the communication overhead of the attestation protocol is decreased compared to the aforementioned protocol. Because we can use two communication rounds instead of four rounds (i.e., attestation request round to transmit the challenge C_i and attestation response round to get the checksum CS_i). Accordingly, the attestation protocol for the synchronized WSN consists of the following communication steps:

1. **Attestation request**: In this phase, BS sends an attestation request to the targeted sensor node in order to initiate memory verification. The attestation request includes a challenge C_i that is used in the next step for memory traversing. Appending the timestamp TS_i to the MAC protects the sensor from the reply attack and guarantees request freshness.

   BS $\rightarrow N_i$:

   $$Req_i, ID_{BS}, C_i, TS_i, MAC_{K_{BS,N_i}}(ID_{BS}, C_i, TS_i)$$

2. **Attestation response**: As the sensor N_i computes the checksum CS_i over the whole memory layout like what we mentioned before, N_i sends the checksum CS_i and $MAC_{K_{BS,N_i}}(ID_{BS}, C_i, CS_i)$ back to the BS.

   $N_i \rightarrow$ BS:

   $$Res_i, N_i, CS_i, MAC_{K_{BS,N_i}}(N_i, C_i, CS_i)$$

5 Group-based Attestation Scheme

To attest all nodes in the sensor network, an attestor must be able to conduct the attestation against nodes that are even muti-hop away from the attestor. The first attempt to solve this problem was in SCUBA (19). The BS should send the attestation request as an expanding ring starts within the directly connected nodes. Then, BS sends the challenge over multi-hop and delegates the checking process for the directly connected node. However, because of SCUBA's time sensitivity, the expanded ring will incur two main consequences: (i) There is no guarantee that the authorized node in the path will not be compromised in between the other node verification time and node operation time. Thus, BS can not trust the checksums coming from the nodes in the path to attested nodes. (ii) Since the attestation scheme is time dependent and also the long path between the attested node and the BS will incur a long and unexpected time delay because of the underlying MAC protocol of the WSN, some false positive attestation results might happen. Thus, this time gain by the attacker increases the opportunity of the attacker to successfully establish an attack against the attestation process without being detected.

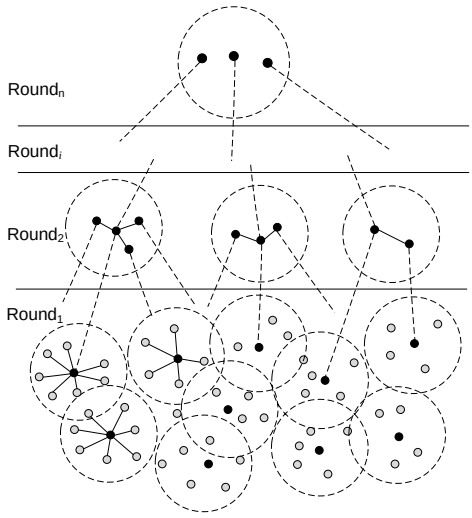
In this section, we are going to propose a group-based attestation scheme by distributing the role of attestor to normal nodes in the basic attestation schemes proposed in Section 4 for the attestation of the whole network. This distributed approach will make node-to-node attestation possible instead of base station-to-node attestation. The concept of a group-based attestation is that a group of nodes cooperate to attest each other and check if there is any compromised node in the whole network in order to inform the central authority. Group attestation will *localize the attestation process*, and thus, can reduce the communication and time needed for the attestation. Also, it solves the problem of unexpected time delay because of the underlying MAC protocol in case of expanding ring. Moreover, it will decrease the overhead on the central authority, i.e. the base station.

Most of the current attestation schemes cannot achieve the attestation of nodes multi-hop away from the base station without significant security degradation. To overcome this problem, we are going to take advantage of two important properties of the basic schemes, which are the time insensitivity and lightweightness of the attestation process by which a normal sensor node can conduct the attestor's role. Thus, our group attestation scheme can attest all the nodes of the sensor network in a distributed manner.

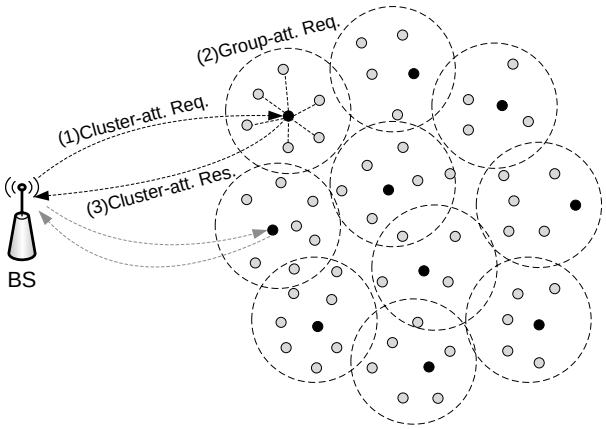
5.1 Attestation Scheme without BS Involvement

For the first proposal of group attestation, we are going to propose an attestation scheme without any BS involvement. The group attestation is triggered periodically in order to see if there is any misbehaving node in the network. We assume either the sensor nodes form a cluster-based network or each collection of neighboring nodes constructs a group. The WSN formation is shown in Figure 8(a). The attestation protocol consists of four steps: attestor request and challenge computing, nodes checksum computing, nodes response, and neighbors' verification. The steps are described as follows:

1. **attestor request**: At the time of attestation, the nodes of the cluster cooperatively need to compute a fresh challenge C_i for the attestation process.



(a) Attestation protocol propagation over WSN groups without BS involvement, where the black nodes are the cluster heads and gray nodes are the normal nodes.



(b) Attestation protocol with BS authorization, where BS sends a request for each cluster.

Figure 8 Group-based attestation protocols with and without BS involvement.

The challenge must be fresh in order to prevent the attacker from pre-computing the response of the attestation protocol. Here, we used a simple and efficient technique where each node sends its challenge C_i to the cluster head (CH_i). Then, CH_i computes the attestation challenge as $C_{CH_i} = H(C_1, C_2, \cdots, C_j)$, where $|j|$ is the cluster size and is usually < 10. After that, CH_i broadcasts the attestation challenge as:

$CH_i \rightarrow *$:

$$\{C_1, C_2, \cdots, C_j\}, C_{CH_i}$$

2. **Node checksum computing**: Each sensor involved in the attested cluster uses the computed challenge to run the attestation process. Thus, each node N_i computes its response as $H_{N_i} = H(C_{CH_i}, CS_i, N_i)$, where C_{CH_i} is the computed challenge from the previous step, and CS_i is the attestation process's checksum over the sensor node N_i memory. Then, the node broadcasts this hash value to the neighboring nodes.

   $N_i \rightarrow *$:

   $$N_i, H_{N_i} = H(C_{CH_i}, CS_i, N_i)$$

3. **Attestor verification**: As the memory content of the uncompromised nodes are typical, the value CS_i of all nodes of the attested cluster are equal. Thus, each node N_j in the cluster attests each received checksum H_{N_i} from their neighbors as $H'_{N_i} = H(C_{CH_i}, CS_j, N_i)$ and checks if $H'_{N_i} \stackrel{?}{=} H_{N_i}$.

Since the introduced protocol is an attestation instance, where a group of nodes (i.e., nodes of one cluster) are verifying each other, the protocol is scalable for the case of the attestation among the cluster heads. Thus, in the first round of the protocol run, the nodes of each cluster are attested. Then, since the cluster head optionally do an extra operation of reporting the compromised node, it needs to be attested in a second round, where it is considered as a regular node. Therefore, in the second round, another virtual cluster are formed among the neighbor cluster heads for a new round of the group attestation as illustrated in Figure 8(a). This group attestation is recursively repeated until all sensor nodes are attested, as shown in Figure 8(a).

5.2 Attestation Scheme with BS Authorization

The previous group attestation scheme localized the attestation process to each group of sensors in order to avoid the network delay. However, the sensor nodes have to concur in the challenge computation. Also, sensors must run the attestation protocol for a predetermined time (i.e., periodically) in order to prevent DoS attack. Group attestation protocol with BS involvement relaxes the computation and communication overhead of the previous scheme and keeps the localization property of the previous scheme the same. Also, since the BS is involved, the initiation of an attestation request is much more flexible. In fact, it might be event-driven or periodic. Thus, the result of the attestation is immediately available to the intrusion detection system. The attestation protocol is shown in Figure 8(b) and is described as follows:

1. **BS attestation request**: In this phase, BS attests a CH_i. Then, the CH_i sends back a list of node identifiers to the BS. If the CH_i is not compromised, BS sends authentic attestation coupons to CH_i which gives it the authority to attest the other nodes in the cluster.

   BS $\rightarrow CH_i$:

$$Req_i, ID_{BS}, CH_i, C_i, MAC_{K_{BS,CH_i}}(ID_{BS}, C_i, TS_i), (Coup_1, Coup_2 \ldots, Coup_j),$$

where the coupon of each N_i in the cluster is $Coup_i = H(N_i, CH_i, TS_i, K_{N_i})$.

2. **Group attestation request**: As the CH_i receives the attestation coupons, it initiates group attestation by passing the coupons to the cluster's nodes.

3. **Node checksum computing**: Each N_i authenticates the request by checking $Coup_i$ and starts computing the response CS_i as the previous group attestation protocol. Moreover, each node N_i sends a MAC value that can be used by the CH_i to report the node status.

   $N_i \rightarrow CH_i$:

   $$N_i, H_{N_i}, MAC_{K_{BS,N_i}}(N_i, CH_i, C_i, CS_i), \quad where\ H_{N_i} = H(C_i, CS_i, N_i).$$

4. **attestor verification**: As the memory content of the uncompromised nodes are typical, the value CS_i of all group nodes are equal. Thus, CH attests each received checksum H_{N_i} of node N_i from its cluster by computing $H'_{N_i} = H(C_i, CS'_i, N_i)$ of node N_i and checks if $H'_{N_i} \stackrel{?}{=} H_{N_i}$. Finally, CH_i replies the hashed value of the whole correct nodes' responses to the base station. Also, CH_i generates a report for the compromised node with their reported MAC values.

   $CH_i \rightarrow$ BS:

   $$Res_i, CH_i, CS_{CH_i}, MAC_{K_{BS,CH_i}}(CH_i, C_i, CS_{CH_i}), [E_{K_{BS,CH_i}}(C_i, Report, TS_i)]$$

5. **BS verification**: By receiving the report from the CH_i, BS emulates the node's checksums computations and verifies both the the CH_i's memory and the report correctness.

This protocol is also an attestation instance, where a group of nodes (i.e., one cluster nodes) are verifying each other. Thus, for attesting the whole WSN, the BS transmits a challenge for each cluster head and receives a report from it, as shown in Figure 8(a).

6 Security Analysis

In this section, we look over the security of the attestation schemes under the assumptions and the threat model given in Section 3.1. Furthermore, we explore the security properties of our protocols against the possible attacks, which are considered separately as follows:

Replay attack and checksum pre-computing: The attacker who compromises a sensor node tries to compute the checksum over the whole memory and store all possible values of the checksums before installing a malicious code. Then, when attestation begins, the compromised node may attempt to respond with the pre-computed checksum. However, since the attestation schemes depend on the challenge and response mechanism in the attestation protocols with a uniformly fresh random challenge each attestation attempt, these attacks are prevented.

Memory copy attack: In the memory copy attack, the attacker can do two things: (*i*) copy the original sensor's code in another location of the sensor's memory and replace it with a malicious code or (*ii*) keep the original code and copy the malicious code to any place in the sensor memory. In (19; 21), the attack was only detectable by measuring the exact time of the attestation response because the empty memory was not filled and the attacker was able to deploy a malicious code freely. Thus, adopting one of these schemes requires careful consideration of the WSN, where response timing is sensitive to external factors such as network channel collision and network delay. However, the memory copy attack is completely prevented in our attestation protocols where all free memory (i.e., RAM and Flash) is filled by high-entropy noise from our memory filling techniques and there is no free space for any malicious code. Hence, in the case of the full hash traversal, if a malicious code is installed by overwriting certain random values on sensor node memory, the attacker must compute online the content of the used blocks (i.e., random block on demand). We will show in the security analysis section how hard this scenario is, since it requires the attacker to do heavy computations and accordingly has a large attestation time. On the other hand, for the case of pseudo-random traversal algorithm, the attacker, who tries to modify or replace any random filled block, must predict in advance the traversal algorithm jumps. Also, we show in the security analysis section the detection probability if the attacker modifies either filled blocks or program instructions.

A Rootkit-based attack: This attack was proposed by Castelluccia *et al.* (4), and their attack was based on return-oriented rootkit that exploited the characteristics of the sensor micro-controller and bootloader. The attacker in this attack inserted a hook to the attestation function. As the attestation function was called, the hook triggers attacker rootkit code in the bootloader which deletes the malicious content and the hook from the program memory. Once the attestation is over, the rootkit restores itself into program memory by using both the procedure stored in the bootloader and the attacker routines in the data memory. However, since our attestation scheme filled the empty space of the program

and data memory, attacker is not able to restore the random filling used by the malicious code and routines.

Compression attack: This attack was also proposed by Castelluccia *et al.* (4), and the attack depends on compressing the original code and exploiting the small gained space in order to apply the return-oriented rootkit attack and store a malicious code. However, this attack is not applicable in our schemes for several reasons. Initially, the attack assumes the existence of the code of compression and decompression functions, which requires a considerable space in the program and data memories. However, this is not the case for many sensor's application that do not use these functions. Also, the attack uses RAM to store return-oriented rootkit routines and decompression Meta data, which is not feasible for our scheme because the RAM is already filled by random for the attestation. Since the program memory and the free RAM space have been filled in the attestation scheme, the only available space to the attacker is the gained space from compressing the original code, where it must store: the decompression function, decompression Meta data, return-oriented rootkit routines, and attacker malicious code. However, following the author's example, the compression gain was 4208 bytes, while the required space to execute the return-oriented rootkit, even without a malicious code, is 6359 bytes. This shows the difficulty of applying this attack without enough memory space, which was already filled in our attestation scheme.

Online computing of random block on demand The attack is related to the full hash traversing technique, where the malicious code generates required blocks at the time when the hash function needs the blocks. Let n be the number of blocks of empty memory. Then n-th block can be generated from the seed by n times hashing. Similarly, $(n-1)$-th block can be generated. This method requires lots of hash calculations about $O(n^2) = n + (n-1) + \cdots + 1$. As we know, the number of hash chain calculation by "clean node" is $\Theta(n)$. If n is large enough, $O(n^2)$ can be a heavy burden to execute in the malicious node. In some cases, n can be compromised of several hundreds to thousands of blocks. (Note that n is the number of hash blocks and thus, if we use SHA-1, one hash block is 20 bytes long.) There is a more sophisticated method for the attacker to reduce the hash calculation from $O(n^2)$ to $O(n\lg n)$ with more memory $O(\lg n)$ (6). Although $O(n\lg n)$ is smaller than $O(n^2)$, the verifier still has, however, a sufficient difference between $O(n)$ and $O(n\lg n)$ by which it can discriminate a normal attestation response from an abnormal one. Setting a proper threshold prevents the attacker from having enough time to compute the hash value with enough memory to keep its malicious code and valid original code. We will show the inapplicability of this attack in the simulation mentioned in Section 7.

Proxy attack: The attack happens if a part of the memory content of a compromised sensor node is copied to another compromised node. Subsequently, the malicious code is installed on the evacuated part (19). Later, when the attestation request is received, the compromised node cooperates with the nodes by having its partial memory contents let the attested node bypass the attestation and send the correct checksum. This incurs an expensive communication overhead by the attested node with other cooperated nodes to evaluate the intermediate checksum over what partial memory content each node has. The attack is applicable if the traversing technique is sequential. The full memory traversing technique, since the filling occurred horizontally and the checksum was computed vertically over small blocks from each column of the memory matrix. This makes each column being accessed by the traversal algorithm as much as the number of rows of the matrix. The attested node which cooperates with other nodes needs to access the network as much as the traversal algorithm accesses each memory column individually. The proxy attack in the case of the full memory traversal is upper bounded by the number of blocks in each memory column. On the other hand, the proxy attack in the pseudo-random traversals is harder and needs an unpredictable number of network access by the compromised node because the memory blocks to be traversed are unpredictable. This results in a distinguishable delay in the attestation's response time, which would be detected easily by the attestor.

7 Simulation Results

7.1 Performance Evaluation

In this section, we show the feasibility of our attestation schemes in the scenarios of BS-to-node attestation and group attestation. Thus, a Java simulation was implemented for this purpose. The goal of our simulation was to analyze and evaluate the performance of our attestation schemes in terms of efficiency, power consumption, and time. The analyses included individual algorithms (i.e., data memory noise filling, FMT and pseudo-random traversal techniques) for checksum computing. Also, we evaluated the communication overhead of our basic challenge-response communication protocols and group based attestation schemes. Following our threat model, we intend to protect the data memory and program memory, i.e., RAM and Program memory respectively, since the writing time for the external EEPROM is too slow (14) and cannot be used by an attacker without being detected. The size of the external EEPROM in most of the sensors is about a few mega-bytes, and in order to fill that, it will take several thousand seconds. For example, if the size of EEPROM is 4M bytes, and the writing time for 1 byte is $1ms$, it takes over 1 hour 9 minutes 54 seconds. However, the writing time in RAM is about 2 clk_{cpu} cycle (1). Also, since we fill the data and program memories and attest them, the attacker can not access the EEPROM without a hook in either data or program memory. First, evaluation of the

attestation checksum computing was done when a sensor node had both the on fly data memory filling, and either pre- or post-deployment filling was applied to the data memory and program memory, respectively. In addition, the FMT traversing technique was used for checksum computing. We estimated the total execution time T of the attestation process in this scenario by

$$T = T_r + (T_h + T_a) \times S_E \tag{1}$$

where T_r is the time needed to calculate the checksum over the memory including the data memory and program memory, T_h is the time needed to generate one random block for the data memory, and T_a is the time needed for writing one block to the memory.

In this case, at the time of attestation, the estimated execution time of the data memory filling is just counting the hash operations needed to fill the RAM. Let U_f be the number of hash rounds for filling the empty RAM space, and δ be the time to execute one round. Then $T_h \times S_E$ can be rewritten by $U_f \times \delta$. Then

$$U_f = \lceil |\text{RAM}|/S_{ho} \rceil \tag{2}$$

where S_{ho} is the output size of hash function, and $|\ |$ denotes the size of specific memory. (S_{ho} is 160 if the hash function is SHA-1, and is 128 if MD5.)

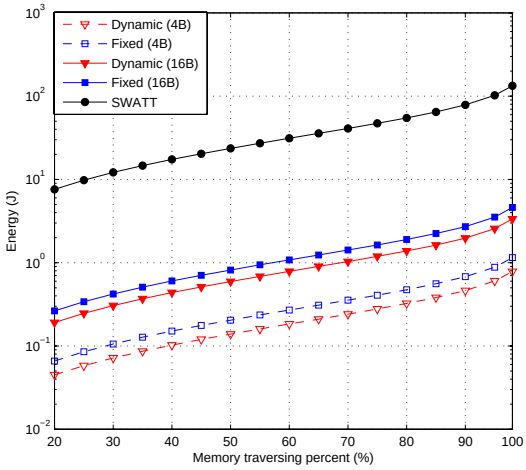
Let U_r be the number of hash rounds for making checksum CS_i, then

$$U_r = \lceil (|\text{RAM}| + |\text{Flash}|)/S_{hi} \rceil + 1 \tag{3}$$

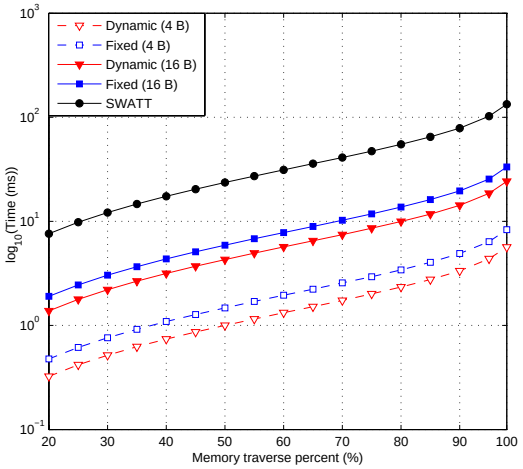
where S_{hi} is the input size of one round of hash function. (S_{hi} is 512 commonly. +1 is needed because of the initial padding operation of the hash function.) Then T_r can be rewritten by $U_r \times \delta$. Finally,

$$T = U_r \times \delta + U_f \times \delta = (U_r + U_f) \times \delta \tag{4}$$

A simulation of the internal attestation process on different sensor node platforms was carried out for evaluating the data memory filling and full hash traversal technique. The platforms used in the simulation are MICA2 sensor with ATmega128 Micro-controller (14), MTM-CM2000-MSP sensor with MSP430F1611 Micro-controller (15) and MSP430F149 Micro-controller (10). Also, the derived performance and analyses model of the cryptographic hash functions follows Ganesan *et al.* (8) analyses. We show in table 2 the total attestation time of the data memory filling and full hash traversal. The first row in the table shows the time in seconds for the normal node in the case of two rounds data memory filling. Obviously, the attestation total time will be increased as the number of rounds increased; This small time difference happens because of increasing the number of rounds will have great influence on the security of attestation as will see in the security analysis section. Also, simulation was conducted to evaluate the



(a) Energy consumption of SWATT pseudo-random traversal and the other two different BPMT algorithms.



(b) Execution time of SWATT pseudo-random traversal and the other two different BPMT algorithms.

Figure 9 The power and time consumption of the various pseudo-random memory traversal algorithms on Mica2 sensor.

performance of our pseudo-random traversal technique in Mica2, which uses TinySec (12) link layer security architecture with a variable 37-byte packet size (i.e., 29 byte payload).

Figure 9(a) shows the power consumption of several pseudo-random traversing algorithms, where the energy consumption of $RC5$ comes from (12). It shows the energy consumption level of the cell based (i.e., SWATT like algorithm) and the other two BPMT algorithms is increased as the amount of the covered memory is increased. Apparently, the dynamic BPMT is the most energy efficient algorithm. For instance, $1J$ is the energy consumption when 90% of the memory is traversed by the dynamic BPMT algorithm compared to 23.2 J in the case of a cell based approach. Also, the dynamic BPMT is 10 times faster than SWATT (21) and double times than the fixed BPMT, as shown in Figure 9(b). Moreover, the pseudo-random algorithms are faster than

Table 2 The total attestation time of the data memory filling and full hash traversal technique in different sensor's platforms.

	MICA2 (ATmega128)		MSP430F149		MTM-CM2000-MSP (MSP430F1611)	
# of filling Rounds	MD5 (Sec.)	SHA1 (Sec.)	MD5 (Sec.)	SHA1 (Sec.)	MD5 (Sec.)	SHA1 (Sec.)
2	2.98	10.75	2.13	8.71	5.11	14.55
3	3.74	12.32	2.51	9.49	7.02	18.47

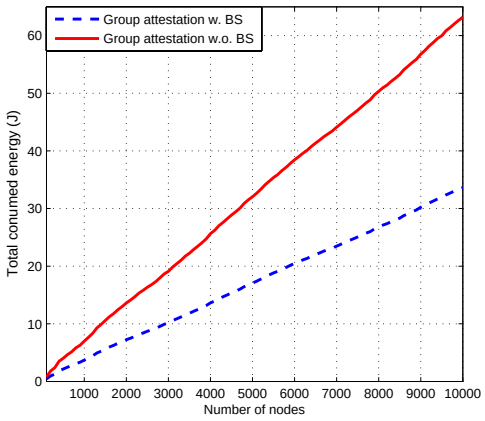


Figure 10 Total energy consumption of the group-based attestation schemes in different WSN networks size.

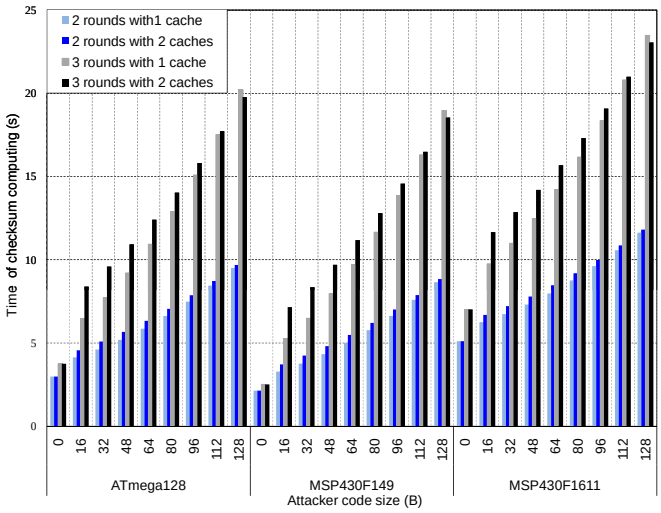


Figure 11 Verification time of valid node and malicious node using the full memory traversal technique in different sensor's platforms.

the full memory traversal because the full memory traversal will hash detrministically the whole memory and will not depends on the probability that some blocks are not traversed. For the communication overhead, we analyzed our basic attestation protocols from section 4.3 and the communication overhead of the group-based attestation was able to be estimated since it is the multi-round version of the basic protocols. According to (23), Mica2 has a data transmission rate of 26 μs/bit, a receiving current of 7.0 mA, a sending current of 21.5 mA, and 3V power supply. The energy consumption becomes as follows:

- $E_{Transmission} = 3 \times 21.5\text{mA} \times (26 \times 10^{-6}\text{s/bit} \times 37 \text{ bytes}) = 0.5426$ mJ/message.
- $E_{Reception} = 3 \times 7.0\text{mA} \times (26 \times 10^{-6}\text{s/bit} \times 37 \text{ bytes}) = 0.1617$ mJ/message.

Hence, the communication overhead of the challenge and response protocol of section 4.3 is 2 received and 2 transmitted messages with a total energy consumption of (1.4084 mJ). However, the communication overhead of the attestation protocol that uses time synchronization is 1 received and 1 transmitted messages with a total energy consumption of (0.7042 mJ). Since the group based attestation schemes are a generalization of the basic attestation with extra communication overhead for the challenge preparation phase, we show in Figure 10 the total communication overhead of our proposed group-based schemes in terms of network size. We used LEACH (11) algorithm as an underlying cluster formation algorithm with cluster size around 10 nodes/cluster.

7.2 Security Evaluation

An evaluation of the critical security attacks against our traversal algorithms was simulated. On one hand, this simulation estimated the the online computing of random block on demand attack mentioned in section 6 for the case of FMT algorithm. On the other hand, we estimate the detection probability of an attacker's code injection in case of applying the BPMT algorithm. For the Online computing of random block on demand attack, let n be the number of blocks of the empty memory. Then, the n-th block can be generated by an attacker from the seed by $\lceil \|S_E\|/S_{ho} \rceil \times 2 - n$ times hashing because the n-th block is calculated by $R_n \oplus R_{\lceil \|S_E\|/S_{ho} \rceil \times 2-n}$ as shown in the example of Figure 2.

We show in table 3 the attacker total time required to compute the attestation checksum over the sensor memories. The first row with zero attacker code means the attestation time of the uncompromised node. Thus, the simulation results showed that the required time is exponentially increased as the attacker code size is increased. In figure 11, further simulation results about

Table 3 Attacker's total attestation time with various malicious code size.

	MICA2 (ATmega128)		MSP430F149		MTM-CM2000-MSP (MSP430F1611)	
Malicious Code(B)	MD5 (Sec.)	SHA1 (Sec.)	MD5 (Sec.)	SHA1 (Sec.)	MD5 (Sec.)	SHA1 (Sec.)
0	2.98	10.75	2.13	8.71	5.11	14.55
16	52.08	137.24	26.68	71.96	127.84	330.78
32	76.62	200.48	38.27	103.58	188.56	488.89
48	101.17	263.73	51.22	135.20	250.56	647.00
64	125.71	263.73	63.50	135.20	311.93	647.00
80	150.26	326.97	75.77	166.82	373.29	805.11
96	174.80	390.21	88.04	198.45	434.66	963.22
112	199.35	453.46	100.32	230.07	496.02	1121.33
128	223.90	516.70	112.59	261.69	557.39	1279.44

the attacker total attestation time in the case of two and three memory filling rounds were illustrated. Since the results presented in table 3 show only the computation time of compromised node with various malicious code injected, Figure 11 shows the attestation time when the attacker has the advantage of having different cache blocks that he can use to speed up the computation time. To have cache, a smart attacker can inject his code then exploit the xor relation of computed random blocks at the filling time by storing intermediate cache values in order to reduce the number of online hashes needed at the checksum computing phase. Thus, the best use of caches (i.e. how many caches and which xor-ed values should be cached) might optimally reduce the time. This problem is known as the time-memory trade off hard problem (16)

However, the attacker's decision of the caches positions (i.e., xor-ed values that should be cached at the filling phase) should be on the fly for each attestation challenge which is also hard. The cache size in our simulation was the input size of the hash function (e.g., 512 bits in SHA-1). For our attestation scheme, we found that two rounds are sufficient to make a distinguishable time gap between the valid and the malicious node which runs an online computing of the random blocks on demand, as shown in Figure 12(a). We observed that the time gap of checksum computing between the valid node and malicious node increased as either the attacker injected code or the filling algorithm rounds increased. From this analysis, we can see that WSN administrator can make a trade-off between security (distinguishable time gap) and power consumption by adjusting the number of rounds. On the other hand, we ran the dynamic BPMT algorithm and simulated an attacker that modified a variable number of instructions. In Figure 12(b), we show the relationship between the detection probability of the number of modified instructions and the random memory coverage by the BPMT algorithm. The figure shows that the detection probability obviously increased as the traversal algorithm either covered more memory or as the number of modified instructions increased.

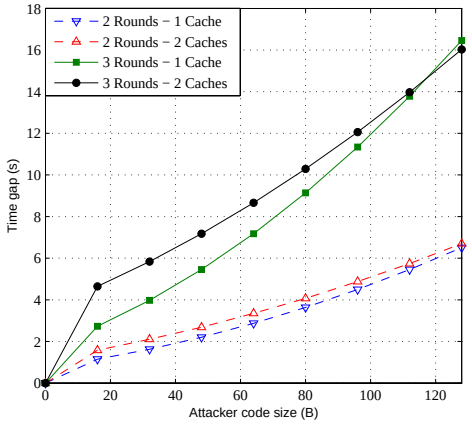
8 Conclusion

Node attestation is considered to be a crucial technique for detecting compromised nodes in WSN. Nevertheless, it is considered to be a challenging problem due to the limited resources of the sensor node. Installing TPM in individual sensor nodes is infeasible because of the fundamental requirements of WSN such as large scale and low cost deployment. Therefore, software-based code attestation is a promising solution for detecting compromised nodes. However, previously proposed software-based protocols had several limitations. These limitations related to time dependency makes the accuracy of compromised node detection and its computation overhead questionable. In this paper, various random filling and memory traversing techniques were presented and a simple and practical protocol for one hop attestation was shown. Also, two distributed group based protocols were introduced by exploiting the basic single hop protocol. Both the one hop and group attestation protocols have various advantages, such as loose time dependency compared to conventional attestation protocols, such as implementation feasibility, immunity to known attacks in code attestation, and their flexibility to all types of sensor network applications.

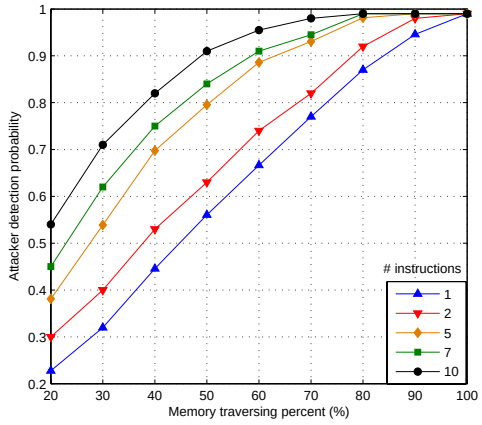
References

[1] Atmel 8-bit avr microcontroller with 128k bytes in-system programmable flash. atmega128 atmega128l rev. 2467n.avr.03/06.

[2] Tamer AbuHmed, Nandinbold Nyamaa, and DaeHun Nyang. Software-based remote code attestation in wireless sensornetwork. In *GLOBECOM*, pages 1–8, 2009.



(a) Verification time gap of a valid node and malicious node after using the full memory traversal technique.



(b) Attacker detection probability for either instructions modification or injection in the case of pseudo-random traversal technique.

Figure 12 The security analyses of the full memory traversal and pseudo-random memory traversal techniques in MICA2 sensor node.

[3] Micheal Mitzenmacher and. Probability and computing: Randomized algorithms and probabilistic analysis. In *Cambridge University Press*, 2005.

[4] Claude Castelluccia, Aurélien Francillon, Daniele Perito, and Claudio Soriente. On the difficulty of software-based attestation of embedded devices. In *CCS '09: Proceedings of the 16th ACM conference on Computer and communications security*, pages 400–409, 2009.

[5] Young-Geun Choi, Jeonil Kang, and DaeHun Nyang. Proactive code verification protocol in wireless sensor network. In *ICCSA (2)*, pages 1085–1096, 2007.

[6] Don Coppersmith and Markus Jakobsson. Almost optimal hash sequence traversal. In *Financial Cryptography*, pages 102–119, 2002.

[7] Jing Deng, Carl Hartung, Richard Han, and Shivakant Mishra. A practical study of transitory master key establishment forwireless sensor networks. In *SECURECOMM '05: Proceedings of the First International Conference on Security and Privacy for Emerging Areas in Communications Networks*, pages 289–302. IEEE Computer Society, 2005.

[8] Prasanth Ganesan, Ramnath Venugopalan, Pushkin Peddabachagari, Alexander Dean, Frank Mueller, and Mihail Sichitiu. Analyzing and modeling encryption overhead for sensor network nodes. In *WSNA '03: The 2nd ACM international conference on Wireless sensor networks and applications*, pages 151–159. ACM, 2003.

[9] Technical report Group, T.C.:Trusted Platform Module (TPM) specifications. http://www.trustedcomputinggroup.org/specs/tpm, Nov., 2010.(date of access).

[10] MSP430x1xx Family User's Guide. Texas instruments ltd. product documentation, 2004.

[11] Wendi Rabiner Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *HICSS '00: Proceedings of the 33rd Hawaii International Conference on System Sciences-Volume 8*, page 8020, Washington, DC, USA, 2000. IEEE Computer Society.

[12] Chris Karlof, Naveen Sastry, and David Wagner. Tinysec: a link layer security architecture for wireless sensor networks. In *SenSys*, pages 162–175, 2004.

[13] Christoph Krauß, Frederic Stumpf, and Claudia M. Eckert. Detecting node compromise in hybrid wireless sensor networks using attestation techniques. In *ESAS*, pages 203–217, 2007.

[14] MICA2 Wireless Module. Crossbow technology inc., 2003. www.xbow.com/products/productdetails.aspx?sid=174, Nov., 2010. (Data of access).

[15] MTM-CM2000-MSP Wireless Module. Maxfor technology inc., 2008. http://maxfor.co.kr/eng/en_sub5_1_2.html, Nov., 2010. (Data of access).

[16] Philippe Oechslin. Making a faster cryptanalytic time-memory trade-off. In *CRYPTO*, pages 617–630, 2003.

[17] Taejoon Park and Kang G. Shin. Soft tamper-proofing via program integrity verification in wireless sensor networks. *IEEE Trans. Mob. Comput.*, 4(3):297–309, 2005.

[18] Ronald L. Rivest. The rc5 encryption algorithm. In *FSE*, pages 86–96, 1994.

[19] Arvind Seshadri, Mark Luk, Adrian Perrig, Leendert van Doorn, and Pradeep K. Khosla. Scuba: Secure code update by attestation in sensor networks. In *Workshop on Wireless Security*, pages 85–94, 2006.

[20] Arvind Seshadri, Mark Luk, Elaine Shi, Adrian Perrig, Leendert van Doorn, and Pradeep K. Khosla. Pioneer: verifying code integrity and enforcing untampered code execution on legacy systems. In *SOSP*, pages 1–16, 2005.

[21] Arvind Seshadri, Adrian Perrig, Leendert van Doorn, and Pradeep K. Khosla. Swatt: Software-based attestation for embedded devices. In *IEEE Symposium on Security and Privacy*, pages 272–, 2004.

[22] Elaine Shi, Adrian Perrig, and Leendert van Doorn. Bind: A fine-grained attestation service for secure distributed systems. In *IEEE Symposium on Security and Privacy*, pages 154–168, 2005.

[23] Victor Shnayder, Mark Hempstead, Bor-rong Chen, Geoff Werner Allen, and Matt Welsh. Simulating the power consumption of large-scale sensor network applications. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 188–200, 2004.

[24] Diomidis Spinellis. Reflection as a mechanism for software integrity verification. *ACM Trans. Inf. Syst. Secur.*, 3(1):51–62, 2000.

[25] Stumpf, Omid Tafreschi, Patrick Roder, and Claudia M. Eckert. A robust integrity reporting protocol for remote attestation. In *WATC'06. Second Workshop on Advanced in Trusted Computing*, 2006.

[26] Bharath Sundararaman, Ugo Buy, and Ajay D. Kshemkalyani. Clock synchronization for wireless sensor networks: a survey. *Ad Hoc Networks*, 3(3):281 – 323, 2005.

[27] Yang Xiao, Venkata Krishna Rayi, Bo Sun, Xiaojiang Du, Fei Hu, and Michael Galloway. A survey of key management schemes in wireless sensor networks. *Comput. Commun.*, 30(11-12):2314–2341, 2007.

[28] Yi Yang, Xinran Wang, Sencun Zhu, and Guohong Cao. Distributed software-based attestation for node compromise detection in sensor networks. In *SRDS*, pages 219–230, 2007.