The Pennsylvania State University
The Graduate School

TOWARDS COMPROMISE RESILIENT WIRELESS SENSOR NETWORKS

A Dissertation in
Computer Science and Engineering
by
Yi Yang

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

August 2010

The dissertation of Yi Yang was reviewed and approved* by the following:

*Signatures are on file in the Graduate School.

Abstract

Sensor networks are envisioned to be powerful and economical solutions for both civilian and military applications, because of their capability and flexibility built on top of low-cost and small-size sensors. Meanwhile, sensor networks are also targets of all kinds of security attacks due to their extremely scarce resources and their unattended, hostile operating environment. Among all the potential attacks, insider attacks from compromised sensors are very challenging and difficult to be addressed. Compromised sensors have valid credentials, so that they can take part in the normal network operations and provide correct responses once challenged. *My objective is to reduce or remove the possible damages that compromised sensors might bring to the network and further to identify and revoke those compromised sensors.* In the literature, a lot of techniques have been proposed to achieve the above goal. I work on the following important issues that have not been well solved yet.

First, I propose SDAP, a secure hop-by-hop data aggregation protocol for sensor networks. Data aggregation is a very important technique in sensor networks to reduce data redundancy and communication overhead during sensor data collection. SDAP can preserve the efficiency of the ordinary hierarchical hop-by-hop data aggregation and meanwhile provide high assurance on the trustworthiness of the final aggregation result, even though multiple compromised sensors may collude to trick the base station to accept bogus data from them.

Then, I devise two distributed software-based attestation schemes to detect compromised sensor nodes, in which multiple neighbors of a suspicious node collaborate in a challenge-response process and make a joint decision regarding the trustworthiness of a suspicious sensor node in a distributed manner. The schemes only involve regular neighboring nodes and no trusted verifier or BS is included, so the attestation could be finished in a timely and distributed manner; also, they do not rely on response time difference to distinguish between a benign node and a compromised node, so the result is more accurate and reliable.

Last but not least, I target on a potentially (if not the most) severe attack from compromised sensors, which is named sensor worm attack. We not only validate the possibility for worm attack to happen in sensor networks, but also propose an effective defensive scheme based on software diversity. To the best of our knowledge, this is the first work investigating in depth on sensor worm defense.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would like to express my gratitude to all those who gave me selfless help to complete this dissertation. First of all, I am extremely grateful to my advisor, Dr Sencun Zhu, for always being there with his encouraging words and guiding me through the course of research work. His keen professional insight and critical suggestions made this dissertation possible. His influence to me is in not only my research but also my life. He teaches me how to be a complete person through his personal example.

A significant acknowledgement must be made to my committee members, Dr Guohong Cao, Dr Trent Jaeger, and Dr Runze Li, for having spent their valuable time reviewing the draft of this dissertation and excellent feedback that helped me to shape my research skills. I also would like to thank Dr Thomas LaPorta, Dr Bhuvan Urgaonkar, and other professors in the department for their great suggestions and help for my career.

I am also deeply indebted to my co-authors and colleagues in the department for providing a nice academic environment. Their continuous support always made me feel as if I was under the safe shelter of a big family, always having someone around to back me up and cheer me up.

I want to thank all my friends either in PennState or far away, for their kindness, trust, and emotional support that played a big role during the course of this study.

Finally, I would like to express my deep appreciation to my parents who have provided endless love and encouragement to me. Special thanks need be given to my husband, Zhixi, for his love and unconditional support and always being with me. Without their love and support, I cannot finish this work.

I sincerely thank you all!

Chapter 1

Introduction

1.1 Security Challenges from Sensor Node Compromise

Wireless sensor network is envisioned to be an economical solution to many important applications, such as real-time traffic monitoring, military surveillance, and homeland security [1]. This is due to the surprisingly power, convenience, and flexibility provided by a sensor network consisting of hundreds or even thousands of low-cost, tiny sensors. In such a network, each sensor acts as an information source, sensing and collecting data from the environment for a given task. There may also exist one or more *base stations* (or *data sinks*), which have access to the Internet infrastructure, distributing queries to the sensors and collecting data from the network. The base stations then report the query results to a *remote user* who subscribes to specific data streams.

A representative example of sensors is Mica2 mote [2], a small ($58\times$ $32\times$ 7 mm) sensor unit with a processor, radio, power source, and optional sensing elements. The processor is 8MHz 8-bit ATmega128L with 128KB of program flash memory, 4KB of RAM for data, 4KB configuration EEPROM, and 512KB of measurement (serial) flash memory. The radio is a 916MHz low-power radio from RFM, delivering up to 40 Kbps bandwidth on a single shared channel and with a range of up to a few dozen meters or so. An optional sensor board allows mounting of a temperature sensor, magnetometer, sounder, and other sensing elements. The whole device is powered by two AA batteries, which provide approximately 2850 mAh at 3V. The processor consumes 5.5mA (at 3V) when active, and two orders of magnitude less power when sleeping. The RFM radio consumes 4.8mA (at 3V) in receiving mode, up to 12mA in transmission mode, and

$5\mu A$ in sleeping mode. Hence, communication (message transmissions) dominates the energy expenditure of sensors.

Sensor networks subject to various security attacks because of the following reasons. First, sensors have scarce resources in communication, computation and storage, as explained above. Expensive cryptographic mechanisms, such as public key encryption/decryption, are not feasible to be applied in sensor networks. Second, due to the low manufacturing cost (e.g., less than one dollar for smartdust microsensors), it is unlikely that we will use expensive *tamper-resistant* hardware for sensors. Third, sensors often operate in an *unattended, harsh*, or *hostile* environment. An attacker may launch numerous attacks, to obtain important information from sensor networks, to interfere with their normal operations, or even to destroy them. The question is how much we can do to increase the defensive capability and survivability of a sensor network that is under attacks.

Up-to-date sensor networks normally employ modern cryptographic techniques to guarantee *data confidentiality, data integrity* and *data availability*. For example, data encryption/decryption by secure symmetric keys can be exploited to ensure data confidentiality; message authentication code is effective to provide data integrity; denial of service defensive techniques are useful in improving data availability. Based on these cryptographic mechanisms, in the literature, researchers have proposed security techniques to defend against many kinds of attacks in sensor networks, e.g., dodging communication channel jamming [3, 4], thwarting MAC layer attacks [5], countering attacks against routing protocols [6], preventing false data injection attacks [7], providing attack-resilient node localization [8, 9] and time synchronization [10, 11].

From all these experiences, we have learned the security challenges in wireless sensor networks. The attacks from an outsider, who does not join network operations, are relatively easy to be addressed by encryption or authentication. Security challenges actually come from insider attacks launched by compromised sensor nodes which are parts of the sensor network. These nodes compromised by attackers possess valid credentials, so that they can easily give correct responses to pass the tests from other contemporary sensors. Therefore, these compromised nodes are insidious destructors, which are very difficult to be detected.

The attacker has strong motivations to compromise sensors. Breaking cryptographic mechanisms employed by sensor networks takes long time and considerable computa-

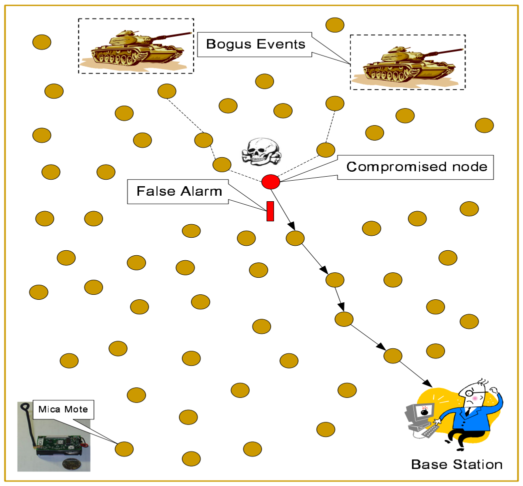


Figure 1.1. An example sensor network application. Here a compromised node may report a false alarm on the presence of tanks to mislead the network administrator.

tion resources. Alternatively, the attacker could simply corrupt a sensor to obtain everything he needs. According to [12], it takes only less than one minute for the attacker to extract all the information located onboard the node's memory. Therefore, the attacker may capture a fraction of the sensors in the network, reprogram them with malicious code, and redeploy them back into the network. Besides the exposure of secret information (e.g., cryptographic keys), the compromised nodes may launch all kinds of passive and active attacks, under the full control of the attacker. An example of bogus alarm from compromised node is illustrated in Figure 1.1. A recent study [13] has shown that a single compromised node is sufficient to take over the entire network and prevent communications among sensor nodes in all of the fourteen routing protocols examined. The potential damages that might be brought by colluded multiple compromised sensors are even harder to be estimated.

Among all the possible attacks that may be launched by the compromised nodes, we have special interest in the impact of compromised nodes to the result of sensor data aggregation, because hop-by-hop data aggregation is a very important technique for reducing the communication overhead and energy expenditure in the process of sensor data collection. Since individual sensor readings are lost in the per-hop aggregation process, compromised nodes in the network may forge arbitrary false values as whatever the attacker wants, causing the final aggregation result calculated by the base station to far deviate from the true measurement. In many situations, values computed by the BS provide a basis for critical decisions of remote users; hence, false or biased aggregation values may cause catastrophic consequences. Unfortunately, most sensor data aggregation techniques [14, 15, 16, 17, 18, 19, 20] were not designed with security in mind. A fundamental security challenge that we try to solve here is: *how can the base station obtain a good approximation of the fusion result when a fraction of sensor nodes are compromised?*

Nevertheless, the more important issue is not reactive strategies to alleviate the damage of an attack from compromised sensors after it happens, but the preventive methods to find out those compromised sensor nodes. Very desirably, if we can identify and further revoke those corrupted nodes in a timely manner, the potential damages caused by them could be minimized. As discussed before, this is a tough problem to be solved. Intrusion detection mechanisms such as watchdog [21] and reputation-based system [22] have been proposed to detect abnormal nodes, but they rely on accurate observation and reasoning of node's misbehavior. Not surprisingly, behavior-based detection is error-prone and could be easily bypassed by an attacker who knows the decision threshold. *Our goal is to detect node's compromise with an overwhelmingly high probability irrespective of node's behavior*.

Furthermore, we study a potentially the most dangerous attack that may be launched by the compromised nodes. We name this attack *sensor worm attack*. More specifically, we define sensor worms as crafted messages that exploit software vulnerability of sensor nodes in a sensor network, causing sensor nodes to crash or taking control of sensor nodes. Clearly, sensor worm attack could be the most destructive one if an attacker simply sends a single message to compromise the entire sensor network, defeating the mission of the sensor network. The only related work we know of is one on modelling sensor worm propagation [23]; it, however, neither discussed the feasibility of launching sensor worms nor proposed any solution to address sensor worms. The questions that we want to answer are: *is it possible for worm attacks to occur in sensor networks? If it is possible, what can we do to defend against such sensor worm attacks?*

1.2 Summary of Contributions

To answer all the above questions in my mind, I make the following three contributions in my dissertation research study.

The first contribution is SDAP, a Secure Data Aggregation Protocol for sensor networks, which can preserve the efficiency of the ordinary hierarchical hop-by-hop data aggregation and meanwhile provide high assurance on the trustworthiness of the final aggregation result.

SDAP takes the approach of reducing the trust on high-level nodes, which is realized by the principle of *divide-and-conquer*. More specifically, by using a probabilistic grouping method, SDAP *dynamically* partitions the topology tree into multiple *logical* groups (subtrees) of similar sizes. Then, SDAP performs hop-by-hop aggregation in each logical group and generates one aggregate from each group. In addition, based on the principle of *commit-and-attest*, SDAP enhances an ordinary hop-by-hop aggregation protocol with commitment capability, which ensures that once a group commits its aggregate this group cannot deny it later. After the BS collects all the group aggregates, it then identifies suspicious groups based on a bivariate multiple-outlier detection algorithm. Each group under suspect needs to prove the correctness of its group aggregate. The final aggregate by the BS is calculated over all the group aggregates that are either normal or have passed the attestation procedure.

The second contribution is the design of two novel distributed software-based attestation schemes for node compromise detection in sensor networks. First, noises (pseudorandom numbers) are filled into the empty program memory of each node before deployment. Then, either the seed for generating the noise is distributed to multiple neighbors based on threshold secret sharing, or a random digest of the program memory content is assigned to each neighbor. When an attestation is triggered, multiple neighbors of a suspicious node collaborate in a challenge-response process and make a joint decision regarding the trustworthiness of this node in a distributed manner.

Our distributed schemes differ from the previous ones in that (1) they only involve regular neighboring nodes and no trusted verifier or BS is included, so the attestation could be finished in a timely and distributed manner, and (2) they do not rely on response time difference to distinguish between a benign node and a compromised node. These nice features make them very attractive for unattended sensor networks where a BS or trusted verifier could be the single point of failure from both security and operation perspectives.

The third contribution is an innovative idea of using software diversity approach to improve the sensor network immunity under worm attacks. To the best of our knowledge, this is the first work not only to validate the feasibility of launching worm attack in sensor networks but also to propose sensor worm defense through software diversity.

It is widely believed that the Harvard architecture of many sensor microcontrollers (which has separate memories for program and data) effectively precludes the happening of buffer overflow attack on sensors. However, as a trial study, we conducted experiments on Mica2 motes and found that a buffer overflow vulnerability, the common trigger of worm attacks, may lead to unpredictable system behaviors such as re-initialization of the sensors. Under some circumstances, this may cause the corruption of sensors as well as the propagation of sensor worm packets. Moreover, in the spirit of *survivability through heterogeneity* philosophy, we explore the technique of *software diversity* to combat sensor worms. We design a general color assignment algorithm and analyze the impact of sensor deployment error to the effectiveness of sensor worm containment.

1.3 Dissertation Organization

The remainder of the dissertation is organized as follows. The next chapter discusses the related work in sensor data aggregation, software-based attestation for sensor compromise detection, and sensor worm defense. Chapter 3 introduces a secure hop-by-hop data aggregation protocol for sensor networks. Chapter 4 presents our distributed software-based attestation schemes for sensor node compromise detection. Chapter 5 describes experiment results of sensor worm as well as sensor worm defense through software diversity. Chapter 6 concludes the dissertation.

Chapter 2

Related Work

2.1 Secure Data Aggregation in Sensor Networks

Many data aggregation protocols [14, 15, 16, 17, 18, 20, 24] have been proposed, but none of them were designed with security in mind. Until recently very little work has been focusing on secure data aggregation.

After analyzing the possible attacks on the existing aggregation primitives, Wagner [25] proposed a mathematical framework for formally evaluating the security of several resilient aggregation techniques. For example, median is a more robust estimator than mean; truncation and trimming can be used to eliminate possible outliers. This work, however, is not really about data aggregation because it assumes the BS has already collected all the raw data. Also, abnormal data are discarded without further reasoning.

Hu and Evans [26] proposed a secure hop-by-hop data aggregation scheme that works if one node is compromised. They also assume that only leaf nodes in the tree topology sense data whereas the intermediate nodes do not have their own readings. SDAP can tolerate more compromised nodes and allows every node to input its own readings.

Du et al. [27] proposed a mechanism that allows the base station to check the aggregated values submitted by several designated aggregators, based on the endorsements provided by a certain number of witness nodes around the aggregators. Their scheme does not provide per-hop aggregation. Additionally, it is assumed that sensing nodes can be trusted and witness nodes will not collude with the aggregators. However, these conditions may not always hold in practice.

Przydatek et al.[28, 29] presented SIA, a Secure Information Aggregation scheme for sensor networks where a fraction of sensors may be compromised. In their model, the aggregator collects the authenticated raw data from all the sensors in the network. The aggregator then computes an aggregation result over the raw data together with a commitment to the data based on a Merkle-hash tree and then sends them to a trustable remote home server, which later challenges the aggregator to verify the aggregate. They assume that the bandwidth between a remote home server and an aggregator is a bottleneck; therefore, their protocol is for reducing this bandwidth overhead while providing a means to detect with high probability if the aggregator is compromised. The main difference between SIA and SDAP is that SIA does not deal with hierarchical per-hop aggregation because it assumes the raw data are first collected by the aggregator. Since SIA and SDAP work in different stages with different network models (e.g., in SDAP there is no remote home server), in our future work we will investigate the potential of integrating these two.

Later on, Chan et al. [30] proposed a secure hierarchical data aggregation scheme for sensor networks. Their technique is effective to provide provably security guarantee even under general hierarchical aggregator topologies and multiple malicious sensor nodes. Different from ours, they have different aggregation algorithms for different aggregation functions. Roy et al. [31] augmented the normal data aggregation framework such as synopsis diffusion [32] with a set of countermeasures against values falsified by compromised nodes. They consider a ring topology for aggregation whereas ours is an aggregation tree. He et al. [33] devised privacy-preserving data aggregation schemes in sensor network, which is also interesting. We may address privacy issues and combine privacy-preserving techniques in our scheme for our future work.

Several other works [34, 35, 36] also proposed various solutions to prevent false data injection attacks in sensor networks. In their models, it is assumed that a set of sensors are deployed as a cluster in an area of interest. When these sensors reach an agreement on an event, each of them will contribute a MAC over the event report. If a forwarding node shares a MAC key with the endorsing sensors, it will be able to verify the authenticity of the report. It drops the report if the verification fails. In this way, an injected false data packet could be discarded before it reaches the BS, saving the forwarding energy. We notice that although these schemes also address the problem of false data injection, they do not involve data aggregations.

2.2 Software-based Attestation for Node Compromise Detection

Assuming that program and associated data are stored in continuous memory locations, Spinellis [37] used cryptographic hashes computed over two overlapped random memory ranges (they together cover the entire memory space) to check the integrity of installed software in a machine. Shaneck et al. [38] proposed remote software attestation by sending a piece of attestation code to a sensor node. The attestation code is obfuscated to make static analysis of the code difficult for adversaries. Though interesting, it is however unclear how this idea could be implemented in real sensors. In PIV (Program Integrity Verification) [39], a new hash algorithm is generated for each attestation and verification servers are widely distributed to verify hashes received from attested nodes. Different from this work, our schemes only involve regular nodes.

Seshadri et al. [40] proposed SWATT in which a pseudorandom memory traversal algorithm is used to compute the checksum over all the traversed memory cells. According to Coupon Collector's Problem [41], as long as this process iterates for more than $O(m \ln m)$ times where m is the memory size, the entire memory space of interrogated node can be covered with a high probability. Our block-based memory traversal algorithm improves the cell-based algorithm of SWATT in that for each traversal a block of cells rather than one cell are accessed. Hence, less iterations and computational cost are involved in the attested node. More important, based on this algorithm we propose two distributed schemes where an attestation is fulfilled by multiple neighbors instead of a trustable verifier as in SWATT.

Software-based attestation could also be used in computer systems as well as distributed systems. Seshadri et al. [42] took advantage of SWATT-like pseudorandom memory traversal to construct a trusted computing base, based on which hashes of executables were measured to provide untampered code execution. Shi et al. [43] tied data integrity with code integrity and performed a fine-grained service that only attested to the critical code right before the time-of-use, which provided a general solution in establishing a trusted environment for distributed systems.

It was proposed in [44, 45] to incorporate meta-information such as cache/TLB miss in the computation of checksum over memory traversal. Although [46] argued that architecture-dependent side effects such as TLB miss are not sufficient to authenticate software, we do not intend to verify this claim because our focus is to authenticate program running on sensor nodes with microcontrollers that do not support virtual memory. There also exist hardware-assisted code attestation approaches. Sailer et al. [47] presented a TCG-based integrity measurement architecture, in which the measurement and its ordered list are protected by the trusted platform module that is part of the TCG standard [48]. However, we prefer software-based attestation technique because by using them sensors' costs will not be increased. Moreover, Wurster et al. [49] found that in an architecture where code and data reads are separated, checksumming-based tamper resistance is subject to such an attack that checksum is computed over the original program whereas the code that gets executed is actually the malicious version. Similar situation exists in sensor networks where an adversary keeps a copy of the original program in sensor's empty memory which is accessible during checksum computations.

There are a lot of tools for software protection [50, 51], such as watermarking, obfuscation and tamper-proofing. What may be useful to us is the software tamper proofing technique [52, 53] in which a program is crafted in such a way that the attackers cannot maliciously alter the code without being detected. There are three principal ways to detect the tampering: 1) examine the program itself to see whether it is identical to the original one. Message digests can be used here to reduce the overhead; 2) program (result or self) checking to examine the validity of intermediate results produced by the program; 3) encryption can prevent anyone from easily modifying the code unless he or she is able to decrypt it first. These techniques could be combined with ours to defend against sensor node compromises more effectively.

2.3 Sensor Worm Defense

Internet worm attacks [54] as well as buffer overflow vulnerabilities [55] have been extensively studied. Both proactive and reactive strategies are proposed to defend against worm attacks in Internet [56]. Also, Internet worm propagation is modeled and the impact of network topology on the size of the final infected population has been analyzed in [57, 58]. [59] proposes generic techniques for blocking buffer overflow attacks based on some inherent distinctions between exploit code and random data.

The related work we know on sensor worm mainly models the worm propagation. [60] models node compromise spread in wireless sensor networks using epidemic theory and identifies key factors determining potential outbreaks. However, the sensor deployment error and sensor compromise containment strategies are not considered in this work. Some techniques [61, 62] are also designed for sensor memory protection. These work together with [63] improve the defensive capabilities of individual sensors. It is necessary for us to enhance sensor network immunity under worm attack in a systematic way, since the defensive capability of individual sensors is limited. In parallel to our work, [64] illustrates that a mal-packet with only specially crafted data can exploit memory-related vulnerabilities and utilize existing application codes in a sensor to propagate itself without disrupting sensors' functionality. [65] presents a code injection attacks on Harvard-architecture devices, which supports our idea from another angle.

Inspired by diversity, an important source of robustness in biological systems, software diversity in computer systems [66] as well as computer networking [67, 68] bears a lot of attention recently. A variety of randomization techniques [69] have been proposed to enhance the intrusion resistance of computer systems by increasing software complexity without degrading functionalities and performance. Diversification at the network level is achieved by applying different operating systems, critical software components [70] or communication protocols [71] on different machines of the network.

Similar work is conducted in preventing epidemics in the context of computer worms or viruses [72]. In [73], it is stated that selective immunization should be enforced according to the node's degree, i.e., nodes with high degree should be installed different softwares because they are more important in the network connectivity.

Graph coloring (especially vertex coloring) [74], a famous problem in graph theory, ensures that there are no two adjacent nodes sharing the same color. So, its solution is a natural option for making globally optimal decision in software diversity. In the distributed coloring algorithms proposed in [67], the initially random assignment of nodes' colors will cause high communication overhead in the following color adjustment and negotiation with neighbors. Also, the algorithms may not converge to a few colors due to the high density of sensor networks, which in practice may result in a high cost for software implementations. Our sensor worm defense scheme has already considered all these factors as well as the sensor deployment errors, thus is well-tailored for sensor networks.

Chapter 3

Secure Hop-by-Hop Data Aggregation for Sensor Networks

3.1 System Model and Design Goals

In this section, we describe the system model and design goals, followed by the notations used in the description of protocol.

3.1.1 Network Model and Key Setup

Network Model We assume a sensor network consisting of a large number of resource-limited sensor nodes (e.g., Mica2 motes) that are distributed in a certain density. In addition, there exists a powerful BS that connects the sensor network to the outside infrastructure such as the Internet. As in other data aggregation protocols [18, 26], we assume a topological tree rooted at the BS. We call this a topology tree.

Definition 1. *A topology tree T is a tree rooted at the BS that describes the parent-child relationship among neighbors according to the physical topology of the sensor network.*

There are various methods for constructing the topology tree according to different application requirements, one of which is introduced in Section 3.2.2. However, SDAP does not rely on a specific tree construction algorithm as long as there is one. Based on this topology tree, data flow from source nodes to the BS, forming a reversed multicast-like tree, which we call an aggregation tree.

Definition 2. *An aggregation tree T_a is a subtree constructed from the topology tree T ($T_a \subseteq T$), which contains all the data sources concerning the same event as well as those nodes on the way from these data sources to the BS. All the intermediate nodes in the tree act as aggregators, fusing data from downstream nodes.*

An aggregation tree may be identical to the topology tree or just a subtree of it, depending on the communication model. All the trees we mention later refer to the aggregation trees. There are two communication models in the aggregation tree due to two different stimuli of data collection. One is a query-response model where the network is configured to collect data periodically after receiving a query from the BS. The other one is an event-trigger model in which emergent events are reported to the BS from sensing nodes. Our following discussion is best suitable for the query-response model (i.e., global observation of the same event). However, if in the event-trigger model (i.e., local observation of the same event) the aggregation tree is big enough, then hop-by-hop aggregation can also be conducted and our scheme can be easily adapted to this scenario too.

In a real application, a topology tree may be dynamic due to node or link failures. In TinyOS [75], a beaconing message is flooded every 30 seconds to reconstruct the broadcast tree. Clearly, it will be too costly for the BS to keep track of the network topology for every topology change, because each topology discovery may require every node to report its parent/child information to the BS. As such, in our scheme, we assume that the BS does not know the shape of the tree and its distance (in number of hops) from every node although it may want to discover the tree topology occasionally for other purposes.

To concentrate on the security aspects of data aggregation, in the protocol part we do not address the general issues regarding data aggregation, e.g., what sensor applications might benefit from the technique of data aggregation or how to ensure time synchronization among nodes. Also, we assume that there is a reliable transmission mechanism, for example, by using a link-layer hop-by-hop acknowledgment protocol. Thus, the various types of packets in our scheme will not be lost.

Key Setup We assume the BS cannot be compromised and it has a secure mechanism (e.g., μTESLA [76]) to authenticate its broadcast messages to all the nodes in the tree and every node can verify the received broadcast messages. We also assume every sensor node has an individual secret key shared with the BS. Furthermore, there is a unique pairwise key shared between each pair of neighboring nodes [77, 78, 79, 80, 81].

3.1.2 Attack Model

Since a standard authentication primitive, e.g., Message Authentication Code (MAC), can be employed to easily defeat an outsider adversary (who do not have any authentication keys) from launching many attacks, we assume an adversary can compromise a (small) fraction of sensor nodes to obtain the keys as well as reprogram these sensors with attacking code. There may be several potential attacks against a tree-based aggregation protocol. One type of attacks is behavior-based, in which the goal of an attacker is to disrupt the normal operation of the sensor network. For example, once a sensor node in the tree is compromised, it can attack the underlying routing protocol, drop other nodes' reports on purpose, or cause denial of message attacks [82] to deprive other nodes from receiving broadcast messages from the BS.

In this work, however, we are not addressing any of these behavior-based attacks (although we discuss the robustness of our scheme under these attacks in Section 3.3.3.2); instead, we focus on defending against *false data injection attacks* where the goal of an attacker is to make the BS to accept false sensor reports. In many situations, values received by the BS provide a basis for critical decisions; hence, false or biased values may cause catastrophic consequences. For example, when forwarding other sensor nodes' reported values, a compromised node may modify their values; it may also forge some false sensor readings on its own behalf. Because the measurements of the physical world are inherently noisy, if an attacker forges sensor readings that have negligible influence on the final aggregation result, the gain is little. Therefore, we assume that an attacker aims to inject false values that deviate from the true measurements in a noticeable scale. Meanwhile, the attacker does not want to be detected when launching this attack.

In particular, in the context of data aggregation, an aggregate usually contains not only a data value computed for the required aggregation function but also a count value indicating the number of sensor nodes involved in the aggregation operation. Accordingly, there are two kinds of attacks targeting at the data and count in the aggregate, respectively. We refer to these two types of attacks *value changing attack* and *count changing attack*.

Definition 3. *Value changing attack and count changing attack are two special types of* *false data injection attacks during data aggregation, in which a node compromised by the attacker forges a false aggregation value and/or a large count, in order to make this false value account for a large fraction in the computed final aggregation result.*

Next we show through an example why value changing attack and count changing attack are severe attacks. Suppose the BS queries the network for the average temperature and any sensed value must be between 32F and 150F. Let us assume a compromised node receives from its child nodes the aggregated data 100F and the count value 50. If the compromised node cannot modify the received aggregate, i.e., it can only forge a false reading of its own, then the aggregation data may range from 98.7F($\frac{100*50+32}{51}$) $\sim$ 101F($\frac{100*50+150}{51}$), which does not deviate far away from the true average value. However, if it can launch a count changing attack by reporting a bogus large count value, then it may make the average result be any value in the range from 100F to 150F (assuming its own reported temperature is 150F). Similarly, if the compromised node can launch a value changing attack by modifying the data value in its child nodes' aggregate, it can easily make the average result be either 150F or 32F as desired. Obviously, if possible, an attacker can combine these two attacks to affect the final aggregate without being detected.

Note that we do not consider the attack where a compromised node forges a false reading on its own behalf as a value changing attack. The reasons are as follows. First, as shown in the above example, the impact of such an attack is usually limited. Second, such a compromised node is very much like a faulty sensor node. In this case, we have to rely on an outlier detection algorithm or the content-based attestation proposed in Section 3.3.4 to solve this problem.

3.1.3 Design Goal

Our design goal is to defend against the false data injection attacks that trick the BS into accepting false aggregation results, and we will focus on two kinds of false data injection attacks, value changing attacks and count changing attacks. Specifically, our design goal includes:

- *Effectiveness*: The BS should have a high probability to detect the injected false values. Once false values are detected, they will be discarded. This is important to ensure the accuracy of the final aggregation result.

- *Low communication overhead*: The purpose of conducting aggregation is to reduce communication overhead. Clearly, if the overhead of our scheme is equivalent to that of a raw data based scheme, there is no need to employ our scheme.

- *Generality*: It is undesirable to design one scheme for each aggregation function, so our scheme should apply to various aggregation functions, such as MEAN, SUM, COUNT, and so forth.

Notations The following notations are used in the description of the protocol:

- u, v, w, x, y are principals, i.e., the identifiers of sensor nodes.

- $K_{u,v}$ is the pairwise key shared between node u and node v, and K_u is the individual key shared between node u and the BS.

- $m1|m2$ denotes the concatenation of two messages $m1$ and $m2$.

- $E(K,m)$ refers to the encryption of message m using key K.

- $MAC(K,m)$ is the Message Authentication Code (MAC) of message m with key K.

In addition, we will use $u \rightarrow v : M$ to denote a one-hop delivery of message M from node u to a neighbor v and $u \rightarrow\rightarrow v : M$ to denote a delivery that may involve multiple hops.

3.2 The Secure Data Aggregation Protocol

In this section, we present our Secure Data Aggregation Protocol (SDAP). We first give an overview of the protocol and then present the details of the protocol.

3.2.1 Protocol Overview

The design of SDAP is based on the principles of *divide-and-conquer* and *commit-and-attest*. First, SDAP uses a novel probabilistic grouping technique to partition the nodes in a tree topology into multiple logical groups (subtrees) of similar sizes. A commitment-based hop-by-hop aggregation is performed in each group to generate a group aggregate. The BS then identifies the suspicious groups based on the set of group aggregates. Finally, each group under suspect participates in an attestation process to prove the correctness of its group aggregate.

Next, we present the details of the protocol, which includes three phases: *query dissemination*, *data aggregation*, and *attestation*.

3.2.2 Tree Construction and Query Dissemination

For completeness, we first describe a simple tree construction algorithm, which is similar to that in [18]. Initially, the root broadcasts a tree construction beaconing message which includes its own id and its depth to be 0. When a node, say x, receives a broadcast message at its first time from a node y, x assigns its depth to be the depth of y plus one, and its parent to be y. After this, it rebroadcasts the message. This process continues until all nodes have received this message.

After constructing the tree, the BS can disseminate the query message through this tree. Besides the aggregation function that represents the BS's request, a random number is added to the query. This random number is generated by the BS as a grouping seed, which is used for the probabilistic grouping as well as the query identification in the next phase. Specifically, a query packet that the BS broadcasts may be:

$$BS \rightarrow\rightarrow * : F_{agg}, S_g, \tag{3.1}$$

where F_{agg} refers to a specific aggregation function, such as MEAN, SUM, and S_g is the random number generated for each query. Also, we may employ μTESLA [76] to provide global broadcast authentication of the query dissemination.

Above we discussed query dissemination after tree construction to make it independent of the tree construction protocol. In practice, we may combine these two steps into one. The query information can be piggybacked in a beaconing message. On the other hand, the dissemination of a query can help reconstruct the tree topology, thus mitigating the tree partition problem due to node or link failures.

3.2.3 Probabilistic Grouping and Data Aggregation

Through the previous phase, all nodes have identified their parents. In this phase, SDAP randomly groups all the nodes into multiple logical groups and performs aggregation in each group. Probabilistic grouping is conducted through the selection of leader node for each group. During the aggregation, every node makes its commitment by embedding some security information to its aggregate. Grouping and aggregation are closely tied to each other. Actually, they are finished in the same bottom-up procedure.

Next, we first describe how group leaders are selected, and then discuss techniques to add security information into the aggregated data.

3.2.3.1 Group Leader Selection

Probabilistic grouping is finished through the selection of group leader nodes. Here we have a definition upon the leader node.

Definition 4. *A group leader is the topmost node in a group, which completes and submits the aggregate for the group. Leader is changed among nodes and selected probabilistically during the process of data aggregation.*

More specifically, group leaders are selected on-the-fly based on the count values (we will see how count c is calculated in the next subsection) and the grouping seed S_g received in the query dissemination phase. Two functions are used in group leader selection. One is a cryptographically secure pseudorandom function H that uniformly maps the input values (node's id and S_g) into the range of $[0,1)$; the other is a grouping function F_g that takes a positive integer (count) as the input and outputs a real number between $[0,1]$. Each node, say x, decides if it is a leader by checking whether the following inequation is true for it:

$$H(S_g|x) < F_g(c). \tag{3.2}$$

If it is true, node x becomes a leader, and all the nodes in its subtree that have not been grouped yet become members of its group. An example of a grouped tree is shown in Fig. 3.1.

The construction principle is that a node with larger count has a higher probability to become a leader. The grouping function F_g is used to control the probability for a node to be chosen as a group leader and it is preloaded in each sensor. Because the output of H is uniformly distributed between 0 and 1, the probability that it is smaller than $F_g(c)$ actually equals to the value of $F_g(c)$. In our construction, $F_g(c)$ increases with the count value c. Thus, if a node has a larger count value, the probability for it to become a leader is higher. By adjusting the grouping function, ideally, the resulted group sizes are roughly even with a small deviation, which provides the basis for our attestation. A specific grouping function is selected and the grouping result is shown in Section 3.4.1. Apparently, we can consider more factors in the construction of grouping function, e.g., taking into account the residual energy (r_e) of sensor nodes. In this way, the grouping function becomes $F_g(c, r_e)$, which increases with c as well as r_e. To simplify the form of grouping function and our following analysis and evaluation, we now only consider $F_g(c)$. We will consider this option in our future work.

The use of the random number S_g as the grouping seed is mainly for security and load balance. With the random number, the BS can change the leaders among nodes instead of fixing their roles, so that the attacker cannot determine in advance which nodes will be the group leaders for each query. Otherwise, the attacker may target at the group leaders and compromise them. Also, because a different S_g is used each time, every node is assigned into a different group that is formed on the fly. This helps thwart some prearranged colluding attacks by multiple compromised nodes. Another advantage is to balance the resource usage of nodes (e.g., storage, computation, and communication) so as to prolong the overall lifetime of the network.

3.2.3.2 Aggregation Commitment

Before describing the data aggregation process, we first introduce the packet format used in the commitment. Each aggregation packet contains the sender's id, an aggregated data value, and a count value to indicate how many nodes contributing to the aggregated data. In addition, a flag field (one bit) is contained in each packet to show whether the aggregate needs to be processed further by the nodes enroute to the root. Flag value '1' means that no further aggregation is needed, whereas '0' means to be aggregated. This flag field is initialized to '0'. After a group leader finishes the aggregation for the group, this flag field is set to '1'. Other nodes on the path to the root just forward those packets with flag '1'.

The pairwise key shared between each pair of parent and child is used to encrypt the aggregate. This encryption in practice provides not only confidentiality but also authentication. This is because the format of content is known to everyone in the network and the value of each item should fall in a certain range. Thus, using encryption saves the

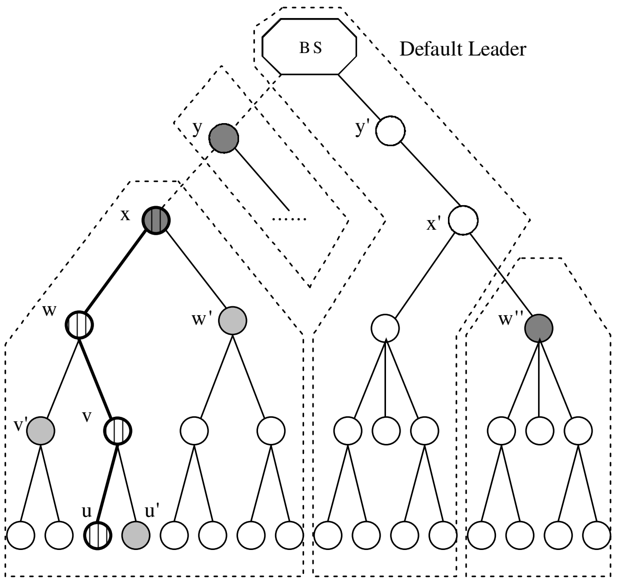


Figure 3.1. An example of a grouped aggregation tree. The nodes x, y and w” with the dark-gray color are leader nodes, and the nodes included in the dashed line are corresponding group members. The BS as the root is a default leader.

bandwidth that will otherwise be used for an additional MAC. In addition, a MAC computed using the key shared with the BS is also attached at the end of each packet, which provides authentication to the BS. Next, we present details of the aggregation process.

Leaf node aggregation: Different from query dissemination, data aggregation starts from the leaf nodes in the aggregation tree towards the BS. Since a leaf node does not need to do aggregation, it just sends its id, data and count value to its parent (it also keeps a local copy of the packet). The packet that a leaf node u sends to its parent v is as follows:

$$\begin{aligned} u \rightarrow v \quad : \quad & u, 0, E(K_{u,v}, 1|R_u|S_g)|MAC_u \\ & MAC_u = MAC(K_u, 0|1|u|R_u|S_g), \end{aligned} \tag{3.3}$$

where ‘0’ is the aggregation flag, ‘1’ is the count value, R_u is the reading of node u, and MAC_u is the MAC value computed by node u with its individual key shared with the BS. Here S_g is included to identify the query and to prevent replay attacks.

Intermediate node aggregation: When an intermediate node receives an aggregate from its child node, it first checks the flag. If the flag is ‘0’, it keeps a local copy of the aggregates (until the attestation phase is done) and performs further aggregation; otherwise, the node directly forwards the packet to its parent node.

In detail, for a report with flag ‘0’ received from a child node, a node first decrypts the data using its pairwise key shared with this child node. It also performs some simple checking on the validity of the count, R_u (if within a certain range), and S_g (if the same as the one received in the query dissemination phase). If the aggregate packet does not pass this checking, it will discard the packet directly. Otherwise, it will further aggregate its own reading with all the aggregates carrying flag ‘0’ received from its child nodes. A new count is also calculated as the sum of the count values in the received aggregates with flag ‘0’ plus one (considering its own reading). The node checks if it is a group leader based on the inequation (3.2) using its own id and the new count as the inputs. The node then encrypts the new count value and aggregation data using the pairwise key shared with its own parent.

As shown in Fig. 3.1, w is the parent of v. Since here node v is not a leader, the packet that v sends to w is as follows:

$$\begin{aligned} v \rightarrow w : v, 0, E(K_{v,w}, 3|Agg_v|S_g)|MAC_v \\ Agg_v = F_{agg}(R_v, R_u, R_{u'}) \\ MAC_v = MAC(K_v, 0|3|v|Agg_v|MAC_u \oplus MAC_{u'}|S_g), \end{aligned} \tag{3.4}$$

where ‘3’ is the count value summed over the count value of u, u' and its own contribution, Agg_v is the aggregation value of node v and MAC_v is the MAC value computed by node v. Note that the MAC of an intermediate node is calculated over not only the previous fields but also the XOR of the MACs from its children. In this way, a MAC value is also computed in a hop-by-hop fashion, thus it can represent the authentication information of all the nodes contributing to this aggregation data. In addition, here we use a general aggregation function F_{agg} instead of a specific one such as MEAN, SUM or MEDIAN. Obviously, our protocol is applicable to multiple aggregation applications.

Leader node aggregation: Now suppose that an intermediate node has processed the aggregates from its child nodes and it finds out that it is a group leader based on the inequation (3.2). Like a regular intermediate node, it also computes a new aggregate, keeps local copies of those packets with flag '0', and appends a corresponding MAC using its individual key. Unlike a regular intermediate node, it sets the flag to '1' in its aggregation packet and encrypts the new aggregate with its individual key shared with the BS. Since in Fig. 3.1 node x is a group leader, the packet it sends upward is as follows:

$$\begin{aligned} x \longrightarrow\!\!\!\!\longrightarrow BS : x, 1, E(K_x, 15|Agg_x|S_g)|MAC_x \\ Agg_x = F_{agg}(R_x, Agg_w, Agg_{w'}) \\ MAC_x = MAC(K_x, 1|15|x|Agg_x|MAC_w \oplus MAC_{w'}|S_g), \end{aligned} \tag{3.5}$$

where Agg_x is the aggregation result of the group and MAC_x is the MAC value computed by the leader node x. Note that the leader node needs to set the flag field to '1', so that data from this group will not be aggregated any more. That is, in Fig. 3.1, when node y receives a packet from x, it forwards the packet towards the BS without any further aggregation and it does not add the count value of x to its own. In an extreme case when all the children of a node are group leaders, this node will only contribute the count value 1 to its parent node, similar to a leaf node. As such, we can see that the importance of a higher level node is reduced as we have desired.

Based on the above aggregation rule, the aggregated data and the corresponding MACs are transmitted to the BS. There may be some nodes left without group membership. In this case, the BS is the default group leader for them.

After the BS receives the aggregates from all groups, it decrypts and saves them in the following format: $(x, c_x, Agg_x, MAC_x, S_g)$, where x is the leader node's id, c_x is the group count, Agg_x is the group aggregation value, MAC_x is the authentication tag computed by the group leader, and S_g is used to identify the query. Note that all the groups are logical groups; no physical partition of the topology tree is involved.

We notice that although the spirit of this technique is similar to Merkle hash tree [83], there are several noticeable differences. First, Merkle hash tree is a data structure, not based on a real topology tree; second, Merkle hash tree is a binary tree whereas in our case the topology tree is arbitrary. Third, in Merkle hash tree only leaves are measurements, all others are hash values. Fourth, the MAC in our scheme is computed over more information.

3.2.3.3 Tracking the Forwarding Path

When a sensor node receives an aggregation packet with flag ‘1’, it records into its forwarding table the following information: S_g, the id of the group leader, the incoming link (i.e., from which node it receives the packet). In this way, when the BS sends out an attestation request later regarding this group, the node knows where to forward this request. This can save some message overhead because otherwise the BS has to flood the request. For example, as shown in Fig. 3.1, when node y receives the packet from x, it forwards the packet to the BS and adds x to its forwarding table. In the future, if the BS wants to attest the group of x, it sends the attestation message directly to its child y. Since x is in y’s forwarding table, y also forwards this attestation message directly to x.

The above solution works fine in most cases. If the aggregation tree is large and there are a large quantity of groups, techniques such as Bloom filters [84] may be used to construct the forwarding table, in order to reduce the storage overhead. We notice that the size of forwarding table does not necessarily keep increasing because this forwarding table is updated for each query.

3.2.4 Verification and Attestation

3.2.4.1 Verifying the Aggregation Messages

After the BS has received the aggregation messages from the group leaders, it needs to verify the authenticity of the aggregated value in each aggregation message. This includes verifying the content of the packet and the authenticity of the leader. First, based on the group leader id, say x, in the message, the BS can find out the individual key of the node (K_x) by which it decrypts the data and gets the information $(x, c_x, Agg_x, MAC_x, S_g)$. The authenticity of the message is provided because the content format is known to the BS and the value of each item should fall in certain range. Second, the BS verifies the legitimacy of the claimed group leader x by checking whether $H(S_g|x) < F_g(c_x)$ because the BS knows H, F_g and the grouping seed S_g. If this does not hold or any item in the packet is invalid, the BS simply drops the packet.

3.2.4.2 Determining Suspicious Groups for Attestation

After the above verification, the BS believes that the aggregate is truly from a legitimate leader x. However, the BS cannot tell whether c_x or Agg_x has been modified because a compromised group leader or member may have modified the data, which can influence the final aggregation result at the BS. Note that authentication cannot solve this insider attack because a compromised node has the valid keys.

We expect the attacker to forge an aggregated data that have a non-trivial influence on the final result; otherwise the attacker could not gain much. As a result, a false aggregate should exhibit certain abnormality. On the other hand, we cannot simply treat all abnormal sensing data as outliers and discard them, since they may indeed reflect the real environment. In many cases we are more interested in abnormal data than in normal ones. For example, for sensors deployed to detect fire events, abnormally high temperature is our special concern. With these in mind, we have to verify the abnormal aggregates before accepting or rejecting them. In other words, the BS should attest the groups with suspicious large count values or doubtful aggregation data. In detail, we use Grubbs' test [85] to identify abnormal groups.

Definition 5. *Grubbs' test is a hypothesis test for detecting data outliers. Given a dataset $\Gamma = \{\chi_1, \chi_2, \cdots, \chi_n\}$, suppose that μ and s are the sample mean and standard deviation of all the data, then the data $\chi_i (1 \leq i \leq n)$ with the largest sample statistic*

$$Z = \frac{|\chi_i - \mu|}{s} \tag{3.6}$$

is an outlier if this statistic falls beyond the range defined by the critical values or this statistic's corresponding p-value is smaller than the predefined significance level α.

In Grubbs' test, H_0 means that there are no outliers in the data set and H_1 means that there is at least one outlier in the data set. More specifically, it first computes the sample statistic for each datum χ in the set by $\frac{|\chi - \mu|}{s}$. The result represents the datum's absolute deviation from the sample mean in units of the sample standard deviation. Based on this, each time the datum with the maximum statistic is picked up and there are two equivalent methods to decide whether H_0 should be accepted or not. One is to check whether the sample statistic falls in the non-rejection range defined by the critical values. The other one is to compare the p-value computed based on the sample statistic with the predefined significance level α (equals to 0.05 typically), where the p-value is the observed level of significance, defined as the probability that the sample statistic is equal to or more than the value obtained from the sample data given that H_0 is true. The smaller the p-value is, the farther the sample statistic deviates from the sample mean. When the p-value is smaller than α, H_0 is rejected and this datum is considered to be an outlier.

We make several extensions so that our Grubbs' test based algorithm can detect *multiple* outliers from *bivariate* data in our setting. First, since Grubbs' test detects one outlier at a time, we expunge the detected outlier from the dataset and iterate the test over the remaining data until no outliers can be found. In this way we can detect multiple outliers. Second, Grubbs' test is normally used for univariate data set, but we will need to detect outliers from bivariate data (i.e., counts and aggregation data). Because counts and data are independent variables, we set the p-value as the product of p-values of these two variables to prevent an attacker from either forging a large count or an extreme value.

Definition 6. *The extended Grubbs' test is designed to detect multiple outliers from a bivariate dataset. Given a dataset $\Gamma' = \{(c_1,\chi_1),(c_1,\chi_2),\cdots,(c_n,\chi_n)\}$, the tuple $(c_i,\chi_i)(1 \le i \le n)$ with p-values P_c for count and P_χ for data is an outlier, if the product*

$$P_c \cdot P_\chi < \alpha. \tag{3.7}$$

Normally, a datum of one variable is considered to be an outlier when its p-value is smaller than 0.05. In our bivariate case, even when each separate one is less like an outlier, we may still consider the combination as an outlier. For instance, for a count and data value pair reported by a group, suppose we get the p-value 0.2 for the count and 0.24 for the data. None of them is smaller than 0.05; however, their product is $0.048 < 0.05$. Thus, we identify this group as a suspicious group. In another example, to avoid detection an attacker may report a very small count value but extreme data. In this case, we may get the p-value of 1.0 for the count, but as long as p-value for the data is less than 0.05, this group will still be selected for attestation.

We have seen that an attacker does not have much motivation for forging a small count. As such, we are only interested in large count values. That is, for count values, the BS will run the one-sided Grubbs' test for computing the p-values. For data values, we may consider a two-sided test for some aggregation applications, such as MEAN. For other operations, it depends on the specific aggregation functions (e.g., MIN/MAX). We may also resort to the content-based attestation introduced in Section 3.3.4 to deal with these data outliers. A formal description of the outlier detection algorithm is shown in Algorithm 1. Note that in this algorithm for simplicity we only consider the related items in each attested tuple - the leader's id, the count and the data.

Algorithm 1: Outlier Detection Algorithm

Input: a set Γ' (with size equal to the total number of groups) of tuples (x, c_x, Agg_x), where x is group leader's id, c_x is group count and Agg_x is group aggregated value;

Output: a set L of leader ids of groups with outliers (which is initialized to $\emptyset$);

Procedure:

```
1:  loop
2:      compute mean $\mu_c$ and standard deviation $s_c$ for all the counts in set $\Gamma'$;
3:      compute mean $\mu_v$ and standard deviation $s_v$ for all the values in set $\Gamma'$;
4:      find the maximum count value $c_x$ in set $\Gamma'$;
5:      compute statistic $Z_c = \frac{c_x - \mu_c}{s_c}$ for count $c_x$;
6:      compute $p$-value $P_c$ based on the statistic $Z_c$;
7:      compute statistic $Z_v = \frac{|Agg_x - \mu_v|}{s_v}$ for the corresponding value $Agg_x$;
8:      compute $p$-value $P_v$ based on the statistic $Z_v$;
9:      if $(P_c * P_v) < \alpha$ then
10:         $\Gamma' = \Gamma' - \{(x, c_x, Agg_x)\}$;
11:         $L = L \cup \{x\}$;
12:     else
13:         break;
14:     end if
15: end loop
16: return $L$;
```

3.2.4.3 Generating and Forwarding Attestation Requests

After the BS has decided which group(s) to attest, it will need to decide how to attest the group. The challenge is due to the fact that the BS only knows the group leader id - it does not know what the other nodes are and how they form the group subtree. In this case, how can it prevent the group leader from making up the group topology and attestation results? Next, we show a simple while effective way to address this challenge.

The BS broadcasts an attestation message including the leader's id of the attested group, a random number S_a, and the grouping seed S_g. S_a is used as the seed for the attestation and it will determine a unique and verifiable attestation path as shown shortly. S_g is included for identifying the query. Let x be the leader's id of the attested group. Then, the attestation request from the BS is

$$BS \rightarrow\rightarrow x : x, S_a, S_g. \tag{3.8}$$

Again, we can use μTESLA to provide broadcast authentication.

Suppose that y is the id of the node from which the BS received the group aggregate (BS also maintains a forwarding table), then the BS will first send this request to node y. The attestation request from the BS will be disseminated downward in the tree. Every node receiving this request searches its forwarding table using the leader's id as the index to get the next-hop node's id. It then forwards the request to that next-hop node.

3.2.4.4 Group Attestation

During a group attestation process, a physical attestation path between the group leader and a leaf node (in the group subtree) is dynamically formed. For ease of presentation, we first describe a construction aiming at count changing attack, then we discuss a variant to deal with the combination of count changing attack and value changing attack.

Definition 7. *An attestation path is a path in the attested group formed from the leader node to a leaf node in the logical subtree, one node in each level.*

Specifically, after the leader node receives the attestation request from the BS, it decides the next hop on the attestation path as follows. Suppose the attested leader node x have d children with counts c_1, c_2, $\cdots$, c_d in the logical group (not all the child nodes in the physical tree because some child nodes may become group leaders themselves). Node x first adds up all the count values of its child nodes, i.e., calculates $\sum_{k=1}^{d} c_k$. This can be done since the parent node stored all the count values from the children in the aggregation phase. Then, it calculates the value $\sum_{k=1}^{d} c_k \cdot H(S_a|id)$, with the attestation seed S_a and its own id, based on the pseudorandom function H. The parent picks up

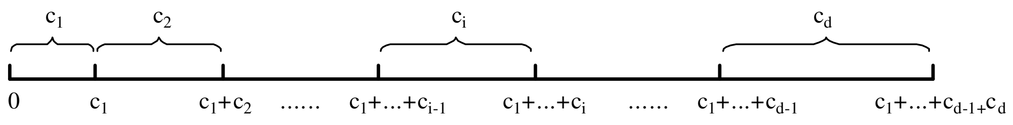


Figure 3.2. Count intervals for a parent with *d* children.

the i^{th} child for attestation if this calculated value falls in the i^{th} child's count interval $[\sum_{k=1}^{i-1} c_k, \sum_{k=1}^{i} c_k)$, as shown in Fig. 3.2. The selected child runs the same process to select one of its own children to form the path until a leaf node is reached. Recursively, an attestation path between the leader and a leaf node in the logical group subtree is formed. The attestation path selection algorithm is described in Algorithm 2[1].

The construction principle is that a node with larger counts and/or abnormal data has a higher possibility to be attested. Next we prove that this construction ensures that the probability for a child node to be selected on the path is proportional to its count value reported in the aggregation phase. Thus, a child with a larger count will be attested with a higher probability.

Lemma 3.2.1. *Suppose a parent has d children with counts c_1, c_2, ... , c_d respectively in a logical group. The probability that this parent selects the ith child with count c_i for attestation is*

$$P(i,d) = \frac{c_i}{\sum_{k=1}^{d} c_k}. \tag{3.9}$$

Proof. Because the value of $H(S_a|id)$ is uniformly distributed in the range $[0,1)$, it can be treated as a random variable X that follows a uniform distribution with the pdf (probability density function)

$$f_X(x) = \begin{cases} 1, & \text{if } 0 \le x < 1 \\ 0, & \text{otherwise.} \end{cases}$$

Thus, the value of $\sum_{k=1}^{d} c_k \cdot H(S_a|id)$ can be treated as another random variable Y uniformly distributed in $[0, \sum_{k=1}^{d} c_k)$, whose pdf is given by

$$f_Y(y) = \begin{cases} \frac{1}{\sum_{k=1}^{d} c_k}, & \text{if } 0 \le y < \sum_{k=1}^{d} c_k \\ 0, & \text{otherwise.} \end{cases}$$

[1]Instead of only using count values, a variant of this algorithm is to use some function (e.g., multiplication) of count and data as the criteria to detect compromised nodes that forge small count but extreme data.

From Fig. 3.2, we can see that the probability for a parent to select the ith child equals to

$$P(i,d) = \int_{\sum_{k=1}^{i-1} c_k}^{\sum_{k=1}^{i} c_k} \frac{1}{\sum_{k=1}^{d} c_k} dy = \frac{c_i}{\sum_{k=1}^{d} c_k},$$

which is proportional to c_i, the count value of this child. □

Algorithm 2: Attestation Path Selection Algorithm

Input: attested group leader's id x, attestation seed S_a, and pseudorandom function H;

Output: a set ρ of attested nodes' ids in the attestation path, initialized to $\{x\}$;

Procedure:

```
1:  parent = x;
2:  loop
3:    τ = ids of all the children of parent;
4:    d = size of τ;
5:    if $d == 0$ then
6:      break;
7:    else
8:      compute $h = H(S_a|\text{parent})$;
9:      compute $sum = \sum_{k=1}^{d} c_k$, for counts of all the d children;
10:     compute $\rho = sum \cdot h$;
11:     if $\rho \in [\sum_{k=1}^{i-1} c_k, \sum_{k=1}^{i} c_k)$ then
12:       $\rho = \rho \cup \{i^{th}$ child of parent$\}$;
13:       parent = $i^{th}$ child of parent;
14:     end if;
15:   end if;
16: end loop
17: return ρ;
```

Each node on the path sends back its count and its own reading. Their sibling nodes (except leaf nodes which only need send counts and readings) send back counts, aggregation data and MACs. Except the leader node, all nodes also attach their parents' ids while sending back corresponding values, so that the BS gets to know the relationship among nodes in the subtree. Similarly, the attestation seed S_a is appended to identify the attestation, and the use of encryption here also provides authentication since the format of packet is publicly known.

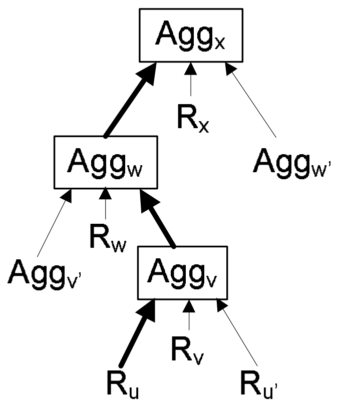
Fig. 3.1 illustrates one example. Assume that the BS wants to attest the group with leader node *x* and according to our previously introduced mechanism the chosen attestation path in this group is x—w—v—u. Then, the messages sent back to the BS from this group are:

$$\begin{aligned}
x \rightarrow\rightarrow BS &: x, E(K_x, x|15|R_x|S_a) \\
w \rightarrow\rightarrow BS &: w, E(K_w, w|x|7|R_w|S_a) \\
w' \rightarrow\rightarrow BS &: w', E(K_{w'}, w'|x|7|Agg_{w'}|MAC_{w'}|S_a) \\
v \rightarrow\rightarrow BS &: v, E(K_v, v|w|3|R_v|S_a) \\
v' \rightarrow\rightarrow BS &: v', E(K_{v'}, v'|w|3|Agg_{v'}|MAC_{v'}|S_a) \\
u \rightarrow\rightarrow BS &: u, E(K_u, u|v|1|R_u|S_a) \\
u' \rightarrow\rightarrow BS &: u', E(K_{u'}, u'|v|1|R_{u'}|S_a).
\end{aligned}$$

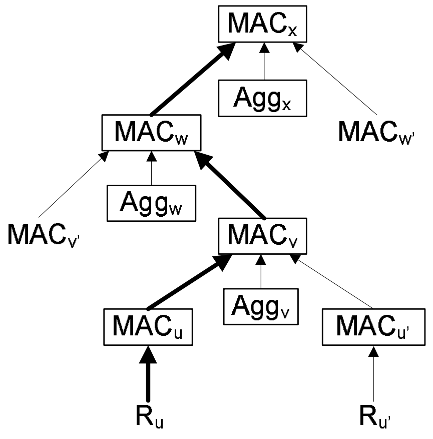
These messages are encrypted by the individual keys of corresponding sensor nodes. Note that from the analysis and evaluation in Section 3.4 we can see that the BS can almost accurately locate the abnormal groups and only nodes around the attestation path need send back values for attestation, so the message overhead is not a big issue here.

After the BS decrypts the received data and reconstructs the subtree topology according to the parent id information, it first verifies whether *w*, *v* and *u* are really the nodes on the attestation path based on S_a, these nodes' ids and counts. Then, it verifies whether the count value of every node is the sum of its children's counts plus one. If this check succeeds, it aggregates the data by itself and reconstructs the aggregation result of this group, Agg_x, to examine whether nodes on the path have forged the aggregation results in the aggregation phase:

$$\begin{aligned}
Agg_v &= F_{agg}(R_v, R_u, R_{u'}) \\
Agg_w &= F_{agg}(R_w, Agg_v, Agg_{v'}) \\
Agg_x &= F_{agg}(R_x, Agg_w, Agg_{w'}).
\end{aligned}$$



(a) Reconstruction of aggregates.



(b) Reconstruction of MACs.

Figure 3.3. The reconstruction operation flow in BS. Values in the black frames are those recomputed by the BS. Others are those sent back to the BS. Thick arrows represent the attestation path.

It can also reconstruct MAC_x using these data:

$$\begin{aligned}
MAC_u &= MAC(K_u, 0|1|u|R_u|S_g) \\
MAC_{u'} &= MAC(K_{u'}, 0|1|u'|R_{u'}|S_g) \\
MAC_v &= MAC(K_v, 0|3|v|Agg_v|MAC_u \oplus MAC_{u'}|S_g) \\
MAC_w &= MAC(K_w, 0|7|w|Agg_w|MAC_v \oplus MAC_{v'}|S_g) \\
MAC_x &= MAC(K_x, 1|15|x|Agg_x|MAC_w \oplus MAC_{w'}|S_g).
\end{aligned}$$

The reconstruction operation flow in the BS is shown in the Fig. 3.3. Note that here some of the reconstructions may not be necessary. For example, if the BS compares the reconstructed aggregation result with the previously received one and finds that they are not consistent, then there is no need for the BS to recompute the MAC value. Only when both the aggregation result and the MAC value match the previously received commitment, the BS accepts the data and use them to compute the final aggregation result. Otherwise, the BS may find out the compromised nodes in this group and recompute the aggregate without values from them or just simply discard aggregate from this group.

Attesting Multiple Paths The above technique is for one path attestation. To improve the detection capability, we may select multiple attestation paths. One straightforward solution is to send multiple attestation seeds, each of which is used to determine one path. A more efficient way is as follows. In its attestation request the BS adds n_p, the number of paths to be attested. When a group node selects its child nodes, it evaluates $H(S_a|id|k)$, where $k = 1, 2, ..., n_p$, each determining an attested node in one of the attestation paths. Clearly, these multiple paths may overlap; if a node appears in multiple paths, it only needs to send back one report. Thus, the cost of attestation is sublinear with respect to the number of attestation paths. The improvement of detection rate under the condition of multiple attestation paths is analyzed in Section 3.3.2.1.

Dealing with Value Changing Attack Because a compromised node may forge small count but extreme data to avoid the attestation, for detecting value changing attack or a combination of count and value changing attacks, it is not effective to determine an attestation path only based on counts. Instead, we need take into account the data value by using some function of count and data as the criteria to select a path. For example, we may simply replace the counts in the above path selection rule with $c \cdot |R - R_{normal}|$, where c is the count, R is the corresponding data value, and R_{normal} is the normal data value (e.g., the normal indoor temperature). We call this *cr*-product. Of course, other appropriate functions can also be applied.

3.3 Security Analysis

This section discusses how SDAP defends against several attacks, and also presents the analytical results about its detection capability.

These notations are used in the following analysis and evaluation:

- $T_a(n, d, h)$ is used to model the aggregation tree, where n means the total number of nodes, d is the degree of the tree (e.g., $d = 2$ represents a binary tree), and h is the height of the tree;
- $g (1 \leq g \leq n)$ is the average group size.
- n_a is the number of attested groups;
- n_p is the number of attestation paths in the attested group (for ease of expression, we assume the number of attestation path in each attested group is the same);

3.3.1 General Security Analysis

Our commit-and-attest technique aims to ensure that once a group has committed its aggregation result, if being attested later, every involved node in the group has to report its original aggregate. Otherwise, the group attestation process will detect the attack by finding the inconsistency between the committed aggregate/MAC and the reconstructed aggregate/MAC. This technique is secure as long as we use a cryptographically secure MAC function such as HMAC, although we will not give a rigorous proof here. Readers could refer to Merkle hash tree [83] on this.

Due to our probabilistic grouping scheme, an attacker cannot selectively compromise nodes to ensure his optimal attack strategy: for example, making multiple of the compromised nodes or no more than one to appear in the same group. Because grouping is a dynamic process, a node cannot know in advance whether it will become a group leader or which group it will belong to. Also, because the aggregates from all the groups are encrypted, a compromised node cannot know if its own aggregate will become an outlier which could possibly be detected by Grubbs' test. Further, a node does not know whether it will be selected on the attestation path because the attestation path is also dynamically selected in a probabilistic fashion.

3.3.2 Detection Rate Analysis

We discuss the effectiveness of SDAP to detect the count changing attack and the value changing attack in this section. For clarification, when we refer to a compromised node, we always assume that this node makes either count changing attack or value changing attack, i.e., a compromised node following our protocol has no difference with a normal one.

To detect either of these two attacks or a combination of them, the first step is to identify suspicious groups. In the previous section, we proposed to use the extension of Grubbs' test for this purpose (certainly other appropriate outlier detection algorithms may also be applied), thus the probability of an abnormal group being selected is determined by the power of Grubbs' test. The second step is to locate compromised nodes within groups based on group attestation, given that the attacked group has been identified. Accordingly, in the following, we first analyze the false positive rate of the Grubbs' test in our scheme, then we derive the detection rate of our group attestation.

Lemma 3.3.1. *The false positive rate F_p of the Grubbs' test in our scheme is 0, i.e.,*

$$F_p = 0. \tag{3.10}$$

Proof. The Grubbs' test may identify an attack-free group as a suspicious one. However, from security viewpoint this is not an issue because this group will pass the group attestation anyway. From performance point of view, this is also fine because the number of attested groups are very low if there are no attacks, according to our simulations in Section 3.4.2 (the detection capability of Grubbs' test when there are attacks will also be shown there). Within groups, if honest nodes are selected on the attestation path, it does not matter, either. Actually, the BS could detect other compromised nodes (if there are any), based on the authenticated data from these honest nodes. Therefore, the Grubbs' test in our scheme has zero false positive rate. □

Next, we will focus on analyzing the detection rate of our scheme through group attestation, i.e., the possibility that compromised nodes are selected on one of the attestation paths. Similar to section 3.2.4.4, we first analyze the detection rate upon count changing attack, then we discuss the situation to deal with the combination of count changing attack and value changing attack.

3.3.2.1 Count Changing Attack Detection

For ease of presentation, let us first consider the case that there is only one compromised node in an attested group, although multiple compromised nodes, if they are in a logical group, may collude in launching attacks. A count changing attack will be detected if the compromised node is selected on one of the attestation paths.

Since an attestation path always starts from the leader node, if the leader node of a group made the count changing attack, the detection rate is 100%. Next we analyze the probability that a regular node is selected on the attestation path. We use the notation $c_{j,i}$ to denote the count value of a node in depth j (i.e., its distance from the leader node is j), which is also the ith child of its parent, as shown in Fig. 3.4. In the notation, $0 \leq j \leq h$ where h is the height of the group subtree and $1 \leq i \leq d_j$ where in depth j there are totally d_j children for selection. Therefore, the count of the leader node is denoted by $c_{0,1}$. Counts of leader's children are denoted by $c_{1,1}$ to c_{1,d_1} from left to right. In depth j, the attested node is denoted by u_j.

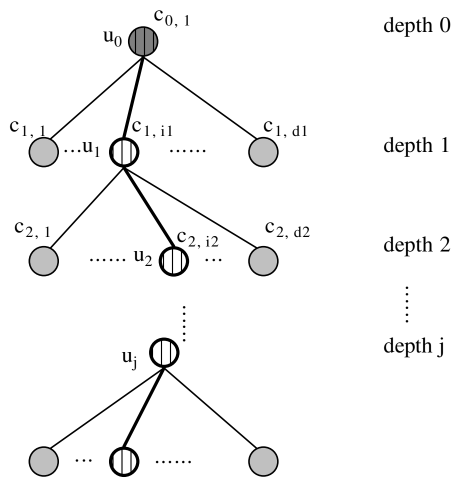


Figure 3.4. Choosing one attestation path in a group based on counts.

Lemma 3.3.2. *The detection rate to the attack launched by a compromised node u_j in depth j, i.e., the probability for node u_j to be selected on the attestation path, is*

$$D_r(j) = \prod_{l=1}^{j} P(i_l, d_l). \tag{3.11}$$

Proof. When the sibling of node u_j's parent is selected, the probability to choose node u_j is 0; therefore, the probability of choosing node u_j with the parent u_{j-1} equals to the probability of choosing u_{j-1} multiplied by the probability of choosing u_j under the condition that we have selected the parent u_{j-1}.

According to Lemma 3.2.1, the probability for a child u_1 in depth 1 with count c_{1,i_1} to be selected on the path is $P(i_1, d_1)$, because the probability for us to choose the leader node in depth 0 is 100%. Hence, we have that the detection rate $D_r(j)$, i.e., the probability for node u_j to be selected on the attestation path, equals to

$$\begin{aligned}
P(U_j) &= P(U_j|U_{j-1}) \cdot P(U_{j-1}) \\
&= P(U_j|U_{j-1}) \cdot P(U_{j-1}|U_{j-2}) \cdots P(U_1|U_0)P(U_0) \\
&= P(i_j, d_j) \cdots P(i_1, d_1) \cdot 1 \\
&= \textstyle\prod_{l=1}^{j} P(i_l, d_l),
\end{aligned}$$

where U_j refers to the event that u_j is selected on the attestation path. □

Next, we show the detection rate to the attack if we select multiple paths for attestation.

Lemma 3.3.3. *Suppose we choose n_p independent attestation paths, the detection rate of the count changing attack by a compromised node u_j in depth j becomes*

$$D_r(j,n_p) = 1 - [1 - D_r(j)]^{n_p}. \tag{3.12}$$

Proof. We can treat the selection of n_p attestation paths as n_p independent events, because each time the attestation path is randomly selected. The probability of detecting a compromised node equals to the probability of selecting this node at least once in the n_p events. Suppose A refers to the event that node u_j is selected on the attestation path at least once in the n_p events. On the contrary, $\overline{A}$ means that node u_j is never selected on the attestation path in the n_p events. Based on Lemma 3.3.2, we have

$$P(\overline{A}) \quad = \quad [1 - \textstyle\prod_{l=1}^{j} P(i_l, d_l)]^{n_p} = [1 - D_r(j)]^{n_p}.$$

Therefore, the detection rate $D_r(j,n_p)$ equals to

$$P(A) = 1 - P(\overline{A}) = 1 - [1 - D_r(j)]^{n_p}.$$

□

Since $D_r(j,n_p)$ in equation (3.12) is a function increasing with the value of n_p, we can see that if we perform the attestation for multiple times (i.e., $n_p > 1$), then the detection rate will be higher, which means that we have more chances to detect the attack. Assume the node v in the group with leader x in Fig. 3.1 is an attacking node, our detection rate of the attack through multiple paths is shown in Fig. 3.5. For instance, if node v changes its count value from 3 to 12. Accordingly, the count values of node w and x becomes 16 and 24, respectively. If we choose only one attestation path, the detection rate of this attack is 55.7%, but if we choose four attestation paths, the detection rate is increased to 96.1%.

Finally, we consider the case when multiple compromised nodes are in the attested logical group. The detection rate is subject to the distribution of these compromised nodes, for example, whether more than one compromised nodes locate on a same path, no two nodes locate on a same path, or a hybrid of these two scenarios. If multiple com-

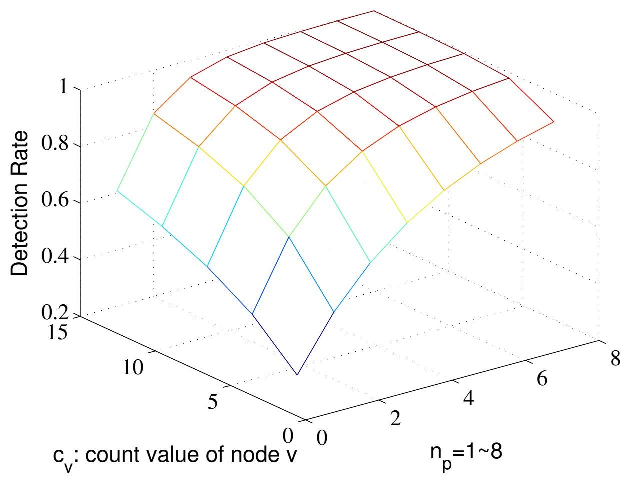


Figure 3.5. Detection rate to the count changing attack launched by node *v* in the group with leader node *x*. For c_v=3, 6, 9, 12, 15, respectively. The number of attestation paths equals to $1 \sim 8$.

promised nodes are on the same attestation path, then the detection rate of the attack equals to the probability that the highest-level compromised node is selected; when all these compromised nodes are on different attestation paths, we can detect the attack as long as we can choose any of these paths, so the detection rate is the sum of the probability for choosing each one. For the hybrid case, the detection rate can be computed as the sum of the probability that we attest the highest-level compromised node in each of these paths. From the above analysis, we can see that if there are more than one compromised nodes in the same attested group, the detection rate becomes higher unless these nodes are all on the same path.

3.3.2.2 Value Changing Attack Detection

Similar to a count changing attack, a value changing attack or the combination of count and value changing attacks could be detected when the attacking node is selected on the attestation path. In this case, we should use the attestation path selection criterion introduced in Section 3.2.4.4 (i.e., replacing count c with the cr-product: $c \cdot |R - R_{normal}|$), trying to pick up all nodes with either abnormally large count or extreme data. Then, the probability that a child is chosen on the attestation path (or attested) is proportional to the cr-product instead of only count. Correspondingly, in lemmas 3.3.2 and 3.3.3, we could simply replace all count c with the cr-product, to obtain correct detection rate upon the combination of count and value changing attacks.

3.3.3 Detection of Other Attacks

Next, we discuss the detection of other attacks, such as the event suppression attack and several outsider attacks.

3.3.3.1 Event Suppression Attack

A data aggregation protocol is vulnerable to a potential event suppression attack, where a compromised node changes its aggregated value corresponding to a real abnormal event to a normal value, thus the BS may not notice the real event, because the extended Grubbs' test might not detect an attacked group reporting a normal value.

Nevertheless, our probabilistic grouping technique can greatly mitigate this attack, because (1) the role of an attack node in an aggregation subtree (group) is not fixed. Since group leader is randomly selected, every node might become a leaf node in an aggregation subtree; (2) some other nodes belonging to other aggregation subtrees may detect the real event and report it to the BS; (3) because all the group reports are encrypted, the attacker might not know what is normal for sure. If all the other groups report the real event, then this group becomes abnormal; (4) to change the value from abnormal to normal, the attacker must also forge a large count to be effective (e.g., in the case of MEAN, SUM), which could be caught by our bivariate Grubbs' test.

3.3.3.2 Outsider Attacks

The previously discussed false data injection attacks, including count changing attack and value changing attack, are launched by insider compromised nodes who have legitimate keys. We might also check the robustness of our protocol under outsider attacks, such as eavesdropping, replaying and dropping packets through jamming.

For eavesdropping, without authorized keys, the messages could be obtained by the attacker are at most the node id and flag. From node id, the attacker can only get some general information about the network, such as the network size. No sensitive information will be exposed. Although the flag field may leak the information about leaders, the aggregation of current round has been finished when the attacker knows this and different leaders are chosen in the next round through probabilistic grouping, so the benefit that could be gained by the attacker through compromising these leader nodes is limited.

As for replaying, we encrypt messages by corresponding individual or pairwise keys and attach random numbers in both aggregation and attestation. Therefore, simply replaying packets by an outsider is ineffective.

The influence of jamming attack depends on what packets are dropped by this attack. If the dropped packets carry important information about the real environment, e.g., an emergent abnormal event, then its impact is similar to the event suppression attack that we discussed before. If the dropped messages are redundant, then the attack is not as damaging as in the previous case. However, in the real condition, the attacker may selectively drop packets to maximize the damages to the network as well as the benefits to themselves. Fortunately, our probabilistic strategy could ease this impact in a large degree. Also, the count information is included in each packet. If finally the count computed by the BS is smaller than the actual network size (assume that the BS knows some general information about the network, such as network size), then the BS gets to know this kind of attack probably has happened during the aggregation. Furthermore, our group attestation is effective in detecting the count inconsistency caused by packet dropping within a group.

3.3.4 Other Attestation Techniques

The group attestation introduced in section 3.2.4.4 is actually a depth-based one because a top-down attestation path is selected in a group. Alternatively, we may use a breadth-based scheme, in which the BS may ask, for example, all the nodes one or two levels below the group leader to supply their aggregates. This approach is good for attesting a more balanced tree because the higher level nodes usually have larger counts than the lower level nodes. However, it may be ineffective for arbitrary tree topology, and it is more vulnerable to colluding attacks by several topologically consecutive compromised nodes.

Some aggregation functions are found inherently insecure, such as MIN/MAX, because the change to a single sensor reading can cause a noticeable change to the final result [25]. Another situation is that it is hard for us to verify the input values from leaf nodes (actually this is unnecessary in most cases since readings from leaf nodes bring very small impact on the final aggregation result calculated by the BS), because there are not any downstream nodes under leaf nodes which can provide proofs to support or reject those values. Therefore, we also consider a content-based attestation approach to validate an outlier. Of course, this work could be further eased if the BS leverages the topological knowledge about the whole tree. Specifically, once the BS notices such an outlier, it requests the sensor readings from the neighbors of this node and compares them. Since these nodes are close to each other, their readings should bear certain spatial or temporal correlation. Based on this knowledge, the BS can decide whether to accept the outlier or not. Since this technique is orthogonal to the presented techniques, we will study it in the future.

3.3.5 Discussion

Next, we try to answer the question like "how many compromised nodes could our scheme tolerate?" The Byzantine generals problem[86, 87], which states that agreement could be reached if the number of compromised nodes is less than one third of the total number of nodes, may give us some hints to solve this problem. However, our model is different from that in the Byzantine generals problem in four ways.

First, the network model is different. We consider a reversed multicast tree network model in which data flow from individual sensors to a *centralized* base station, whereas in Byzantine generals problems it is a strongly connected directed graph model, where all the nodes are *distributed*, which means they are all equally important. Second, the communication model is different. In our scheme, each sensor sends individual sensor readings towards the base station and intermediate sensors may aggregate the data received from children nodes. Meanwhile, in Byzantine generals problem, nodes exchange mutual (two-party) messages. Third, the decision making process is different. Byzantine generals problem exploits majority voting mechanism, which means as long as majority of reports regarding the same node are consistent then this majority value will be used and the system will converge. In our scheme, the base station employs statistical method to detect outliers and make decisions upon which groups are abnormal. Fourth, in Byzantine generals problem, the decision making involves multiple rounds ($m+1$ if there are m compromised nodes), whereas for our scheme each round the base station makes decision and detects new compromised nodes.

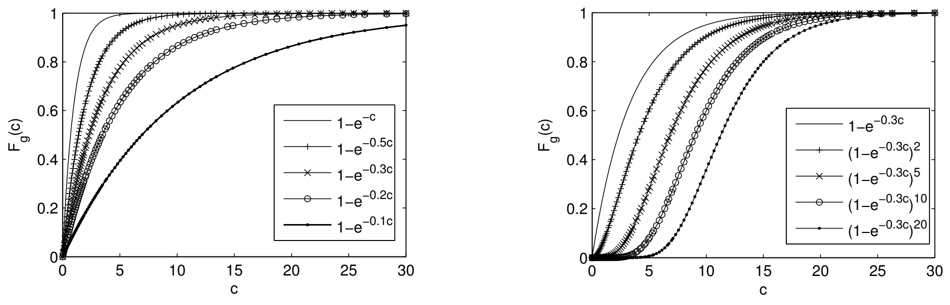


Figure 3.6. $F_g(c)$ as the function of count c with parameters β, γ: $F_g(c) = (1 - e^{-\beta \cdot c})^{\gamma}$ $(0 < \beta \leq 1, \gamma \geq 1)$.

Therefore, the total number of compromised nodes our scheme could tolerate depends on many factors, such as the distribution of compromised nodes, the number of groups/grouping results, and the outlier detection method. The probabilistic nature of our scheme enables it to tolerate potentially more than one third of compromised nodes. The data aggregation tree is randomly partitioned, so the compromised nodes cannot manipulate in this process and maximize their benefits; they could not choose positions in each group at will.

3.4 Performance Evaluation

In this section, we evaluate the performance of SDAP. We first present a grouping function and show that it meets our requirements through simulated grouping results. Then, we evaluate the effectiveness of Grubbs' test in detecting outliers. Last, after we analyze the overhead of the protocol, we further use simulations to support our claim that SDAP only causes little extra overhead compared to hop-by-hop aggregation.

3.4.1 Grouping Function and Grouping Results

We first pick up an appropriate grouping function, then we show the grouping results by applying this function.

3.4.1.1 Grouping Function

In Section 3.2.3.1, a grouping function F_g was used to control the probability for a node to become the group leader. This function generates an output between 0 and 1, based on the input count. Our goal is to select an F_g which ensures that the group sizes are close to each other so that the variance is reduced. Hence, when the BS performs the Grubbs' test, less likely a normal group will become an outlier to be attested, and the attestation overhead could also be decreased.

Specifically, this grouping function should meet the following requirements:

- if $c = 0$, $F_g(c) = 0$;
- if $c = 1$ (leaf node), $F_g(c) \approx 0$;
- if $c \to \infty$, $F_g(c) \to 1$, but $F_g(c) < 1$;
- the gradient of its curve increases slowly at first and decreases towards 0 after a peak value close to 1.

The first three requirements are apparent. Based on the fourth requirement, when the count c is small, the probability of becoming a leader is low, whereas when the count c is sufficiently large, this probability is rapidly increased to a large value (e.g., larger than 50%). As a result, the group sizes are close to each other.

To meet these requirements, we choose $F_g(c) = (1 - e^{-\beta \cdot c})^{\gamma}$ $(0 < \beta \leq 1, \gamma \geq 1)$, where β is used to control the gradient of the curve and γ is used to control the shape of the curve (e.g., concave or convex). As shown in Fig. 3.6, when β increases, the curve becomes sharper. With a large γ, the function satisfies the fourth requirement of our grouping function.

3.4.1.2 Grouping Results

We verify that the grouping function satisfies our requirements through the simulated grouping results. In the simulation, 3000 nodes are randomly distributed in an area of $2000 * 2000 ft^2$. The transmission range is set to be $65 ft$. Tree construction protocol introduced in Section 3.2.2 is used to build up the tree. For the grouping function, β and γ are set to 0.15 and 30, respectively. The simulation is run 5000 rounds. Fig. 3.7 shows the distribution of group leaders to the depth of the tree. The resulted aggregation tree has the height of 44. As discussed in Section 3.2.2, the root has a depth of 0, and nodes in depth 44 are all leaves. Since these nodes only have the count of 1, the chances of being leaders are almost 0. Due to the small counts, nodes in the depth of 40 or more

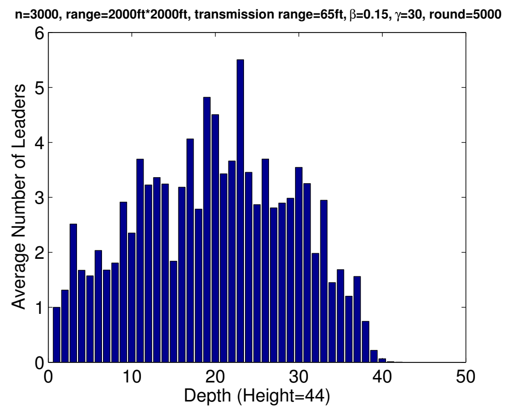


Figure 3.7. Distribution of group leaders.

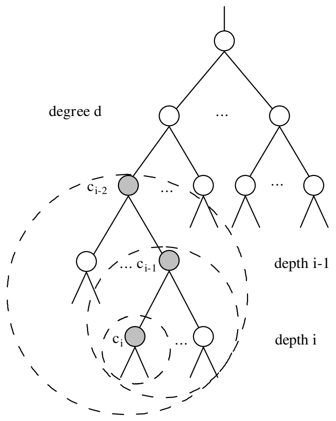


Figure 3.8. Derivation of counts and probabilities.

have no chance of being leaders, either. On the other hand, nodes with depth between 10 and 30 have higher probabilities to become leaders. For example, an average of 5.5 nodes in the depth of 22 are group leaders. The root is always a leader, so the average number of group leader in depth 0 is 1.

Next, we derive and verify the average group size and probability to become leader for each depth. As stated earlier, the aggregation tree T_a could be modeled with parameters (n,d,h), in which n, d, h are the total number of nodes, degree, and height of the tree, respectively. If the grouping procedure runs for many rounds, on average, we can get a probability based derivation function about the group size for each depth. Assume depth $i(0 \leq i \leq h)$ is the first depth from the leaf containing leader nodes and the count value for depth i is c_i (Fig. 3.8). Since it is the first depth having leaders, the probabilities for nodes in depth i to become leaders are the same and equal to

$$p_i = F_g(c_i), \tag{3.13}$$

where F_g is the grouping function in our protocol. Among all the d children of a node in depth $(i-1)$, the number of leader nodes can be treated as a random variable X that follows the binomial distribution with parameters (d, p_i). The probability for the $k(0 \leq k \leq d)$ out of d children of a node in depth $(i-1)$ being chosen as leaders is

$$P(x=k) = \binom{n}{k} p_i^k \cdot (1-p_i)^{d-k}. \tag{3.14}$$

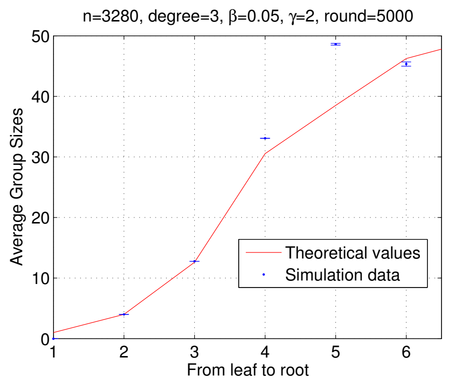


Figure 3.9. Average counts for each depth.

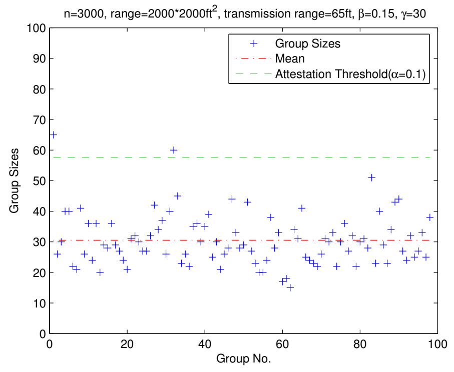


Figure 3.10. Attestation threshold with no attacks.

Thus, according to the definition of a random variable's expectation, the average group size of nodes in depth $(i-1)$ is

$$c_{i-1} = \sum_{k=1}^{d} P(x=k) \cdot [c_i(d-k)+1], \tag{3.15}$$

and the average probability for nodes in depth $(i-1)$ becoming leaders is

$$p_{i-1} = \sum_{k=1}^{d} P(x=k) \cdot F_g[c_i(d-k)+1]. \tag{3.16}$$

Because from the first depth containing leader nodes, we can get the initial values of c_i and p_i. Then, recursively, we can derive the average group size and probability of becoming leaders for each depth, according to equations (3.15) and (3.16).

To verify our theoretical derivation, we run the simulation 5000 times for the same topology tree. The total number of nodes n is 3280, the degree of tree is 3, and the height of the tree is 7. In each round, although the topology tree remains constant, the grouping results are different from each other, since every intermediate node is elected as the leader probabilistically. Simulation results (Fig. 3.9) show that our theoretical analysis can roughly describe the curves of the average group sizes for all the depths. Every point of the simulation data in this figure represents the average of 5000 values.

3.4.2 Performance of Grubbs' Test

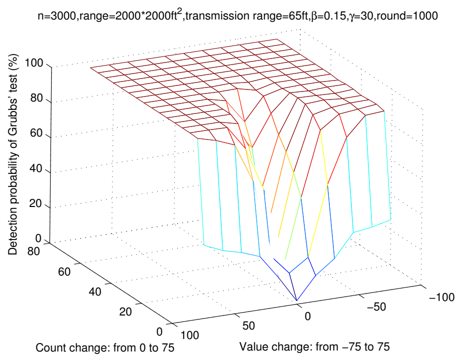
In this section, we first check the performance of Grubbs' test when there are no attacks, i.e., the number of legitimate groups attested. Then, the effectiveness of Grubbs' test in detecting outliers when there are count and/or value changing attacks is also investigated.

As a case study, we take the (weighted) mean as the aggregation function. We can obtain similar results upon other aggregation applications, such as SUM, COUNT and MEDIAN. Two metrics are used to evaluate the performance of Grubbs' test when there are attacks: *detection probability* and *accuracy improving rate*. The detection probability is the possibility that the Grubbs' test can detect the corresponding malicious groups. Suppose $total\_no$ is the total number of malicious groups we introduce and $detect\_no$ is the number of malicious groups that are detected, then the detection probability is defined as $\frac{detect\_no}{total\_no}$. Accuracy improving rate reflects the accuracy improvement on the aggregation result calculated by the BS after the detected unverifiable outliers are removed from the aggregate set. Let agg be the real aggregation result without attacks, agg_1 be the aggregation result with attacks, and agg_2 be the aggregation result after unverifiable outliers are excluded. Then the accuracy improving rate is defined as: $\frac{|agg_1 - agg_2|}{agg} \times 100\%$. Because our grouping method is probabilistic, we run each simulation 1000 times to the same aggregation tree and calculate the average values.

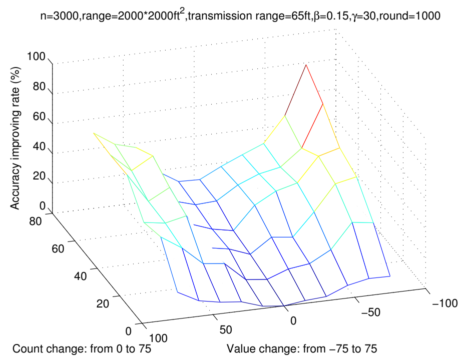
3.4.2.1 Grubbs' test without attacks

Normally, when there are no value changing attacks, group aggregation values are close to each other, so the deviation is small and no legitimate groups will be attested.

Fig. 3.10 shows the distribution of group sizes without count changing attacks. As can be seen, the mean of group size is 30, and the resulted group sizes do not deviate much from the mean. More specifically, most group sizes are limited between 20 and 40. This can provide a good basis for the attestation. According to the critical values in Grubbs' test, when the total group number is 98 and $\alpha = 0.1$ (one tailed test, $\alpha/2 = 0.05$), the attestation threshold is 57.57. If the attacker increases the group size larger than this threshold, the BS can detect this attack by choosing the corresponding group for attestation. From the figure, we can see that the number of attestation is very small when there are no attacks, since only 2 out of 98 legitimate groups have group sizes



(a) Detection probability of Grubbs' test.



(b) Accuracy improving rate.

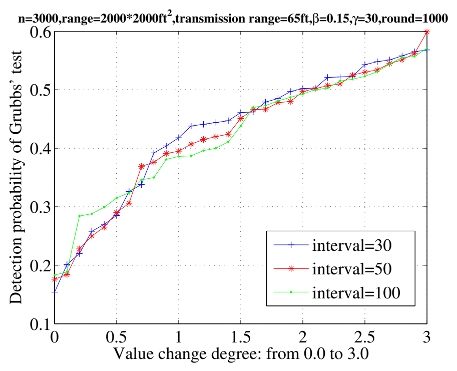
Figure 3.11. The detection probability of Grubbs' test and accuracy improving rate when there is one malicious node.

larger than this threshold. Although the BS will choose these two groups for attestation, the BS will accept their aggregates after the attestation because they are both legitimate groups.

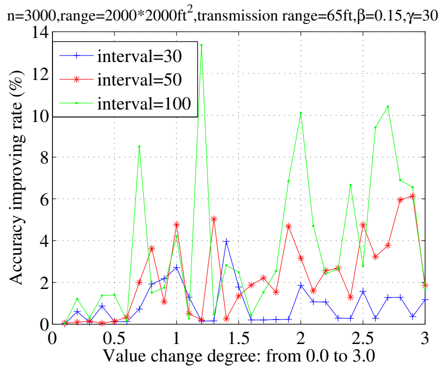
3.4.2.2 Grubbs' test with single malicious node

Fig. 3.11 illustrates the detection probability of Grubbs' test and the accuracy improving rate when there is one malicious node launching count and/or value changing attacks. In the simulation, sensor readings are uniformly distributed over a range of [70, 100]. To simulate the combination of count and value changing attacks by one malicious node, we randomly pick up a node and change its count as well as aggregation value in different degrees.

From Fig. 3.11(a), we can see that the Grubbs' test is very effective in detecting the combination of count and value changing attacks, since the detection probability is rapidly increased to a high value ($\geq 80\%$), after the aggregate is changed by a certain degree, e.g, value changed by 15 and count changed by 20. The larger is this degree, the higher the detection probability becomes, until finally it reaches 100%. Also, Grubbs' test is more effective in detecting count changing attack compared with value changing attack. The detection probability of Grubbs' test is about 30% to detect a value changed by 40, but if we change the count by the same degree the detection probability is actually 100%. The reason is as follows: the increase of one normal node's count directly changes the group count, whereas the change to group aggregation value is influenced by



(a) Impact to detection probability.



(b) Impact to accuracy improving rate.

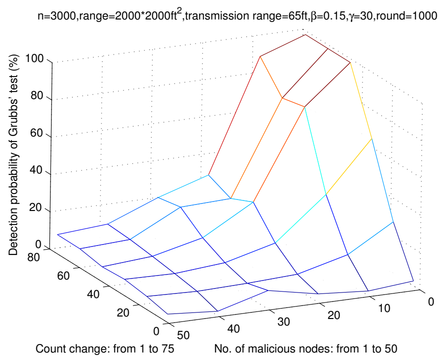
Figure 3.12. The impact of sensor readings' distribution range to detection probability and accuracy improving rate.

many other factors, such as the distance between the malicious node and group leader, the aggregates from other sibling nodes, and so on.

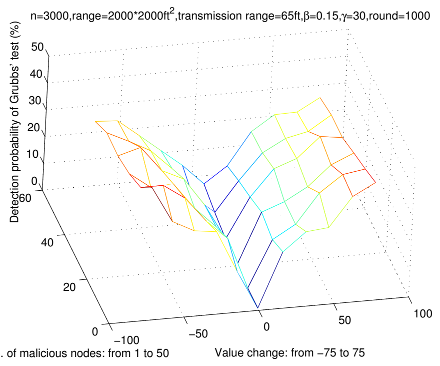
Fig. 3.11(b) shows that the accuracy improving rate increases with a larger count or value change. Obviously, if the aggregate is changed by a larger degree, after removing this outlier, we can obtain a higher accuracy improvement. Additionally, if we increase and decrease the aggregation value by the same degree, we get two accuracy improving rates that are close to each other, so this figure is symmetric from this point of view.

We also check the impact of sensor readings' distribution to the detection of value changing attack (it is irrelevant to counts). Fig. 3.12 shows the influence of sensor readings' distribution range to the detection probability and accuracy improving rate. We check three ranges, which are [70, 100], [70, 120] and [70, 170], with distribution interval of 30, 50, 100, respectively. Change degree represents the degree of the interval that the compromised node's aggregation value is changed by. In another word, we randomly pick up one node and change its aggregation value by interval*degree.

From Fig. 3.12(a), we can see that under the same distribution interval if the change degree becomes larger, then the detection probability is higher, because the aggregation value is changed by a larger number. However, the detection probability of Grubbs' test is not influenced much by the change of distribution interval. Normally, with fixed change degree and higher interval, the aggregation value is also changed by a larger value, but in this case the variance of group aggregations is actually increased, which decreases the detection probability. As a result, the detection probability is slightly influenced by the increase of distribution interval. As shown in Fig. 3.12(b), the accuracy

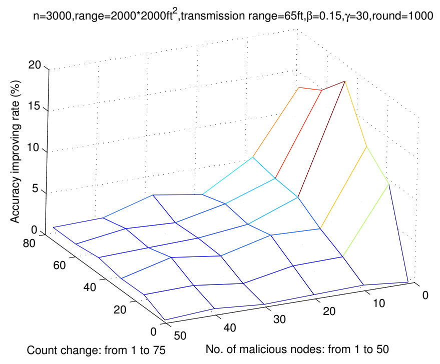


(a) Count changing attack.

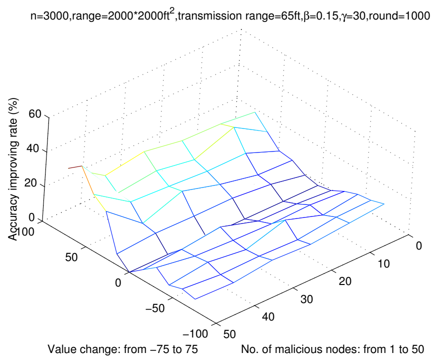


(b) Value changing attack.

Figure 3.13. The detection probability with multiple attacks.



(a) Count changing attack.



(b) Value changing attack.

Figure 3.14. The accuracy improving rate with multiple attacks.

improving rate vibrates with the rise of value change degree due to the particularity of value changing attack mentioned above, but generally speaking, the accuracy improving rate is higher with a larger distribution interval. Apparently, we can obtain a similar result if we decrease the aggregation value instead.

3.4.2.3 Grubbs' test with multiple malicious nodes

Instead of changing only one node's aggregate, we randomly choose multiple nodes and change their aggregates in this simulation. For simplicity, the changes to the aggregates of all the compromised nodes are the same. Fig. 3.13 shows the detection probability of Grubbs' test under multiple attacks. As can be seen from Fig. 3.13(a), the detection probability becomes higher with larger count changes, but this probability decreases with more malicious nodes. The reason is that the increase in the number of malicious nodes also directly raises the variance of group sizes, which causes a lower detection probability. When the number of malicious nodes is increased to more than half of the total number of groups, this probability is decreased to about 10%. Similarly, as shown in Fig. 3.13(b), the detection probability increases with a larger value change and decreased when there are more malicious nodes. However, this decrease is not as obvious as that in the count changing attack, because the change of normal nodes' aggregation values may not influence the group aggregates in a obvious way.

We get similar results in the accuracy improving rate with multiple attacks, as shown in Fig. 3.14. From Fig. 3.14(a), we can see that the accuracy improving rate increases with a larger count change, but decreases if there are more malicious nodes launching count changing attacks. As shown in Fig. 3.14(b), the accuracy improving rate is higher with a larger value change, but it is not influenced much by the increase in the number of malicious nodes.

3.4.3 Overhead Evaluation

In this section, the overhead of SDAP is evaluated from the following three aspects: computation, storage and communication.

3.4.3.1 Computational Cost

We assume that the BS has sufficient resources including computation, so we only consider the computational cost in sensor nodes. During the aggregation, generally speaking, each node in the aggregation tree needs to compute one decryption, one pseudorandom function value, one grouping function value, one aggregation, one MAC, and one encryption. In practice, the computational cost is actually lower, since some nodes may not need to do all the operations. For example, leaf nodes only compute one encryption and one MAC. During the attestation, each node on the attestation path needs compute a pseudorandom function value. Moreover, these nodes and their sibling nodes encrypt the data that are sent back to the BS.

Given the scarce resources in sensors, we can use block cipher RC5 [88] to implement all these cryptographic primitives, so as to reuse code and save memory space, because RC5 has advantages due to its versatility [76, 89, 80]. The encryption/decryption can directly use RC5. The message authentication code (MAC) and pseudorandom function could be implemented by cipher block chaining CBC-MAC based on RC5. Furthermore, computation time spent on encryption and MAC are almost the same [76].

Therefore, during the aggregation, each node need compute four MACs (the computation of aggregation value and grouping function value only involves simple mathematical operations, so the cost is much less). In addition, during the attestation, each node on the attestation path computes two MACs, and each sibling node computes one MAC. Hence, the possibly maximal computation cost for one node during the whole process is to compute six MACs. The energy a sensor node uses in computing one MAC is about the same as that used for transmitting one byte [34]. Thus, from energy point of view, the energy used by a sensor node for both aggregation and attestation is about the same as that used in transmitting six bytes, so we believe this is an reasonable overhead for the current-generation sensor nodes.

3.4.3.2 Storage Requirement

Each node within groups need keep local copies of packets with flag ‘0’ received from children, except the leaf nodes which only keep a local copy of their own packets. Each aggregation packet is 19-byte (2 bytes for id, 9 bytes for data including count and grouping seed, 8 bytes for MACs). Also, each node on the way to the BS need construct a table recording the forwarding path, with each item having 8 bytes (2 bytes for leader node’s id, 2 bytes for incoming node’s id and 4 bytes for grouping seed). In our simulation with a big aggregation tree (3000 nodes and about 100 groups), the average degree of node is close to 10 and the total number of groups below a leader is also in tens. Therefore, the total storage requirement for one node is at most several kilobytes. Considering that the current-generation sensor nodes such as MICA2 Motes have 128KB program memory, 4KB data RAM, and 512KB non-volatile measurement flash memory, the storage overhead is not a concern, either. This will also be verified by our prototype implementation in Section 3.4.4.

3.4.3.3 Communication Overhead

Next, we focus on analyzing the communication overhead of our scheme. Specifically, we first analyze the communication overhead of our protocol and then further use simulations to verify our claim that our protocol only causes little extra overhead compared to hop-by-hop aggregation.

To accurately measure the overhead, we use the metrics of $packet * hop$ and $byte * hop$ (product of the data size and the message traveling distance), because message overhead is proportional to the traveling distance of sensing data. To help understand the communication overhead of our protocol, we also compare it with the no-aggregation and hop-by-hop aggregation approaches. For ease of exposition, we do not consider the impact of packet retransmission due to the unreliable channel. Although packet retransmission will increase the absolute performance overhead of SDAP, we expect its *relative* performance overhead compared to the other two approaches will be similar because packet retransmissions also occur in these approaches. We will verify the above intuition quantitatively in our future work.

Analytical Results in packet*hop Since all the three schemes have the query broadcast overhead, we only compare the communication overhead in the aggregation and attestation. In the hop-by-hop data aggregation approach, the number of packets is equal to the number of edges in the broadcast tree. Hence, the communication overhead of the hop-by-hop aggregation approach is as follows:

$$C_{hop-by-hop} = n - 1 = \Theta(n).$$

On the other hand, without in-network aggregation, i.e., every sensor node sends its reading (with a MAC) separately to the BS, the communication overhead can be expressed by:

$$\begin{aligned} C_{no-aggregation} &= \sum_1^h i \cdot d^i \\ &= \frac{hd^{h+2} - (h+1)d^{h+1} + d}{(d-1)^2} \\ &= \Theta(n \cdot logn), \end{aligned}$$

because h can be approximated by $logn$. The upper bound is $O(n^2)$ in case of a linear tree ($h = n$, $d = 1$).

In our protocol, the total number of groups is $\lfloor n/g \rfloor + 1$, considering the extra group with the BS as the default leader. The height of the group can be approximated by $\lceil h/2 \rceil$, and then the average distance to the BS from a leader is $\lfloor h/2 \rfloor$. Based on the results shown in Fig. 3.7, this assumption is reasonable because we only consider the average case. Therefore, the communication overhead of our protocol during the aggregation phase is $(g-1)(\lfloor n/g \rfloor + 1) + \lfloor n/g \rfloor \lfloor h/2 \rfloor$.

The overhead for attestation depends on the number of attested groups and the attestation paths that we have chosen. The overhead of disseminating the attestation request is $n_a n_p \lfloor h/2 \rfloor$, and the overhead of sending the data back to the BS is $n_a n_p [\lfloor h/2 \rfloor + \sum_1^{\lceil h/2 \rceil} (\lfloor h/2 \rfloor + i)d]$. Therefore, the total overhead is:

$$\begin{aligned} C_{our} &\leq (g-1)(\lfloor n/g \rfloor + 1) + \lfloor n/g \rfloor \lfloor h/2 \rfloor + n_a n_p \lfloor h/2 \rfloor \\ &\quad + n_a n_p [\lfloor h/2 \rfloor + \textstyle\sum_1^{\lceil h/2 \rceil} (\lfloor h/2 \rfloor + i)d] \\ &\approx n + \lfloor nh/2g \rfloor + n_a n_p h + \frac{n_a n_p d h (3h+2)}{8}. \end{aligned}$$

This formula actually gives us an upper bound of the communication overhead because in case of multiple attestation paths, a node locating on multiple paths only needs to report one copy of its aggregate. Also, the n_p attestation requests for a group could actually be piggybacked into one packet.

Therefore, the communication overhead of our protocol depends on the average group size g. If g is as large as n, the overhead is about $O(n)$. Otherwise, if g is small and can be treated as a constant number, the overhead is $O(n \cdot logn)$. In either case, the overhead of our protocol is lower than the no-aggregation approach and higher than the hop-by-hop aggregation approach.

To quantify the difference, data results in $packet * hop$ can be seen in Fig. 3.15. The results are based on the following parameter setup: $n = 3280$, $d = 3$, $h = 7$ and $n_p = 1$. As shown in the figure, the communication overhead of our protocol is between $3.4K$ and $4.4K$. Using the same parameters, we can easily calculate the cost of the other approaches. Specifically, the communication overhead of the hop-by-hop aggregation approach is $3K$, and the communication overhead of the no-aggregation approach is $21K$. Thus, our protocol does not add much overhead compared to the hop-by-hop aggregation.

Analytical Results in byte*hop With the results of last subsection, we can easily calculate the overhead in $byte * hop$. Each packet includes node id (2 bytes), data (4 bytes) and MAC (8 bytes), so the overhead of the hop-by-hop aggregation and the no-aggregation approaches in $byte * hop$ are the results in $packet * hop$ multiplied by 14(bytes).

Although we did not consider the query dissemination overhead, for fair comparison, we should consider the extra overhead of our protocol, due to the 4-byte random number

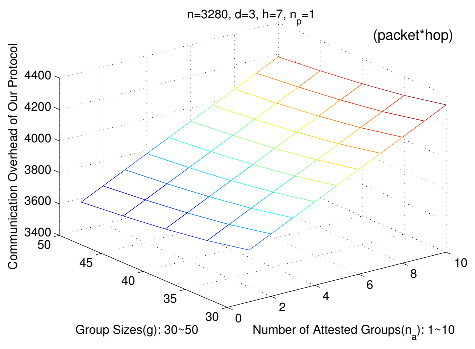


Figure 3.15. Communication overhead in packet*hop.

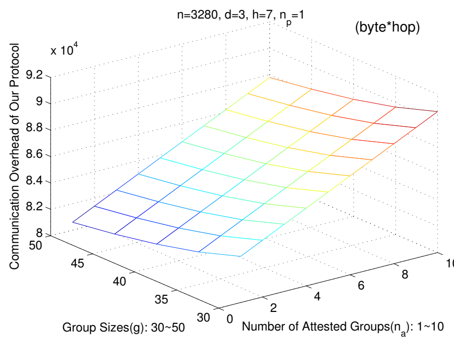


Figure 3.16. Communication overhead in byte*hop.

used in the query dissemination. The total extra communication overhead for the query broadcast from the BS is about $4(n-1)$. The committed aggregation packet is of the same format and with a size of 19 bytes (2 bytes for id, 9 bytes for data including count and grouping seed, 8 bytes for MAC), thus the overhead of the aggregation is $19[(g-1)(\lfloor n/g\rfloor+1)+\lfloor n/g\rfloor\lfloor h/2\rfloor]$. The size of the attestation request from the BS is 10 bytes, 2 bytes for the leader id and 8 bytes for grouping/attestation seeds. Thus, the overhead to disseminate the attestation request is $10n_an_p\lfloor h/2\rfloor$. The overhead of sending data back to the BS is $21n_an_p[\lfloor h/2\rfloor+\sum_1^{\lceil h/2\rceil}(\lfloor h/2\rfloor+i)]+13n_an_p\sum_1^{\lceil h/2\rceil}(\lfloor h/2\rfloor+i)(d-1)$, because the packets sent back to the BS from the nodes on the attestation path have the size of 13 bytes (4 bytes for node and its parent ids, 9 bytes for data including count and seed) and packets from other nodes in the attestation are of the size 21 bytes (4 bytes for node and its parent ids, 9 bytes for data including count and seed, 8 bytes for MAC). The total communication overhead of our protocol in $byte*hop$ is given by

$$\begin{aligned}
C'_{our} &\leq 4(n-1)+19[(g-1)(\lfloor n/g\rfloor+1)+\lfloor n/g\rfloor\lfloor h/2\rfloor] \\
&\quad +31n_an_p\lfloor h/2\rfloor+21n_an_p\textstyle\sum_1^{\lceil h/2\rceil}(\lfloor h/2\rfloor+i) \\
&\quad +13n_an_p\textstyle\sum_1^{\lceil h/2\rceil}(\lfloor h/2\rfloor+i)(d-1) \\
&\approx 23n+\lfloor\frac{19nh}{2g}\rfloor+\lfloor\frac{31n_an_ph}{2}\rfloor+\frac{n_an_ph(3h+2)(13d+8)}{8}.
\end{aligned}$$

The result in $byte*hop$ (Fig. 3.16) is based on the same parameter setup: $n=3280$, $d=3$, $h=7$ and $n_p=1$. As shown in the figure, the communication overhead of our protocol is between $80K$ and $92K$, whereas the communication overhead of the hop-by-hop aggregation approach is $45.9K$ and the communication overhead of the no-

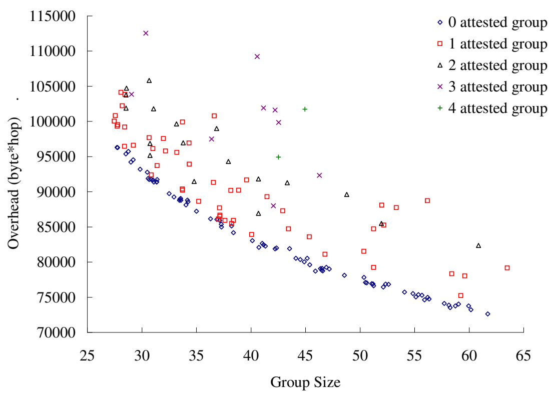


Figure 3.17. Communication overhead based on simulations.

aggregation approach is $298.5K$.

Simulation Results The previous analytical results are applicable to balanced trees with fixed degrees. To evaluate the communication overhead for more general cases, we also setup simulations to check the results. In our simulation, 3000 nodes are randomly distributed in an area of $2000*2000ft^2$. The transmission range is set to $60ft$. To test different group sizes, (β, γ) takes 8 different groups of values: $(0.15, 30)$, $(0.14, 33)$, $(0.13, 36)$, $(0.12, 39)$, $(0.11, 42)$, $(0.10, 45)$, $(0.09, 48)$, $(0.08, 51)$. For each pair of parameters, we run the simulation 20 times, each time with a different grouping seed. Based on our Grubbs' test, among all the 160 simulation runs, there are no attested groups in 79 simulations, 1 attested group in 52 simulations, 2 attested groups in 18 simulations, 3 attested groups in 9 simulations, and 4 attested groups in 2 simulations. We choose one attestation path in each attested group. The simulation result in $byte*hop$ is shown in Fig. 3.17. As can be seen from the figure, the overhead of our protocol including attestation is between 70K∼115K. With the same parameters, through simulation, we get the communication overhead of the hop-by-hop aggregation approach to be $42K$, and the communication overhead of the no-aggregation approach to be $1202K$.

In summary, through both analytical and simulation results, we can see that our protocol does not add much overhead compared to the hop-by-hop aggregation approach, but it is more secure. On the other hand, as the no-aggregation approach, our protocol provides security, but with much less communication overhead.

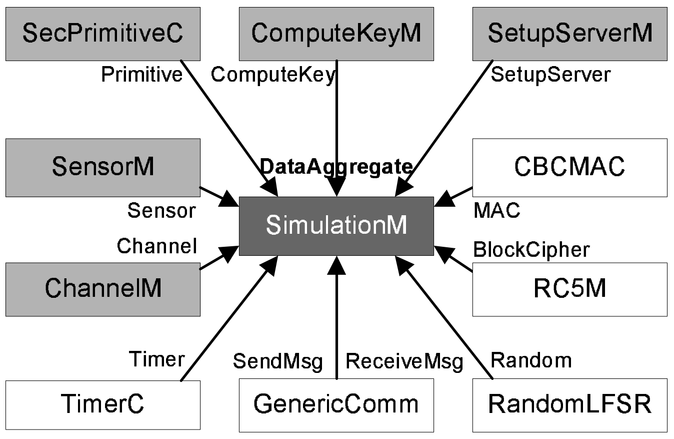


Figure 3.18. The relationship among components in our implementation: Dark gray components are developed by us; light gray ones are adapted from TinyKeyMan; white components are directly adopted from TinyOS and TinySec.

3.4.4 Prototype Implementation

We implement the prototype of our SDAP scheme on top of the TinyOS. The illustration of component relationship can be seen in Fig. 3.18. The simulation module (SimulationM) is the main module of our implementation. It provides the interface DataAggregate whose implementation realizes the main function of secure hop-by-hop data aggregation and attestation. The timer, communication and random number generator are adopted from TinyOS [90]. The MAC, encryption and decryption mechanisms are used from TinySec [89]. The pairwise key establishment mechanism is adapted from TinyKeyMan [91].

Every node constructs and keeps four tables to record information for successful aggregation and attestation. The first one is AssociateTable, which stores basic information of each node, such as parent id, children's ids, reading, count and aggregate. The second table is ValueTable, which stores counts and aggregates received from children for later aggregation. The third one is KeyTable, which stores an individual key shared with the BS and all the pairwise keys established with neighbors for encryption, decryption and MAC computation/verification. The fourth table is RoutingTable, which stores routing information for the attestation, including incoming node id and corresponding leader node id.

In more detail, after initialization, first the command DataAggregate.init() is called, so that the aggregation tree topology is constructed and all the maintained tables are initialized. Then, after deployment, except the BS (with id of 0) every node establishes a pairwise key with its parent from the first node (in highest level) to the last leaf node (in lowest level). After this, to reduce channel collision/packet loss and also guarantee proper time synchronization for aggregation, the aggregation process is scheduled in a reverse order, i.e., from the last leaf node to the first node in the highest level, every node aggregates counts and values received from children if there are any and sends an aggregation packet on its own behalf to its parent by calling DataAggregate.sendAggToParent() with parent id as input. Parent node receiving this packet checks the flag field: if it is 1, then node records the source and leader node ids into the RoutingTable, and resends this packet to its own parent; otherwise if it is 0, then this node decrypts count and aggregate data, verifies the MAC. If both succeed, then this node records the data from children into the ValueTable for its own aggregation operation happened later. For attestation, the BS sends out an attestation packet by calling DataAggregate.sendAttToNode() with the attested leader's id as input. Every node including BS itself forwards the attestation packet to the next hop by looking up the RoutingTable until this packet reaches the attested group leader. Then, the attestation within group starts. The group leader notifies the node on the attestation path to send back counts and readings, and also informs their sibling nodes to send back counts, aggregates and MACs. Nodes on the attestation path repeat this process until leaf nodes are reached. All the data are sent back to the BS by calling DataAggregate.sendAttAckToBS() with the BS's id 0 as input.

There are six types of packets transmitted in our scheme: aggregation, attestation, path node notification, sibling node notification, path node response, and sibling node response. The packet formats of four of them are shown in Fig. 3.19 (the packet formats of path node and sibling node notifications only include header, one field of type, and CRC, which are too simple to be shown here). By specifying "export DBG=am" under TOSSIM, we can check all the transmitted messages in the system. In Fig. 3.19(a), the format of aggregation packet includes header, data and CRC fields. The header has the information about forwarding address (2 bytes), message type in Active Message (AM) model (1 byte), group number (1 byte), and message length (1 byte). In the example, this is an aggregation message sent from leader node 1 to the BS 0. Therefore, the forwarding address is the destination node id in hex 0x0000; Normally, our packets are AM_INTMSG, which is specified in TinyOS to have the type of 4; The default group

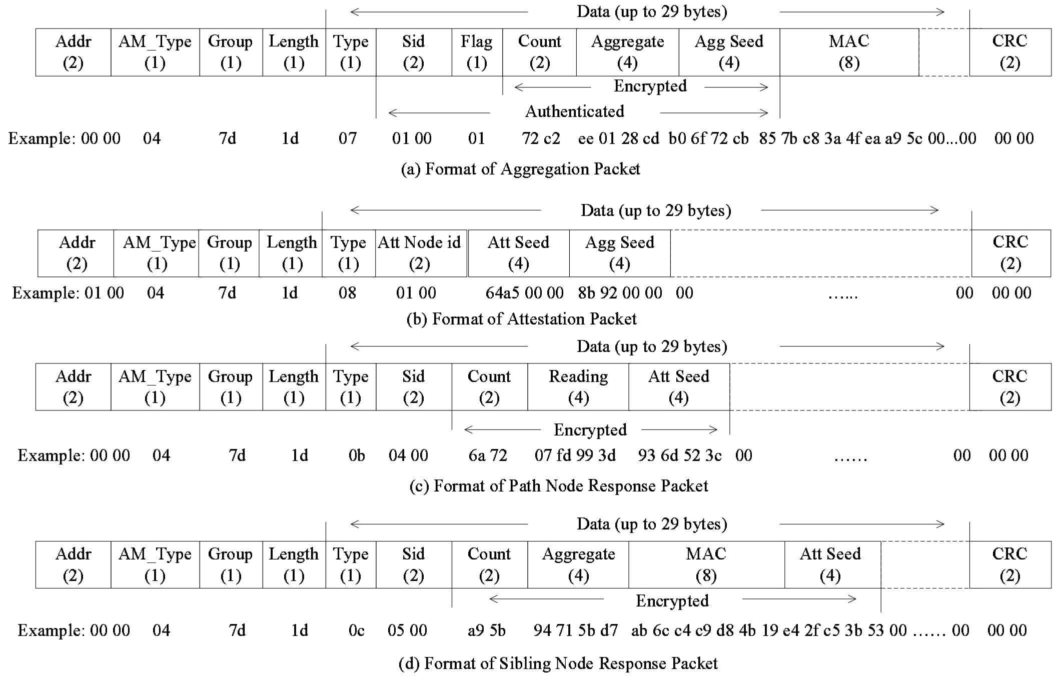

Data (up to 29 bytes)
Addr (2) | AM_Type (1) | Group (1) | Length (1) | Type (1) | Sid (2) | Flag (1) | Count (2) | Aggregate (4) | Agg Seed (4) | MAC (8) | CRC (2)
Encrypted
Authenticated
Example: 00 00 04 7d 1d 07 01 00 01 72 c2 ee 01 28 cd b0 6f 72 cb 85 7b c8 3a 4f ea a9 5c 00...00 00 00
(a) Format of Aggregation Packet
Data (up to 29 bytes)
Addr (2) | AM_Type (1) | Group (1) | Length (1) | Type (1) | Att Node id (2) | Att Seed (4) | Agg Seed (4) | CRC (2)
Example: 01 00 04 7d 1d 08 01 00 64a5 00 00 8b 92 00 00 00 ...... 00 00 00
(b) Format of Attestation Packet
Data (up to 29 bytes)
Addr (2) | AM_Type (1) | Group (1) | Length (1) | Type (1) | Sid (2) | Count (2) | Reading (4) | Att Seed (4) | CRC (2)
Encrypted
Example: 00 00 04 7d 1d 0b 04 00 6a 72 07 fd 99 3d 93 6d 52 3c 00 ...... 00 00 00
(c) Format of Path Node Response Packet
Data (up to 29 bytes)
Addr (2) | AM_Type (1) | Group (1) | Length (1) | Type (1) | Sid (2) | Count (2) | Aggregate (4) | MAC (8) | Att Seed (4) | CRC (2)
Encrypted
Example: 00 00 04 7d 1d 0c 05 00 a9 5b 94 71 5b d7 ab 6c c4 c9 d8 4b 19 e4 2f c5 3b 53 00 ...... 00 00 00
(d) Format of Sibling Node Response Packet

Figure 3.19. The formats of packets in our implementation. One example of each packet type is given to illustrate the specific format.

number in hex is 0x7d; The maximum length of data field is 29-byte in TinyOS, which is 0x1d in hex. To differentiate our six types of packets, we define a type attribute (1 byte) in the data field: 7 is the type of aggregation packet, 8 is the type of attestation packet, 9 is the type of path node notification packet, 10 is the type of sibling node notification packet, 11 is the type of path node response packet, and 12 is the type of sibling node response packet (1-6 are already predefined in TinyKeyMan). Sid records the source node id (2 byte). The source id of node 1 in our example is 0x0100 (lower byte is presented first). Flag (1 byte) shows whether the data need to be aggregated further. Here the leader node 1 sets it to be 1. The 2-byte count, 4-byte aggregate, and 4-byte aggregation seed in the packet are encrypted by the pairwise key. An 8-byte MAC computed over the previous fields by individual key is attached at the end of the data field to provide authentication information to the BS. Finally, the 2-byte CRC field for error checking is unused in our system yet (i.e., remains 0x0000). Similarly, the formats of attestation, path node response, and sibling node response packets are shown in Fig. 3.19(b), Fig. 3.19(c), and Fig. 3.19(d), respectively.

On Mica2 motes, ROM space needed for our prototype implementation is 20.7KB out of 128KB program memory. The RAM space changes with the maximum degree of the tree because if the tree has larger maximum or average degree more information will need to be maintained for successful aggregation and attestation. The RAM space needed is around 1.3KB out of 4KB data memory, which is shown in Table 3.1. These results indicate that our scheme is practical when applied on current generation sensors such as Mica2 motes.

Table 3.1. Code size as a function of the maximum degree in the aggregation tree (the total number of nodes is 50).

Maximum degree	2	4	6	8	10
RAM(bytes)	1286	1314	1342	1370	1398

Chapter 4

Distributed Software-based Attestation for Node Compromise Detection

4.1 System Model and Assumptions

Network Model We consider a sensor network composed of a large quantity of low-cost sensors that are constrained by scarce resources in power, computation, communication and storage (e.g., 128KB program memory and 4 KB RAM in the case of Mica2 motes). We assume that sensors are densely deployed in the network so that each node has multiple immediate neighbors and the neighbors can communicate with each other. There is a way for sensors to discover neighbors (e.g., by sending a HELLO message [92]) and to establish a pairwise key shared with every neighbor [93].

Attack Model We assume that an attacker can capture a (small) fraction of sensors, reprogram them with malicious code, and redeploy them back into the network. Since in our setting all the nodes are equally important and vulnerable to compromises, we assume that every node is equally likely to be compromised. Like all the previous schemes [37, 38, 40, 39], we assume that the attacker does not enlarge the sensor's memory; unlike the time delay based attestation schemes [38, 40] that further assume the processor speed and memory access rate are not increased, our schemes do not make such assumptions.

Once compromised, sensors are under the full control of the attacker. They may launch various attacks, including passive attacks such as eavesdropping, and active attacks such as false data injection. Detection of node's abnormal behavior (e.g., by

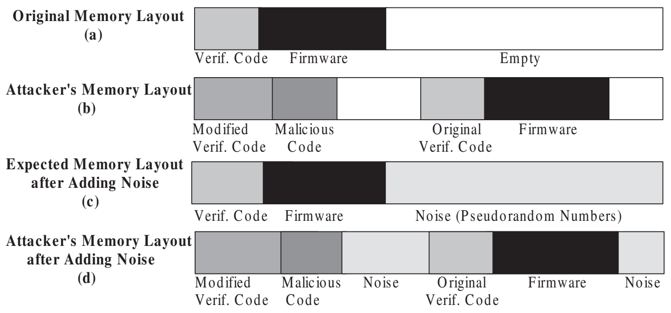


Figure 4.1. Program memory layout before and after adding noise.

watchdog [21]) may trigger our attestation protocol; however, it is not necessarily the only condition. For example, our attestation protocol may be executed with a reasonably large (e.g., days) and random trigger interval to reduce the attestation cost and also to prevent the time-of-check-to-time-of-use (TOCTTOU) attack.Furthermore, we assume the compromised nodes in the same neighborhood may collude by sharing their knowledge and resources during an attestation.

Note that our main concern is sensor's program space instead of data space, because of the following reasons. Data space is normally used to store current execution information, such as program stack and temporary data. Therefore, certain parts of the data memory cannot be overwritten by the attacker. Otherwise, it may cause the sensor program to crash. This not only largely reduces the amount of available space for the attacker, but also requires the attacker to be very careful when considering where to put the copy of the original code. If we also take into account the fact that the size of data space is typically two orders of magnitude smaller than that of the program space, we assume that the attacker will not put original code to sensor's data space.

4.2 Preliminaries

Besides the normal functioning code, a piece of verification code is also loaded at the beginning of a sensor's program memory to support code attestation. As shown in Figure 4.1(a)[1], the original layout of memory contains the verification code, firmware and some empty space. The empty space exists because sensor nodes have fixed-size code space that is usually not filled completely. Figure 4.1(b) illustrates the modified layout of sensor's memory by an attacker, where some malicious code takes over the memory

[1]This is a schematic figure. The actual layout may be slightly different.

space of the original code and the latter is moved to the empty space. Our idea to prevent this attack is to proactively fill the empty space of every node with pseudorandom numbers (also called *noise*, Figure 4.1(c)) before node deployment. Noises are derived from a secret seed that is unique for each node. The attacker does not know the seed for generating the noise, thus is unable to compress the noise to make empty space for his own malicious code. If the attacker moves the original code to overwrite the noise space (Figure 4.1(d)), our attestation schemes will guarantee that once challenged the probability for the compromised node to return a correct checksum is negligible.

In the following, we first present a specific noise generation method, then describe a block-based pseudorandom memory traversal algorithm as the underlying memory traversal mechanism for our distributed software-based attestation schemes.

Noise Generation To generate noises for our purpose, we use the block cipher RC5 running in *CTR (counter) mode* as Pseudo Random Number Generator (PRNG). The design principle is for the attestor's convenience in reconstructing the attested node's memory image. This is done by taking a seed as the input and each time encrypting an incremental counter, i.e., first encrypting a 0, then encrypting a 1, and so on. We call this seed to the PRNG as *noise-generation seed*, which contains information about all the generated noise. Since each time we are outputted with an 8-byte noise, we only need $m'/8$ counters to generate all the noise, if m' is empty memory size (in byte) to be filled.

Counter mode offers a nice property that the (1-byte) noise in each memory cell is individually accessible by directly encrypting the right counter and finding out the correct offset in the corresponding 8-byte value. As such, in the block-based traversal algorithm introduced below, the attestor keep only three variables in its data space during memory traversal: the current checksum, the block value, and the accessed memory cell within the block. This is highly desirable because an attestor consumes little data space to locally generate a correct answer for comparison. Note if noise is generated by *CBC mode*, i.e., first encrypting the seed, then repeatedly applying encryption over the previous output, the attestor will spend a lot of memory in constructing the expected memory image of attested node because all the noise must be generated sequentially. This is infeasible in our schemes since an attestor is just a regular sensor whose own program space is filled up with noise and data space is limited.

Block-based Pseudorandom Memory Traversal There are two memory traversals,

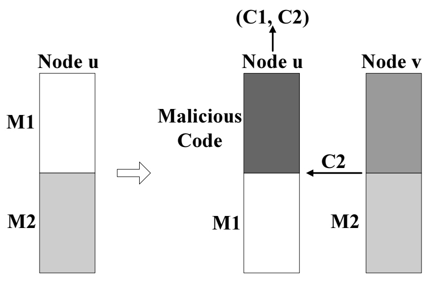


Figure 4.2. Collusion between node *u* and *v* in sequential memory traversal.

sequential and pseudorandom. In a sequential traversal scheme [39], a checksum is computed over a block of memory and a checksum vector over all the blocks reflects integrity of the entire memory. This scheme, however, is ineffective against collusion among nodes. Here is an example (Figure 4.2). Let the original memory of a node u be divided into two halves, M_1 and M_2. Node u may store M_1 and resort to its neighbor v to store the other half M_2. Thus, node u could use the other half for malicious code. When challenged, node v computes checksum C_2 over M_2, then passes C_2 to node u, which locally computes C_1 over M_1 and returns the correct result (C_1, C_2).

Pseudorandom memory traversal [40] is effective to defend against the above attack because the order of memory access is unpredictable to the attacker. However, it has much higher traversal overhead than a sequential algorithm, because each memory cell is accessed multiple times to cover the entire memory space of an attested node with a high probability. Based on the Coupon Collector's problem, on average $O(m \ln m)$ traversals are needed for the memory size of m; that is, $O(\ln m)$ traversals per cell. For Mica2 mote [2] with $128KB$ program space, this means 11.7 traversals per cell. We call this algorithm *cell-based* pseudorandom memory traversal, as shown in Figure 4.3(a). To reduce the overhead, we propose a *block-based* pseudorandom memory traversal algorithm (Algorithm 1), in which we traverse memory block-by-block instead of cell-by-cell and efficient 'XOR' operations are executed within blocks. As shown in Figure 4.3(b), each block has b continuous memory cells.

In Algorithm 1, the PRNG used for memory traversal is RC5 in counter mode. Taking a seed, each time RC5 outputs an 8-byte value which can be treated as four memory addresses, since each memory address is 2-byte (e.g., Mica2 mote). Therefore, from one RC5 computation we get 4 addresses for memory traversals, so that the total number of RC5 computations is actually $i_t/4$, if i_t is the number of traversal iterations to

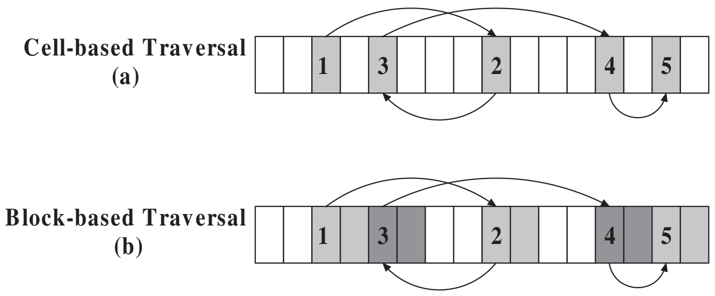


Figure 4.3. Cell-based and block-based ($b = 2$) pseudorandom memory traversal.

Algorithm 1 Block-based Memory Traversal Algorithm

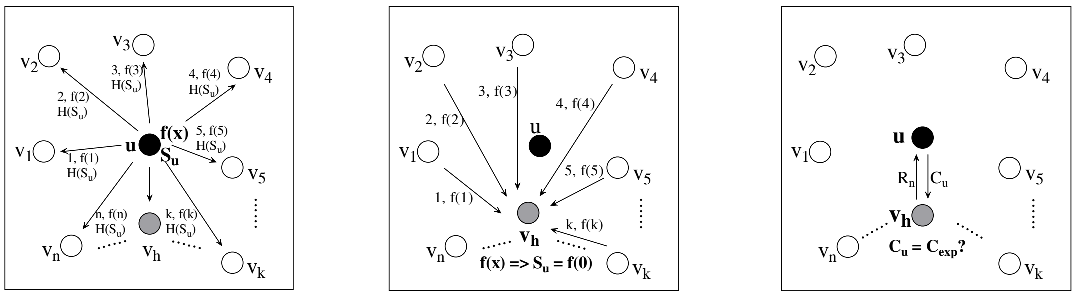
Input: i_t-number of iterations, j-current byte of checksum, b-block size, m-memory size, A-traversed memory address;
Output: a checksum $C(C_0, \cdots, C_7)$;
Procedure:

```
1: $C$ is initialized as a 64-bit random number;
2: $j$ is initialized to point to $C_0$, the first byte of $C$;
3: for $i = 1$ to $i_t/4$ do
4:     $(A_0, A_1, A_2, A_3) = RC5_i$;
5:     for $k = 0$ to $3$ do
6:         $C_j = C_j + (Mem[A_k \pmod m] \oplus \cdots \oplus Mem[A_k + b - 1 \pmod m])$;
7:         $j = (j+1) \mod 8$;
8:     end for
9: end for
10: return $C$;
```

cover the whole memory space. We call this seed to the PRNG as *memory-traversal seed*, which determines the order of pseudorandom memory traversals. Using the same memory-traversal seed, in the following attestation, the attestor and the attested node should get the same checksum results over the same memory content. Every time we construct a 16-bit pseudorandom address for memory read, we update the corresponding 8-bit checksum based on the bitwise 'XOR' result of a block of cells. This procedure is repeated until sufficient number of iterations are finished to access each memory cell at least once.

We have a theorem governing the number of iterations needed in Algorithm 1. (All the proofs could be found in Appendix.)

Theorem 4.2.1. *In block-based pseudorandom memory traversal, suppose b is block size, m is memory size, and random variable Y represents the number of traversal iterations needed to cover each memory cell at least once, then $E(Y) = O(\frac{m \ln m}{b})$ and $Pr[Y > \frac{cm \ln m}{b}] \le m^{1-c}$, where c is a constant factor.*



(a) Share distribution (b) Seed recovery (c) Attestation

Figure 4.4. Scheme I: A Basic Threshold Secret Sharing Scheme.

From Theorem 4.2.1, we can see that $i_t \approx O(\frac{m \ln m}{b})$. Also, this theorem indicates that if we traverse the node's memory in a block of b cells at a time, then the number of traversal iterations is $1/b$ of that in cell-based traversal to ensure each cell has the same level of possibility being accessed. Although the total number of cells to be traversed is still the same, the computational overhead is reduced through efficient 'XOR' computations. Note that cell-based traversal is a special case of block-based traversal where $b = 1$. This does not mean it is absolutely better if b is larger. If $b = m$ then we traverse all the memory cells sequentially by 'XOR' operations and update the checksum only once. This is less secure because there is a high probability of collision and an attacker may simply remember the XOR of all cells. Therefore, there is a tradeoff between performance and security, and we need choose an appropriate value of b. For Mica2 mote, every read of program memory returns 16 bytes instead of 1 byte, it is therefore recommended that we set b to be 16 (or its multiples).

4.3 Proposed Schemes

We propose two distributed software-based attestation schemes based on the block-based pseudorandom memory traversal algorithm and analyze their security properties.

4.3.1 Scheme I: A Basic Threshold Secret Sharing Scheme

There are three basic steps in the first scheme: *noise generation*, *secret share distribution*, *secret seed recovery and attestation*. First, the empty memory space of each node, say u, is filled with pseudorandom numbers derived from a unique noise-generation seed S_u. This is finished off-line before node deployment. Then, after node deployment, node u distributes one share of its noise-generation seed to each of its neighbors.

Later, when an attestation is triggered against node u, neighbors collaborate to recover S_u, reconstruct the memory image of node u, and attest its memory content through pseudorandom memory traversals. Since we already introduce the technique for noise generation in the previous section, next we discuss details of secret share distribution as well as secret seed recovery and attestation.

Secret Share Distribution We assume that it takes an attacker at least a time period T_{min} to compromise a sensor [92, 94]. Besides the noise that are filled into the empty memory space, the noise-generation seed is also preloaded into sensor's memory. After a sensor node u is deployed, e.g., by aerial scattering, it does the following:

- It discovers neighbors (e.g., by sending a HELLO message) and meanwhile it starts a timer which will expire after T_{min}. It also establishes a pairwise key with each neighbor on the fly[2].
- It splits S_u into multiple shares and sends a separate share to each neighbor; a hash value $H(S_u)$ computed by one-way hash function H is also included in the message, which enables every neighbor to easily verify the correctness of a recovered seed in the future while preventing a neighbor from deriving S_u from $H(S_u)$.
- When the timer expires, it removes S_u from memory.

If we rely on a trusted verifier or BS, it is relatively easy to recover the memory image and validate the response. In our setting, however, no nodes are absolutely trustable. To address this issue, we adopt Shamir's (k,n) threshold secret sharing [95, 96] for every node to distribute shares of the noise-generation seed to multiple neighbors, which will later collaborate in recovering the seed for attestation. In our case, n is the number of neighbors and k ($1 \leq k \leq n$) is a system threshold which relates to the network density and reflects a tradeoff between security and performance. More specifically, node u distributes secret shares to its neighbors as follows. First, node u randomly picks up $k-1$ constants denoted by $a_1, a_2, \ldots, a_{k-1}$ in a prime finite field Z_p and constructs a univariate polynomial $f(x) = S_u + a_1 x + a_2 x^2 + \ldots + a_{k-1} x^{k-1}$, where $S_u = f(0)$ is the noise-generation seed. Then, as shown in Figure 4.4(a), a tuple (i, f(i)) is distributed to neighbor v_i ($1 \leq i \leq n$) securely.

Note that we assume there exists a lower bound on the time interval T_{min} that is necessary for an adversary to compromise a sensor node [92], and that the time T_{est} for a

[2]Since then, all the messages transmitted between two nodes will be encrypted by corresponding pairwise keys, unless mentioned otherwise.

newly deployed sensor node to discover its immediate neighbors and to distribute secret shares of noise-generation seed is smaller than T_{min}. Based on real experiments, Deng et al. [94] showed that it is possible for an experienced attacker to obtain copies of all the memory and data of a Mica2 mote in tens of seconds or minutes after a node is captured. Zhu et al. [97] showed through experiments that T_{est} is in the order of several seconds for a network of a reasonable node density (up to 20 neighbors). Therefore, we believe that it is a reasonable assumption in practice that $T_{min} > T_{est}$. As a result, within time T_{min} after deployment, each node stores a pairwise key, a secret share and a hash value for every neighbor. These are kept in data space, so we do not need to verify this part of node u's memory.

Secret Seed Recovery and Attestation An attestation is triggered if a sufficient number (e.g., more than half) [3] of neighbors detect the abnormal behavior of node u. Then, all the neighbors $v_1, v_2, \ldots, v_n$ of node u elect a cluster head[4], denoted by $v_h (1 \leq h \leq n)$, among themselves based on an appropriate election algorithm [98]. The role of cluster head is rotated among all the neighbors for each attestation due to security and performance (load balance) reasons. The current cluster head v_h sends an authenticated challenge R_n, which is a random number, to node u. While node u computes the response over its own memory space by i_t traversals based on our block-based algorithm with R_n as the memory-traversal seed, v_h does the following:

- It collects k secret shares from neighbors of node u, so that the polynomial $f(x)$ is uniquely determined (by Lagrange interpolation) and S_u is recovered by evaluating this polynomial at 0, as shown in Figure 4.4(b).
- It verifies whether the recovered noise-generation seed is valid by comparing its hash value with the locally kept $H(S_u)$. If they do not match, it requests secret shares from another set of k neighbors until a correct noise-generation seed is reconstructed.
- It locally computes the expected memory traversal checksum C_{exp}, based on S_u and R_n. Specifically, from the recovered noise-generation seed S_u, it knows the expected memory image of node u (our counter-mode based noise generation enables it to readily regenerate the noise for each empty space traversal upon node u).

[3]In this way, few number of colluded malicious neighbors cannot accuse an honest node u by exhausting its energy for attestation.

[4]It is called cluster head hereafter although there are no topological clusters here.

Then, taking R_n as the memory-traversal seed, our block-based memory traversal algorithm outputs the correct checksum C_{exp} after i_t traversals.

- It compares C_{exp} with the responded checksum C_u from node u and concludes that the interrogated node u has been compromised if these two checksums are different, as shown in Figure 4.4(c).

Security Analysis We analyze the security properties of Scheme I in terms of the detection rate (i.e., the probability for neighbors to successfully detect a compromised node). First, we have a lemma discussing under what conditions Scheme I is able to detect a compromised node.

Lemma 4.3.1. *Scheme I is able to successfully detect a compromised node u, if: (1) cluster head is trustable; (2) cluster head obtains $\geq k$ correct shares of noise-generation seed S_u from neighbors of node u; (3) the attacker does not obtain $\geq k$ shares to recover the seed S_u.*

Next, we derive the detection rate of Scheme I, based on Lemma 4.3.1.

Theorem 4.3.2. *Assuming that the probability for each node in the network to be compromised is the same and equals to $p_0 (0 < p_0 < 1)$, then the detection rate of Scheme I is*

$$P_{bs}(k,n) = \begin{cases} \sum_{i=k-1}^{n-1} \binom{n-1}{i} (1-p_0)^{i+1} p_0^{n-1-i}, & \textit{if } n < 2k \\ \sum_{i=n-k}^{n-1} \binom{n-1}{i} (1-p_0)^{i+1} p_0^{n-1-i}, & \textit{if } n \geq 2k. \end{cases}$$

When $n = 15$, the detection rate of Scheme I is shown in Figure 4.5, from which we can see that if p_0 is larger (i.e., nodes in the network have a higher possibility to be compromised), then P_{bs} is lower. In addition, suppose p_0 is fixed, then P_{bs} has a peak value with $k = \frac{n+1}{2}$ (if n is an odd integer) or $k = \lceil \frac{n+1}{2} \rceil, \lfloor \frac{n+1}{2} \rfloor$ (if n is an even integer), and P_{bs} is symmetric when k is larger or smaller than these values. This means when we choose the values of k and n we should make k as close to half of n as possible so that Scheme I has a higher detection rate. For example, suppose p_0 is 0.05 and $n = 15$, the detection rates when k equals to 7, 8 are both about 95%.

Scheme I could be enhanced if we are able to prevent the attacker from obtaining more than k secret shares. We can adopt proactive secret sharing [99] to periodically update shares of S_u without changing S_u itself. As a result, even if an attacker has managed to obtain as many as $k-1$ secret shares (the threshold is k), a proactive share renewal process will render them useless. Specifically, after a certain time period, a

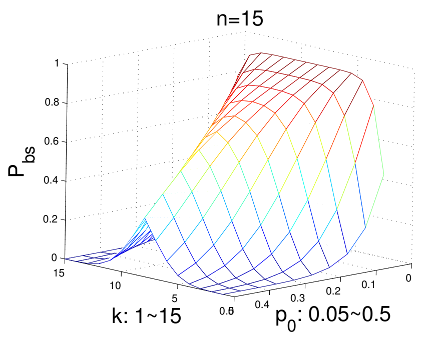


Figure 4.5. Detection rate of Scheme I.

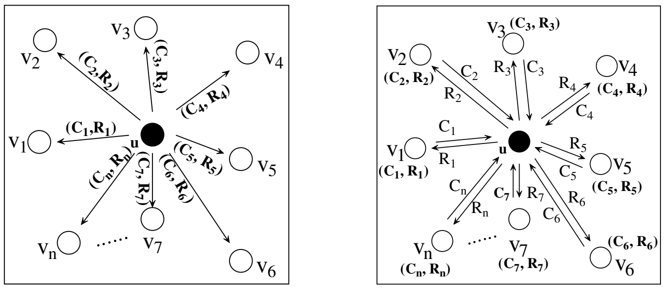
$(k-1)$-degree polynomial $g(x)$ is randomly selected by node u over Z_p, satisfying that $g(0) = 0$. After the neighbor v_i receives the distributed share $(i, g(i))$, it renews the share it kept by $(i, f(i) + g(i))$. Since $f(0) + g(0) = S_u + 0 = S_u$, the noise-generation seed S_u is not changed upon this renewal.

We note that in this scheme a compromised neighbor may contribute an erroneous share. The cluster head will need to pick another set of k shares to redo the seed recovery, thus consuming additional energy. This problem could be solved if a node has some additional data memory space. For example, besides storing $H(S_u)$, each neighbor of node u also stores a hash of every secret share. This will allow it to easily verify the received shares from other neighbors when it becomes the cluster head, thus compromised neighbors are deterred from contributing wrong shares. We will investigate more practical tradeoffs in our future work.

4.3.2 Scheme II: A Majority Voting Based Attestation Scheme

In Scheme I, the cluster head can reconstruct the noise-generation seed and hence has a way to know the expected memory content of the attested node. Based on this knowledge, the cluster head is able to send a random challenge at will for each attestation. This prevents the attested node from precomputing the response based on prediction. However, as we have seen, the compromise of the cluster head may result in a wrong decision about the attested node. Also, once a noise-generation seed is disclosed (e.g., due to neighbor compromises), an attacker may replace the noise with malicious code in a sensor that is later compromised.

To address the above problems, we consider two strategies. First, instead of relying on some specific cluster head to make decisions, we make use of a majority voting scheme among neighbors. Second, instead of distributing and recovering the noise-



(a) Information distribution (b) Attestation

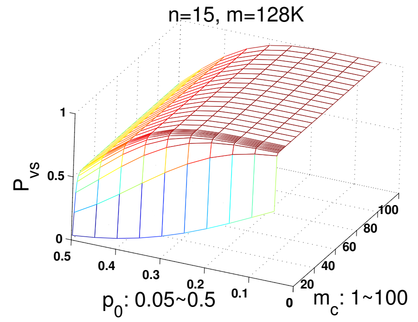
Figure 4.6. Scheme II: A Majority Voting based Attestation Scheme.

generation seed, each neighbor is distributed with and keeps a challenge as well as the corresponding response. During an attestation, each neighbor sends the challenge and waits for the response from the attested node. If the received response is different from the local one, then the attested node is considered compromised.

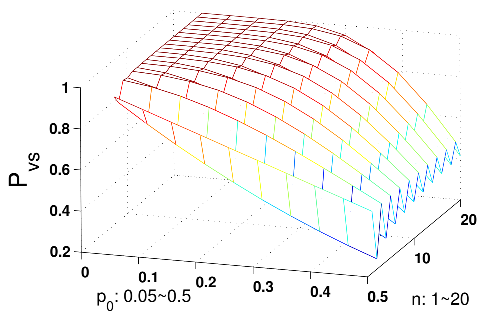
The advantages of this scheme are three-fold. First, neighbors do not need compute any responses locally, greatly reducing the computational overhead involved in the attestors. Second, although a compromised neighbor knows which cells it will traverse, it does not know which cells will be traversed by other neighbors. Hence, the attacker cannot decide which cells are safe to modify for sure. Third, based on majority voting, an innocent node will not be identified as compromised due to one or a few compromised neighbors. Next, we provide the details of information distribution and attestation in this scheme.

Information Distribution Before deployment, noise is preloaded into nodes' empty memory spaces. Each node is also preloaded with n tuples of (C_i, R_i) ($1 \leq i \leq n$, n is no less than the estimated maximum number of neighbors), where C_i is a challenge and R_i is the corresponding response. Each tuple is generated by an offline server as follows. Taking a random challenge C_i as the memory-traversal seed, node u's memory is traversed i_t times based on our block-based pseudorandom memory traversal algorithm and a checksum R_i is returned. After deployment, every node (say u) discovers neighbors, securely delivers to each neighbor $v_i(1 \leq i \leq n)$ a randomly picked tuple (shown in Figure 4.6(a)) within T_{min}, and finally erases all the tuples after T_{min}.

Note that the number of traversal iterations i_t in this scheme is different from that in the previous scheme. Suppose that the memory size of node u is m, then i_t should be $O(\frac{m \ln m}{bn})$ or above, so that all the neighbors may corporately traverse each of node u's



(a) P_{vs}: different m_c, fixed n



(b) P_{vs}: different n, fixed $m_c (> 0)$

Figure 4.7. Detection rate of Scheme II.

memory cells at least once.

Attestation Later on, if more than half of neighbors agree to attest node u, neighbors attest node u in sequence (to prevent channel collisions), as shown in Figure 4.6(b). Specifically, each neighbor v_i securely sends node u its challenge C_i and waits for the response. Then, taking the challenge C_i, node u traverses its memory accordingly based on the block-based pseudorandom memory traversal algorithm with i_t iterations, and reports the resulted checksum as the response. Note that sequence numbers must be added to both the challenge and response messages before encryptions to prevent replay attacks. After that, neighbor v_i compares this checksum with the one it keeps locally (R_i) and makes its own decision about node u. Finally, if the number of neighbors who have negative opinions exceeds the majority (i.e., $\lceil \frac{n+1}{2} \rceil$), then node u will be identified as compromised. In this way, each neighbor of node u has its independent judgement about node u's memory integrity.

Security Analysis According to the Byzantine generals problem [86, 87], if a compromised neighbor cannot modify opinions from other neighbors (i.e., a faulty neighbor may lie on its own behalf, or refuse to relay results received from others, but it cannot alter other's results without betraying itself as faulty), then all the honest neighbors can finally reach an agreement about node u's code memory integrity by employing authentication mechanisms. In our case, the pairwise keys shared and known only between two nodes could provide the same functionality.

The choice of i_t (i.e., the number of memory traversal iterations) reflects a tradeoff between performance and accuracy: if i_t is larger, then the overhead is higher, but the decisions from neighbors is more reliable, because if $i_t = O(\frac{m \ln m}{b})$, then every neighbor will make node u traverse each memory cell at least once with a high probability and hence obtain a sound opinion about node u's memory. However, the total traversal cost involved in node u will be n times of that in Scheme I. On the other hand, if we reduce the number of iterations (e.g., to less than $O(\frac{m\ln m}{bn})$), it is possible that neighbors fail to detect the modified part of the attested node's code memory.

The detection rate of Scheme II is formalized by the following theorem.

Theorem 4.3.3. *Assuming that the probability for each node in the network to be compromised is the same and equals to* $p_0 (0 < p_0 < 1)$. *For Scheme II with regard to node u, suppose* m_c *is the number of changed memory cells of node u, m is node u's memory size, and we choose* $i_t = \frac{m\ln m}{bn}$, *then the detection rate of Scheme II is* $P_{vs}(n, m_c, m) = \sum_{i=\lceil\frac{n+1}{2}\rceil}^{n}[\binom{n}{i}(1-p_0)^i p_0^{n-i}\sum_{j=\lceil\frac{n+1}{2}\rceil}^{i}\binom{i}{j}p_h^j(1-p_h)^{i-j}]$, *where* $p_h = 1-(\frac{m-m_c}{m})^{\frac{m\ln m}{n}}$ *is the probability for an honest neighbor to detect the compromised node u.*

The detection rate of Scheme II with different m_c and fixed n is shown in Figure 4.7(a). Even if the attacker changes only 3 bytes, P_{vs} is already increased to 99% when $p_0 = 0.05$. In this case, p_h is also as high as 90%. The detection rate of Scheme II with different n and fixed m_c can be seen in Figure 4.7(b). P_{vs} vibrates as the total number of neighbors n changes, since n alternates between odd and even numbers in majority voting.

4.4 Performance Evaluation

We first analyze the performance of our schemes, then we conduct simulations to show that our schemes can effectively detect changed memory content of sensor nodes.

4.4.1 Performance Analysis

We quantify the overhead of our schemes from three aspects: computation, communication and storage.

Computational Cost In Scheme I, for the attested node, there are n $(k-1)$-degree polynomial evaluations and one hash computation to distribute shares to n neighbors. Also, there are $O(\frac{m\ln m}{b})$ memory traversals involved during attestation. For neighbors, to recover and validate the noise-generation seed, there are one $(k-1)$-degree polynomial interpolation, one $(k-1)$-degree polynomial evaluation and one hash computation. In attestation, $O(\frac{m\ln m}{b})$ memory traversals are needed to cover the whole memory of the attested node. In Scheme II, since (challenge, response) pairs are generated offline, there are only $O(\frac{m\ln m}{b})$ memory traversals involved in the attested node.

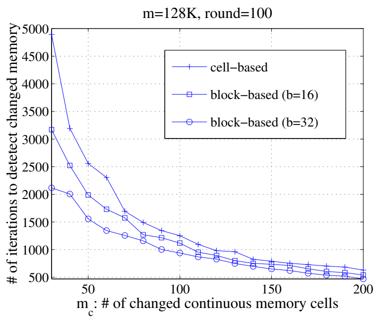


Figure 4.8. Number of iterations for cluster head to detect changed cells in Scheme I.

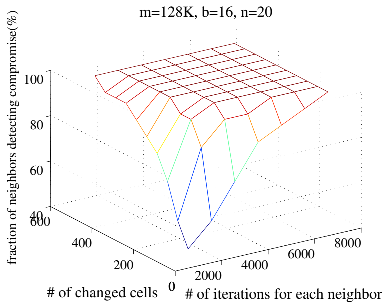


Figure 4.9. Fraction of neighbors successfully detecting changed cells in Scheme II.

Communication Overhead Suppose L_s, L_h and L_c are the lengths of a secret seed or share (they have the same length in our schemes), hash value, and checksum, respectively. In Scheme I, the message overhead includes: secret share distribution of $n(L_s + L_h)$, secret seed recovery by the cluster head of at least $(k-1)L_s$, and the attestation overhead of $L_s + L_c$. For Scheme II, the message overhead includes: challenge and response distribution of $n(L_s + L_c)$ and the attestation overhead of $n(L_s + L_c)$.

Storage Requirement In Scheme I, each node needs to store secret shares for its n neighbors with the storage cost of $n(L_s + L_h)$ bytes. Also, it costs neighbors of the attested node $n(L_s + L_h)$ bytes to store all the secret shares. In Scheme II, it costs neighbors $n(L_s + L_c)$ bytes to store all the challenges and responses for the attested node.

Comparison of Two Schemes Obviously, Scheme II has lower computation overhead than Scheme I. The length of checksum is 8-byte in our algorithm ($L_c = 8$ bytes). Suppose we choose L_s and L_h to be 8-byte, too. Then, the storage overhead of these two schemes are almost the same. Moreover, Scheme II has higher communication overhead than Scheme I.

From the above analysis and comparison, we have the following observation and conclusion. Scheme II has better security properties, but it also has higher communication overhead. Scheme I has lower communication overhead but relatively higher computational cost. As such, which one to use should be based on specific resource configurations and application requirements.

4.4.2 Simulation results

We conduct simulations to verify the effectiveness of our schemes in detecting memory cells changed by the attacker. The attacker in our simulation randomly selects a memory cell, beginning from which the attacker changes the content of m_c continuous memory cells. From the simulation results we find that in practice the overhead to detect changed memory cells can further be largely reduced because the number of traversal iterations needed to detect the modification is actually much less than expected. The number of iterations to access each memory cell at least once could be used to detect even one byte of memory change. In practice, however, the attacker needs many more continuous memory cells (e.g., several hundred bytes) to inject malicious code that can really do harm.

In the simulation of Scheme I, the size of modified memory content m_c is changed from 30 to 200 in bytes and each point obtained in the figure is the average value of 100 rounds. As shown in Figure 4.8, although the number of iterations needed to cover the entire memory space is $O(m \ln m) = 1505K$ when memory size m is $128K$, in Scheme I the actual traversal iterations for the cluster head to detect changed memory content with size as small as 30-byte is only about 3200 if block size $b = 16$. If we increase the block size to 32, then the number of iterations to detect this change is even smaller: about 2100 iterations to detect the changed 30 bytes, while in this case we need 4900 iterations in cell-based traversal.

In the simulation of Scheme II, we fix the memory size m to be $128K$, block size b to be 16, and the number of neighbors n to be 20. With the number of changed continuous memory cells varying from 50 to 500, we examine the fraction of neighbors that can successfully detect the modified memory part, under different number of traversal iterations for each neighbor (as shown in Figure 4.9). When the size of changed memory is small (e.g. $m_c = 50$), the value of $\frac{O(m \ln m)}{bn} = 6272$ gives a good lower bound to make sure every neighbor can detect the change. Nevertheless, if we have some knowledge on the number of changed memory cells, we may adjust or even reduce the number of memory traversal iterations accordingly. In fact, when 500 bytes of memory have been modified, there are only 1000 traversal iterations needed to guarantee that all the neighbors can detect this modification. In this case, to tolerate a certain number of compromised neighbors (e.g., less than half), majority voting is a good idea as long as more than half of neighbors are honest.

4.5 Further Discussions

Next, we discuss some important issues regarding our schemes.

4.5.1 Inaccuracy in Neighbor Number Estimation

In Scheme II, our discussion implicitly assumed that a node knows in advance the number of neighbors n when preloaded the challenge/response pairs. In real networks, the actual number of neighbors may be different from what we have predicted. As such, we should adjust i_t dynamically so that the cell traversal probability and the number of iterations do not vary much. One simple approach to addressing this is as follows. Suppose we know that the maximum number of neighbors in the network is $n_{max} = 16$, then node u is preloaded with n_{max} challenge/response tuples. If the actual number $n = 9$, node u may send $\lceil \frac{n_{max}}{n} \rceil = 2$ tuples to each neighbor. Later on during an attestation, u will make two traversals for each neighbor. Note that after node u erases these n_{max} tuples, some empty space will be created. As such, instead of zeroizing the space, node u could fill it with some noises that are generated on-the-fly based on the noise generation seed and then erase the seed. For correctness, the offline generation of n_{max} responses should be based on these noises, not the challenge/response tuples.

4.5.2 Topology Changes

In the description of our schemes, we mainly consider a static network model. For some sensor networks, nodes may be added and die during the network lifetime. Here we discuss the influence of network topology changes on our schemes.

Node removal is less a concern in our schemes than node addition. Scheme I works well if some neighbors die, as long as the number of neighbors is still more than the threshold value. The removal of neighbors is not a problem in Scheme II as long as the fraction of honest neighbors is still larger than one half. When a new node is added, it discovers its neighbors and then distributes its shares to the neighbors, following the same procedure as before. The challenge arises from getting shares from its neighbors which have done their secret share distributions. According to our protocols, these neighbors have erased their own noise seeds, thus they do not have any additional share to give out to this new node.

To (partially) address the problem, we may apply the following idea. Node deployment is divided into multiple intervals and there is an interval key for each interval. Nodes deployed in interval T_i carry the interval key K_i as well as all the past interval keys. When a node is deployed, it generates more shares than its actual number of neighbors; the extra shares could be encrypted with its interval key and store locally. The node erases the interval keys after T_{min}, thus it cannot decrypt the share itself. However, a new node deployed either in the same time interval or later carries the interval key used in the encryption, so it can decrypt the share. In this way, a new node may collect one encrypted share from each neighbor and joins the neighbor for later attestation operations. Note that the security assumption here is the same as in our schemes; that is, a node will not be compromised within T_{min}. We will study this approach in more details and investigate more secure solutions in our future work.

4.5.3 Attacks in Attestation

There are potential attacks such as compression attack existing in the schemes. In compression attack, the attacker compresses the original code to make some space for the malicious code. For example, a 57KB main.srec file could be compressed to 20KB by WinRAR 3.71. The compression ratio is 35% (data compression ratio is defined as compressed size/uncompressed size [100]). If the size of malicious code is less than 37KB, then the program memory of sensors could accommodate such malicious code. Based on the malicious code, the compromised node could launch a lot of attacks. The lower the compression ratio is, the more powerful the attack may be, depending on the compression tool the attacker employs. When it comes to the attestation, the attacker could decompress to the original code and respond with correct traversal checksum. This is a hard problem and we will try to solve it in our future work.

Chapter 5

Improving Sensor Network Immunity under Worm Attacks through Software Diversity

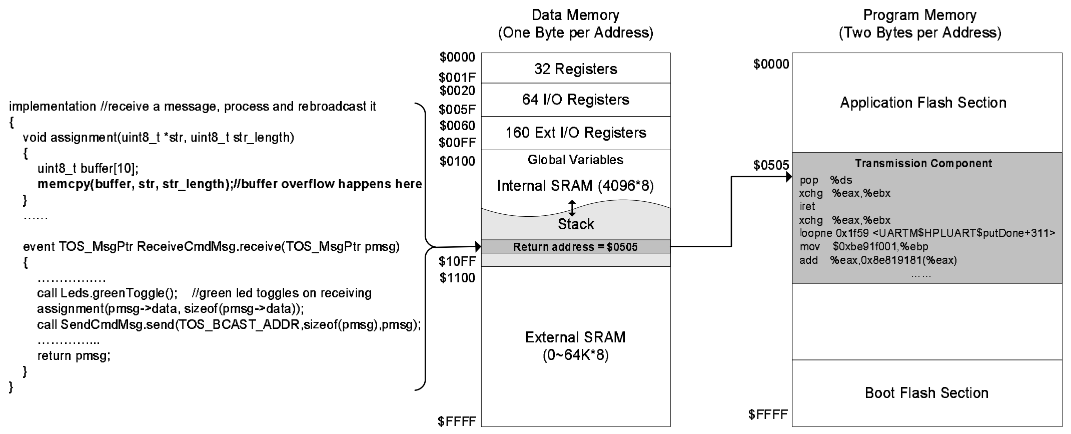
5.1 Sensor Worm Attacks

In this section, we first introduce the memory architecture of sensors. Then, we illustrate the feasibility of launching sensor worms through experiments. At last, we model the propagation of sensor worms using a simple epidemic model.

5.1.1 The Memory Architecture of Sensors

Observing that most of the Internet worms exploit the buffer overflow vulnerabilities in software, we start by studying the buffer overflow problem in sensor nodes. Based on our initial investigation, we find that buffer overflow in sensors is memory architecture dependent. The 8 and 16-bit microcontrollers used in small embedded systems are designed in either of two memory architectures: the Von Neumann Architecture which uses a single physical memory for both program code and data, and the Harvard Architecture which uses separate memories for program and data.

Most of the buffer overflow attacks in the Internet target at the Von Neumann Architecture. By overflowing the boundary of a buffer in the stack or heap, a worm injects a piece of malicious code into a program. If it succeeds, it will cause the return address of



```
implementation //receive a message, process and rebroadcast it
{
   void assignment(uint8_t *str, uint8_t str_length)
   {
      uint8_t buffer[10];
      memcpy(buffer, str, str_length);//buffer overflow happens here
   }
   ......

   event TOS_MsgPtr ReceiveCmdMsg.receive(TOS_MsgPtr pmsg)
   {
      ................
      call Leds.greenToggle();    //green led toggles on receiving
      assignment(pmsg->data, sizeof(pmsg->data));
      call SendCmdMsg.send(TOS_BCAST_ADDR,sizeof(pmsg),pmsg);
      ...............
      return pmsg;
   }
}
```

```
pop    %ds
xchg   %eax,%ebx
iret
xchg   %eax,%ebx
loopne 0x1f59 <UARTM$HPLUART$putDone+311>
mov    $0xbe91f001,%ebp
add    %eax,0x8e819181(%eax)
......
```

Figure 5.1. Buffer overflow experiment on memory structure of ATmega128.

a function to be overwritten; further, the instruction pointer will point to the location of the injected code and start to execute the malicious code. Therefore, if a sensor network consists of sensor nodes that use the Von Neumann architecture (e.g., microcontroller msp430 of Texas Instrument [101]), it is also vulnerable to such worm attacks if the program does not perform careful boundary checking.

The Harvard architecture, which is used by vendors like Atmel and Microchip, makes it hard (if not impossible) to inject malicious code, because the program memory and the data memory are separate (Figure 5.1). The data memory is not executable and typically different instructions are required to access the program and data memory. Mica2 motes [2] use the 8-bit microcontroller ATmega128 from Atmel [102]. The stack residing in the internal SRAM of the data memory is mainly used for storing temporary data, local variables and return addresses for subroutine calls and interrupts. The base of the stack (normally at 0x10FF) is defined in the RAMEND value of the include file (m128def.inc) for ATmega128 and the stack pointer is initialized to be 0x10FF too. This means when we push return address (two bytes) into the stack, we need two push instructions. The stack size is increased by two but the address in stack pointer will be decreased by two. Therefore, the maximum size of the stack equals to the size of the internal SRAM, which is 4KB.

The lowest addresses (around 0x0100) in the internal SRAM are normally allocated for the .bss section including uninitialized global or static variables (the size of this part is calculated for bytes in RAM during compilation).If the stack is overflowed by a large quantity of data, the stack size may grow to overflow the .bss section or even the register parts in the data memory. Thus, unexpected values will be set to the system variables and the registers. This kind of stack overflow, however, can only corrupt the current sensor and the attack will not propagate to other sensors automatically. Hence, during the experiment in the next section, we will focus on another kind of buffer overflow, which could cause the propagation of the attack in the network.

5.1.2 Trial Experiments on Sensor Worms

As a trial study, we are interested in stack-based buffer overflow, which causes the transfer of program flow by changing the return address of a function call. If a senor node has a routine that processes a received packet before relaying them, then by manipulating the content in the packet an attacker may change the return address of the function call to the beginning address of a transmission component in the program memory. Consequently, the sensor sends out the attack packet before it is corrupted. If the attack packet is supposed to reach all nodes (e.g., a broadcast packet), all the sensors will be affected.

To study the consequences of this type of buffer overflow, we performed some simple experiments on Mica2 motes. Suppose that the data structure of the message type contains a pointer to its data payload (string). Figure 5.1 shows the code in a sensor. When a sensor receives a message, it triggers the function Leds.greenToggle() to turn on/off the green LED. The sensor then calls a function assignment(), which copies the data payload to a buffer of size 10 without bound checking. Finally, it rebroadcasts the same packet to the network, and toggles a red LED after a successful transmission.

To start the message propagation process, one sensor broadcasts an attack packet, the data field of which is the beginning address of the transmission program in the sensor (which is 0x0505 in our case). In our experiments, we found that an extra 7-byte in the string is long enough to overflow the return address. Thus, by sending a packet with a 17-byte string in the data payload and setting every byte 0x05, the 2-byte return address of a receiving sensor will be changed to 0x0505.

When buffer overflow does not happen (the function assignment() is disabled), the red LED and green LED flash alternately on these two sensors, which means that two sensors operate normally and the packet is echoed between these two sensors. However, after the function assignment() is enabled, buffer overflow occurs and the return address is modified to point to the transmission component. Thus, both sensors become irresponsive after echoed the received packet once more (their red LED and green LED flash no more).

The above is a trial for constructing a sensor worm. Another possibility is to directly inject malicious code into the program memory through mechanisms such as over-the-air software distribution or network reprogramming [103]. Since the injected malicious code could be arbitrary as the attacker desires, it may bring more severe network-wide damages to the sensor network. We notice that sensor worm may adopt other vulnerabilities than buffer overflow to launch successful attacks. How they could be used is out of the scope of this paper. We may investigate more such possibilities in our future work.

5.1.3 The Modeling of Sensor Worm Attack

We use the classical simple epidemic model [72] to model the propagation of sensor worms when no defense is in place. Here sensors have two states: susceptible and infectious. The overall rate of new infections given by this model is:

$$\frac{dI(t)}{dt} = \frac{\beta I(t)(n - I(t))}{n}, \tag{5.1}$$

where $I(t)$ is the number of infectious nodes at time t, n is the total number of nodes in the network. β is worm's infection rate, which is the average number of probes an infectious node can send out to the population n during a unit time. Specifically, if we consider the topology of the sensor network as a graph, β is the average out-degree of a node. By solving this differential function, we can get the number of infectious sensors at any time t as:

$$I(t) = \frac{n}{1 + e^{-\beta t} C n}, \tag{5.2}$$

where C is a constant factor.

In Section 5.3.2.5, we will show the effectiveness of our proposed scheme in terms of infection fraction by comparing it with this no-defense case.

5.2 Sensor Worm Defense

Message source authentication can block sensor worms from outsider attackers, but it is not much helpful if sensor worms are injected by compromised sensors or base station. Our focus is to increase the survivability of sensor networks against sensor worms. In the spirit of *survivability through heterogeneity* philosophy [104], we will explore *software diversity* to combat sensor worms.

By software diversity, we should resort to multiple realizations (by different people) of the critical programs such as routing protocols and operating systems, then install each sensor node with one of the versions. We expect that different versions of the same functionality will not have the same vulnerability for the attacker to exploit, e.g., the beginning addresses of the transmission program in different implementations are different. On the other hand, due to the implementation cost it is unrealistic to assign every node a different version of a program. Therefore, the research challenge becomes: *given a limited number of versions of a program, how to optimize our global benefits in defending against sensor worms through software diversity?* Next we first present an efficient algorithm for version assignment (before deployment), followed by studying the impact of sensor deployment error (after deployment).

5.2.1 Graph Construction

Graph coloring (especially vertex coloring) [74], a famous problem in graph theory, tries to assign colors to the vertices of a graph such that no two adjacent nodes in the graph share the same color. We may transform our problem into a graph coloring problem by first mapping a sensor network into a graph and mapping colors to the software versions(in the following, we use color and software version interchangeably), and then finding a solution to the corresponding graph coloring problem.

When mapping a sensor network into a graph, an intuitive solution is to map every sensor node into a vertex in the graph [67]. This, however, is not suitable for sensor networks, which usually have high density (e.g., a node may have 20 or more neighbors). A sensor network with high node density results in a high-density graph, making it infeasible to color the graph by using just a few colors. To address this problem, we will consider the geographic locations of sensors for mapping. Specifically, we will first partition the sensor field into small cells, then map every cell into a vertex and map the neighborship of two cells into an edge in the graph. In this model, cell is the unit of consideration and multiple sensors may locate in the same cell. Thus, when we assign a color to a cell, all the nodes in the cell will have the same color. Although in this case once a cell color is compromised more than one sensor will be corrupted, our focus is to

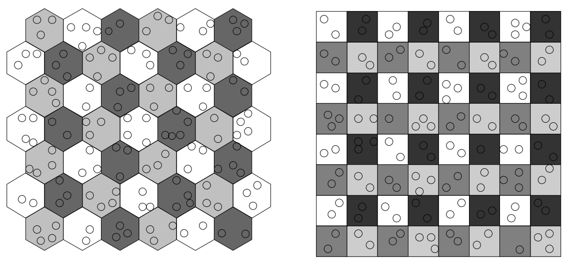


Figure 5.2. Examples of color assignment.

quarantine the propagation of sensor worms so that large-scale node compromises will be prevented. Note that our scheme is suitable for applications in which a small fraction of node compromises may be tolerated.

5.2.2 Color Assignment

Before we describe the color assignment algorithm, we first introduce several concepts [67]. In a graph, an edge whose two endpoints having the same color is called a *defective edge*. Otherwise, it is called an *immune edge*. A *disconnected component* is a subgraph inside which all vertices are connected through defective edges whereas all its boundary edges are immune edges. Formally, suppose the set of cells in the network is denoted by V, the set of software is S and the number of software versions is $s = |S|$. Our purpose is to devise an assignment or mapping $S \mapsto V$, so as to (1) minimize the number of defective edges and (2) maximize the number of disconnected components. Indeed, these two goals are not orthogonal. This is because an increase in the total number of disconnected components will reduce the size of each component as well as the number of defective edges.

According to Four Color Theorem, we only need a limited number of colors (e.g., no more than four) to implement our scheme. This means that our scheme is very practical because a very limited number of implementations for the critical function are sufficient to solve our problem. More specifically, our problem is to find out a color assignment to the graph so that $S(i) \neq S(j)$ if $(i, j) \in E$, given the topology of a graph $G = (V,E)$ and the available colors S. In the case that the number of available colors is larger than the optimally minimum one, the color assignment solutions are often not unique. We try to devise a color assignment algorithm that provides the flexibility of automatically outputting one of the viable solutions in an efficient way.

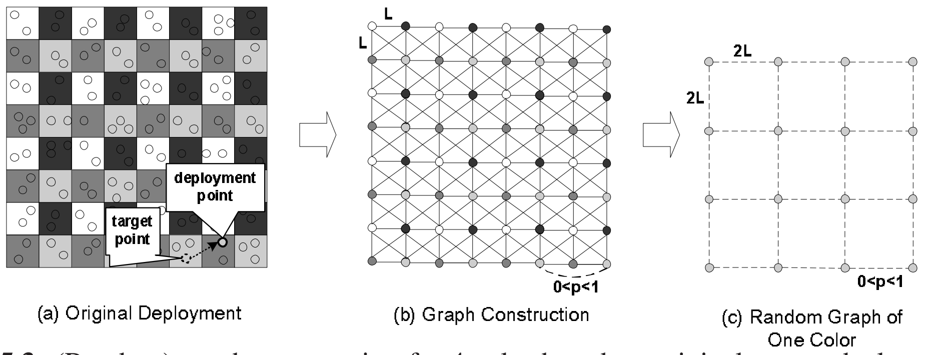


Figure 5.3. (Random) graph construction for 4 color based on original sensor deployment and color assignment.

An intuitive solution is a greedy graph coloring algorithm. This algorithm as well as its improvement heuristics (e.g., first order vertices by decreasing or increasing degree) can find a solution fast, but their results are largely influenced by the order in the permutation of vertices. That is, in some circumstance, the greedy algorithm may have a conclusion that this graph cannot be colored by a certain number of colors, but once we change the vertex traversing order the greedy algorithm may successfully output a color assignment solution.

Our color assignment algorithm is based on backtracking [105]. Backtracking is a type of algorithm that is a refinement of the brute force search. In backtracking, many partial solutions can be eliminated without being explicitly examined, according to the properties of the problem [106]. In our case, once a vertex is assigned a color, then all its neighbors are refrained from being assigned the same color. It is a recursive procedure, independent of the vertex traversing order. In each recursion, the problem is turned to be a smaller problem with the same form. This process is repeated until every vertex is assigned a color or the procedure stops because of failing to assign a color to one of the vertices. The details of our backtracking color assignment algorithm is presented in Algorithm 2.

Let N be the total number of cells in the deployment area, which means that there are N vertices in the constructed graph. Clearly, time complexity of the brute force algorithm is $\Theta(s^N)$, because for each vertex from 0 to $N-1$ there are s choices. The backtracking procedure is a depth-first search on the solution space tree, which is a balanced tree with degree s and height N. Each internal node on the tree is to find the next available color by calling function availableColor(), with the time complexity of $O(sN)$. Our algorithm searches along a path from the root to the leaf in the tree, with the total

Algorithm 2 Backtracking Color Assignment Algorithm

Input: Adjacency matrix $G[0..N-1][0..N-1](N = |V|)$ of graph $G = (V, E)$, where $G[i][j] = 1$ if $(i, j) \in E$ and $G[i][j] = 0$ otherwise; Available colors are represented by integers $1..s$ where $s = |S|$;

Output: A color assignment solution represented by an array X, where $X[i](0 \leq i \leq N-1)$ is the color assigned to vertex i;

Procedure:

```
for $i \leftarrow 0$ to $N-1$ do
    $X[i] \leftarrow 0$; {Initialize array $X$}
end for
backtracking(0); {Array $X$ is updated here}

void backtracking (int k)
loop
    availableColor($k$);
    if $(!X[k])$ then
        break;{No new color for vertex $k$ is available}
    end if
    if $(k == N-1)$ then
        printNodeColors();{A solution of array $X$ is outputted}
        exit(0);
    else
        backtracking($k+1$);
    end if
end loop

void availableColor (int k)
loop
    $X[k] \leftarrow (X[k]+1)\%(s+1)$;{Try the next color}
    if $(!X[k])$ then
        return;{All colors have been tried}
    end if
    for $i \leftarrow 0$ to $N-1$ do
        if $(G[k][i] \&\& (X[k] == X[i]))$ then
            break;{Vertices sharing an edge cannot have the same color}
        end if
    end for
    if $(i == N)$ then
        return;{A new color is found}
    end if
end loop
```

number of internal nodes as N. Hence, the time complexity of the backtracking algorithm is $N \cdot O(sN) = O(sN^2)$. This polynomial time is much less than the exponential time of the brute force method.

We check the effectiveness of the above algorithm by applying it to the 3- and 4-color cases (the 2-color case is simply a column-based partition of the sensor field). For the 3-color case we may partition the sensor field using hexagons because its corresponding graph is 3-colorable. If we have four or more colors, then grid-based partition is applicable, because its corresponding abstract graph is 4-colorable. As shown in Figure 5.2, we construct one example of the solutions for each case, based on the outputs from Algorithm 1. We notice that both the 3-color and 4-color cases are optimal with minimum number of defective edges (which is zero) and maximum number of disconnected component (i.e., each cell is a disconnected component). To ease our presentation, our following discussion is mainly based on the graph topology of 4-color case in Figure 5.2(b).

The next research issue is: what should be the size of each cell? To answer this question, we need to take into account the transmission range of sensors, say R. Let the shortest distance between two cells of the same color be L. Clearly, we should make sure that $L > R$, so that there is no defective edge between them. As shown in Figure 5.2, L is actually the edge length of each cell, so the edge length of each cell should be larger than the transmission range R.

5.2.3 Sensor Node Deployment

So far we have been focused on color assignment in the ideal planning phase; we have not discussed how sensor nodes are deployed in a sensor field and whether a real deployment may cause issues. In real applications, there could be many ways to deploy sensor nodes in a field, subject to factors such as the requirement of the application and the safety of the deployment environment. In one extreme, we may be able to place every sensor node precisely in a planned location. If so, we may directly apply our color assignment algorithm. This however is unlikely for deploying a large-scale sensor network. In the other extreme, the real location of a sensor node is completely random in the sensor field. In this case, multiple sensors with different colors might be dropped in the same cell, rendering our algorithm falling back to a random coloring algorithm [67]. Inspired by the group-based sensor deployment model in [107, 108], we will study the

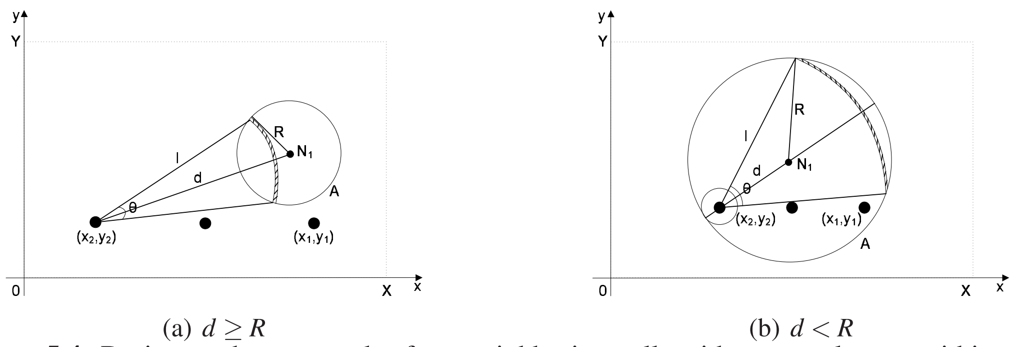


Figure 5.4. Derive p_0 that two nodes from neighboring cells with same color are within transmission range.

case that the distance between the actual location of a sensor node and its targeted location follows a two-dimensional normal distribution (although some other probabilistic distributions may be considered as well). This is a reasonable model because in real applications sensors are normally prearranged according to their targeted locations and are then dropped out from a moving vehicle group by group.

More specifically, suppose that the deployment area $(X \times Y)$ is divided into $h \times v$ cells and $(i, j)(1 \leq i \leq h, 1 \leq j \leq v)$ is the cell in row i, column j. If the target point of cell (i, j) is (x_i, y_j), we have the mean of sensor location distribution from this cell as $\mu = (x_i, y_j)$. The position (x, y) of sensors in this cell follows the pdf (probability density function) of a two-dimensional normal distribution $f_{i,j}(x, y)$:

$$\begin{aligned} f_{i,j}(x,y) &= f(x-x_i, y-y_j) \\ &= \frac{1}{2\pi\sigma^2} e^{-[(x-x_i)^2+(y-y_j)^2]/2\sigma^2}, \end{aligned} \tag{5.3}$$

where σ is the standard deviation of the distribution.

Clearly, the deployment error of sensor nodes will have some impacts on our objective. If two cells with the same color are separated by one cell of a different color according to our planning, there will be a chance that these two cells are connected due to deployment errors. In our setting, with some simplification(i.e., we do not consider cells in diagonal as neighboring in the random graph because they are farther away than cells neighboring in horizontal or vertical), we could construct a random graph for each color; that is, it consists of cells of the same color, as shown in Figure 5.3. There are four colors available, so four random graphs could be generated from the original sensor deployment and graph construction, one for each color. The probability p that an edge exists between two adjacent vertices in the random graph is related to the size of a cell (L), the deployment error (σ), the number of sensors in each cell (m), and the node transmission range (R).

We denote two cells separated by one cell of a different color as (i_1, j_1) and (i_2, j_2), with target points of (x_1, y_1) and (x_2, y_2) respectively. To derive p, we first consider the probability p_0 that one sensor N_1 from cell (i_1, j_1) is within the transmission range of another sensor N_2 from cell (i_2, j_2). Let us consider an infinitesimal rectangle $\mathrm{d}x \times \mathrm{d}y$ in the deployment area. The probability p_{n1} that sensor N_1 resides in this small area is:

$$\begin{aligned} p_{n1} &= f_{i_1,j_1}(x,y)\mathrm{d}x\mathrm{d}y \\ &= \tfrac{1}{2\pi\sigma^2} e^{-[(x-x_1)^2+(y-y_1)^2]/2\sigma^2}\mathrm{d}x\mathrm{d}y, \end{aligned} \tag{5.4}$$

because the probability density over this very small area can be treated as even.

Suppose the distance between node N_1 and the target point (x_2, y_2) of N_2 is d. Then, the probability that the sensor N_2 resides in the round area A centered at the location of sensor N_1 with radius R is different when $d \geq R$ and $d < R$. If $d \geq R$, as shown in Figure 5.4(a), the probability p_{n2} that sensor N_2 resides in the round area A is a double integration:

$$p_{n2} = \int\int_A f_{i2,j2}(x,y)\mathrm{d}A.$$

We consider $\mathrm{d}A$ as arc $\mathrm{d}A = \theta l \mathrm{d}l$. As θ is a function of l, i.e., $\theta = 2arccos(\frac{l^2+d^2-R^2}{2ld})$, the above formula can be simplified to the integration:

$$\begin{aligned} p_{n2} &= \int_{d-R}^{d+R} f_{i2,j2}(x,y)\theta l \mathrm{d}l \\ &= \int_{d-R}^{d+R} 2arccos(\frac{l^2+d^2-R^2}{2ld}) f_{i2,j2}(x,y) l \mathrm{d}l. \end{aligned} \tag{5.5}$$

If $d < R$, as shown in Figure 5.4(b), the probability that sensor N_2 resides in the round area A equals to the same double integration, but now θ as a function of l has changed to:

$$\theta = \begin{cases} 2\pi, & \text{if } 0 \leq l \leq R-d; \\ 2arccos(\frac{l^2+d^2-R^2}{2ld}), & \text{if } R-d \leq l \leq R+d. \end{cases}$$

Hence, in this case the double integration can be simplified to the integration:

$$p_{n2} = \int_0^{R-d} 2\pi l f_{i2,j2}(x,y)\mathrm{d}l +$$

$$\begin{aligned}
& \int_{R-d}^{R+d} 2arccos(\frac{l^2+d^2-R^2}{2ld}) f_{i2,j2}(x,y) l \mathrm{d}l \\
= & \ 1-e^{-\frac{(R-d)^2}{2\sigma^2}} + \\
& \int_{R-d}^{R+d} 2arccos(\frac{l^2+d^2-R^2}{2ld}) f_{i2,j2}(x,y) l \mathrm{d}l.
\end{aligned} \tag{5.6}$$

Because node N_1 may appear at any place in the deployment area and the positions of two nodes from two different cells are two independent events, the probability p_0 that two nodes from cells $(i1, j1)$ and $(i2, j2)$ are within each other's transmission range is:

$$\begin{aligned}
p_0 &= \int_{x=0}^{X}\int_{y=0}^{Y} p_{n2} \cdot f_{i1,j1}(x,y) \mathrm{d}x\mathrm{d}y \\
&= \int_{x=0}^{X}\int_{y=0}^{Y} \frac{p_{n2}}{2\pi\sigma^2} e^{-[(x-x_1)^2+(y-y_1)^2]/2\sigma^2} \mathrm{d}x\mathrm{d}y.
\end{aligned} \tag{5.7}$$

Next, we derive the probability p that two neighboring cells of the same color (in the random graph) are connected, based on p_0. The fact that two cells are connected means that at least one pair of nodes from these two cells respectively are within each other's transmission range. Let m be the number of nodes in one cell, then there are totally m^2 such kind of pairs. Therefore, the probability that two cells are connected is:

$$p = 1-(1-p_0)^{m^2}. \tag{5.8}$$

The above derivations will be validated by our simulation results in Section 5.3.2.1.

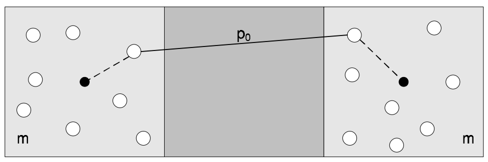
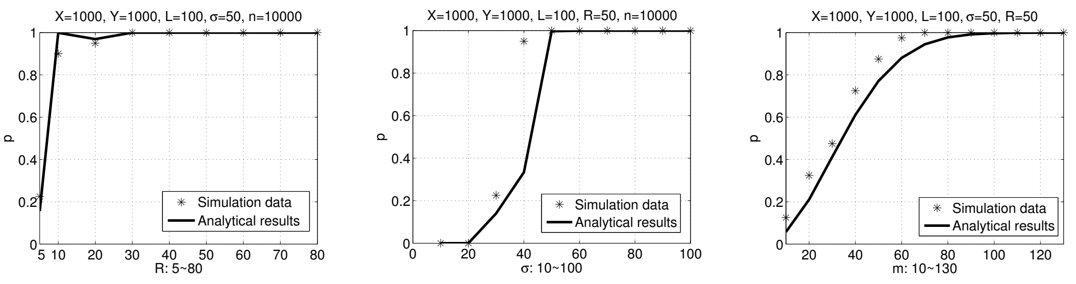


Figure 5.5. The derivation of probability p.

Based on the value of probability p, our next task is to check the impact of p on our goal of containing sensor worms. To address this problem, we resort to the percolation theory [109]. Percolation theory deals with fluid flow (or any other similar process) in random media. In mathematics, percolation theory describes the behavior of connected clusters in a random graph. In our application, we will consider bond percolation (related to edges in graph) instead of site percolation (related to vertices in graph). Bond percolation means that the probability that one edge appears in the graph is $p (0 < p < 1)$.



(a) p as a function of R (b) p as a function of σ (c) p as a function of m

Figure 5.6. Probability p of two neighboring cells with the same color connected, as a function of R, σ, m.

The problem is to find out whether there is a percolation cluster (i.e., infinitely extended connected cluster) going through the entire graph, which in our scenario means that sensor worm can successfully propagate throughout the entire network.

Clearly, as p increases, the average connected cluster size s_{av} also increases, because the probability of continuing a cluster by finding an adjacent connected edge becomes larger. By the size of a connected cluster, we mean the number of cells that are bound together by connected edges in the cluster. The cluster size distribution is typically expressed as a discrete function $\phi(\lambda)$, where $\phi(\lambda)$ is the number of clusters of size λ. Actually, when p is not close to 1, the probability of encountering a cluster of size λ is in the order of p^{λ}. For example, if $\lambda = 1$, then $\phi(\lambda) = p$, because if we choose an edge at random then the chance of it being connected is p. Similarly, if $\lambda = 2$, then $\phi(\lambda) = 6p^2$, and so on. More specifically, since the number of edges occurring in the clusters of size λ is proportional to $\lambda\phi(\lambda)$, the weighted mean of cluster sizes is given by:

$$s_{av} = \frac{\sum_{\lambda=1}^{n_r} \lambda^2 \phi(\lambda)}{\sum_{\lambda=1}^{n_r} \lambda \phi(\lambda)}, \tag{5.9}$$

where $n_r (n_r \leq n)$ is the total number of nodes with the same color in the random graph. $n_r = n$ when there is only one available color. From the above formula, we know that the average cluster size s_{av} approaches n_r when p is close to a critical value p_c. For instance, $p_c \approx 0.65$ for hexagons (Figure 5.2(a)), and $p_c \approx 0.5$ for the square lattice in two dimension (Figure 5.2(b)). When $p < p_c$ the probability that a percolation cluster exists is close to 0, but when p is increased to around or above p_c the probability that a percolation cluster exists rapidly approaches 1. This is verified by our following simulation results in Section 5.3.2.2.

Hence, we should choose p_0 as close to 0 as possible so that p becomes small and lower than p_c (according to Eq.(8)). In real applications, normally the transmission range and deployment error level are fixed (although in Section 5.3.2.4 we will check the situation that the attacker can increase the transmission range of compromised nodes to gain more from the attack), so intuitively we should make the cell size L as large as we can to obtain small p_0 and p (when other parameters such as σ and R are fixed, if L is larger, then p_0 is smaller so that p is also smaller). However, we notice that it is not always better if L is larger, because in that case there will be more sensors in each cell. Suppose the length of the edge of the deployment area is $|X| = |Y| = \delta$. Then, the number of nodes in each cell is:

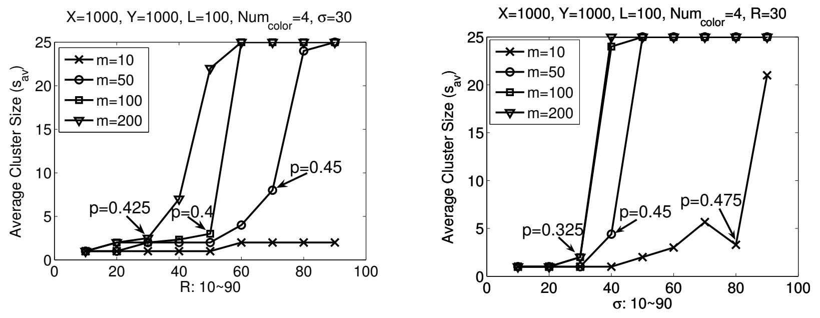
$$m = \frac{n}{(\delta/L)^2} = \frac{nL^2}{\delta^2},$$

assuming that each cell has the same number of sensors. Clearly, m increases with L, when δ and n are fixed. One extreme case is that L equals to δ. Then, all the sensors in the whole deployment area have the same color. At this time, the worm can easily propagate over the whole area.

The above formulation provides us a way to pick up the parameter L for cell size. Suppose the maximum size of connected clusters exists in the random graph is s_{max}. Then, the number of sensor nodes that can be compromised at one time by the worm attack is ms_{max}, because there are m sensors in each cell. Our objective is to keep this number as small as possible. To achieve this, if L is larger, then s_{max} is smaller because p is smaller, but at the same time, m becomes larger. Therefore, we can have a minimum of ms_{max} under a certain value of L. We will use simulations to find out this optimal value of L in Section 5.3.2.3.

5.2.4 The Case of Loading Multiple Colors

Above we assume that each sensor can only take one color, which is the color of its targeted cell. We notice that if the program memory of a sensor node has extra space to hold multiple versions of code (OSes or other programs) and it allows dynamic transition of codes, it may be preloaded with multiple versions altogether. In addition, we can preload each sensor node with the optimal planning map, which marks the color of each cell corresponding to the case of no deployment error. After its deployment, a sensor node first figures out the cell it falls in based on an attack-resilient localiza-



(a) Average cluster size vs R (b) Average cluster size vs σ

Figure 5.7. The average cluster size changes as a function of R and σ.

tion scheme [8], and then sets its color according to the planning map. In this case, deployment error will have no impact on the scheme.

In [67], O'Donnell and Sethu proposed several distributed coloring algorithms for networks consisting of machines that can be loaded with multiple colors. In their algorithms, every machine is mapped to a vertex. After deployment, a machine may randomly select a color out of a set of available colors (called *random coloring*) or flips its color based on its neighbor's colors (called *color flipping*). These algorithms however are not well tailored for sensor networks that feature high network connectivity; moreover, a relatively high communication cost is needed for a node to negotiate colors with its neighbors in a sensor network. Our scheme does not have these limitations. We will quantitatively compare our scheme with theirs in the next section.

5.3 Performance Evaluation

In this section, we will validate our analytical model and show the performance of our scheme under different parameter settings. The results will also be compared with those of randomized coloring and color flipping schemes in [67].

5.3.1 Simulation Setup

In the simulation, the deployment area of X×Y=1,000m× 1,000m is divided into different sizes of cells with edge lengths from 25m to 250m respectively. The target points of each cell is the center. The total number of sensors in the entire network changes from 1,000 to 64,000. The transmission range varies from 5m to 90m. The standard deviation of the two-dimensional normal distribution is in the range of 10 to 100. The number of

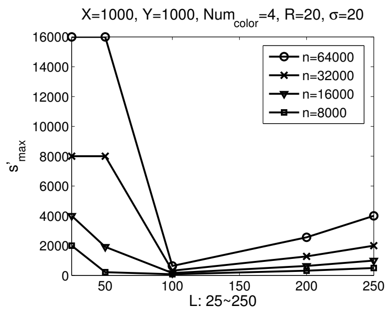


Figure 5.8. s'_{max} changes as a function of cell size L under different n.

available colors is either 4 or 5.

In our simulation, by default, there are totally 10,000 sensors distributed over the deployment area and each cell has the edge length of 100m. On average, the node density over the entire area is 100 sensors per cell (100m$\times$100m), i.e., one sensor per 10m$\times$10m area. We may adjust the distribution of sensors by changing σ. For example, under the same cell size, when the target points are 2σ=100m apart (i.e., $\sigma = 50$), the overall probability distribution function is close to uniform. Note that each point in the following figures is averaged over 100 rounds.

5.3.2 Simulation results

5.3.2.1 Probability p of Two Cells Being Connected

We check how p changes with R, σ and m through both analytical results and simulations. In the simulation, p is computed as the number of edges that are connected divided by the total number of edges in the random graph (of the same color). As shown in Figure 5.6, the simulation results match our analytical results well.

In Figure 5.6(a), with R being changed to above 10, p is increased rapidly from 0.2 to 1 under $\sigma = 50, L = 100$ and $n = 10,000$. In Figure 5.6(b), when σ is increased to above 40, p also quickly increases from 0 to 1. In Figure 5.6(c), when the number of sensors in each cell increases, p is increased to 1 gradually.

5.3.2.2 The Average Cluster Size

Figure 5.7 shows that the average cluster size changes as a function of R and σ under different m. When L is 100, there are totally 10×10 cells in the network. Since there are 4 colors, each random graph of one color has 25 vertices and 40 edges. When R or σ is larger than a threshold, the average cluster sizes are increased rapidly to cover the entire random graph. In reality, this means that we should keep R and σ to be relatively small values, e.g., much less than 40, so that sensor worms cannot propagate over the deployment area.

We also check the values of p at the turning points in each line of Figure 5.7. p is normally between 0.4 and 0.5 at those points, which means the critical value p_c is around $0.4 \sim 0.5$ in our scheme. This confirms our derivations based on percolation theory and Eq.(9) in Section 5.2.3.

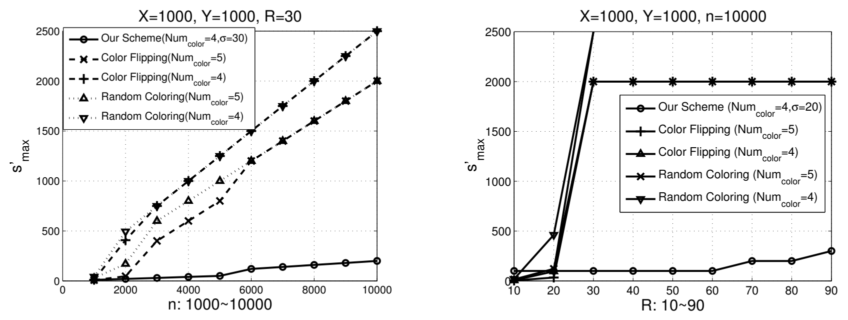
5.3.2.3 Maximum Number of Node Compromises

Next, we study the maximum number (s'_{max}) of node compromises that could happen at one time. In the simulation, s'_{max} is calculated by multiplying the maximum size of connected cluster with the number of sensors at each cell ($s_{max} \times m$). As shown in Figure 5.8, s'_{max} changes as a function of cell size L and the total number of sensors n.

Under a certain number n, when L is larger, the probability p that two cells with the same color are connected becomes smaller, so the maximum size of connected cluster (s_{max}) is also reduced. At $R = 20$ and $\sigma = 20$, it is decreased to the size of 1 (i.e., containing only one cell) quickly until L reaches 100. After this, s_{max} remains 1, but the number of sensors in each cell is increased with L. Thus, s'_{max} starts to increase (slowly). Therefore, we achieve the minimum s'_{max} when $L = 100$ under this condition. On the other hand, if we decrease n, the above trend is the same, but the absolute values of s'_{max} become smaller.

5.3.2.4 Comparison with Previous Schemes

We compare the performance of our scheme with the previous schemes, random coloring and color flipping [67], in terms of s'_{max}. In random coloring, n (changing from 1,000 to 10,000) sensors are randomly deployed over an area of 1,000m$\times$1,000m, each of which has a random color. Under the same setting, in color flipping every sensor changes its color if necessary according to its neighbors' colors, so as to reduce the defective edges. Since there is no concept of cell in random coloring and color flipping, s'_{max} is the maximum number of connected sensors with the same color.



(a) s'_{max} as a function of n (b) s'_{max} as a function of R

Figure 5.9. Comparing s'_{max} of our scheme with others.

We first examine s'_{max} as a function of the total number of sensors n. The simulation results are presented in Figure 5.9(a). In both these two schemes, s'_{max} rapidly approaches the maximum number of sensors with the same color (2,500 in 4-color case and 2,000 in 5-color case), when the total number of sensors increases. On the other hand, more available colors help only when n is relatively small, because in the figure only when $n < 6,000$ s'_{max} is smaller with 5 colors than with 4 colors. In our scheme, there are totally $\frac{n}{100}$ sensors in each cell when $L = 100$. s'_{max} does not change much under appropriate parameters such as a small $\sigma(= 30)$. Even when $n = 10,000$, s'_{max} is as low as 200.

We also check s'_{max} as a function of the transmission range R. Although sensors normally have a fixed transmission range, transmission range R in our simulation varies because we notice that after the attacker compromises one sensor he may increase the transmission range of the compromised node by enhancing its RF power level or antenna configuration to maximize his gain from the attack. The simulation results are presented in Figure 5.9(b). Similarly, in both random coloring and color flipping, s'_{max} is increased rapidly to close to the maximum number of sensors with the same color in the network, even when R is as small as 30 under $n = 10,000$. Compared with Figure 5.9(a), more available colors (from 4 to 5) is even useless, because the 4-color and 5-color lines are almost overlapped together. In our scheme, there are totally 100 sensors in each cell when $L = 100$ and $n = 10,000$. If R is small, sensor worm will not propagate outside one single cell, so the maximum number of sensors that could be compromised at one time is 100, which may be slightly higher than those in the other two schemes at the beginning. However, our scheme significantly outperforms the other two as R becomes larger, because s'_{max} does not change much under appropriate parameters.

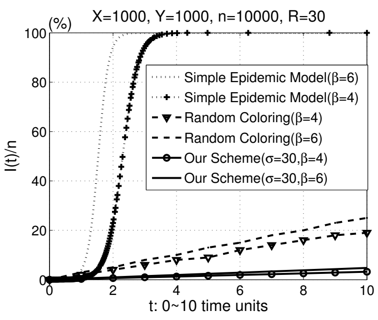


Figure 5.10. Comparing $I(t)/n$.

The above results indicate that our scheme with proper parameters can effectively prevent the propagation of sensor worms, compared with the previous two schemes.

5.3.2.5 Effectiveness of Sensor Worm Containment

At last, we simulate the infection rate $I(t)$ in our scheme and compare the results with those in the simple epidemic model and the random coloring scheme. We check the percentage of infectious nodes as a function of time units for probing rate $\beta = 4$ and $\beta = 6$ (although in reality sensor worm has broadcast nature in probing, we check these two cases merely for comparison reasons). As shown in Figure 5.10, $I(t)$ in our scheme increases much slower than those in the original theoretical model and the random coloring scheme. The comparison results indicate that our scheme can substantially reduce the possibility that sensor worm propagates in the network.

Through simulations, we find that the simple epidemic model can roughly describe the propagation rate of sensor worm when all the sensors have the same color in the network. In this case, sensor worm can propagate throughout the entire network of 10,000 sensor nodes (100% coverage) within 2 or 3 time units. Moreover, when the probing rate β becomes larger (e.g., increased from 4 to 6), i.e., an infectious node sends more probes to the population at one time, the number of infectious nodes is increased even faster. Fortunately, having multiple colors (in random coloring) can slow down this process in a large degree. The percentage of infectious nodes in random coloring is eventually increased to 25% when the number of available colors is 4 at the 10th time unit. Compared with random coloring, our scheme can further improve the defensive capability against sensor worm by quarantining sensor worm within as few cells as possible. As a result, at the 10th time unit, there are only 3.2% infectious nodes when $\beta = 4$ and 4.76% infectious nodes when $\beta = 6$, under the condition that $\sigma = R = 30$ and $L = 100$.

5.4 Further Discussions

Overall, to defend against buffer overflow [55] and sensor worm, there are two directions we can work on. The first direction is hardware. Sensors could employ compromise resilient hardware to protect from being broken in. However, this will increase the cost of sensors, which is infeasible in practice. The second direction is software. Sensors could employ some software-based techniques to protect themselves. This is the direction that we are interested in. We will elaborate on it in the following.

Sensor worm exploits buffer overflow vulnerability in sensors' program to propagate. In the Harvard architecture of most sensors, it exhibits the form of jump-to-libc attack [110]. That is to say, buffer overflow causes that the return address on the stack is replaced by the address of another instruction and an additional portion of the stack is overwritten to provide arguments to this function. This allows attackers to call preexisting functions without the need to inject malicious code into a program. Obviously, a non-executable stack can prevent some buffer overflow exploitation but not a jump-to-libc attack because in this attack only existing executable code is used. However, several other techniques are effective against the jump-to-libc attack.

For example, similar to StackGuard [111], we can employ a "canary" value on the stack, which when destroyed shows that a buffer preceding it in memory has been overflowed. One benefit of this technique is that the performance penalty by this technique is almost negligible. This technique actually works to detect the buffer overflow. To prevent buffer overflow, address space layout randomization [112] is effective because the memory locations of functions are random. However, how effective this technique is on (only) 16-bit program memory of sensor architecture remains unclear, since it may be defeated by brute force.

Therefore, the above techniques are complementary to our mechanisms in defending against jump-to-libc attack, but due to the scarce resources of sensors, enhancing the defensive capabilities of individual sensor is limited. In this work, we focus on improving the immunity of entire sensor network through appropriately configuring the program versions of neighboring sensors. In this way, the whole sensor network becomes more powerful under software diversity.

Chapter 6

Conclusion

In the dissertation, I have made three contributions. First, I propose a secure hop-by-hop data aggregation protocol for wireless sensor networks. Second, I present two distributed schemes on software-based attestation to detect compromised sensor nodes. Finally, the possibility for worm attack to happen in sensor networks is validated and a software diversity based defensive solution is discussed.

For secure data aggregation, by using *divide-and-conquer*, we partition the aggregation tree into groups to reduce the importance of high-level nodes in the aggregation tree. Also, we enhance the logical groups with commitment capability by applying *commit-and-attest*, so that the BS has a way to verify the group aggregates. Extensive analytical and simulation results show that SDAP is effective in defending against both count and value changing attacks and SDAP is efficient with respect to the reasonable overhead it causes. Prototype implementation on top of TinyOS shows that our scheme is practical to be applied in the current generation sensor nodes such as Mica2 motes. As future work, several directions are worth of investigation. First, we will further enrich the protocol in more detail. For example, the breadth-based attestation and the content-based attestation techniques may also be included in the protocol. Second, we will check the influence of unreliable transmission channel to our protocol. Third, we may implement and show the benefits of Bloom Filter in our protocol. Last, the potential of integrating with different network model, such as that in SIA, will be examined, too.

The detection of node compromise is a critical but challenging problem for resource-constrained sensors deployed in an unattended or hostile environment. Recent work on software-based code attestation has shed light on accurately identifying compromised nodes. However, they are not readily applied into regular sensor networks due to one or another limitations. In the work of distributed software-based attestation for node compromise detection, we have presented two distributed schemes towards making software-based attestation more practical. Our schemes do not depend on response time measurement by mobile verifiers or the base station. Instead, neighbors of a suspicious node collaborate in the attestation process to make a joint decision. In the future, we will further investigate solutions for noise-generation seed update and network topology change (i.e., nodes join and leave the network). Another issue we have not addressed yet is how to schedule the message transmissions of neighbors (for secret share distribution and collection) to minimize channel collision. Finally, we will study how to apply our schemes to different sensor memory architectures.

Finally, we not only illustrate the feasibility of launching worm attacks in sensor network, but also propose a concrete defense scheme based on the idea of software diversity. We show by assigning each sensor an appropriate version of software among a few versions, we can significantly increase the survivability of sensor networks under worm attacks. We also analyze the impact of sensor deployment error on the capability of our scheme, which provides a guidance for our selection of system parameters. In the future, we may investigate the impact of different cell shapes, such as hexagon, triangle, and other polygons.

Bibliography

[1] AKYILDIZ, I., W. SU, Y. SANKARASUBRAMANIAM, and E.CAYIRCI (2002) "Wireless Sensor Networks: A Survey," *Computer Networks*, **38**(4).

[2] "Mica Motes," `http://www.xbow.com`.

[3] WOOD, A. and J. STANKOVIC (2002) "Denial of service in sensor networks," *IEEE Computer*.

[4] XU, W., T. WOOD, W. TRAPPE, and Y. ZHANG (2004) "Channle surfing and spatial retreats: defenses against wireless denial of service," in *ACM workshop on wireless security*.

[5] RAYA, M., J. HUBAUX, and I. AAD (2004) "Domino: A system to detect greedy behavior in IEEE 802.11 hotspots," in *MobiSys*.

[6] KARLOF, C. and D. WAGNER (2003) "Secure routing in sensor networks: Attacks and Countermeasures," in *Proceedings of First IEEE Workshop on Sensor Network Protocols and Applications*.

[7] ZHU, S., S. SETIA, S. JAJODIA, and P. NING (2004) "An interleaved hop-by-hop authentication scheme for filtering of injected false data in sensor networks," in *Proceedings of IEEE Symp. on Security and Privacy*.

[8] CAPKUN, S. and J. HUBAUX (2004) *Secure positioning in sensor networks*, *Tech. Rep. Technical Report EPFL/IC/200444*.

[9] LAZOS, L. and R. POOVERDRAN (2004) "Serloc: Secure range-independent localization for wireless sensor networks," in *Wise'04*.

[10] SONG, H., S. ZHU, and G. CAO (2005) "Attack-resilient time synchronization for wireless sensor networks," in *MASS'05*.

[11] SUN, K., P. NING, and C. WANG (2006) "Secure and resilient clock synchronization in wireless sensor networks," *IEEE Journal on Selected Areas in Communications*.

[12] HARTUNG, C., J. BALASALLE, and R. HAN (2005) *Node compromise in sensor networks: the need for secure systems*, *Technical report CU-CS-990-05*, Department of computer science, university of colorado at boulder.

[13] KARLOF, C. and D. WAGNER (2003) "Secure Routing in Sensor Networks: Attacks and Countermeasures," in *Proceedings of First IEEE Workshop on Sensor Network Protocols and Applications*.

[14] ESTRIN, D., R. GOVINDAN, J. HEIDEMANN, and S. KUMAR (1999) "Next Century Challenges: Scalable Coordination in Sensor Networks," in *Proceedings of ACM Mobicom*, ACM, Seattle, Washington, USA, pp. 263–270.

[15] INTANAGONWIWAT, C., D. ESTRIN, R. GOVINDAN, and J. HEIDEMANN (2002) "Impact of Network Density on Data Aggregation in Wireless Sensor Networks," in *ICDCS*, pp. 457–458.

[16] INTANAGONWIWAT, C., R. GOVINDAN, and D. ESTRIN (2000) "Directed diffusion: a scalable and robust communication paradigm for sensor networks," in *MOBICOM*, pp. 56–67.

[17] KRISHNAMACHARI, B., D. ESTRIN, and S. WICKER (2002) "The Impact of Data Aggregation in Wireless Sensor Networks," in *International Workshop on Distributed Event-Based Systems, (DEBS '02)*, Vienna, Austria.

[18] MADDEN, S., M. J. FRANKLIN, J. M. HELLERSTEIN, and W. HONG (2002) "TAG: A Tiny AGgregation Service for Ad-Hoc Sensor Networks," in *OSDI*.

[19] CASTELLUCCIA, C., E. MYKLETUN, and G. TSUDIK (2005) "Efficient Aggregation of Encrypted Data in Wireless Sensor Networks," in *Mobile and Ubiquitous Systems: Networking and Services MobiQuitous 2005*.

[20] CHEN, J.-Y., G. PANDURANGAN, and D. XU (2005) "Robust computation of aggregates in wireless sensor networks: distributed randomized algorithms and analysis," in *IPSN*, pp. 348–355.

[21] MARTI, S., T. J. GIULI, K. LAI, and M. BAKER (2000) "Mitigating routing misbehavior in mobile ad hoc networks," in *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pp. 255–265.

[22] GANERIWAL, S. and M. SRIVASTAVA (2004) "Reputation-based Framework for High Integrity Sensor Networks," in *Proceedings of ACM workshop SASN'04*.

[23] KHAYAM, S. A. and H. RADHA (2005) "A topologically-aware worm propagation model for wireless sensor networks," in *SDCS*.

[24] YAO, Y. and J. GEHRKE (2002) "The Cougar Approach to In-Network Query Processing in Sensor Networks," *SIGMOD Record*, **31**(3), pp. 9–18.

[25] WAGNER, D. (2004) "Resilient aggregation in sensor networks," in *Proceedings of ACM Workshop SASN '04*.

[26] HU, L. and D. EVANS (2003) "Secure aggregation for wireless networks," in *Workshop on Security and Assurance in Ad hoc Networks*.

[27] DU, W., J. DENG, Y. S. HAN, and P. K. VARSHNEY (2003) "A Witness-Based Approach for Data Fusion Assurance in Wireless Sensor Networks," in *Proc. of IEEE GLOBECOM '03)*.

[28] PRZYDATEK, B., D. SONG, and A. PERRIG (2003) "SIA: secure information aggregation in sensor networks," in *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pp. 255–265.

[29] CHAN, H., A. PERRIG, B. PRZYDATEK, and D. SONG (2007) "SIA: Secure Information Aggregation in Sensor Networks," *Journal of Computer Security, Special Issue on Adhoc and Sensor Networks (JCS)*.

[30] CHAN, H., A. PERRIG, and D. SONG (2006) "Secure hierarchical in-network aggregation in sensor networks," in *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, pp. 278–287.

[31] ROY, S., S. SETIA, and S. JAJODIA (2006) "Attack-resilient hierarchical data aggregation in sensor networks," in *SASN '06: Proceedings of the fourth ACM workshop on Security of ad hoc and sensor networks*, pp. 71–82.

[32] NATH, S., P. GIBBONS, S. SESHAN, and Z. ANDERSON (2004) "Synopsis diffusion for robust aggregation in sensor networks," in *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pp. 250–262.

[33] HE, W., X. LIU, H. NGUYEN, K. NAHRSTEDT, and T. ABDELZAHER (2007) "PDA: Privacy-preserving Data Aggregation in Wireless Sensor Networks," in *IEEE Infocom 2007*.

[34] YE, F., H. LUO, S. LU, and L. ZHANG (2004) "Statistical En-route Filtering of Injected False Data in Sensor Networks," in *Proceedings of IEEE Infocom'04*.

[35] ZHANG, W. and G. CAO (2005) "Group Rekeying for Filtering False Data in Sensor Networks: A Predistribution and Local Collaboration-Based Approach," *IEEE INFOCOM*.

[36] ZHU, S., S. SETIA, S. JAJODIA, and P. NING (2004) "An Interleaved Hop-by-hop Authentication Scheme for Filtering of Injected False Data in Sensor Networks," in *Proceedings of IEEE Symp. on Security and Privacy*, pp. 259–271.

[37] SPINELLIS, D. (2000) "Reflection as a mechanism for software integrity verification," *ACM Trans. Inf. Syst. Secur.*, **3**(1).

[38] SHANECK, M., K. MAHADEVAN, V. KHER, and Y. KIM (2005) "Remote Software-Based Attestation for Wireless Sensors." in *ESAS*.

[39] PARK, T. and K. G. SHIN (2005) "Soft Tamper-Proofing via Program Integrity Verification in Wireless Sensor Networks." *IEEE Trans. Mob. Comput.*, **4**(3), pp. 297–309.

[40] SESHADRI, A., A. PERRIG, L. VAN DOORN, and P. KHOSLA (2004) "SWATT: SoftWare-based ATTestation for Embedded Devices," in *IEEE Symposium on Security and Privacy*.

[41] MITZENMACHER, M. and E. UPFAL (2005) *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*, Cambridge University Press.

[42] SESHADRI, A., M. LUK, E. SHI, A. PERRIG, L. VAN DOORN, and P. KHOSLA (2005) "Pioneer: Verifying Integrity and Guaranteeing Execution of Code on Legacy Platforms," in *SOSP*, pp. 1–15.

[43] SHI, E., A. PERRIG, and L. V. DOORN (2005) "BIND: A Fine-Grained Attestation Service for Secure Distributed Systems," in *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, pp. 154–168.

[44] KENNELL, R. and L. H. JAMIESON (2003) "Establishing the genuinity of remote computer systems," in *Proceedings of the 12th USENIX Security Symposium*.

[45] ——— (2004) *An analysis of proposed attacks against genuinity tests*, *Tech. rep.*, Purdue University, cERIAS TR 2004-27.

[46] SHANKAR, U., M. CHEW, and J.D.TYGAR (2004) "Side effects are not sufficient to authenticate software," in *Proceedings of the 13th USENIX Security Symposium*.

[47] SAILER, R., X. ZHANG, T. JAEGER, and L. VAN DOORN (2004) "Design and Implementation of a TCG-based Integrity Measurement Architecture," in *Proceedings of the 13th USENIX Security Symposium*.

[48] "TCG," Http://www.trustedcomputinggroup.org.

[49] WURSTER, G., P. C. VAN OORSCHOT, and A. SOMAYAJI "A Generic Attack on Checksumming-Based Software Tamper Resistance," in *SP' 2005*, pp. 127–138.

[50] COLLBERG, C. S. and C. THOMBORSON (2002) "Watermarking, Tamper-Proofing, and Obfuscation - Tools for Software Protection," in *IEEE Transactions on Software Engineering*, vol. 28, pp. 735–746.

[51] VAN OORSCHOT, P. (2003) "Revisiting Software Protection," *Information Security, LNCS*, **2851**, pp. 1–13.

[52] AUCSMITH, D. (1996) "Tamper Resistant Software: An Implementation," in *Proceedings of the First International Workshop on Information Hiding*, pp. 317–333.

[53] AUCSMITH, D. and G. GRAUNKE (1999), "Tamper Resistant Methods and Apparatus," Us patent 5,892,899, assignee: Intel Corporation.

[54] STANIFORD, S., V. PAXSON, and N. WEAVER (2002) "How to Own the Internet in Your Spare Time," in *Proceedings of the 11th USENIX Security Symposium*.

[55] ONE, A. "Smashing the stack for fun and profit," *Phrack 49*, http://www.phrack.org/show.php?p=49a=14.

[56] BRUMLEY, D., L.-H. LIU, P. POOSANKAM, and D. SONG (2006) "Design Space and Analysis of Worm Defense Strategies," in *ASIACCS*.

[57] GANESH, A., L. MASSOULIE, and D. TOWSLEY (2005) "The effect of network topology on the spread of epidemics," in *Infocom*.

[58] DRAIEF, M., A. GANESH, and L. MASSOULIE "Thresholds for virus spread on networks," in *ValueTools'06*.

[59] X. WANG, P. L., C. PAN and S. ZHU. "SigFree: A Signature-free Buffer Overflow Attack Blocker," in *USENIX Secucrity'06*.

[60] DE, P., Y. LIU, and S. K. DAS "Modeling Node Compromise Spread in Wireless Sensor Networks Using Epidemic Theory," in *WOWMOM '06*.

[61] KUMAR, R., E. KOHLER, and M. SRIVASTAVA (2007) "Harbor: Software-based Memory Protection for Sensor Nodes," in *IPSN*.

[62] REGEHR, J., N. COOPRIDER, W. ARCHER, and E. EIDE (2006) *Memory Safety and Untrusted Extensions for TinyOS*, *Tech. Rep. UUCS-06-007*, University of Utah.

[63] ALARIFI, A. and W. DU (2006) "Diversify Sensor Nodes to Improve Resilience Against Node Compromise," in *SASN'06*.

[64] GU, Q. and R. NOORANI "Towards Self-propagate Mal-packets in Sensor Networks," in *WiSec'08*.

[65] Francillon, A. and C. Castelluccia "Code Injection Attacks on Harvard-Architecture Devices," in *CCS'08*.

[66] Forrest, S., A. Somayaji, and D. Ackley "Building Diverse Computer Systems," in *HOTOS '97*.

[67] O'Donnell, A. J. and H. Sethu "On achieving software diversity for improved network security using distributed coloring algorithms," in *CCS '04*.

[68] Bailey, M. G. "Malware resistant networking using system diversity," in *SIGITE '05*.

[69] G.S.Kc, A.D.Keromytis, and V.Prevelakis (2003) "Countering code injection attacks with instruction set randomization," in *CCS*.

[70] M.C.Mont, A.Baldwin, Y.Beres, K.Harrison, M.Sadler, and S.Shiu (2002) "Towards diversity of cots software applications: Reducing risks of widespread faults and attacks," in *Technical Report HPL-2002-178*.

[71] N.Roux, J-S.Pegon, and M.Subbarao (2000) "Cost adaptive mechanism to provide network diversity for manet reactive routing protocols," in *MILCOM*.

[72] Bailey, N. (1975) *The mathematical theory of infectious diseases and its applications*, Hafner Press, New York.

[73] Pastor-Satorras, R. and A. Vespignani (2002) *Epidemics and immunization in scale-free networks*, chap. Handbook of graphs and networks: from the genome to the Internet.

[74] Jensen, T. R. (1995) *Graph Coloring Problems*, Wiley.

[75] Hill, J., R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister (2000) "System architecture directions for networked sensors," *Proc. of ASPLOS IX*.

[76] Perrig, A., R. Szewczyk, V. Wen, D. Culler, and J.D.Tygar (2001) "SPINS: Security protocols for sensor networks," in *International Conference on Mobile Computing and Networking (MobiCom 2001)*.

[77] Eschenauer, L. and V. D. Gligor (2002) "A key-management scheme for distributed sensor networks," in *CCS '02: Proceedings of the 9th ACM conference on Computer and communications security*, pp. 41–47.

[78] Du, W., J. Deng, Y. S. Han, and P. Varshney (2003) "A Pairwise Key Predistribution Scheme for Wireless Sensor Networks," in *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS)*, Washington DC, USA, pp. 42–51.

[79] LIU, D. and P. NING (2003) "Establishing Pairwise Keys in Distributed Sensor Networks," in *Proceedings of ACM CCS*.

[80] ZHU, S., S. SETIA, and S. JAJODIA (2003) "LEAP: Efficient Security Mechanisms for Large-Scale Distributed Sensor Networks," in *Proceedings of ACM CCS*.

[81] ZHANG, W., H. SONG, S. ZHU, and G. CAO (2005) "Least Privilege and Privilege Deprivation: Towards Tolerating Mobile Sink Compromises in Wireless Sensor Networks," in *ACM MobiHoc*.

[82] MCCUNE, J., E. SHI, A. PERRIG, and M. REITER (2005) "Detection of Denial-of-Message Attacks on Sensor Network Broadcasts," in *IEEE Symposium on Security and Privacy*, pp. 64–78.

[83] MERKLE, R. (1989) "A certified digital signature," in *Proceedings of Advances in Crypto-89*, pp. 218–238.

[84] BLOOM, B. H. (1970) "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, **13**(7), pp. 422–426.

[85] FRANK, G. (1969) "Procedures for Detecting Outlying Observations in Samples," *Technometrics*, **11**(1), pp. 1–21.

[86] PEASE, M., R. SHOSTAK, and L. LAMPORT (1980) "Reaching agreement in the presence of faults," in *Journal of ACM*, vol. 27, pp. 228–234.

[87] LAMPORT, R. SHOSTAK, AND M. PEASE, L. (1982) "The Byzantine Generals Problem," *ACM Transactions on Programming Languages and Systems*.

[88] R.L.RIVEST (1995) "The RC5 encryption algorithm," in *Workshop on Fast Software Encryption*, pp. 86–96.

[89] KARLOF, C., N. SASTRY, and D. WAGNER (2004) "TinySec: A Link Layer Security Architecture for Wireless Sensor Networks," in *Second ACM Conference on Embedded Networked Sensor Systems (SenSys 2004)*.

[90] "TinyOS," Http://www.tinyos.net/.

[91] "TinyKeyMan," http://discovery.csc.ncsu.edu/.

[92] ZHU, S., S. SETIA, and S. JAJODIA (2003) "LEAP: efficient security mechanisms for large-scale distributed sensor networks," in *CCS '03*, pp. 62–72.

[93] LIU, D., P. NING, and R. LI (2005) "Establishing Pairwise Keys in Distributed Sensor Networks," *ACM Transactions on Information and System Security*, **8**(1), pp. 41–77.

[94] DENG, J., R. HAN, and S. MISHRA (2005) "A Practical Study of Transitory Master Key Establishment For Wireless Sensor Networks," in *SecureComm 2005*, pp. 289–299.

[95] SHAMIR, A. (1979) "How to share a secret," *Commun. ACM*, **22**(11), pp. 612–613.

[96] ALFRED J. MENEZES, P. C. V. O. and S. A. VANSTONE (1996) *Handbook of Applied Cryptography*, CRC Press, http://www.cacr.math.uwaterloo.ca/hac/.

[97] ZHU, S., S. SETIA, and S. JAJODIA, "LEAP+: Efficient Security Mechanisms for Large-Scale Distributed Sensor Networks," http://www.cse.psu.edu/ szhu/papers/leap.pdf, to appear in ACM TOSN.

[98] PIRRETTI, M., S. ZHU, V. NARAYANAN, P. MCDANIEL, M. KANDEMIR, and R. BROOKS (2005) "The sleep deprivation attack in sensor networks: analysis and methods of defense," in *ICA DSN*.

[99] HERZBERG, A., S. JARECKI, H. KRAWCZYK, and M. YUNG (1995) "Proactive Secret Sharing Or: How to Cope With Perpetual Leakage," in *CRYPTO '95*, pp. 339–352.

[100] "Data Compression Ratio," http://en.wikipedia.org/wiki/Data_compression_ratio.

[101] *MSP430 Microcontrollers*, texas Instrument. http://www.ti.com/.

[102] *ATmega128(L)*, http://www.atmel.com/dyn/resources/prod-documents/doc2467.pdf.

[103] KULKARNI, S. and L. WANG (2005) "MNP: Multihop network programming for sensor networks," *Proc. of International Conference on Distributed Computing Systems*.

[104] ZHANG, Y., H. VIN, L. ALVISI, W. LEE, and S. K. DAO "Heterogeneous networking: A New Survivability Paradigm," in *NSPW'01*.

[105] HOROWITZ, E., S. SAHNI, and S. RAJASEKARAN (1996) *Computer Algorithms/C++*, Computer Science Press.

[106] "Backtracking," Http://en.wikipedia.org/wiki/Backtracking.

[107] DU, W., J. DENG, Y. S. HAN, S. CHEN, and P. VARSHNEY (2004) "A Key Management Scheme for Wireless Sensor Networks Using Deployment Knowledge," in *IEEE INFOCOM*.

[108] DU, W., J. DENG, Y. S. HAN, and P. VARSHNEY (2006) "A Key Predistribution Scheme for Sensor Networks Using Deployment Knowledge," *IEEE Transactions on Dependable and Secure Computing*.

[109] "Percolation theory," Http://en.wikipedia.org/wiki/Percolation-theory.

[110] "Return to libc attack," Http://en.wikipedia.org/wiki/Return-to-libc_attack.

[111] COWAN, C., C. PU, D. MAIER, H. HINTON, J. WALPOLE, P. BAKKE, S. BEATTIE, A. GRIER, P. WAGLE, and Q. ZHANG (1998) "StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks," in *7th Usenix security syposium*, pp. 63–78.

[112] SHACHAM, H., M. PAGE, B. PFAFF, E.-J. GOH, N. MODADUGU, and D. BONEH (2004) "On the effectiveness of address-space randomization," in *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, pp. 298–307.

[113] LIPMAA, H. (2006), "Secret Sharing, Threshold Cryptography, MPC," Lecture notes. http://www.cs.ut.ee/ lipmaa/teaching/MTAT.07.005/.

Vita

Yi Yang

Yi Yang was born in Hubei, China. She received the B.S. and M.S. degree in Computer Science from Huazhong Normal University, Hubei, China, in the years of 2000 and 2003, respectively. She enrolled in the Ph.D. program of Computer Science and Engineering at The Pennsylvania State University in January 2005. She is a student member of the IEEE and ACM.

PUBLICATIONS

1. **Yi Yang** and Sencun Zhu, “Anonymous routing”,“Key Management”, and “Sensor code attestation”, in Encyclopedia of Cryptography and Security (2nd Edition).

2. Min Shao, Sencun Zhu, Wensheng Zhang, Guohong Cao, and **Yi Yang**, “pDCS: Security and Privacy Support for Data-Centric Sensor Networks,” *IEEE Transaction on Mobile Computing*, Volume 8, Number 8, August 2009.

3. **Yi Yang**, Sencun Zhu, Guohong Cao, and Thomas LaPorta, “An Active Global Attack Model in Sensor Source Location Privacy: Analysis and Countermeasures,” *SecureComm 2009.*

4. **Yi Yang**, Sencun Zhu, and Guohong Cao, “Improving Sensor Network Immunity under Worm Attacks: A Software Diversity Approach,” *ACM Mobihoc 2008,* acceptance ratio 14.7%.

5. Min Shao,**Yi Yang**, Sencun Zhu, and Guohong Cao, “Towards Statistically Strong Source Anonymity for Sensor Networks,” *IEEE Infocom 2008,* acceptance ratio 20.5%.

6. **Yi Yang**, Min Shao, Sencun Zhu, Bhuvan Urgaonkar, and Guohong Cao, “Towards Event Source Unobservability with Minimum Network Traffic in Sensor Networks,” *ACM WiSec 2008,* acceptance ratio 16.7%.

7. **Yi Yang**, Xinran Wang, Sencun Zhu, and Guohong Cao, “Distributed Software-based Attestation for Node Compromise Detection in Sensor Networks,” *IEEE SRDS*, October 2007, acceptance ratio 15%.

8. **Yi Yang**, Xinran Wang, Sencun Zhu, and Guohong Cao, “SDAP: A Secure Hop-by-hop Data Aggregation Protocal for Sensor Networks,” *ACM Mobihoc 2006,* acceptance ratio 9.8%. Journal version was accepted by ACM Transactions on Information and System Security (TISSEC), July 2008.