Secure Hierarchical In-Network Aggregation in Sensor Networks[*]

Haowen Chan
Carnegie Mellon University
haowenchan@cmu.edu

Adrian Perrig
Carnegie Mellon University
perrig@cmu.edu

Dawn Song
Carnegie Mellon University
dawnsong@cmu.edu

ABSTRACT

In-network aggregation is an essential primitive for performing queries on sensor network data. However, most aggregation algorithms assume that all intermediate nodes are trusted. In contrast, the standard threat model in sensor network security assumes that an attacker may control a fraction of the nodes, which may misbehave in an arbitrary (Byzantine) manner.

We present the first algorithm for provably secure hierarchical in-network data aggregation. Our algorithm is guaranteed to detect any manipulation of the aggregate by the adversary beyond what is achievable through direct injection of data values at compromised nodes. In other words, the adversary can never gain any advantage from misrepresenting intermediate aggregation computations. Our algorithm incurs only $O(\Delta \log^2 n)$ node congestion, supports arbitrary tree-based aggregator topologies and retains its resistance against aggregation manipulation in the presence of arbitrary numbers of malicious nodes. The main algorithm is based on performing the SUM aggregation securely by first forcing the adversary to commit to its choice of intermediate aggregation results, and then having the sensor nodes independently verify that their contributions to the aggregate are correctly incorporated. We show how to reduce secure MEDIAN, COUNT, and AVERAGE to this primitive.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and Protection*

General Terms

Security, Algorithms

[*]This research was supported in part by CyLab at Carnegie Mellon under grant DAAD19-02-1-0389 from the Army Research Office, and grant CNS-0347807 from the National Science Foundation, and by a gift from Bosch. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of ARO, Bosch, Carnegie Mellon University, NSF, or the U.S. Government or any of its agencies.

Keywords

Secure aggregation, Sensor Networks, Data aggregation

1. INTRODUCTION

Wireless sensor networks are increasingly deployed in security-critical applications such as factory monitoring, environmental monitoring, burglar alarms and fire alarms. The sensor nodes for these applications are typically deployed in unsecured locations and are not made tamper-proof due to cost considerations. Hence, an adversary could undetectably take control of one or more sensor nodes and launch active attacks to subvert correct network operations. Such environments pose a particularly challenging set of constraints for the protocol designer: sensor network protocols must be highly energy efficient while being able to function securely in the presence of possible malicious nodes within the network.

In this paper we focus on the particular problem of securely and efficiently performing aggregate queries (such as MEDIAN, SUM and AVERAGE) on sensor networks. In-network data aggregation is an efficient primitive for reducing the total message complexity of aggregate sensor queries. For example, in-network aggregation of the SUM function is performed by having each intermediate node forward a single message containing the sum of the sensor readings of all the nodes downstream from it, rather than forwarding each downstream message one-by-one to the base station. The energy savings of performing in-network aggregation have been shown to be significant and are crucial for energy-constrained sensor networks [9, 11, 20].

Unfortunately, most in-network aggregation schemes assume that all sensor nodes are trusted [12, 20]. An adversary controlling just a few aggregator nodes could potentially cause the sensor network to return arbitrary results, thus completely subverting the function of the network to the adversary's own purposes.

Despite the importance of the problem and a significant amount of work on the area, the known approaches to secure aggregation either require strong assumptions about network topology or adversary capabilities, or are only able to provide limited probabilistic security properties. For example, Hu and Evans [8] propose a secure aggregation scheme under the assumption that at most a single node is malicious. Przydatek et al. [17] propose Secure Information Aggregation (SIA), which provides a statistical security property under the assumption of a single-aggregator model. In the single-aggregator model, sensor nodes send their data to a single aggregator node, which computes the aggregate and sends it to the base station. This form of aggregation reduces communications only on the link between the aggregator and the base station, and is not scalable to large multihop sensor deployments. Most of the algorithms in SIA (in particular, MEDIAN, SUM and AVERAGE) cannot be directly adapted to a hierarchical aggregation model since they involve sorting all of the input values; the final aggregator in the hierarchy thus needs to access all the data values of the sensor nodes.

In this paper, we present the first provably secure sensor network data aggregation protocol for general networks and multiple adversarial nodes. The algorithm limits the adversary's ability to manipulate the aggregation result with the tightest bound possible for general algorithms with no knowledge of the distribution of sensor data values. Specifically, an adversary can gain no additional influence over the final result by manipulating the results of the in-network aggregate computation as opposed to simply reporting false data readings for the compromised nodes under its control. Furthermore, unlike prior schemes, our algorithm is designed for general hierarchical aggregator topologies and multiple malicious sensor nodes. Our metric for communication cost is *congestion*, which is the maximum communication load on any node in the network. Let n be the number of nodes in the network, and Δ be the maximum degree of any node in the aggregation tree. Our algorithm induces only $O(\Delta \log^2 n)$ node congestion in the aggregation tree.

2. RELATED WORK

Researchers have investigated resilient aggregation algorithms to provide increased likelihood of accurate results in environments prone to message loss or node failures. This class of algorithms includes work by Gupta et al. [7], Nath et al. [15], Chen et al. [3] and Manjhi et al. [14].

A number of aggregation algorithms have been proposed to ensure *secrecy* of the data against intermediate aggregators. Such algorithms have been proposed by Girao et al. [5], Castelluccia et al. [2], and Cam et al. [1].

Hu and Evans [8] propose securing in-network aggregation against a single Byzantine adversary by requiring aggregator nodes to forward their inputs to their parent nodes in the aggregation tree. Jadia and Mathuria [10] extend the Hu and Evans approach by incorporating privacy, but also considered only a single malicious node.

Several secure aggregation algorithms have been proposed for the single-aggregator model. Przydatek et al. [17] proposed Secure Information Aggregation (SIA) for this topology. Also for the single-aggregator case, Du et al. [4] propose using multiple *witness* nodes as additional aggregators to verify the integrity of the aggregator's result. Mahimkar and Rappaport [13] also propose an aggregation-verification scheme for the single-aggregator model using a threshold signature scheme to ensure that at least t of the nodes agree with the aggregation result. Yang et al. [19] describe a probabilistic aggregation algorithm which subdivides an aggregation tree into subtrees, each of which reports their aggregates directly to the base station. Outliers among the subtrees are then probed for inconsistencies.

Wagner [18] addressed the issue of measuring and bounding malicious nodes' contribution to the final aggregation result. The paper measures how much damage an attacker can inflict by taking control of a number of nodes and using them solely to inject erroneous data values.

3. PROBLEM MODEL

In general, the goal of secure aggregation is to compute aggregate functions (such as SUM, COUNT or AVERAGE) of the sensed data values residing on sensor nodes, while assuming that a portion of the sensor nodes are controlled by an adversary which is attempting to skew the final result. In this section, we present the formal parameters of the problem.

3.1 Network Assumptions

We assume a general multihop network with a set $S = \{s_1, \ldots, s_n\}$ of n sensor nodes and a single (untrusted) base station R, which is able to communicate with the querier which resides outside of the network. The querier knows the total number of sensor nodes n, and that all n nodes are alive and reachable.

We assume the aggregation is performed over an *aggregation tree* which is the directed tree formed by the union of all the paths from the sensor nodes to the base station (one such tree is shown in Figure 1(a)). These paths may be arbitrarily chosen and are not necessarily shortest paths. The optimisation of the aggregation tree structure is out of the scope of this paper—our algorithm takes the structure of the aggregation tree as given. One method for constructing an aggregation tree is described in TaG [11].

3.2 Security Infrastructure

We assume that each sensor node has a unique identifier s and shares a unique secret symmetric key K_s with the querier. We further assume the existence of a broadcast authentication primitive where any node can authenticate a message from the querier. This broadcast authentication could, for example, be performed using μTESLA [16]. We assume the sensor nodes have the ability to perform symmetric-key encryption and decryption as well as computations of a collision-resistant cryptographic hash function H.

3.3 Attacker Model

We assume that the attacker is in complete control of an *arbitrary number* of sensor nodes, including knowledge of all their secret keys. The attacker has a network-wide presence and can record and inject messages at will. The sole goal of the attacker is to launch what Przydatek et al. [17] call a *stealthy attack*, i.e., to cause the querier to accept a false aggregate that is higher or lower than the true aggregate value.

We do not consider denial-of-service (DoS) attacks where the goal of the adversary is to prevent the querier from getting any aggregation result at all. While such attacks can disrupt the normal operation of the sensor network, they are not as potentially hazardous in security-critical applications as the ability to cause the operator of the network to accept arbitrary data. Furthermore, any maliciously induced extended loss of service is a detectable anomaly which will (eventually) expose the adversary's presence if subsequent protocols or manual intervention do not succeed in resolving the problem.

3.4 Problem Definition and Metrics

Each sensor node s_i has a data value a_i. We assume that the data value is a *non-negative* bounded real value $a_i \in [0, r]$ for some maximum allowed data value r. The objective of the aggregation process is to compute some function f over all the data values, i.e., $f(a_1, \ldots, a_n)$. Note that for the SUM aggregate, the case where data values are in a range $[r_1, r_2]$ (where r_1, r_2 can be negative) is reducible to this case by setting $r = r_2 - r_1$ and add nr_1 to the aggregation result.

Definition 1 *A* ***direct data injection*** *attack occurs when an attacker modifies the data readings reported by the nodes under its direct control, under the constraint that only legal readings in $[0, r]$ are reported.*

Wagner [18] performed a quantitative study measuring the effect of direct data injection on various aggregates, and concludes that the aggregates addressed in this paper (truncated SUM and AVERAGE, COUNT and Φ-QUANTILE) can be resilient under such attacks.

Without domain knowledge about what constitutes an anomalous sensor reading, it is impossible to detect a direct data injection attack, since they are indistinguishable from legitimate sensor readings [17, 19]. Hence, if a secure aggregation scheme does not make assumptions on the distribution of data values, it cannot limit the adversary's capability to perform direct data injection. We can thus define an optimal level of aggregation security as follows.

Definition 2 *An aggregation algorithm is **optimally secure** if, by tampering with the aggregation process, an adversary is unable to induce the querier to accept any aggregation result which is not already achievable by direct data injection.*

As a metric for communication overhead, we consider node *congestion*, which is the worst case communication load on any single sensor node during the algorithm. Congestion is a commonly used metric in ad-hoc networks since it measures how quickly the heaviest-loaded nodes will exhaust their batteries [6, 12]. Since the heaviest-loaded nodes are typically the nodes which are most essential to the connectivity of the network (e.g., the nodes closest to the base station), their failure may cause the network to partition even though other sensor nodes in the network may still have high battery levels. A lower communication load on the heaviest-loaded nodes is thus desirable even if the trade-off is a larger amount of communication in the network as a whole.

For a lower bound on congestion, consider an unsecured aggregation protocol where each node sends just a single message to its parent in the aggregation tree. This is the minimum number of messages that ensures that each sensor node contributes to the aggregation result. There is $\Omega(1)$ congestion on each edge on the aggregation tree, thus resulting in $\Omega(d)$ congestion on the node(s) with highest degree d in the aggregation tree. The parameter d is dependent on the shape of the given aggregation tree and can be as large as $\Theta(n)$ for a single-aggregator topology or as small as $\Theta(1)$ for a balanced aggregation tree. Since we are taking the aggregation tree topology as an input, we have no control over d. Hence, it is often more informative to consider per-edge congestion, which can be independent of the structure of the aggregation tree.

Consider the simplest solution where we omit aggregation altogether and simply send all data values (encrypted and authenticated) directly to the base station, which then forwards it to the querier. This provides perfect data integrity, but induces $O(n)$ congestion at the nodes and edges nearest the base station. For an algorithm to be practical, it must cause only sublinear edge congestion.

Our goal is to design an **optimally secure** aggregation algorithm with only **sublinear edge congestion**.

4. THE SUM ALGORITHM

In this section we describe our algorithm for the SUM aggregate, where the aggregation function f is addition. Specifically, we wish to compute $a_1 + \cdots + a_n$, where a_i is the data value at node i. We defer analysis of the algorithm properties to Section 5, and discuss the application of the algorithm to other aggregates such as COUNT, AVERAGE and MEDIAN in Section 6.

We build on the aggregate-commit-prove framework described by Przydatek et al. [17] but extend their single aggregator model to a fully distributed setting. Our algorithm involves computing a cryptographic commitment structure (similar to a hash tree) over the data values of the sensor nodes as well as the aggregation process. This forces the adversary to choose a fixed aggregation topology and set of aggregation results. The individual sensor nodes then independently audit the commitment structure to verify that their respective contributions have been added to the aggregate. If the adversary attempts to discard or reduce the contribution of a legitimate sensor node, this necessarily induces an inconsistency in the commitment structure which can be detected by the affected node. This basic approach provides us with a lower bound for the SUM aggregate. To provide an upper-bound for SUM, we can reuse the same lower-bounding approach, but on a complementary aggregate called the COMPLEMENT aggregate. Where SUM is defined as $\sum a_i$, COMPLEMENT is defined as $\sum(r - a_i)$ where r is the upper bound on allowable data values. When the final aggregates are computed, the querier enforces the constraint that SUM + COMPLEMENT $= nr$. Hence any adversary that wishes to increase SUM must also decrease COMPLEMENT, and vice-versa, otherwise the discrepancy will be detected. Hence, by enforcing a lower-bound on COMPLEMENT, we are also enforcing an upper-bound on SUM.

The overall algorithm has three main phases: query dissemination, aggregation-commit, and result-checking.

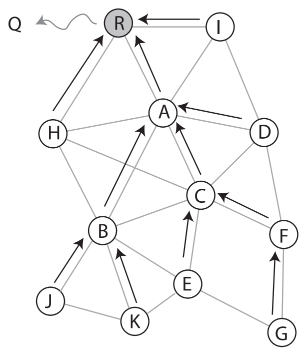
Query dissemination. The base station broadcasts the query to the network. An *aggregation tree*, or a directed spanning tree over the network topology with the base station at the root, is formed as the query is sent to all the nodes, if one is not already present in the network.

Aggregation commit. In this phase, the sensor nodes iteratively construct a commitment structure resembling a hash tree. First, the leaf nodes in the aggregation tree send their data values to their parents in the aggregation tree. Each internal sensor node in the aggregation tree performs an aggregation operation whenever it has heard from all its child sensor nodes. Whenever a sensor node s performs an aggregation operation, s creates a commitment to the set of inputs used to compute the aggregate by computing a hash over all the inputs (including the commitments that were computed by the children of s). Both the aggregation result and the commitment are then passed on to the parent of s. After the final commitment values are reported to the base station (and thus also to the querier), the adversary cannot subsequently claim a different aggregation structure or result. We describe an optimisation to ensure that the constructed commitment trees are perfectly balanced, thus requiring low congestion overhead in the next phase.

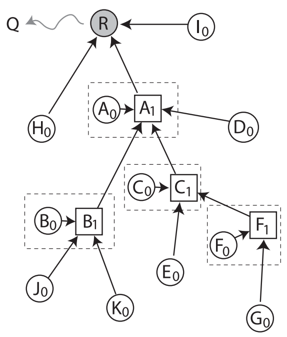
Result-checking. The result-checking phase is a novel distributed verification process. In prior work, algorithms have relied on the querier to issue probes into the commitment structure to verify its integrity [17, 19]. This induces congestion nearest the base station, and moreover, such algorithms yield at best probabilistic security properties. We show that if the verification step is instead fully distributed, it is possible to achieve provably *optimal* security while maintaining sublinear edge congestion.

The result-checking phase proceeds as follows. Once the querier has received the final commitment values, it disseminates them to the rest of the network in an authenticated broadcast. At the same time, sensor nodes disseminate information that will allow their peers to verify that their respective data values have been incorporated into the aggregate. Each sensor node is responsible for checking that its own contribution was added into the aggregate. If a sensor node determines that its data value was indeed added towards the final sum, it sends an authentication code up the aggregation tree towards to the base station. Authentication codes are aggregated along the way with the XOR function for communication efficiency. When the querier has received the XOR of all the authentication codes, it can then verify that all the sensor nodes have confirmed that the aggregation structure is consistent with their data values. If so, then it accepts the aggregation result.

We now describe the details of each of the three phases in turn.



(a) Example network graph. Arrows: Aggregation tree. R: Base station. Q: Querier.



$R = \langle 12, v_R, \overline{v}_R, H[N||12||v_R||\overline{v}_R||H_0||A_1||I_0] \rangle$

$A_1 = \langle 9, v_{A_1}, \overline{v}_{A_1}, H[N||9||v_{A_1}||\overline{v}_{A_1}||A_0||B_1||C_1||D_0] \rangle$

$C_1 = \langle 4, v_{C_1}, \overline{v}_{C_1}, H[N||4||v_{C_1}||\overline{v}_{C_1}||C_0||E_0||F_1] \rangle$

$F_1 = \langle 2, v_{F_1}, \overline{v}_{F_1}, H[N||2||v_{F_1}||\overline{v}_{F_1}||F_0||G_0] \rangle$

$G_0 = \langle 1, a_G, r - a_G, G \rangle$

(b) Naive commitment tree, showing derivations of some of the vertices. For each sensor node X, X_0 is its leaf vertex, while X_1 is the internal vertex representing the aggregate computation at X (if any). On the right we list the labels of the vertices on the path of node G to the root.

Figure 1: Aggregation and naive commitment tree in network context

4.1 Query Dissemination

First, an aggregation tree is established if one is not already present. Various algorithms for selecting the structure of an aggregation tree may be used. For completeness, we describe one such process, while noting that our algorithm is directly applicable to any aggregation tree structure. The Tiny Aggregation Service (TaG) [11] uses a broadcast from the base station where each node chooses as its parent in the aggregation tree, the node from which it first heard the tree-formation message.

To initiate a query in the aggregation tree, the base station originates a query request message which is distributed following the aggregation tree. The query request message contains an attached nonce N to prevent replay of messages belonging to a prior query, and the entire request message is sent using an authenticated broadcast.

4.2 Aggregation-Commit Phase

The goal of the aggregation-commit phase is to iteratively construct a series of cryptographic commitments to data values and to intermediate in-network aggregation operations. This commitment is then passed on to the querier. The querier then rebroadcasts the commitment to the sensor network using an authenticated broadcast so that the rest of the sensor network is able to verify that their respective data values have been incorporated into the aggregate.

4.2.1 Aggregation-Commit: Naive Approach

We first describe a naive approach that yields the desired security properties but has suboptimal congestion overhead when sensor nodes perform their respective verifications. In the naive approach, when each sensor node performs an aggregation operation, it computes a cryptographic hash of all its inputs (including its own data value). The hash value is then passed on to the parent in the aggregation tree along with the aggregation result. Figure 1(b) shows a *commitment tree* which consists of a series of hashes of data values and intermediate results, culminating in a set of final commitment values which is passed on by the base station to the querier along with the aggregation results. Conceptually, a commitment tree is a hash tree with some additional aggregate accounting information attached to the nodes. A definition follows. Recall that N is the query nonce that is disseminated with each query.

Definition 3 *A **commitment tree** is a tree where each vertex has an associated label representing the data that is passed on to its parent. The labels have the following format:*

$$\langle \texttt{count, value, complement, commitment} \rangle$$

Where `count` *is the number of leaf vertices in the subtree rooted at this vertex;* `value` *is the* SUM *aggregate computed over all the leaves in the subtree;* `complement` *is the aggregate over the* COMPLEMENT *of the data values; and* `commitment` *is a cryptographic commitment. The labels are defined inductively as follows:*

*There is one leaf vertex u_s for each sensor node s, which we call the **leaf vertex of** s. The label of u_s consists of* `count`*=1,* `value`*=a_s where a_s is the data value of s,* `complement`*=$r - a_s$ where r is the upper bound on allowable data values, and* `commitment` *is the node's unique ID.*

Internal vertices represent aggregation operations, and have labels that are defined based on their children. Suppose an internal vertex has child vertices with the following labels: $u_1, u_2, \ldots, u_q$, where $u_i = \langle c_i, v_i, \overline{v}_i, h_i \rangle$. Then the vertex has label $\langle c, v, \overline{v}, h \rangle$, with $c = \sum c_i$, $v = \sum v_i$, $\overline{v} = \sum \overline{v}_i$ and $h = H[N||c||v||\overline{v}||u_1||u_2||\cdots||u_q]$.

For brevity, in the remainder of the paper we will often omit references to labels and instead refer directly to the `count`, `value`, `complement` or `commitment` of a vertex.

While there exists a natural mapping between vertices in a commitment tree and sensor nodes in the aggregation tree, a vertex is a logical element in a graph while a sensor node is a physical device. To prevent confusion, we will always refer to the *vertices* in the commitment tree; the term *nodes* always refers to the physical sensor node device.

Since we assume that our hash function provides collision resistance, it is computationally infeasible for an adversary to change any of the contents of the commitment tree once the final commitment values have reached the root.

With knowledge of the root commitment value, a node s may verify the aggregation steps between its leaf vertex u_s and the root of the commitment tree. To do so, s needs the labels of all its *off-path* vertices.

Definition 4 *The set of **off-path** vertices for a vertex u in a tree is the set of all the siblings of each of the vertices on the path from u to the root of the tree that u is in (the path is inclusive of u).*

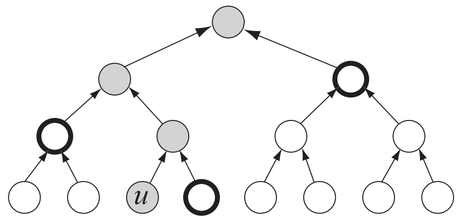


Figure 2: Off-path vertices for u are highlighted in bold. The path from u to the root of its tree is shaded grey.

Figure 2 shows a pictorial depiction of the off-path vertices for a vertex u in a tree. For a more concrete example, the set of off-path commitment tree vertices for G_0 in Figure 1 is $\{F_0, E_0, C_0, B_1, A_0, D_0, H_0, I_0\}$. To allow sensor node G to verify its contribution to the aggregate, the sensor network delivers labels of each off-path vertex to G_0. Sensor node G then recomputes the sequence of computations and hashes and verifies that they lead to the correct root commitment value.

Consider the congestion on the naive scheme. Let h be the height of the aggregation tree and Δ be the maximum degree of any node inside the tree. Each leaf vertex has $O(h\Delta)$ off-path vertices, and it needs to receive all their labels to verify its contribution to the aggregate, thus leading to $O(h\Delta)$ congestion at the leaves of the commitment tree. For an aggregation tree constructed with TaG, the height h of the aggregation tree depends on the diameter (in number of hops) of the network, which in turn depends on the node density and total number of nodes n in the network. In a 2-dimensional deployment area with a constant node density, the best bound on the diameter of the network is $O(\sqrt{n})$ if the network is regularly shaped. In irregular topologies the diameter of the network may be $\Omega(n)$.

4.2.2 Aggregation-Commit: Improved Approach

We present an optimization to improve the congestion cost. The main observation is that, since the aggregation trees are a subgraph of the network topology, they may be arbitrarily unbalanced. Hence, if we decouple the structure of the commitment tree from the structure of the aggregation tree, then the commitment tree could be perfectly balanced.

In the naive commitment tree, each sensor node always computes the aggregate sum of *all* its inputs. This can be considered a strategy of *greedy aggregation*. Consider instead the benefit of *delayed aggregation* at node C_1 in Figure 1(b). Suppose that C, instead of greedily computing the aggregate sum over its own reading (C_0) and both its child nodes E_0 and F_1, instead computes the sum *only* over C_0 and E_0, and passes F_1 directly to A along with $C_1 = C_0 + E_0$. In such a commitment tree, F_1 becomes a child of A_1 (instead of C_1), thus reducing the depth of the commitment tree by 1. Delayed aggregation thus trades off increased communication during the aggregation phase in return for a more balanced commitment tree, which results in lower verification overhead in the result-checking phase. Greenwald and Khanna [6] used a form of delayed aggregation in their quantile summary algorithm.

Our strategy for delayed aggregation is as follows: we perform an aggregation operation (along with the associated commit operation) if and only if it results in a *complete, binary* commitment tree.

We now describe our delayed aggregation algorithm for producing balanced commitment trees. In the naive commitment tree, each sensor node passes to its parent a single message containing the label of the root vertex of its commitment subtree T_s. In the delayed aggregation algorithm, each sensor node now passes on the labels of the root vertices of a *set* of commitment subtrees $F = \{T_1, \ldots, T_q\}$. We call this set a *commitment forest*, and we enforce the condition that the trees in the forest must be complete binary trees, and no two trees have the same height. These constraints are enforced by continually combining equal-height trees into complete binary trees of greater height.

Definition 5 *A* ***commitment forest*** *is a set of complete binary commitment trees such that there is at most one commitment tree of any given height.*

A commitment forest has at most n leaf vertices (one for each sensor node included in the forest, up to a maximum of n). Since all the trees are complete binary trees, the tallest tree in any commitment forest has height at most $\log n$. Since there are no two trees of the same height, any commitment forest has at most $\log n$ trees.

In the following discussion, we will for brevity make reference to "communicating a vertex" to another sensor node, or "communicating a commitment forest" to another sensor node. The actual data communicated is the *label* of the vertex and the *labels of the roots* of the trees in the commitment forest, respectively.

The commitment forest is built as follows. Leaf sensor nodes in the aggregation tree originate a single-vertex commitment forest, which they then communicate to their parent sensor nodes. Each internal sensor node s originates a similar single-vertex commitment forest. In addition, s also receives commitment forests from each of its children. Sensor node s keeps track of which root vertices were received from which of its children. It then combines all the forests to form a new forest as follows.

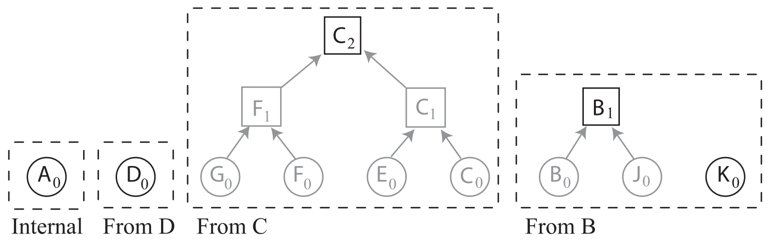
Suppose s wishes to combine q commitment forests $F_1, \ldots, F_q$. Note that since all commitment trees are complete binary trees, tree heights can be determined by inspecting the `count` field of the root vertex. We let the intermediate result be $F = F_1 \cup \cdots \cup F_q$, and repeat the following until no two trees are the same height in F: Let h be the smallest height such that more than one tree in F has height h. Find two commitment trees T_1 and T_2 of height h in F, and merge them into a tree of height $h+1$ by creating a new vertex that is the parent of both the roots of T_1 and T_2 according to the inductive rule in Definition 3. Figure 3 shows an example of the process for node A based on the topology in Figure 1.

The algorithm terminates in $O(q \log n)$ steps since each step reduces the number of trees in the forest by one, and there are at most $q \log n + 1$ trees in the forest. Hence, each sensor node creates at most $q \log n + 1 = O(\Delta \log n)$ vertices in the commitment forest.

When F is a valid commitment forest, s sends the root vertices of each tree in F to its parent sensor node in the aggregation tree. The sensor node s also keeps track of every vertex that it created, as well as all the inputs that it received (i.e., the labels of the root vertices of the commitment forests that were sent to s by its children). This takes $O(d \log n)$ memory per sensor node.

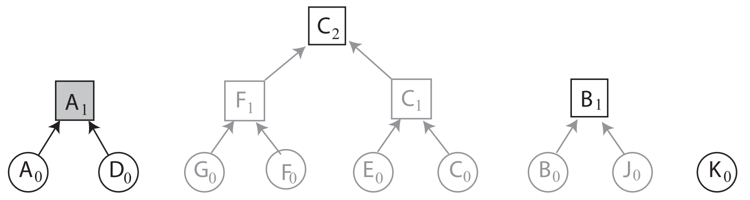
Consider the communication costs of the entire process of creating the final commitment forest. Since there are at most $\log n$ commitment trees in each of the forests presented by any sensor node to its parent, the per-node communication cost for constructing the final forest is $O(\log n)$. This is greater than the $O(1)$ congestion cost of constructing the naive commitment tree. However, no path in the forest is longer than $\log n$ hops. This will eventually enable us to prove a bound of $O(\log^2 n)$ edge congestion for the result-checking phase in Section 5.2.

Once the querier has received the final commitment forest from the base station, it checks that none of the SUM or COMPLEMENT aggregates of the roots of the trees in the forest are negative. If



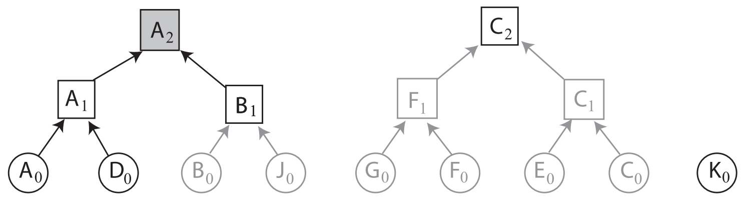
$$A_0 = \langle 1, a_A, r - a_A, A \rangle$$
$$D_0 = \langle 1, a_D, r - a_D, D \rangle$$
$$C_2 = \langle 4, v_{C_2}, \overline{v}_{C_2}, H[N||4||v_{C_2}||\overline{v}_{C_2}||F_1||C_1] \rangle$$
$$B_1 = \langle 2, v_{B_1}, \overline{v}_{B_1}, H[N||2||v_{B_1}||\overline{v}_{B_1}||B_0||J_0] \rangle$$
$$K_0 = \langle 1, a_K, r - a_K, K \rangle$$

(a) Inputs: A generates A_0, and receives D_0 from D, C_2 from C, and (B_1, K_0) from B. Each dashed-line box shows the commitment forest received from a given sensor node. The solid-line box shows the vertex labels, each solid-line box below shows the labels of the new vertices.



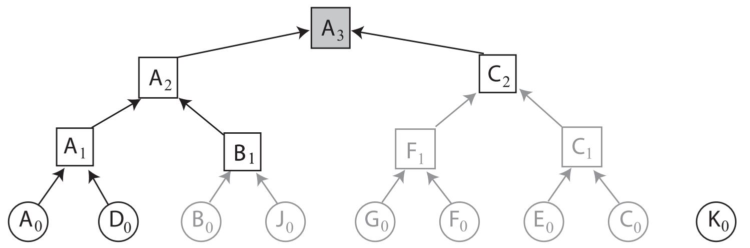
$$v_{A_1} = a_A + a_D$$
$$\overline{v}_{A_1} = r - a_A + r - a_D$$
$$A_1 = \langle 2, v_{A_1}, \overline{v}_{A_1}, H[N||2||v_{A_1}||\overline{v}_{A_1}||A_0||D_0] \rangle$$

(b) First merge: Vertex A_1 created



$$v_{A_2} = v_{A_1} + v_{B_1}$$
$$\overline{v}_{A_2} = \overline{v}_{A_1} + \overline{v}_{B_1}$$
$$A_2 = \langle 4, v_{A_2}, \overline{v}_{A_2}, H[N||4||v_{A_2}||\overline{v}_{A_2}||A_1||B_1] \rangle$$

(c) Second merge: Vertex A_2 created



$$v_{A_3} = v_{A_2} + v_{C_2}$$
$$\overline{v}_{A_3} = \overline{v}_{A_2} + \overline{v}_{C_2}$$
$$A_3 = \langle 8, v_{A_3}, \overline{v}_{A_3}, H[N||8||v_{A_3}||\overline{v}_{A_3}||A_2||C_2] \rangle$$

(d) Final merge: Vertex A_3 created. A_3 and K_0 are sent to the parent of A in the aggregation tree.

Figure 3: Process of node *A* (from Figure 1) deriving its commitment forest from the commitment forests received from its children.

any aggregates are negative, the querier rejects the result and raises an alarm: a negative aggregate is a sure sign of tampering since all the data values (and their complements) are non-negative. Otherwise, the querier then computes the final pair of aggregates SUM and COMPLEMENT. The querier verifies that SUM + COMPLEMENT $= nr$ where r is the upper bound on the range of allowable data values on each node. If this verifies correctly, the querier then initiates the *result-checking* phase.

4.3 Result-checking phase

The purpose of the result-checking phase is to enable each sensor node s to independently verify that its data value a_s was added into the SUM aggregate, and the complement $(r - a_s)$ of its data value was added into the COMPLEMENT aggregate. The verification is performed by inspecting the inputs and aggregation operations in the commitment forest on the path from the leaf vertex of s to the root of its tree; if all the operations are consistent, then the root aggregate value must have increased by a_s due to the incorporation of the data value. If each legitimate node performs this verification, then it ensures that the SUM aggregate is at least the sum of all the data values of the legitimate nodes. Similarly, the COMPLEMENT aggregate is at least the sum of all the complements of the data values of the legitimate nodes. Since the querier enforces SUM + COMPLEMENT $= nr$, these two inequalities form lower and upper bounds on an adversary's ability to manipulate the final result. In Section 5 we shall show that they are in fact the tightest bounds possible.

A high level overview of the process is as follows. First, the aggregation results from the aggregation-commit phase are sent using authenticated broadcast to every sensor node in the network. Each sensor node then individually verifies that its contributions to the respective SUM and COMPLEMENT aggregates were indeed counted. If so, it sends an authentication code to the base station. The authentication code is also aggregated for communication effi-

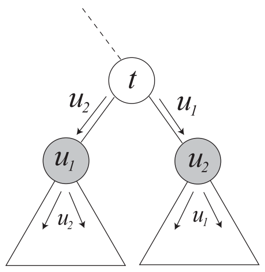


Figure 4: Dissemination of off-path values: t sends the label of u_1 to u_2 and vice-versa; each node then forwards it to all the vertices in their subtrees.

ciency. When the querier has received all the authentication codes, it is then able to verify that all sensor nodes have checked that their contribution to the aggregate has been correctly counted.

For simplicity, we describe each step of the process with reference to the commitment tree visualised as an *overlay network* over the actual aggregation tree. Hence, we will refer to *vertices* in the commitment tree sending information to each other; in the physical world, it is the sensor node that created the vertex is the physical entity that is responsible for performing communications and computations on behalf of the vertex. Each edge in the commitment tree may involve multiple hops in the aggregation tree; the routing on the aggregation tree is straightforward.

Dissemination of final commitment values. After the querier has received the labels of the roots of the final commitment forest, the querier sends each of these labels to the entire sensor network using authenticated broadcast.

Dissemination of off-path values. To enable verification, each leaf vertex must receive all its off-path values. Each internal vertex t in the commitment forest has two children u_1 and u_2. To disseminate off-path values, t sends the label of u_1 to u_2, and vice-versa (t also attaches relevant information tagging u_1 as the right child and u_2 as the left child). Vertex t also sends any labels (and left/right tags) received from its parent to both its children. See Figure 4 for an illustration of the process. The correctness of this algorithm in delivering all the necessary off-path vertex labels to each vertex is proven in Theorem 14 in Section 5.2. Once a vertex has received all the labels of its off-path vertices, it can proceed to the verification step.

Verification of inclusion. When the leaf vertex u_s of a sensor node s has received all the labels of its off-path vertices, it may then verify that no aggregation result-tampering has occurred on the path between u_s and the root of its commitment tree. For each vertex t on the path from u_s to the root of its commitment tree, u_s derives the label of t (via the computations in Definition 3). It is able to do so since the off-path labels provide all the necessary data to perform the label computation. During the computation, u_s inspects the off-path labels: for each node t on the path from u_s to the root, u_s checks that the input values fed into the aggregation operation at t are never negative. Negative values should never occur since the data and complement values are non-negative; hence if a negative input is encountered, the verification fails. Once u_s has derived the label of the root of its commitment tree, it compares the derived label against the label with the same `count` that was disseminated by the querier. If the labels are identical, then u_s proceeds to the next step. Otherwise, the verification fails and u_s may either immediately raise an alarm (for example, using broadcast), or it may simply do nothing and allow the aggregate algorithm to fail due to the absence of its confirmation message in the subsequent steps.

Collection of confirmations. After each sensor node s has successfully performed the verification step for its leaf vertex u_s, it sends an authentication code to the querier. The authentication code for sensor node s is $\text{MAC}_{K_s}(N||OK)$ where OK is a unique message identifier and K_s is the key that s shares with the querier. The collation of the authentication codes proceeds as follows (note that we are referring to the *aggregation* tree at this point, not the commitment tree). Leaf sensor nodes in the aggregation tree first send their authentication codes to their parents in the aggregation tree. Once an internal sensor node has received authentication codes from all its children, it computes the XOR of its own authentication code with all the received codes, and forwards it to its parent. At the end of the process, the querier will receive a single authentication code from the base station that consists of the XOR of all the authentication codes received in the network.

Verification of confirmations. Since the querier knows the key K_s for each sensor node s, it verifies that every sensor node has released its authentication code by computing the XOR of the authentication codes for all the sensor nodes in the network, i.e., $\text{MAC}_{K_1}(N||OK) \oplus \cdots \oplus \text{MAC}_{K_n}(N||OK)$. The querier then compares the computed code with the received code. If the two codes match, then the querier accepts the aggregation result. Otherwise, the querier rejects the result. A rejection may indicate the presence of the adversary in some unknown nodes in the network, or it may be due to natural factors such as node death or message loss. The querier may either retry the query or attempt to determine the cause of the rejection. For example, it could directly request the leaf values of every sensor node: if rejections due to natural causes are sufficiently rare, the high cost of this direct query is incurred infrequently and can be amortised over the other successful queries.

5. ANALYSIS OF SUM

In this section we prove the properties of the SUM algorithm. In Section 5.1 we prove the security properties of the algorithm, and in Section 5.2 we prove bounds on the congestion of the algorithm.

5.1 Security Properties

We assume that the adversary is able to freely choose **any** arbitrary topology and set of labels for the final commitment forest. We then show that any such forest which passes all the verification tests must report an aggregate result that is (optimally) close to the actual result. First, we define the notion of an *inconsistency*, or evidence of tampering, at a given node in the commitment forest.

Definition 6 *Let $t = \langle c_t, v_t, \bar{v}_t, H_t \rangle$ be an internal vertex in a commitment forest. Let its two children be $u_1 = \langle c_1, v_1, \bar{v}_1, H_1 \rangle$ and $u_2 = \langle c_2, v_2, \bar{v}_2, H_2 \rangle$. There is an* ***inconsistency*** *at vertex t in a commitment tree if either (1) $v_t \neq v_1 + v_2$ or $\bar{v}_t \neq \bar{v}_1 + \bar{v}_2$ or (2) any of $\{v_1, v_2, \bar{v}_1, \bar{v}_2\}$ is negative.*

Informally, an inconsistency occurs at t if the sums don't add up at t, or if any of the inputs to t are negative. Intuitively, if there are no inconsistencies on a path from a vertex to the root of the commitment tree, then the aggregate value along the path should be non-decreasing towards the root.

Definition 7 *Call a leaf-vertex u* ***accounted-for*** *if there is no inconsistency at any vertex on the path from the leaf-vertex u to the root of its commitment tree, including at the root vertex.*

Lemma 8 *Suppose there is a set of accounted-for leaf-vertices with distinct labels $u_1, \ldots, u_m$ and committed data values $v_1, \ldots, v_m$ in* *the commitment forest. Then the total of the aggregation values at the roots of the commitment trees in the forest is at least $\sum_{i=1}^{m} v_i$.*

Lemma 8 can be rigorously proven using induction on the height of the subtrees in the forest (see Appendix A). Here we present a more intuitive argument.

PROOF. (Sketch) We show the result for $m = 2$; a similar reasoning applies for arbitrary m. Case 1: Suppose u_1 and u_2 are in different trees. Then, since there is no inconsistency on any vertex on the path from u_1 to the root of its tree, the root of the tree containing u_1 must have an aggregation value of at least v_1. By a similar reasoning, the root of the tree containing u_2 must have an aggregation value of at least v_2. Hence the total aggregation value of the two trees containing u_1 and u_2 is at least $v_1 + v_2$.

Case 2: Now suppose u_1 and u_2 are in the same tree. Since they have distinct labels, they must be distinct vertices, and they must have a lowest common ancestor t in the commitment tree. The vertices between u_1 and t (including u_1) must have aggregation value at least v_1 since there are no inconsistencies on the path from u_1 to t, so the aggregation value could not have decreased. Similarly, the vertices between u_2 and t (including u_2) must have aggregation value at least v_2. Hence, one of the children of t has aggregation value at least v_1 and the other has aggregation value at least v_2. Since there was no inconsistency at t, vertex t must have aggregation value at least $v_1 + v_2$. Since there are no inconsistencies on the path from t to the root of the commitment tree, the root also must have aggregation value at least $v_1 + v_2$.

Negative root aggregate values are detected by the querier at the end of the aggregate-commit phase, so the total sum of the aggregate values of the roots of all the trees is thus at least $v_1 + v_2$. □

The following is a restatement of Lemma 8 for the COMPLEMENTARY SUM aggregate; its proof follows an identical structure and is thus omitted.

Lemma 9 *Suppose there is a set of accounted-for leaf vertices with distinct labels $u_1, \ldots, u_m$ with committed complement values $\overline{v}_1, \ldots, \overline{v}_m$ in the commitment forest. Then the total* COMPLEMENT *aggregation value of the roots of the commitment trees in the forest is at least $\sum_{i=1}^{m} \overline{v}_i$.*

Lemma 10 *A legitimate sensor node will only release its confirmation MAC if it is accounted-for.*

PROOF. By construction, each sensor node s only releases its confirmation MAC if (1) s receives an authenticated message from the querier containing the query nonce N and the root labels of all the trees in the final commitment forest and (2) s receives all labels of its off-path vertices (the sibling vertices to the vertices on the path from the leaf vertex corresponding to s to the root of the commitment tree containing the leaf vertex in the commitment forest), and (3) s is able to recompute the root commitment value that it received from the base station and correctly authenticated, and (4) s verified that all the computations on the path from its leaf vertex u_s to the root of its commitment tree are correct, i.e., there are no inconsistencies on the path from u_s to the root of the commitment tree containing u_s. Since the hash function is collision-resistant, it is computationally infeasible for an adversary to provide s with false labels that also happen to compute to the correct root commitment value. Hence, it must be that s was accounted-for in the commitment forest. □

Lemma 11 *The querier can only receive the correct final XOR check value if all the legitimate sensor nodes replied with their confirmation MACs.*

PROOF. To compute the correct final XOR check value, the adversary needs to know the XOR of all the legitimate sensor nodes that did not release their MAC. Since we assume that each of the distinct MACs are unforgeable (and not correlated with each other), the adversary has no information about this XOR value. Hence, the only way to produce the correct XOR check value is for all the legitimate sensor nodes to have released their relevant MACs. □

Theorem 12 *Let the final* SUM *aggregate received by the querier be S. If the querier accepts S, then $S_L \leq S \leq (S_L + \mu r)$ where S_L is the sum of the data values of all the legitimate nodes, μ is the total number of malicious nodes, and r is the upper bound on the range of allowable values on each node.*

PROOF. Suppose the querier accepts the SUM result S. Let the COMPLEMENT SUM received by the querier be $\overline{S}$. The querier accepts S if and only if it receives the correct final XOR check value in the result-checking phase, and $S + \overline{S} = nr$. Since the querier received the correct XOR check value, we know that each legitimate sensor node must have released its confirmation MAC (Lemma 11), and so the leaf vertices of each legitimate sensor node must be accounted-for (Lemma 10). The set of labels of the leaf vertices of the legitimate nodes is distinct since the labels contain the (unique) node ID of each legitimate node. Since all the leaf vertices of the legitimate sensor nodes are distinct and accounted-for, by Theorem 8, $S \geq S_L$ where S_L is the sum of the data values of all the legitimate nodes. Furthermore, by Theorem 9, $\overline{S} \geq \overline{S_L}$, where $\overline{S_L}$ is the sum of the complements of the data values of all the legitimate nodes. Let L be the set of legitimate sensor nodes, with $|L| = l$. Observe that $\overline{S_L} = \sum_{i \in L} r - a_i = lr - S_L = (n - \mu)r - S_L = nr - (S_L + \mu r)$. We have that $S + \overline{S} = nr$ and $\overline{S} \geq nr - (S_L + \mu r)$. Substituting, $S = nr - \overline{S} \leq S_L + \mu r$. Hence, $S_L \leq S \leq (S_L + \mu r)$. □

Note that nowhere was it assumed that the malicious nodes were constrained to reporting data values between $[0, r]$: in fact it is possible to have malicious nodes with data values above r or below 0 without risking detection if $S_L \leq S \leq (S_L + \mu r)$.

Theorem 13 *The* SUM *algorithm is optimally secure.*

PROOF. Let the sum of the data values of all the legitimate nodes be S_L. Consider an adversary with μ malicious nodes which only performs direct data injection attacks. Recall that in a direct data injection attack, an adversary only causes the nodes under its control to each report a data value within the legal range $[0, r]$. The lowest result the adversary can induce is by setting all its malicious nodes to have data value 0; in this case the computed aggregate is S_L. The highest result the adversary can induce is by setting all μ nodes under its control to yield the highest value r. In this case the computed aggregate is $S_L + \mu r$. Clearly any aggregation value between these two extremes is also achievable by direct data injection. The bound proven in Theorem 12 falls exactly on the range of possible results achievable by direct data injection, hence the algorithm is optimal by Definition 2. □

The optimal security property holds regardless of the number or fraction of malicious nodes; this is significant since the security property holds in general, and not just for a subclass of attacker multiplicities. For example, we do not assume that the attacker is limited to some ρ fraction of the nodes in the network.

5.2 Congestion Complexity

We now consider the congestion induced by the secure SUM algorithm. Recall that node congestion is defined as the communication load on the most heavily loaded sensor node in the network, and edge congestion is the heaviest communication load on a given link in the network. We only need to consider the case where the adversary is not performing an attack. If the adversary attempts to send more messages than the proven congestion bound, legitimate nodes can easily detect this locally and either raise an alarm or refuse to respond with their confirmation values, thus exposing the presence of the adversary. Recall that when we refer to a *vertex* sending and receiving information, we are referring to the commitment tree overlay network that lies over the actual physical aggregation tree.

Theorem 14 *Each vertex u receives the labels of its off-path vertices and no others.*

PROOF. Since, when the vertices are disseminating their labels in the result-checking phase, every vertex always forwards any labels received from its parents to both its children, it is clear that when a label is forwarded to a vertex u', it is eventually forwarded to the entire subtree rooted at u'.

By definition, every off-path vertex u_1 of u has a parent p which is a node on the path between u and the root of its commitment tree. By construction, p sends the label of u_1 to its sibling u_2 which is on the path to u (i.e., either u_2 is an ancestor of u, or $u_2 = u$). Hence, the label u_1 is eventually forwarded to u. Every vertex u'_1 that is not an off-path vertex has a sibling u'_2 which is not on the path between u and the root of its commitment tree. Hence, u is not in the subtree rooted at u'_2. Since the label of u'_1 is only forwarded to the subtree rooted at its sibling and nowhere else, the label of u'_1 never reaches u. □

Theorem 15 *The* SUM *algorithm induces $O(\log^2 n)$ edge congestion (and hence $O(\Delta \log^2 n)$ node congestion) in the aggregation tree.*

PROOF. Every step in the algorithm except the label dissemination step involves either broadcast or convergecast of messages that are at most $O(\log n)$ size. The label-dissemination step is the dominating factor.

Consider an arbitrary edge in the commitment-tree between parent vertex x and child vertex y. In the label dissemination step, messages are only sent from parent to child in the commitment tree. Hence the edge xy carries exactly the labels that y receives. From Theorem 14, y receives $O(\log n)$ labels, hence the total number of labels passing through xy is $O(\log n)$. Hence, the edge congestion in the *commitment* tree is $O(\log n)$. Now consider an arbitrary aggregation tree edge with parent node u and child node v. The child node v presents (i.e., sends) at most $\log n$ commitment-tree vertices to its parent u, and hence the edge uv is responsible for carrying traffic on behalf of at most $\log n$ commitment-tree edges — these are the edges incident on the commitment tree vertices that v presented to u. Note that v may not be responsible for creating all the vertices that it presents to u, but v is nonetheless responsible for forwarding the messages down to the sensor nodes which created those vertices. Since each edge in the commitment tree has $O(\log n)$ congestion, and each edge in the aggregation tree carries traffic for at most $\log n$ commitment-tree edges, the edge congestion in the aggregation tree is $O(\log^2 n)$. The node-congestion bound of $O(\Delta \log^2 n)$ follows from the $O(\log^2 n)$ edge congestion and the definition of Δ as the greatest degree in the aggregation tree. □

6. OTHER AGGREGATION FUNCTIONS

In this section we briefly discuss how to use the SUM algorithm as a primitive for the COUNT, AVERAGE and Φ-QUANTILE aggregates.

The COUNT Aggregate. The query COUNT is generally used to determine the total number of nodes in the network with some property; without loss of generality it can be considered a SUM aggregation where all the nodes have value either 1 (the node has the property) or 0 (otherwise). More formally, each sensor node s has a data value $a_s \in \{0,1\}$, and we wish to compute $f(a_1,\ldots,a_n) = a_1 + a_2 + \cdots + a_n$. Since count is a special case of SUM, we can use the basic algorithm for SUM without modification.

The AVERAGE Aggregate. The AVERAGE aggregate can be computed by first computing the SUM of data values over the nodes of interest, and then the COUNT of the number of nodes of interest, and then dividing the SUM by the COUNT.

The Φ-QUANTILE Aggregate. In the Φ-QUANTILE aggregate, we wish to find the value that is in the Φn-th position in the sorted list of data values. For example, the median is a special case where $\Phi = 0.5$. Without loss of generality we can assume that all the data values are distinct; ties can be broken using unique node IDs.

If we wished to verify the correctness of a proposed Φ-quantile q, we can perform a COUNT computation where each node s presents a value $a'_s = 1$ if its data value $a_s \le q$ and presents $a'_s = 0$ otherwise. If q is the Φ-quantile, then the computed sum should be equal to Φn. Hence, we can use any insecure approximate Φ-quantile aggregation scheme to compute a proposed Φ-quantile, and then securely test to see if the result truly is within the approximation bounds of the Φ-quantile algorithm.

7. CONCLUSION

In-network data aggregation is an important primitive for sensor network operation. The strong standard threat model of multiple Byzantine nodes in sensor networks requires the use of aggregation techniques that are robust against malicious result-tampering by covert adversaries.

We present the first optimally secure aggregation scheme for arbitrary aggregator topologies and multiple malicious nodes. This contribution significantly improves on prior work which requires strict limitations on aggregator topology or malicious node multiplicity, or which only yields a probabilistic security bound. Our algorithm is based on a novel method of distributing the verification of aggregation results onto the sensor nodes, and combining this with a unique technique for balancing commitment trees to achieve sublinear congestion bounds. The algorithm induces $O(\Delta \log^2 n)$ node congestion (where Δ is the maximum degree in the aggregation tree) and provides the strongest security bound that can be proven for any secure aggregation scheme without making assumptions about the distribution of data values.

8. REFERENCES

[1] H. Cam, S. Ozdemir, P. Nair, D. Muthuavinashiappan, and H. O. Sanli. Energy-efficient secure pattern based data aggregation for wireless sensor networks. *Computer Communications*, 29:446–455, 2006.

[2] C. Castelluccia, E. Mykletun, and G. Tsudik. Efficient aggregation of encrypted data in wireless sensor networks. In *Proceedings of The Second Annual International Conference on Mobile and Ubiquitous Systems*, 2005.

[3] J.-Y. Chen, G. Pandurangan, and D. Xu. Robust computation of aggregates in wireless sensor networks: distributed randomized algorithms and analysis. In *Proceedings of the Fourth International Symposium on Information Processing in Sensor Networks*, 2005.

[4] W. Du, J. Deng, Y. Han, and P. K. Varshney. A witness-based approach for data fusion assurance in wireless sensor

networks. In *Proceedings of the IEEE Global Telecommunications Conference*, 2003.

[5] J. Girao, M. Schneider, and D. Westhoff. CDA: Concealed data aggregation in wireless sensor networks. In *Proceedings of the ACM Workshop on Wireless Security*, 2004.

[6] M. B. Greenwald and S. Khanna. Power-conserving computation of order-statistics over sensor networks. In *Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 2004.

[7] I. Gupta, R. van Renesse, and K. P. Birman. Scalable fault-tolerant aggregation in large process groups. In *Proceedings of the International Conference on Dependable Systems and Networks*, 2001.

[8] L. Hu and D. Evans. Secure aggregation for wireless networks. In *Workshop on Security and Assurance in Ad hoc Networks*, 2003.

[9] C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidemann. Impact of network density on data aggregation in wireless sensor networks. In *Proceedings of the 22nd International Conference on Distributed Computing Systems*, 2002.

[10] P. Jadia and A. Mathuria. Efficient secure aggregation in sensor networks. In *Proceedings of the 11th International Conference on High Performance Computing*, 2004.

[11] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: a tiny aggregation service for ad-hoc sensor networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):131–146, 2002.

[12] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. The design of an acquisitional query processor for sensor networks. In *Proceedings of the 2003 ACM International Conference on Management of Data*, 2003.

[13] A. Mahimkar and T. Rappaport. SecureDAV: A secure data aggregation and verification protocol for sensor networks. In *Proceedings of the IEEE Global Telecommunications Conference*, 2004.

[14] A. Manjhi, S. Nath, and P. B. Gibbons. Tributaries and deltas: efficient and robust aggregation in sensor network streams. In *Proceedings of the ACM International Conference on Management of Data*, 2005.

[15] S. Nath, P. B. Gibbons, S. Seshan, and Z. R. Anderson. Synopsis diffusion for robust aggregation in sensor networks. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, 2004.

[16] A. Perrig, R. Szewczyk, J. D. Tygar, V. Wen, and D. E. Culler. SPINS: Security protocols for sensor networks. *Wirel. Netw.*, 8(5):521–534, 2002.

[17] B. Przydatek, D. Song, and A. Perrig. SIA: Secure information aggregation in sensor networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, 2003.

[18] D. Wagner. Resilient aggregation in sensor networks. In *Proceedings of the 2nd ACM Workshop on Security of Ad-hoc and Sensor Networks*, 2004.

[19] Y. Yang, X. Wang, S. Zhu, and G. Cao. SDAP: A secure hop-by-hop data aggregation protocol for sensor networks. In *Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 2006.

[20] Y. Yao and J. Gehrke. The COUGAR approach to in-network query processing in sensor networks. *SIGMOD Rec.*, 31(3):9–18, 2002.

APPENDIX

A. PROOF OF LEMMA 8

We first prove the following:

Lemma 16 *Let F be a collection of commitment trees of height at most h. Suppose there is a set U of accounted-for leaf-vertices with distinct labels $u_1, \ldots, u_m$ and committed values $v_1, \ldots, v_m$ in F. Let the set of trees that contain at least one member of U be T_F. Define $val(X)$ for any forest X to be the total of the aggregation values at the roots of the trees in X. Then $val(T_F) \geq \sum_{i=1}^{m} v_i$.*

PROOF. Proof: By induction on h.

Base case: $h = 0$. Then all the trees are singleton-trees. The total aggregation value of all the singleton-trees that contain at least one member of U is exactly $\sum_{i=1}^{m} v_i$.

Induction step: Assume the theorem holds for h, and consider an arbitrary collection F of commitment trees with at most height $h+1$ where the premise holds. If there are no trees of height $h+1$ then we are done. Otherwise, let the set R be all the root vertices of the trees of height $h+1$. Consider $F' = F \backslash R$, i.e., remove all the vertices in R from F. The result is a collection of trees with height at most h. Let $T_{F'}$ be the set of trees in F' containing at least one member of U. The induction hypothesis holds for F', so $val(T_{F'}) \geq \sum_{i=1}^{m} v_i$. We now show that replacing the vertices from R cannot produce an T_F such that $val(T_F) < val(T_{F'})$. Each vertex r from R is the root of two subtrees of height h in F. We have three cases:

Case 1: Neither subtree contains any members of U. Then the new tree contains no members of U, and so is not a member of T_F.

Case 2: One subtree t_1 contains members of U. Since all the members of U are accounted-for, this implies that there is no inconsistency at r. Hence, the subtree without a member of U must have a non-negative aggregate value. We know that r performs the aggregate sum correctly over its inputs, so it must have aggregate value at least equal to the aggregate value of t_1.

Case 3: Both subtrees contain members of U. Since all the members of U are accounted-for, this implies that there is no inconsistency at r. The aggregate result of r is exactly the sum of the aggregate values of the two subtrees.

In case 2 and 3, the aggregate values of the roots of the trees of height $h+1$ that were in T_F, was no less than the sum of the aggregate values of their constituent subtrees in $T_{F'}$. Hence, $val(T_F) \geq val(T_{F'}) \geq \sum_{i=1}^{m} v_i$. $\square$

Let the commitment forest in Lemma 8 be F. Let the set of trees in F that contain at least one of the accounted-for leaf-vertices be T. By the above lemma, $val(T) \geq \sum_{i=1}^{m} v_i$. We know that there are no root labels with negative aggregation values in the commitment forest, otherwise the querier would have rejected the result. Hence, $val(F) \geq val(T) \geq \sum_{i=1}^{m} v_i$. $\blacksquare$