SDAP: A Secure Hop-by-Hop Data Aggregation Protocol for Sensor Networks

YI YANG, XINRAN WANG, SENCUN ZHU, and GUOHONG CAO
Pennsylvania State University

Hop-by-hop data aggregation is a very important technique for reducing the communication overhead and energy expenditure of sensor nodes during the process of data collection in a sensor network. However, because individual sensor readings are lost in the per-hop aggregation process, compromised nodes in the network may forge false values as the aggregation results of other nodes, tricking the base station into accepting spurious aggregation results. Here a fundamental challenge is how can the base station obtain a good approximation of the fusion result when a fraction of sensor nodes are compromised?

To answer this challenge, we propose SDAP, a Secure Hop-by-hop Data Aggregation Protocol for sensor networks. SDAP is a general-purpose secure data aggregation protocol applicable to multiple aggregation functions. 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 dynamically 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 base station 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. The aggregate by the base station is calculated over all the group aggregates that are either normal or have passed the attestation procedure. Extensive analysis and simulations show that SDAP can achieve the level of efficiency close to an ordinary hop-by-hop aggregation protocol while providing high assurance on the trustworthiness of the aggregation result. Last, prototype implementation on top of TinyOS shows that our scheme is practical on current generation sensor nodes such as Mica2 motes.

Categories and Subject Descriptors: C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless communication*; D.4.6 [**Operating Systems**]: Security and Protection—*Cryptographic controls*; K.6.5 [**Management of Computing and Information Systems**]: Communication Networks—*Security and Protection*

This work was supported in part by the Army Research Office (W911NF-05-1-0270, W911NF-07-1-0318) and the National Science Foundation (NSF-0524156, NSF-0627382, and CAREER NSF-0643906).
A preliminary version of this article appeared in *Proceedings of the 7th ACM International Symposium on Mobile Ad Hoc Networking and Computing (Mobihoc'06)*, Florence, Italy.
Author's address: S. Zhu (corresponding author), Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA 16802; email: szhu@cse.psu.edu.

General Terms: Security, Algorithm, Design

Additional Key Words and Phrases: Data aggregation, probabilistic grouping, commit-and-attest, hop-by-hop, sensor network security

ACM Reference Format:
Yang, Y., Wang, X., Zhu, S., and Cao, G. 2008. SDAP: A secure hop-by-hop data aggregation Protocol for sensor networks. ACM Trans. Inf. Syst. Secur. 11, 4, Article 18 (July 2008), 43 pages. DOI = 10.1145/1380564.1380568. http://doi.acm.org/10.1145/1380564.1380568.

1. INTRODUCTION

Wireless sensor networks are envisioned to be economic solutions to many important applications, such as real-time traffic monitoring, military surveillance, and homeland security [Akyildiz et al. 2002]. A sensor network may consist of hundreds or even thousands of low-cost sensors, each of which 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 subscribe to specific data streams by distributing interests or queries. The sensors in the network then push relevant data to a querying base station (BS). However, it is very inefficient for every sensor node to report their raw data because every data packet need traverse many hops to reach the BS, especially considering that sensor nodes are often constrained by scarce resources in energy, communication, computation, and memory. On the other hand, as in many cases sensor nodes in an area detect the common phenomena, there is high redundancy in their raw data. Thus, reporting raw data back to the BS is often unnecessary.

Recently, many data aggregation protocols [Estrin et al. 1999; Intanagonwiwat et al. 2002, 2000; Krishnamachari et al. 2002; Madden et al. 2002; Castelluccia et al. 2005; Chen et al. 2005] have been proposed to eliminate the data redundancy in sensor data of the network, hence reducing the communication cost and energy expenditure in data collection. During a typical data aggregation process, sensor nodes are organized into a tree hierarchy rooted at a BS, because the tree topology, in which there is only one path from each node to the root, has a nice property of duplicate suppression [Nath et al. 2004]. Each nonleaf node in the tree acts as an aggregator, fusing the data collected from their child nodes before forwarding the results toward the BS. In this way, data are processed and fused at each hop on the way to the BS, and communication overhead can be largely reduced.

Hop-by-hop aggregation, however, opens a new door to false data injection attacks. Sensor nodes are often deployed in open and unattended environments, so they are vulnerable to physical tampering due to the low manufacturing cost. An adversary can obtain the confidential information (e.g., cryptographic keys) from a compromised sensor and reprogram it with malicious code. The compromised node may then report an arbitrary false fusion result to its parent node in the tree hierarchy, causing the final aggregation result to far deviate from the true measurement. This attack becomes more damaging when multiple compromised nodes collude in injecting false data.

The above attack is extremely difficult, if not impossible, to prevent or detect. From the viewpoint of information theory, data aggregation is a lossy data compression process because all the individual sensor readings are lost in the per-hop aggregation process. Hence, it is impossible for the BS to verify the correctness of an aggregated result without knowing the original readings. Unfortunately, the requirement of knowing the original readings effectively precludes any data aggregation techniques. As such, in practice a tradeoff between efficiency and accuracy must be made. The challenge now becomes: how can the BS obtain a good approximation of the aggregation result without losing the efficiency of per-hop data aggregation when a fraction of sensor nodes are compromised?

To answer this challenge, we propose SDAP, a Secure Hop-by-hop Data Aggregation Protocol for sensor networks. The design of SDAP is motivated by the following observations. During a normal hop-by-hop aggregation process in a tree hierarchy, (implicitly) we need to place more trust on high-level nodes (i.e., nodes closer to the root) than low-level nodes, because the aggregated result calculated by a high-level node is from a larger number of sensor nodes. In other words, if a compromised node is closer to the root, the bogus aggregated data from it will have a larger impact on the final result computed by the BS. However, in reality none of these low-cost sensors should be more trustworthy than others. As such, 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. Since fewer nodes will be under a high-level node in a logical subtree, the potential security threat from a compromised high-level node is reduced.

To preserve the efficiency of per-hop aggregation, 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 has collected all the group aggregates, it then identifies the suspicious groups based on a bivariate multiple-outlier detection algorithm. Finally, each group under suspect participates in an attestation process to prove the correctness of its group aggregate. The BS will discard the individual group aggregate if a group under attestation fails to support its earlier commitment made in the collection phase; the final aggregate is calculated over all the group aggregates that are either normal or have passed the attestation procedure.

Unlike the trimming-based resilient aggregation [Wagner 2004] that simply ignores some fraction of highest and lowest values without any reasoning, our attestation scheme provides effective means to validate and then probably accepts the abnormal values, as oftentimes we are more interested in those abnormal values than normal ones. Therefore, there is zero false positive in SDAP. Moreover, SDAP is a general-purpose secure aggregation protocol applicable to various aggregation functions, such as MEAN, SUM, and COUNT. Our analysis and simulations show that SDAP can achieve the level of efficiency close to an ordinary hop-by-hop aggregation protocol while providing high assurance on the trustworthiness of the aggregation result. We implement the prototype of SDAP on top of TinyOS, with the results showing that our scheme is practical for current generation sensor nodes such as Mica2 motes.

The remainder of this article is organized as follows. We introduce the related work in Section 2. Then Section 3 defines our system model and design goals. After that, Section 4 presents the secure data aggregation protocol composed of grouping, aggregation, and attestation. Security analysis and performance evaluation of our scheme are illustrated in Section 5 and Section 6, respectively. At last, our work is summarized and the future work is discussed in Section 7.

2. RELATED WORK

Many data aggregation protocols [Estrin et al. 1999; Intanagonwiwat et al. 2002, 2000; Krishnamachari et al. 2002; Madden et al. 2002; Chen et al. 2005; Yao and Gehrke 2002] have been proposed, but none of them were designed with security in mind. Until recently very little work has focused on secure data aggregation.

After analyzing the possible attacks on the existing aggregation primitives, Wagner [2004] 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 [2003] 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. [2003b] 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. [2003]; Chan et al. [2007] 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. [2006] 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. [2006] augmented the normal data aggregation framework such as synopsis diffusion [Nath et al. 2004] 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. [2007] 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 [Ye et al. 2004; Zhang and Cao 2005; Zhu et al. 2004] 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.

3. 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 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 [Madden et al. 2002; Hu and Evans 2003], we assume a topological tree rooted at the BS. We call this a topology tree.

Definition 3.1. 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 4.2. 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 3.2. 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 [Hill et al. 2000], 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 [Perrig et al. 2001]) 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 [Eschenauer and Gligor 2002; Du et al. 2003a; Liu and Ning 2003; Zhu et al. 2003; Zhang et al. 2005].

3.2 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 [McCune et al. 2005] to deprive other nodes from receiving broadcast messages from the BS.

In this article, however, we are not addressing any of these behavior-based attacks (although we discuss the robustness of our scheme under these attacks in Section 5.3.2); 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 as *value changing attack* and *count changing attack*.

Definition 3.3. 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 5.4 to solve this problem.

3.3 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*: Since it is undesirable to design one scheme for each aggregation function, 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.

4. 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.

4.1 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*.

4.2 Tree Construction and Query Dissemination

For completeness, we first describe a simple tree construction algorithm, which is similar to that in Madden et al. [2002]. 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:

$$BS \rightarrow\rightarrow * : F_{agg}, S_g, \tag{1}$$

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 [Perrig et al. 2001] 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.

4.3 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.

4.3.1 *Group Leader Selection.* Probabilistic grouping is finished through the selection of group leader nodes. Here we have a definition upon the leader node.

Definition 4.1. 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:

$$H(S_g|x) < F_g(c). \tag{2}$$

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 Figure 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 6.1. 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

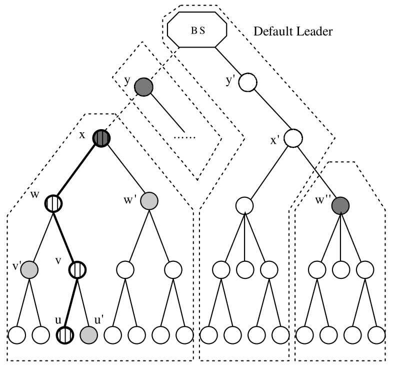


Fig. 1. An example of a grouped aggregation tree. The nodes x, y, and w with the dark-gray color are leader nodes, and the nodes included in the dashed line are corresponding group members. The BS as the root is a default leader.

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 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) to prolong the overall lifetime of the network.

4.3.2 *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 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:

$$\begin{aligned} u \rightarrow v \; : \; & u, 0, E(K_{u,v}, 1|R_u|S_g)|MAC_u \\ & MAC_u = MAC(K_u, 0|1|u|R_u|S_g), \end{aligned} \tag{3}$$

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 (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 Figure 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:

$$\begin{aligned} v \rightarrow w : v, 0, E(K_{v,w}, 3|Agg_v|S_g)|MAC_v \\ Agg_v = F_{agg}(R_v, R_u, R_{u'}) \\ MAC_v = MAC(K_v, 0|3|v|Agg_v|MAC_u \oplus MAC_{u'}|S_g), \end{aligned} \tag{4}$$

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 (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 Figure 1 node x is a group leader, the packet it sends upward is as follows:

$$\begin{aligned} x \rightarrow\rightarrow BS : x, 1, E(K_x, 15|Agg_x|S_g)|MAC_x \\ Agg_x = F_{agg}(R_x, Agg_w, Agg_{w'}) \\ MAC_x = MAC(K_x, 1|15|x|Agg_x|MAC_w \oplus MAC_{w'}|S_g), \end{aligned} \quad (5)$$

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 Figure 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 [Merkle 1989], 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.

4.3.3 *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 Figure 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 [Bloom 1970] 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.

4.4 Verification and Attestation

4.4.1 *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.

4.4.2 *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 nontrivial 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 [Frank 1969] to identify abnormal groups.

Definition 4.2. 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

$$Z = \frac{|\chi_i - \mu|}{s} \tag{6}$$

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 nonrejection 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 4.3. 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

$$P_c \cdot P_\chi < \alpha. \tag{7}$$

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 5.4 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 $\Gamma'$ (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 $L$;
```

4.4.3 *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 but 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

$$BS \rightarrow\rightarrow x : x, S_a, S_g. \tag{8}$$

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

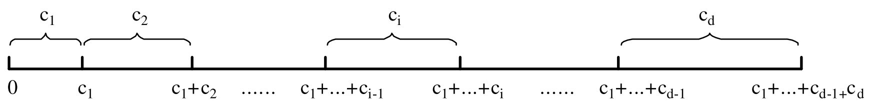


Fig. 2. Count intervals for a parent with *d* children.

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.

4.4.4 *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 4.4. 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 has 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 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 Figure 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 4.5. *Suppose a parent has d children with counts $c_1, c_2, \ldots, c_d$ respectively in a logical group. The probability that this parent selects the ith child with count c_i for attestation is*

$$P(i, d) = \frac{c_i}{\sum_{k=1}^{d} c_k}. \tag{9}$$

[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.

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)

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

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

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

From Fig. 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 $\rho$ of attested nodes' ids in the attestation path, initialized to $\{x\}$;
Procedure:
 1: parent = x;
 2: loop
 3:     $\tau$ = ids of all the children of parent;
 4:     $d$ = size of $\tau$;
 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 $\varrho = sum \cdot h$;
11:         if $\varrho \in [\sum_{k=1}^{i-1} c_k, \sum_{k=1}^{i} c_k)$ 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 $\rho$;
```

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.

Figure 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 6 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:

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

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 Figure 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.

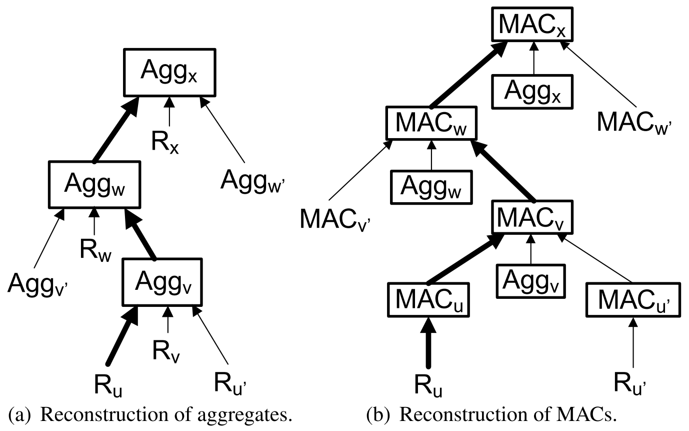


Fig. 3. The reconstruction operation flow in the BS. Values in the black frames are those recomputed by the BS. Others are those sent back to the BS. Thick arrows represent the attestation path.

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 5.2.1.

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.

5. 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).

5.1 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 [Merkle 1989] 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.

5.2 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 5.1. *The false positive rate F_p of the Grubbs' test in our scheme is 0, i.e.,*

$$F_p = 0. \tag{10}$$

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 a 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 6.2 (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 4.4.4, 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.

5.2.1 *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. 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.

LEMMA 5.2. *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*

$$D_r(j) = \prod_{l=1}^{j} P(i_l, d_l). \tag{11}$$

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 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}.

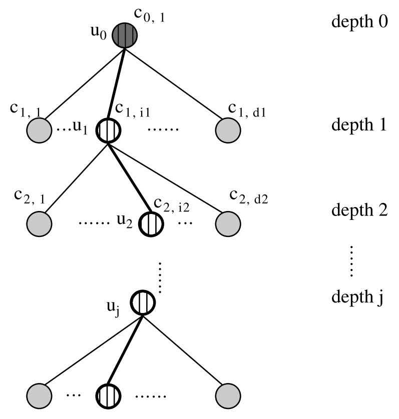


Fig. 4. Choosing one attestation path in a group based on counts.

According to Lemma 4.5, 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 5.3. *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*

$$D_r(j, n_p) = 1 - [1 - D_r(j)]^{n_p}. \tag{12}$$

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 5.2, we have

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

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

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

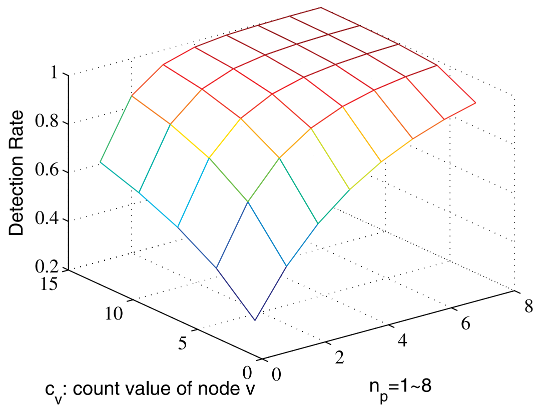


Fig. 5. Detection rate to the count changing attack launched by node v in the group with leader node x. For c_v=3, 6, 9, 12, 15, respectively. The number of attestation paths equals to $1 \sim 8$.

Since $D_r(j, n_p)$ in Equation (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 Figure 1 is an attacking node, our detection rate of the attack through multiple paths is shown in Figure 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 compromised nodes are on the same attestation path, then the detection rate of the attack equals 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.

5.2.2 *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 4.4.4 (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 5.2 and 5.3, 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.

5.3 Detection of Other Attacks

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

5.3.1 *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.

5.3.2 *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.

5.4 Other Attestation Techniques

The group attestation introduced in section 4.4.4 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 [Wagner 2004]. 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.

6. 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.

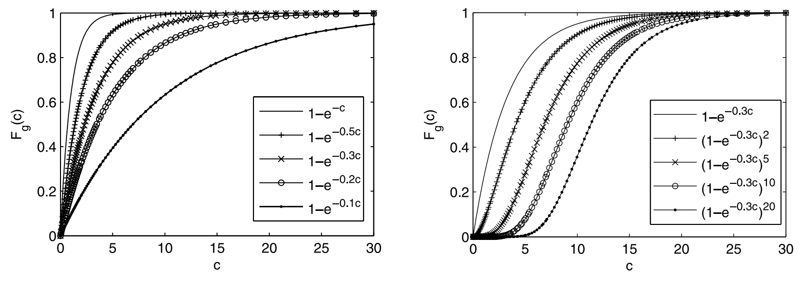


Fig. 6. $F_g(c)$ as the function of count c with parameters β, γ: $F_g(c) = (1-e^{-\beta \cdot c})^{\gamma}$ $(0 < \beta \leq 1, \gamma \geq 1)$.

6.1 Grouping Function and Grouping Results

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

6.1.1 *Grouping Function.* In Section 4.3.1, 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 Figure 6, when β increases, the curve becomes sharper. With a large γ, the function satisfies the fourth requirement of our grouping function.

6.1.2 *Grouping Results.* We verify that the grouping function satisfies our requirements through the simulated grouping results. In the simulation, 3,000 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 4.2 is used to build up the tree. For the grouping function, β and γ

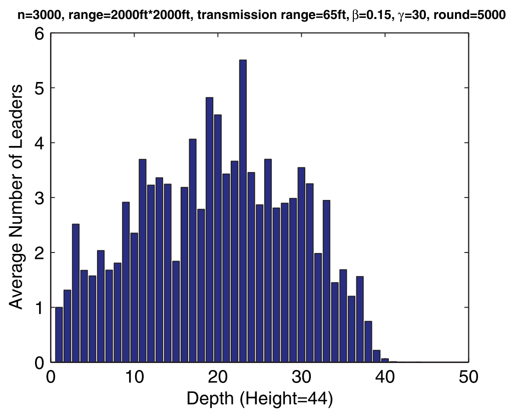


Fig. 7. Distribution of group leaders.

are set to 0.15 and 30, respectively. The simulation is run 5,000 rounds. Figure 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 4.2, 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 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 (Figure 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

$$p_i = F_g(c_i), \tag{13}$$

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

$$P(x = k) = \binom{n}{k} p_i^k \cdot (1 - p_i)^{d-k}. \tag{14}$$

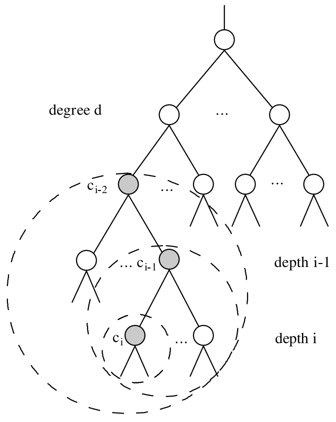


Fig. 8. Derivation of counts and probabilities.

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

$$c_{i-1} = \sum_{k=1}^{d} P(x = k) \cdot [c_i(d-k) + 1], \tag{15}$$

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

$$p_{i-1} = \sum_{k=1}^{d} P(x = k) \cdot F_g[c_i(d-k) + 1]. \tag{16}$$

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 (15) and (16).

To verify our theoretical derivation, we run the simulation 5,000 times for the same topology tree. The total number of nodes n is 3,280, 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 (Figure 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 5,000 values.

6.2 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

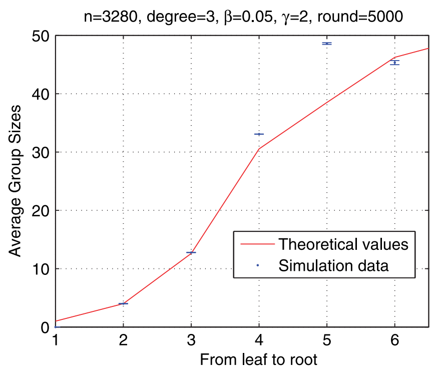


Fig. 9. Average counts for each depth.

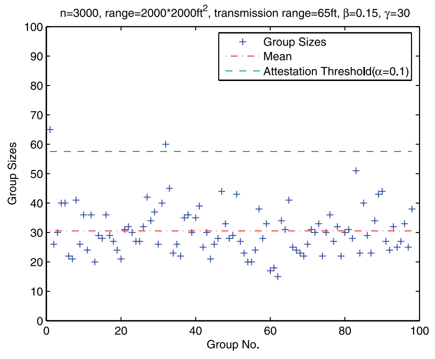
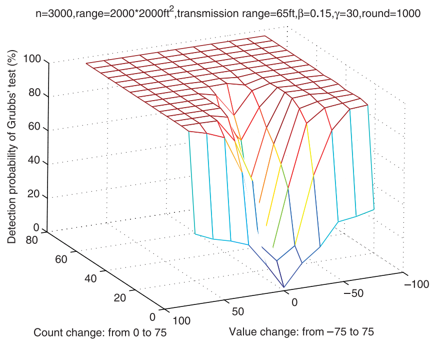


Fig. 10. Attestation threshold with no attacks.

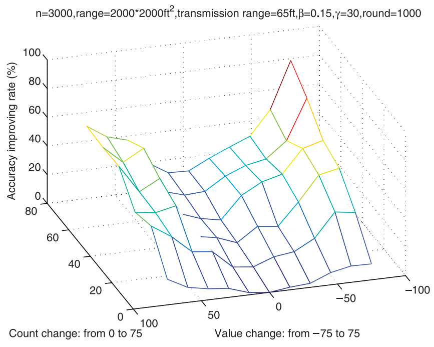
SUM, COUNT, and MEDIAN. Two metrics are used to evaluate the performance of Grubbs' test when there are attacks: *detection probability* and *accuracy improving rate*. The detection probability is the possibility that the Grubbs' test can detect the corresponding malicious groups. Suppose *total_no* 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 1,000 times to the same aggregation tree and calculate the average values.

6.2.1 *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.

Figure 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



(a) Detection probability of Grubbs' test.



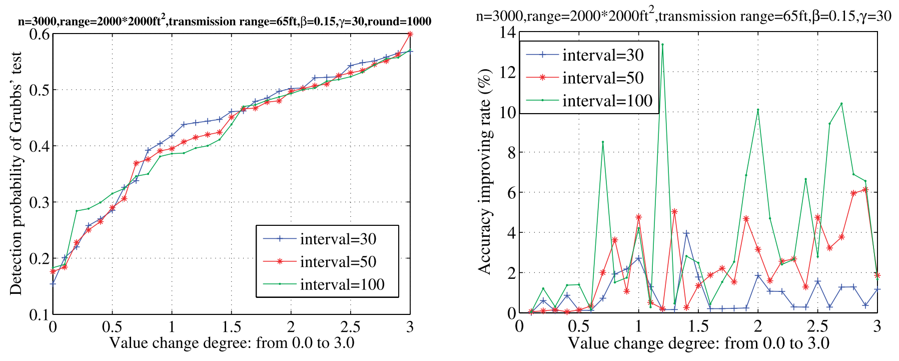
(b) Accuracy improving rate.

Fig. 11. The detection probability of Grubbs' test and accuracy improving rate when there is one malicious node.

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 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.

6.2.2 *Grubbs' Test with Single Malicious Node.* Figure 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 Figure 11a, 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 many other factors, such as the distance between



(a) Impact to detection probability. (b) Impact to accuracy improving rate.

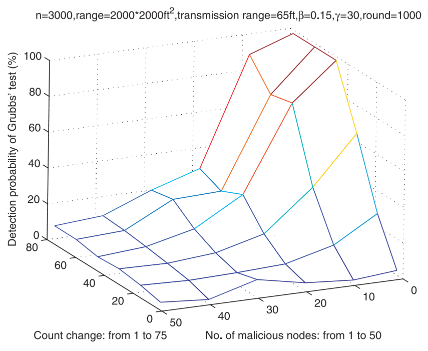
Fig. 12. The impact of sensor readings' distribution range to detection probability and accuracy improving rate.

the malicious node and group leader, the aggregates from other sibling nodes, and so on.

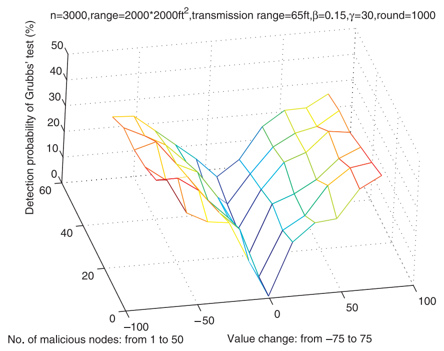
Figure 11b shows that the accuracy improving rate increases with a larger count or value change. Obviously, if the aggregate is changed by a larger degree, after removing this outlier, we can obtain a higher accuracy improvement. Additionally, if we increase and decrease the aggregation value by the 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). Figure 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 Figure 12a, 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 Figure 12b, the accuracy 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.



(a) Count changing attack.



(b) Value changing attack.

Fig. 13. The detection probability with multiple attacks.

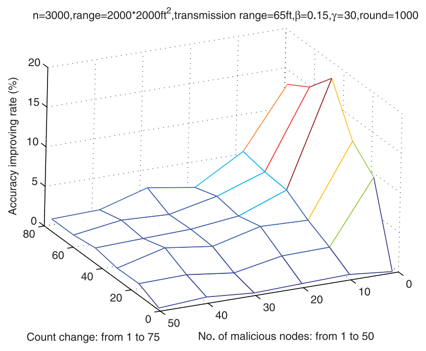
6.2.3 *Grubbs' Test with Multiple Malicious Nodes.* Instead of changing only one node's aggregate, we randomly choose multiple nodes and change their aggregates in this simulation. For simplicity, the changes to the aggregates of all the compromised nodes are the same. Figure 13 shows the detection probability of Grubbs' test under multiple attacks. As can be seen from Figure 13a, 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 Figure 13b, 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 Figure 14. From Figure 14a, 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 Figure 14b, 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.

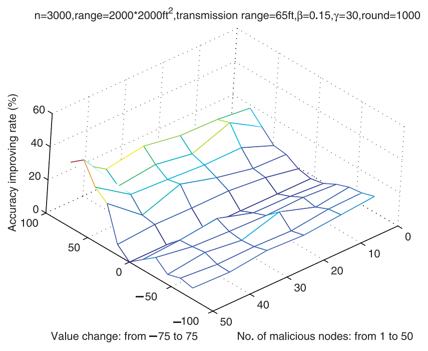
6.3 Overhead Evaluation

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

6.3.1 *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.



(a) Count changing attack.



(b) Value changing attack.

Fig. 14. The accuracy improving rate with multiple attacks.

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 [Rivest 1995] to implement all these cryptographic primitives, so as to reuse code and save memory space, because RC5 has advantages due to its versatility [Perrig et al. 2001; Karlof et al. 2004; Zhu et al. 2003]. 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 [Perrig et al. 2001].

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 [Ye et al. 2004]. 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.

6.3.2 *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 (3,000 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 nonvolatile measurement flash memory, the storage overhead is not a concern, either. This will also be verified by our prototype implementation in Section 6.4.

6.3.3 *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:

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

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. 7, this assumption is reasonable because

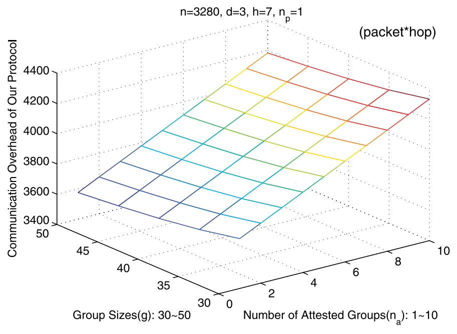


Fig. 15. Comm. overhead in packet*hop.

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

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

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 Figure 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).

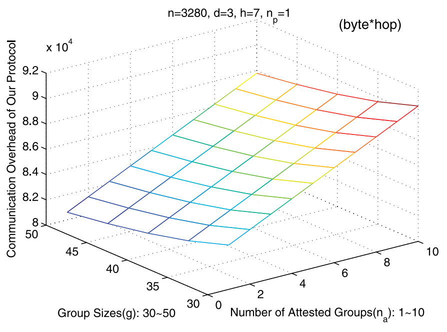


Fig. 16. Comm. overhead in byte*hop.

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 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 $10n_a n_p \lfloor h/2 \rfloor$. The overhead of sending data back to the BS is $21n_a n_p[\lfloor h/2 \rfloor+\sum_1^{\lceil h/2 \rceil}(\lfloor h/2 \rfloor+i)]+13n_a n_p \sum_1^{\lceil h/2 \rceil}(\lfloor h/2 \rfloor+i)(d-1)$, 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, and 8 bytes for MAC). The total communication overhead of our protocol in $byte * hop$ is given by

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

The result in $byte * hop$ (Figure 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-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, 3,000 nodes are randomly distributed in an area of $2000*2000\,ft^2$. The transmission range is set to $60\,ft$. 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),

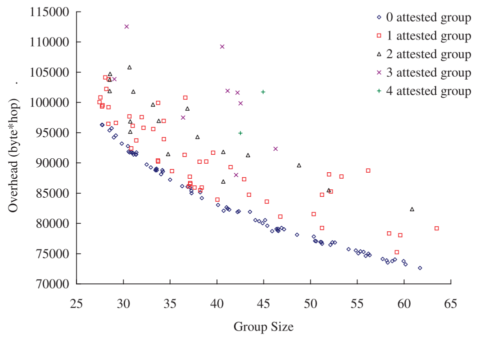


Fig. 17. Communication overhead based on simulations.

(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 Figure 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.

6.4 Prototype Implementation

We implement the prototype of our SDAP scheme on top of the TinyOS. The illustration of component relationship can be seen in Figure 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 [tin b]. The MAC, encryption and decryption mechanisms are used from TinySec [Karlof et al. 2004]. The pairwise key establishment mechanism is adapted from TinyKeyMan [tin a].

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

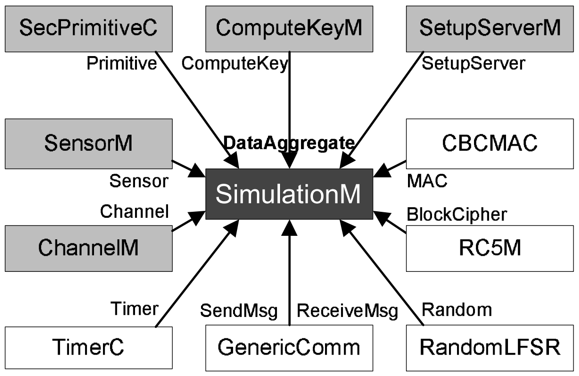


Fig. 18. The relationship among components in our implementation: Dark gray components are developed by us; light gray components are adapted from TinyKeyMan; and white components are directly adopted from TinyOS/TinySec.

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.

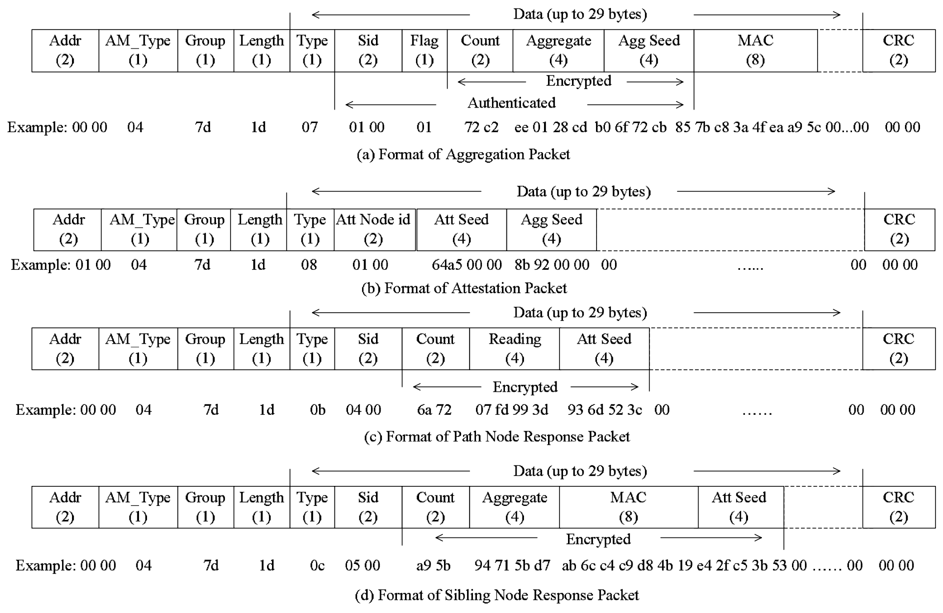

Data (up to 29 bytes)
Addr (2) AM_Type (1) Group (1) Length (1) Type (1) Sid (2) Flag (1) Count (2) Aggregate (4) Agg Seed (4) MAC (8) CRC (2)
Encrypted
Authenticated
Example: 00 00 04 7d 1d 07 01 00 01 72 c2 ee 01 28 cd b0 6f 72 cb 85 7b c8 3a 4f ea a9 5c 00...00 00 00
(a) Format of Aggregation Packet
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

Fig. 19. The formats of packets in our implementation. One example of each packet type is given to illustrate the specific format.

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 Figure 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 Figure 19a, 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 number in hex is 0x7d; and 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

Table I. 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

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 Figure 19b, Figure 19c, and Figure 19d, respectively.

On Mica2 motes, the 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 I. These results indicate that our scheme is practical when applied on current generation sensors such as Mica2 motes.

7. CONCLUSION AND FUTURE WORK

Providing hierarchical data aggregation without losing security guarantee is an interesting and challenging problem in sensor networks. In this article, we propose SDAP, a Secure Hop-by-hop Data Aggregation Protocol for sensor networks. 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.

REFERENCES

AKYILDIZ, I., SU, W., SANKARASUBRAMANIAM, Y., AND CAYIRCI, E. 2002. Wireless sensor networks: A survey. *Comput. Networks 38,* 4 (March).

BLOOM, B. H. 1970. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM 13,* 7, 422–426.

CASTELLUCCIA, C., MYKLETUN, E., AND TSUDIK, G. 2005. Efficient aggregation of encrypted data in wireless sensor networks. In *Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous'05).*

CHAN, H., PERRIG, A., PRZYDATEK, B., AND SONG, D. 2007. SIA: Secure information aggregation in sensor networks. *J. Comput. Secur. Special Issue on Adhoc and Sensor Networks*.

CHAN, H., PERRIG, A., AND SONG, D. 2006. Secure hierarchical in-network aggregation in sensor networks. In *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS'06)* . 278–287.

CHEN, J.-Y., PANDURANGAN, G., AND XU, D. 2005. Robust computation of aggregates in wireless sensor networks: distributed randomized algorithms and analysis. In *Proceedings of the International Symposium on Information Processing in Sensor Networks (IPSN'05)*. 348–355.

DU, W., DENG, J., HAN, Y. S., AND VARSHNEY, P. 2003a. A pairwise key pre-distribution scheme for wireless sensor networks. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS'03)*. Washington DC, 42–51.

DU, W., DENG, J., HAN, Y. S., AND VARSHNEY, P. K. 2003b. A witness-based approach for data fusion assurance in wireless sensor networks. In *Proceedings of the Global Telecommunications Conference (GLOBECOM'03)*.

ESCHENAUER, L. AND GLIGOR, V. D. 2002. A key-management scheme for distributed sensor networks. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS'02)*. 41–47.

ESTRIN, D., GOVINDAN, R., HEIDEMANN, J., AND KUMAR, S. 1999. Next century challenges: Scalable coordination in sensor networks. In *Proceedings of ACM Mobicom (Mobicom'99)*. ACM, Seattle, Washington, 263–270.

FRANK, G. 1969. Procedures for detecting outlying observations in samples. *Technometrics 11,* 1 (February), 1–21.

HE, W., LIU, X., NGUYEN, H., NAHRSTEDT, K., AND ABDELZAHER, T. 2007. PDA: Privacy-preserving data aggregation in wireless sensor networks. In *Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'07)*.

HILL, J., SZEWCZYK, R., WOO, A., HOLLAR, S., CULLER, D., AND PISTER, K. 2000. System architecture directions for networked sensors. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'00)*.

HU, L. AND EVANS, D. 2003. Secure aggregation for wireless networks. In *Proceedings of the Workshop on Security and Assurance in Ad Hoc Networks (SASN'03)*.

INTANAGONWIWAT, C., ESTRIN, D., GOVINDAN, R., AND HEIDEMANN, J. 2002. Impact of network density on data aggregation in wireless sensor networks. In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS'02)*. 457–458.

INTANAGONWIWAT, C., GOVINDAN, R., AND ESTRIN, D. 2000. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Proceedings of the International Conference on Mobile Computing and Networking (MobiCom'02)*. 56–67.

KARLOF, C., SASTRY, N., AND WAGNER, D. 2004. Tinysec: A link layer security architecture for wireless sensor networks. In *Second ACM Conference on Embedded Networked Sensor Systems (SenSys'04)*.

KRISHNAMACHARI, B., ESTRIN, D., AND WICKER, S. 2002. The impact of data aggregation in wireless sensor networks. In *Proceedings of the International Workshop on Distributed Event-Based Systems, (DEBS'02)*. Vienna, Austria.

LIU, D. AND NING, P. 2003. Establishing pairwise keys in distributed sensor networks. In *Proceedings of ACM Computer and Communications Security (CCS'03)*.

MADDEN, S., FRANKLIN, M. J., HELLERSTEIN, J. M., AND HONG, W. 2002. TAG: A tiny aggregation service for ad-hoc sensor networks. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI'02)*.

MCCUNE, J., SHI, E., PERRIG, A., AND REITER, M. 2005. Detection of denial-of-message attacks on sensor network broadcasts. In *IEEE Symposium on Security and Privacy (SP'05)*. 64–78.

MERKLE, R. 1989. A certified digital signature. In *Proceedings of Advances in Cryptology - 9th Annual International Cryptology Conference (CRYPTO'89)*. 218–238.

NATH, S., GIBBONS, P., SESHAN, S., AND ANDERSON, Z. 2004. Synopsis diffusion for robust aggregation in sensor networks. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys'04)*. 250–262.

PERRIG, A., Szewczyk, R., WEN, V., CULLER, D., AND TYGAR, J. D. 2001. SPINS: Security protocols for sensor networks. In *International Conference on Mobile Computing and Networking (MobiCom'01)*.

PRZYDATEK, B., SONG, D., AND PERRIG, A. 2003. SIA: secure information aggregation in sensor networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys'03)*. 255–265.

RIVEST, R. L. 1995. The RC5 encryption algorithm. In *Workshop on Fast Software Encryption (FSE'95)*. 86–96.

ROY, S., SETIA, S., AND JAJODIA, S. 2006. Attack-resilient hierarchical data aggregation in sensor networks. In *Proceedings of the 4th ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN'06)*. 71–82.

TinyKeyMan. http://discovery.csc.ncsu.edu/.

TinyOS. http://www.tinyos.net/.

WAGNER, D. 2004. Resilient aggregation in sensor networks. In *Proceedings of ACM Workshop on Security of Ad Hoc and Sensor Networks (SASN'04)*.

YAO, Y. AND GEHRKE, J. 2002. The Cougar approach to in-network query processing in sensor networks. *SIGMOD Record 31,* 3, 9–18.

YE, F., LUO, H., LU, S., AND ZHANG, L. 2004. Statistical en-route filtering of injected false data in sensor networks. In *Proceedings of the Annual Joint Conference of IEEE Computer and Communications Societies (INFOCOM'04)*.

ZHANG, W. AND CAO, G. 2005. Group Rekeying for Filtering False Data in Sensor Networks: A Predistribution and Local Collaboration-Based Approach. In *Proceedings of the Annual Joint Conference of IEEE Computer and Communications Societies (INFOCOM'05)*.

ZHANG, W., SONG, H., ZHU, S., AND CAO, G. 2005. Least privilege and privilege deprivation: Towards tolerating mobile sink compromises in wireless sensor networks. In *Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'05)*.

ZHU, S., SETIA, S., AND JAJODIA, S. 2003. LEAP: Efficient security mechanisms for large-scale distributed sensor networks. In *Proceedings of ACM Conference on Computers and Communications Security (CCS'03)*.

ZHU, S., SETIA, S., JAJODIA, S., AND NING, P. 2004. An interleaved hop-by-hop authentication scheme for filtering of injected false data in sensor networks. In *Proceedings of IEEE Symposium on Security and Privacy (SP'04)*. 259–271.

Received August 2006; revised August 2007; accepted February 2008