Foundations and Trends® in
sample
Vol. xx, No xx (xxxx) 1–53

DOI: xxxxxx

now
the essence of knowledge

Secure Distributed Data Aggregation

Haowen Chan[1], Hsu-Chun Hsiao[2], Adrian Perrig[3] and Dawn Song[4]

[1] *Carnegie Mellon University, USA, haowenchan@cmu.edu*
[2] *Carnegie Mellon University, USA, hchsiao@cmu.edu*
[3] *Carnegie Mellon University, USA, perrig@cmu.edu*
[4] *University of California, Berkeley, USA, dawnsong@cs.berkeley.edu*

Abstract

We present a survey of the various families of approaches to secure aggregation in distributed networks such as sensor networks. In our survey, we focus on the important algorithmic features of each approach, and provide an overview of a family of secure aggregation protocols which use resilient distributed estimation to retrieve an approximate query result that is guaranteed to be resistant against malicious tampering; we then cover a second family, the commitment-based techniques, in which the query result is exact but the chances of detecting malicious computation tampering is probabilistic. Finally, we describe a hash-tree based approach that can both give an exact query result and is fully resistant against malicious computation tampering.

Contents

1

Introduction

Recent advances in technology have made the application area of highly distributed data collection networks increasingly important. One example of this is sensor networks [39], which are wireless multihop networks composed of a large number of low cost, resource constrained nodes. Another example occurs in distributed systems such as distributed databases [34], peer-to-peer networks [58] or "grid" computing, where a large number of nodes are distributed over the Internet while engaging in some shared data collection or processing task.

A common task in these systems is the transmission of data towards a designated collection point, or *sink*. In sensor networks, the sink is typically a wireless base station that relays the collected data to an off-site server; in other distributed systems the sink may be a designated coordinating server that is responsible for archiving the data and answering user queries. The most straightforward method for collecting data is for each node in the network to send their raw data directly to the sink, via multi-hop routes in which intermediate nodes act as passive message forwarders and neither inspect nor modify the data. However, this approach is communication inefficient since not all of the collected data may be relevant or necessary for the application.

An alternative method for data collection is to observe that, commonly, only very simple *aggregation functions* are queried on the data. An aggregation function takes as inputs all the data values of the nodes in the network, but outputs only a single scalar. Examples of common aggregation functions include the sum of all data values; the count of the number of nodes fulfilling a given predicate, the minimum or maximum data value over the nodes in the network, and various measures such as mean and median. Since the result of computing an aggregation function is only a single value, this computation can be efficiently distributed in the network by having intermediate nodes compute sub-aggregates, leading to an extremely communication-efficient protocol. This technique is known as *in-network aggregation* [39] and is briefly described in Section 2.4.

The efficiency of in-network aggregation comes at a price to resilience, however, since it relies on the honest behavior of intermediate nodes in terms of computing accurate sub-aggregates. For example, for a sum computation, a malicious intermediate node with two children each reporting a data value of '1', could report an inaccurate sub-aggregate value of '100' instead of the correct value of '2', thus skewing the final result by a large amount. Such attacks are not easily preventable since the efficiency of in-network aggregation relies on intermediate sub-aggregators reporting only concise summaries of their received values; since a large fraction of the input data is hidden by necessity, this exposes the network to greater opportunities for attack.

In this article, we provide an overview of the known approaches towards combating such malicious mis-aggregation attacks. Ideally, a secure aggregation protocol should offer three key features: it should (1) produce accurate answers (typically, an accuracy guarantee bounded by some function of the number of malicious nodes in the network), (2) require only low communication overhead, and (3) be resilient against general node compromise models. We present a brief summary of several approaches drawn from a selection of the current literature as well as a more in-depth tutorial in one of the more important frameworks. In our selection of covered literature, our goal is to provide the reader with a general intuitive understanding of the field, rather than to bring the reader exhaustively up to date with all algorithms for the area.

Towards this end, we have opted towards a more tutorial approach in terms of selecting the publications that most clearly exemplify a certain class of approaches (or which have been most influential historically), rather than focusing on breadth or depth of coverage in terms of the most effective or the most recent algorithms.

The remainder of the article is organized as follows. In Section 2 we define the problem and introduce the notion of in-network aggregation more rigorously. In particular, in this article we focus only on aggregation computations for which the secure aggregation problem is feasible: the family of such functions is examined and defined. In Section 3 we highlight some earlier work which show the basic flavors of integrity verification and result checking for secure aggregation. The existing literature on secure aggregation can be broadly divided into two categories: the first category uses *verifiable sampling* to provide resilient probabilistic estimates of the aggregate result; the second category uses *commitment verification*, which, unlike the first category, can provide highly precise results for which any malicious tampering is immediately evident, but at the cost of availability. We cover these approaches in Sections 4 and 5, respectively.

2

Problem Definition

For concreteness and brevity, in the rest of the article, we discuss secure aggregation using the context and terminology of sensor networks. Much of the published literature in secure aggregation is motivated by the application context of sensor networks; in general the models and analysis are often sufficiently general to apply to various other distributed data collection and computation networks such as peer-to-peer and grid computing networks.

2.1 Network Model

A sensor network is a network that consists of one or more base stations and a large number of sensing devices. A base station is often implemented in the form of a gateway connected to a reliable power supply, which communicates via a high bandwidth connection (e.g., through a wired Internet connection, or a LAN) with a reliable server situated in a secure location. The actual gateway device only forwards messages from the sensor network to the remote secure server, and does not perform any cryptographic operations (i.e., the gateway device does not store any keys or sensitive data, and all authentication and encryption

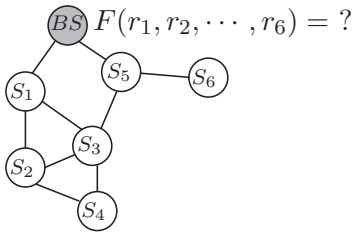


Fig. 2.1 Modeling an aggregation network as a connected graph.

is performed on the secure remote server). Hence, the remote server and the wireless gateway can be conceptually considered a single trustworthy network entity, which we will concisely denote with the term *base station*. For simplicity, we focus on the case where there is only one base station. The network administrator accesses and administers the network through this base station. The remainder of the sensor network consists of a number of *sensor nodes* which communicate to form a wireless mesh network. Sensor nodes typically have relatively lower communication and computational capabilities due to limited power and cost. The main task of the sensor nodes is to collect environmental information and send such information back to the user via the base station. A sensor node typically performs three kinds of tasks: it senses (gathers locally-produced raw data), sends and receives messages from other sensor nodes (often acting as a forwarding router for network), and performs aggregation computations.

Formally, we can model the network as a connected graph with $n+1$ vertices as shown in Figure 2.1. There is a special vertex BS which denotes the base station, and each of the the remaining vertices $s_1, \ldots, s_n$ each represents a sensor node. Every sensor node s_i is associated with a reading r_i. We draw an edge between every pair of vertices that correspond to sensor nodes within mutual transmission/reception range of each other. The distributed aggregation problem is the problem of computing a given function F over all the readings $r_1, \ldots, r_n$, such that the base station BS receives the correct value of $F(r_1, \ldots, r_n)$.

2.2 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 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 [36, 50]. We also assume that an efficient broadcast authentication scheme is available, which allows the base station to send a message M to all nodes in the network such that each node can determine that M was sent from the base station (and not forged by the adversary). A number of highly efficient broadcast authentication schemes have been proposed, including TESLA [47], μTESLA [37], the Guy Fawkes scheme [3], and the hash-tree-based scheme [11]. The specifics of these schemes are out of scope of this article.

2.3 Node Congestion

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 [39]. 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.

2.4 In-Network Aggregation

We first present the notion of *in-network aggregation* as presented in the Tiny Aggregation Protocol (TAG) by Madden et al. [39]. In TAG, a spanning tree is first computed as a subgraph of the communication graph; any well-known method may be used to achieve this. Madden et al. suggest constructing this based on a broadcast tree rooted at the base station BS such that each node is placed on the spanning tree at a

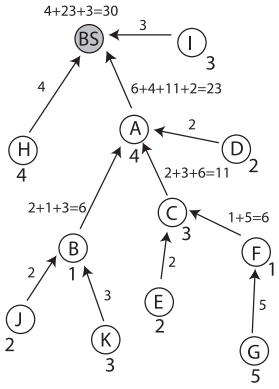


Fig. 2.2 Standard tree-based SUM aggregation as described in TAG. Numbers next to nodes represent data/readings; Numbers and sums on edges represent intermediate sums sent from nodes to parents.

depth equal to its minimum hop distance from BS. All communications in the protocol then take place solely along the edges of this spanning tree.

We describe the aggregation process for real-valued aggregates such as SUM, and later modify this to address more general computations. The aggregation process starts at the leaves. Each node that is a leaf in the tree topology reports its input value (e.g., a sensor reading) to its parent. Once an internal node has received data from each of its children, it evaluates the *intermediate aggregation operator* over this data and its own reading. These internal nodes are called called *aggregators*. In the case of the summation aggregation, the intermediate aggregation operator is addition, i.e., an intermediate sum of the data received from each of the node's children and the node's own reading is performed. The result of the intermediate aggregation operation is reported to the node's parent, which then repeats the process until the final sum is computed at the base station. An example of this process is shown on Figure 2.2.

The above example illustrates that how in-network aggregation can be performed when F is a summation (SUM) function. In addition to SUM, COUNT, AVG, and MAX (MIN) are commonly considered aggregation functions as well. COUNT is a special case of SUM with

each node reporting 1, and AVG can be calculated by SUM/COUNT. In general, the aggregation functions can be classified into two types: duplicate sensitive and duplicate insensitive [46]. Duplicate sensitive functions, such as COUNT and SUM, are functions whose output is altered when incorporating the same input more than once. On the other hand, duplicate insensitive functions, such as MAX and MIN, are functions whose output is invariant to repetitive inputs. For example, let $\{r_1, r_2\} = \{2, 3\}$. $\text{SUM}(r_1, r_2) = 5 \neq \text{SUM}(r_1, r_2, r_2) = 8$; whereas $\text{MAX}(r_1, r_2) = 3 = \text{MAX}(r_1, r_2, r_2)$.

In-network aggregation greatly reduces the communication overhead (and thus reduces the node congestion as well) because the base station only receives a single scalar rather than all data values in the network. However, the efficiency comes at a price of security, as intermediate nodes can report arbitrary values to bias the final aggregate without being detected. Designing secure in-network aggregation schemes is challenging because despite that the conventional *end-to-end authentication* can prevent intermediate nodes from illegal modification (i.e. tampering with the reported values) it also eliminate the possibility of performing legal modification (i.e. applying in-network aggregation functions).

2.5 Threat Model

An attacker could attempt to subvert the distributed computation in a number of ways, namely: outsider attack, eavesdropping, input falsification, and aggregation manipulation. The last three kind of attacks require the attacker to compromise at least one node in the distributed networks. It has been shown that unattended nodes such as sensors are extremely vulnerable to physical compromise key extraction [30].

(1) *Outsider Attack:* Outsiders are attackers who have no control over any devices or cryptographic keys in the network. Hence, to falsify the final results without being detected, an outsider could drop packets selectively or replay others' packets. Such attacks are relatively easy to deal with using cryptographic counters to detect replays and resilient communications (such

as retransmissions or multi-path forwarding) to deal with packet dropping.

(2) *Eavesdropping:* An attacker controlling a malicious insider node may inspect the data of incoming messages and discover information about the sensor readings in the rest of the network. In this article, we will focus on the property of integrity, and not examine methods for providing secrecy from in-network aggregators. Work on providing secrecy for aggregation include the following: Cam et al. [10] proposes privacy-preserving aggregation scheme to hide raw readings by replacing them with predefined pattern codes. Concealed Data Aggregation (CDA) [9, 57] leverages homomorphism [19] to aggregate encrypted data. PDA [31] achieves privacy-preserving aggregation under additive aggregation function using clustering and slicing.

(3) *Input Falsification:* A compromised node may choose to lie about its own data value (and its own value only). Such an attack is usually difficult to defend against within the network since the raw data value of a given node is known only to that node. The detection of such input anomalies is impossible unless certain prior assumptions about data correlation are made. For example, various algorithms for outlier detection [1, 7, 8, 52, 54, 55] can detect abnormal sensor readings based on statistical analysis. However, in order to focus on the general problem of providing computational integrity, most secure aggregation protocols do not consider such anomaly detection mechanisms. In other words, the protocols in this article all assume that the data value reported by a node is by definition the "true" value. Various robust aggregation functions can be selected to ensure that the inaccuracy inherently allowed by this assumption is bounded; these measures are discussed in Section 2.6.

(4) *Aggregation Manipulation:* A more powerful adversary may be able to compromise some aggregators. A malicious aggregator can report arbitrary subcomputation results and

behave in arbitrary ways to attempt to evade detection.

In this article, we will deal primarily with the forth and most powerful kind of adversary, the Aggregation Manipulator. Typically, a secure aggregation protocol should be able to tolerate the colluding behavior of a significant fraction of malicious aggregators, with the accuracy of any accepted result being bounded by some function of the relative fraction of compromised vs legitimate aggregators. Since we are primarily concerned with the accuracy of the final result, we do not consider denial-of-service attacks which primarily serve to ensure that *no* result, false or otherwise, gets accepted at the base station.

2.6 Input-Resilient Functions

As discussed in Section 2, the literature in secure aggregation commonly focuses on general models for which the prior distribution of sensor readings are not known, thus removing the possibility of defending against an input falsification attack. Under this scenario, the best we can hope for is defense against the aggregation manipulation attacks. An important concern is whether it is sufficient to only defend against aggregation manipulation while admitting input falsification. It turns out that certain aggregates are more resilient than others regarding input falsification. Such resilient aggregation functions are called *robust statistics* in the statistics literature [33]. For example, the median is considered a robust measure of central tendency, while the mean is not. Robust statistics can be leveraged in designing secure sensor network aggregation protocols robust against input falsification. In particular, when performing statistical estimation tasks, we should preferably choose aggregation operators that are resilient to input falsification, since nothing much can be done for non-resilient functions.

Wagner proposed a formalized notion of a resilient aggregation function [56]. In this framework, we consider an aggregate function computation as a way to collate a set of signals about a hidden parameter θ. For example, when a sensor network computes the average of a set of temperature sensor readings of the sensors in a room, it is not because the average itself is intrinsically of interest, but because it is correlated

to a parameter θ, which is the "true" ambient temperature of the room. Considered in such a manner, it can be seen that even without the presence of an adversary, naturally occurring error affects the accuracy and resilience of the measure (i.e., the aggregate computation). Such error (with and without the active interference of an adversary) can be exactly quantified. Wagner shows how to precisely evaluate the ability of the adversary to affect the querier's view of θ for various measures.

The immediate conclusion is that certain measures such as MIN, MAX, and unbounded SUM are intrinsically brittle against even small numbers of falsified input values. For example, an adversary can always return the largest allowable value (e.g., "`INT_MAX`") to hide the true maximum reading; similarly for the unbounded SUM operation it can always add an arbitrarily large random value to the computation to completely occlude the true sum.

On the other hand, when the contributions of individual nodes can only have a bounded amount of influence on the measure, then the measure can be resilient, and this resilience is quantifiable. Wagner proposes a metric, α, which indicates the relative inflation factor in a measure's error, under the presence of k adversaries, as compared with a baseline of no adversaries. The larger the α, the more resilient the aggregation function is. It is shown that measures such as truncated average, trimmed average, median and count can be quantifiably resilient against input falsification under certain conditions. These results are shown in Table 2.1. In subsequent discussions, we will assume that the aggregate function being computed by a given protocol satisfies the desired property of resilience against input falsification as quantified here, thus leaving us to focus solely on the mechanisms needed to detect the aggregation manipulation attack.

	Resilience Factor α
$[l, u]$-truncated average	$1 + (u - l)\sigma \cdot k/\sqrt{n}$
5%-trimmed average	$1 + 6.278k/n \quad (\text{for } k < 0.05n)$
median	$\sqrt{1 + 0.101k^2} \quad (\text{for } k < 0.5n)$
count	$1 + O(k/\sqrt{n})$

Table 2.1 Wagner's Resilience Measure for Various Aggreation Functions for k compromised nodes out of n; σ is the assumed s.d. of the sensor reading from the actual value θ

3

Early Work on Secure Aggregation

In this section we review some of the initial work in the area of secure aggregation. Unlike most of the subsequent work in the literature, these protocols provide only limited security properties against limited adversary models. Nonetheless, they are important because they are the first schemes to demonstrate several common design philosophies such as redundancy and verification.

3.1 One-hop Verification

Suppose the attacker assumption is constrained to a single-node attacker (but the actual identity of the compromised node is unknown). To prevent this single malicious node from presenting falsified aggregation results, we can simply try to ensure that the aggregation computation of each node is checked by some other node in the network. A natural candidate for the verifier in a tree topology is the parent of the aggregator. This approach is called *one-hop verification* [32]. The high level algorithmic strategy is for each aggregator node to forward its entire set of inputs to its parent, so that the parent can verify its child's aggregation operation. This ensures that any single node can-

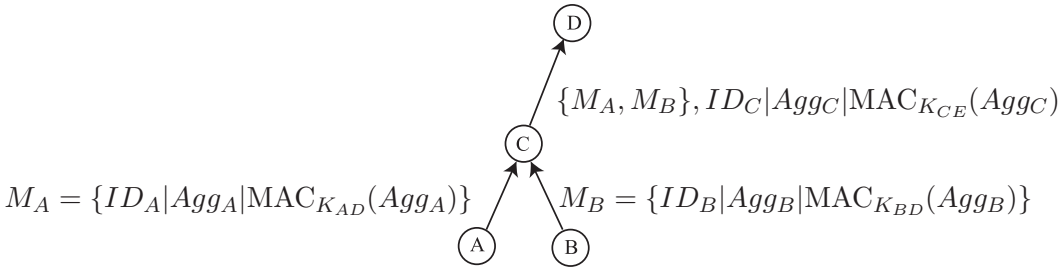


Fig. 3.1 One-hop Verification in the Hu-Evans model

not perform an aggregation manipulation attack since everything that it computes is fully audited by its immediate parent.

The exact protocol is as follows; an example of the general structure of messages is shown in Figure 3.1. Each leaf node s_i in the tree reports its reading r_i to its parent in the form:

$$M_i = \{ID_i \| N \| r_i \| \text{MAC}_{K_i}(r_i)\}$$

In this message, ID_i is the identifier of s_i, r_i is the reading of node s_i, N is a nonce associated with the current query (to prevent replay of messages from previous queries), and K_i is a key associated with node s_i that is shared with the grandparent of s_i. Similarly, each aggregator node s_i that receives incoming messages $M_1, \ldots, M_k$ will construct the following message to its parent:

$$\{M_1, \ldots, M_k\}, ID_i \| N \| Agg_i \| \text{MAC}_{K_i}(Agg_i)$$

In this message, Agg_i is the result of the sub-aggregation computation over the input values of the messages of the children $(M_i, \ldots, M_k)$. For example, if we are evaluating the SUM operation, and the children report values $\{1, 2, 3\}$ in M_1, M_2 and M_3, then $Agg_i = 1 + 2 + 3 = 6$.

The verification process is as follows: since each grandparent node (e.g., node D in the example in Figure 3.1) knows the MAC keys used by all of its grandchildren, it can re-check all the subcomputations of its immediate children (in the example, node D would be able to exactly check the computations performed by C). In this manner, no single adversary node can maliciously cheat in its subcomputation without

being detected by its (legitimate) parent. On the other hand, an adversary controlling two nodes sharing a parent-child relationship can freely forge the aggregation results of the malicious child node without detection because the audit process only checks for the accuracy of computations one-hop down from each legitimate node.

3.2 Witness-based Verification

We now consider the case where only a single aggregator is present, and the purpose of aggregation is to reduce the communication load over the (potentially expensive) link from the single aggregator to the distant querier. In other words, every reading is forwarded to the aggregator without in-network aggregation. Clearly, in this case, we cannot say that the querier should also function as the auditor since this would involve having the aggregator supply all the sensor node readings over the expensive long distance link. A possible approach is then to build in redundancy into the system, by having multiple *redundant aggregators*, or *witnesses*. However, instead of providing multiple aggregate reports to the distant querier, these redundant aggregators can simply attest to the fact that they agree with the aggregate reported by the primary aggregator. In this sense, they are performing a pure validation function, and are hence called witnesses to the authenticity of the aggregation report [20].

Specifically, in this scheme, a set $\{r_1, \ldots, r_n\}$ of readings is forwarded to a single aggregator, which then computes the aggregate function $v = F(r_1, \ldots, r_n)$. For resilience, the proposed scheme replicates the aggregation functionality identically over m redundant aggregators or *witnesses*. The witnesses perform identical functions as the aggregator: Each witness w_i also collects the full set of readings $\{r_1, \ldots, r_n\}$ and computes an aggregation result v_i; however it does not forward the full result directly to the base station but instead computes a short digest of the result: $\mathrm{MAC}_{K_i}(w_i \| v_i)$ using the key K_i shared between w_i and the base station. These digests are collected by the primary aggregator and forwarded to the base station. The base station then inspects the reported result, and recomputes the respective MACs from each of the witnesses: if at least a majority of the witnesses agree that the result is

correct, then it is accepted, otherwise, it is rejected. With sufficiently long cryptographic MACs, the adversary cannot forge the response of a witness unless it directly compromises the witness; hence the attacker needs to compromise a majority of the witness nodes to force acceptance of an incorrect result. Du et al. [20] discuss a variant where the MAC length is shortened to k bits to reduce communication overhead: if the desired upper bound on the chance of an incorrect result being accepted is no more than $2^{-\delta}$ then k must be at least $2(\delta/m + 1)$ bits, thus causing a total additional overhead of $2(\delta + m)$ bits of communication. The coefficient of scaling with increasing m (2 bits per additional witness) is surprisingly small indicating that the witness set should be made as large as possible: in most networks (which would need to use peer sensor nodes as witnesses) the main constraint on the witness set size would be the limited number of nodes which are able to efficiently receive the exhaustive set of sensor readings from the entire network.

3.3 Summary and Discussion

The algorithms in this section demonstrate two important design elements which have been extended and developed by subsequent approaches:

(1) **Data Authentication.** One-hop verification and witness-based verification provide good examples of the necessity of using authentication primitives such as MACs to authenticate the origin of each input and sub-result. Without such authentication, for example, an adversary could insert many false input elements and untraceably skew the aggregation result.
(2) **Commitment and Verification.** Each aggregator node commits to a set of inputs and a result of the computation over these inputs, which is then verified through a process of having redundant elements repeating the computation. For the one-hop verification technique, the redundant verifier is the parent of each aggregator node; for the witness-based verification scheme, the aggregation operation is replicated over

several redundant aggregator nodes. This pattern of commitment and subsequent verification is the basis of more sophisticated schemes in Section 5.

The basic approaches of one-hop verification and witness-based verification have a clear drawback: only a small, fixed number of verifiers need to be compromised in order for the adversary to completely break the security of the system. For the case of one-hop verification, any compromised child-parent pair can successfully avoid audit; in the witness-based scenario, if a majority of the witnesses are compromised, then the adversary can cause any aggregate result to be accepted. Despite these disadvantages, these basic schemes nonetheless have the advantages of being relatively simple to implement, and are relatively efficient for low-threat applications. The additional communication overhead of these schemes is exactly proportional to the amount of redundancy that is built into the system, and if this redundancy factor is considered a constant then the total communication overhead is linear in the size of the network, i.e., the additional per-node communication overhead is constant. All of the more secure algorithms in this article involve asymptotically more overhead than this. In addition, building in redundant structures enables the system to not just detect incorrect results, but also actively *recover* from certain failures (especially, sporadic random errors and failures, which are likely in practice to be more common than malicious attacks). For example, in the witness-based scheme, if the designated aggregator breaks down or returns incorrect results, any of the witnesses are able to take over the role of aggregator, thus ensuring that service is not disrupted.

3.4 References and Further Reading

One-hop verification is first proposed by Hu and Evans [32]. They describe a number of specific optimizations for the basic one-hop verification scheme, including how to efficiently update and use these keys that need to be shared between non-adjacent nodes in the network (the keys are intended to be used only once per aggregation period, hence the lack of nonces in the protocol); for example, through the use of a

one-way hash chain [32]. A number of additional extensions to provide secrecy of the data from a single in-network attacker are also proposed by Jadia and Mathuria [35].

The basic witness-based scheme is described by Du et al. [20]. Cristofaro et al. [18] generalize the witness based approach to hierarchical aggregation, using multiple redundant aggregator nodes per level. Whereas the Du et al. scheme used a strict threshold scheme for determining whether to accept a given aggregation result (accept only if t or more witnesses agree exactly on the result, reject otherwise), Cristofaro et al. use a heuristic metric called "quality of information" (QoI) to measure the level of confidence on a given sub-aggregation result. A result that has a strong level of agreement between witnesses will contribute a high QoI, whereas a large amount of inconsistency would result in a low QoI. This QoI metric is then propagated in the hierarchical aggregation computation process (low QoI sub-results will contribute to lower QoIs on computations dependent on these results). In the end, the base station is presented with both an aggregation result and a measure of confidence on the result, which the base station can use as evidence to accept or reject the final aggregation result. A related approach similar to this QoI metric-based scheme is reputation-based integrity checking [24], where nodes evaluate trustworthiness of other nodes based on their past behavior, and determine whether to accept a received aggregate based on the sender's trustworthiness level.

4

Resilient Estimation

In this section, we discuss a general class of approaches in which accuracy of the aggregation computation is sacrificed in return for making the protocol resilient to malicious manipulation. Specifically, these approaches typically compute a statistical estimator for the aggregate value; while using the estimator is less accurate that computing the precise aggregate, it is easier to secure, because there is less reliance on the honest behavior of intermediate aggregators.

4.1 Authenticated Single Data Values

Suppose communication overhead is not a concern. Then a simple way to ensure the full security of any aggregation computation would be for the base station to request the readings of every sensor node in the network. Each node could return its respective reading, authenticated using the key that it shares with the base station. For example, the message from node s_i could be formatted as follows:

$$ID_i \| N \| r_i \| \mathrm{MAC}_{K_i}(ID_i \| N \| r_i)$$

In this message, the node s_i with identifier ID_i is reporting a read-

ing r_i for the query associated with the nonce N (e.g., N could be a random value selected by the querier to identify the query, or a strictly increasing sequence number). This message is authenticated using the key K_i which is shared only between the node s_i and the base station. Note that this is different from the Hu and Evans method [32] described in Section 3 in which K_i was shared between the node and its grandparent.

The use of the message authentication code ensures that the message cannot be tampered with en-route between any legitimate node and the base station. In other words, this scheme is fully secure against manipulation attacks (malicious nodes may still falsify their own readings, but as explained in Section 2, such attacks are out of the scope of the schemes discussed in this article).

However, exhaustively forwarding the exact data value of every node is extremely communication-intensive. A straightforward method of reducing this communication overhead is to *sample* only a small set of data values in the hope that this contains enough information to *estimate* the actual aggregate. For example, if the base station chooses 10 nodes uniformly at random, and samples each of their readings, it can use the average of the readings reported by the small sample as an unbiased estimator of the average of the readings over the entire sensor node population. Because all the readings from the sample are end-to-end authenticated from the sampled node to the base station, data manipulation attacks (where the data is altered en-route from the node to the base station) remain infeasible. However, it is clear that, the smaller the sample, the greater the potential error of the estimator.

Using a sample set is an example of choosing a set of *authenticated single data values*, which are authenticated from their respective sources, in order to estimate the target aggregate. Typically, aggregate estimators use more sophisticated sampling methods than uniform random sampling. One concept that is often used is the notion of a *predicate test* [60], which is essentially a representative proof of set nonemptiness. Consider the following query: the base station queries the network asking if there are any nodes that satisfy a certain query predicate (e.g., a fire alarm system could query, “are there any sensors

sensing a temperature above 100°C"). The network returns either a negative answer (i.e., "no sensors satisfy the predicate") or the reading of a single sensor that satisfies the predicate, authenticated using a MAC computed with the sensor-base station secret key. In the case of the positive answer, we have the property that the single representative sensor's message is equivalent to a proof that the predicate set is indeed nonempty: specifically, the adversary could not have forged this message if the representative node was not compromised (if the representative node was compromised, then it is equivalent to a data falsification attack, not an aggregation computation manipulation). Hence, if the adversary can be prevented from data falsification (e.g., if the query predicate is structured around an immutable property of the node, such as "are there any sensors with a ID that is above 1000") then this primitive is a powerful tool for resiliently testing a population. Note that if resilient routing is assumed, the querier can obtain an implicit proof of set emptiness in the case of a negative answer (none of the nodes returns a satisfying reading). Moreover, the querier can further obtain an explicit proof of set emptiness by querying the negation of the original predicate. We detail these techniques in the remainder of this section.

4.2 Verifiable Sketches

Further extending the notion of sampling using authenticated single values, a "sketch" is a concise statistical data structure that captures information about aggregate values using just a small set of representatives. An exhaustive coverage of statistical sketches is beyond the scope of this article: we briefly highlight one of the more commonly-used primitives, the "FM-Sketch" [22]. An FM-Sketch is a bitmap of size k that is used to concisely estimate the number of distinct elements in a set. At the start of the algorithm, the entire bitmap is initialized to 0. To insert an element r_i into the sketch, its k-bit hash $h(r_i)$ is computed, and the bit at position j of the sketch is set to be 1, where j is the position of the least significant 1 bit in this hash; this means that all the bits after position j are 0s. This process is shown in Figure 4.1.

The (0-indexed) position of the leftmost 0 bit is an estimator of the total number of distinct elements inserted into sketch. Specifically, let

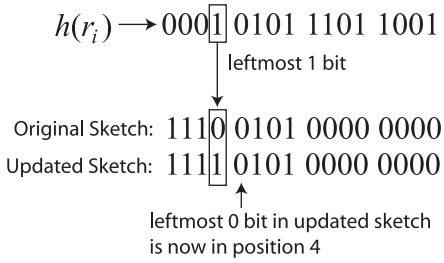


Fig. 4.1 FM-Sketch update process.

R be the random variable denoting the position of the leftmost 0 in the sketch after the insertion of n elements. Then $E(R) = \log_2(\phi n)$ where $\phi \approx 0.77351$. Hence, R can be used to estimate n. The main drawback with the FM-sketch is that the variance of the estimate can be quite large; this error is reduced by using m distinct sketches (each associated with a different hash function) and taking the mean of all the m estimates.

The FM-sketch is concise ($O(\log n)$ bits for a network of size n), updates quickly and efficiently, and most importantly it is *decomposable* (i.e., the sketch of $A \cup B$ is the bitwise OR of the sketch of A and the sketch of B). Decomposability implies that FM-Sketch can be efficiently implemented in a distributed aggregation scenario: each internal aggregator node receives sketches from each of its children, computes the bitwise OR of all the sketches, and forwards the result to the aggregator's parent. Moreover, the FM-sketch is easy to make resilient, because each 1-bit in the sketch is a representative value similar to the one in a set nonemptiness query. In other words, to maintain a *verifiable* FM sketch of size k, we need only ensure that each 1-bit that was set due to the inclusion of some node s_i be associated with a MAC using the key K_i shared only between s_i and the base station. When the base station receives the final sketch, it then checks the MACs associated with every 1 bit in the sketch. Suppose the adversary is unable to freely choose the elements to insert into the sketch (e.g., if we are performing a COUNT operation, each node could be constrained to only report a specific canonical ID as an input into the sketch - since ID_i must be authenticated by the specific key K_i, the adversary cannot make up

arbitrary IDs). Then, the adversary finds it infeasible to force any bit in the sketch to 1 if it would have been set to 0 in a legitimate computation of the sketch, because this would necessarily involve falsifying a MAC with a key that it does not possess.

On the other hand, the adversary could force 1 bits in the sketch to 0 by dropping messages. The problem of avoiding dropped messages is a generic problem of resilient routing. A comprehensive discussion of resilient routing is out of the scope of this article; we discuss the Synopsis Diffusion scheme [46] as an example. The synopsis diffusion framework is motivated by the observation that FM-sketches are *order and duplicate insensitive.* This means that sub-sketches can be arbitrarily replicated and re-routed in a redundant fashion without altering the final result. In particular, for a single base-station network we can use a number of similar-length routes between a node and the base station to forward the message to the base station without an significant increase in communication overhead. To achieve redundancy, Synopsis Diffusion essentially makes use of *all* possible minimum-hop-length paths between a node and the base station to forward data. Essentially, each node is labelled with its hop distance from the base station; aggregation proceeds as normal (with the farthest nodes initiating messages) but instead of using a spanning tree topology where each node forwards just one message to a single parent, in Synopsis Diffusion, each node forwards its computed aggregate sketch to *every* node within transmission range that is one hop closer than itself to the base station. This process is shown in Figure 4.2. The end result is that each data value gets forwarded using a multitude of highly redundant paths; an attacker which does not control a large number of nodes will find it difficult to block a significant fraction of the messages from reaching the base station.

If we assume that the total number of nodes in the network, n, is known, then we can use this knowledge to detect message dropping. For example, in a count c of the number of nodes satisfying a certain predicate, we should be able to expect that the count of the number of nodes *not* satisfying the predicate should be approximately $n - c$. Hence, if we also compute the *complement* of any count, and check

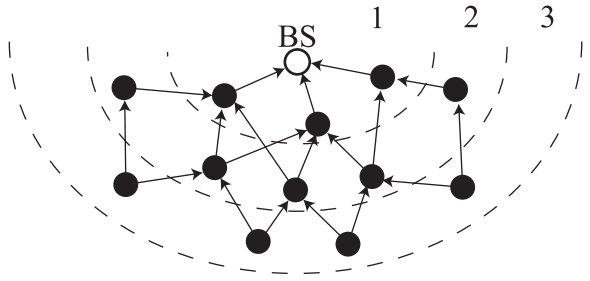


Fig. 4.2 Synopsis Diffusion in the Rings Topology by Nath et al. [46]

that they sum to a value close to n at the end of the computation, this suffices as a probabilistic check that values are not being dropped [25].

If the expected result of the Count query can be a-priori upper and lower bounded by l_u and l_c respectively, then the bits that are below position $a = \lceil \log_2(\phi l_c) \rceil$ are highly likely to be all 1s and the bits that are above position $b = \lfloor \log_2(\phi l_u) \rfloor$ are highly likely to be all 0s. By not explicitly keeping track of these bits, the sketch can be made more concise [51]. When the lower bound l_u is not known, we can instead iteratively compute only the high order (rightmost) w bits of the sketch, until it is determined that with high probability all the bits to the left of the current window are 1 bits. Essentially, the algorithm first computes the rightmost w bits of the sketch, and inspects the result to see if the bit pattern 11 occurs anywhere in the current window. If so, then it is determined that with high probability all bits in the sketch to the left of the current window are also 1-bits. If not, then the window is moved w bits to the left, and the process is repeated. The advantage of starting the window on the right is that the occupancy of the rightmost bits are often extremely sparse and thus very few representative MACs need to be forwarded; as soon as occupancy increases, the incidence of finding a 11 prefix increases very quickly as well and termination quickly ensues.

4.3 Set Sampling

In verifiable sketches, every node is sampled by the query in a single pass, with their responses grouped into buckets of geometrically in-

creasing size (each subset being double the size of the next smallest subset). However, each node is sampled exactly once: it belongs to one and exactly one of these subsets, and many of these subsets are uninformative (for example, the largest subset containing half of the nodes will contain a 1 bit with very high probability). If, instead of doing a single sampling pass, we allow a multiple-pass algorithm, we can instead attempt to quickly zoom into the specific subsets of interest. For example, if we split the set of nodes into two equal halves and determine that the left half has no nodes that satisfy the predicate, then we know this with certainty: we no longer have to issue any further queries into this subset. On the other hand, if the right subset has at least one node that satisfies the predicate, we can imagine performing further probes into that subset to narrow down the range of possibilities for the count of the number of nodes satisfying the predicate. This approach is known as "set sampling" [60].

Set sampling proceeds as follows. Recall that a "predicate test" involves testing a given set for *nonemptiness* with respect to the given predicate; if no node in the set satisfies the predicate, then a negative answer (or no answer at all) is returned; if any node in the set satisfies the predicate, then the specific node ID and reading that satisfies the predicate is returned, authenticated to the base station by a MAC using the node's key. Given sufficiently redundant and reliable routing (e.g., flooding), it can be assumed that the attacker is unable to block messages from being forwarded to the base station, thus a positive answer to the set query cannot be blocked for an extended amount of time (this also implies that a lack of a response after a certain amount of time is equivalent to a negative answer). Despite this useful resilience property, there remains the problem that a malicious node that is not in the sampled set may attempt to flood the network with spurious replies which are eventually discarded at the base station (due to the lack of a correct MAC), but which take up valuable bandwidth and potentially prevent legitimate replies from getting through. To deal with this, we can use a specific form of the predicate test called a "keyed predicate test", which uses a shared group key K to securely specify the subset being tested. Specifically, the base station issues the following query

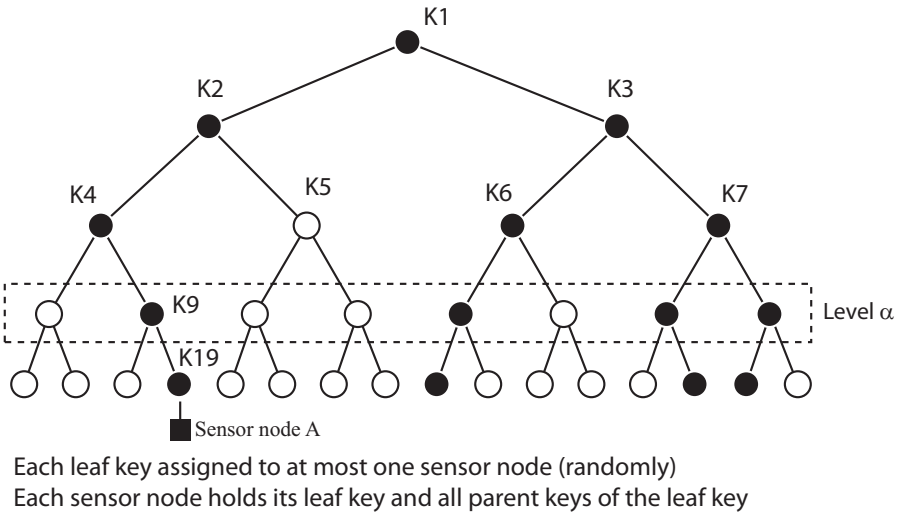


Fig. 4.3 Set Sampling Scheme [60].

using authenticated broadcast (i.e., every receiver of the query can verify that the query is from the base station):

$$\langle \text{name of } K, \text{predicate}, N, H(\text{MAC}_K(N)) \rangle$$

In this query message, N is a nonce associated with the query, and K is a key shared by all nodes in the sampled set. A positive response to the set query is the value $\text{MAC}_K(N)$ (computable by all nodes which know K); a negative response is no response at all. Forwarding nodes, which may not know K, can inspect the value by computing the hash $H(\text{MAC}_K(N))$ and comparing it to the value in the query broadcast; thus if the attacker controls no nodes in the sampled set it cannot generate any value that a legitimate node would forward. This prevents an out-of-set attacker from injecting spurious messages into the set query to DoS the medium.

The keyed predicate test primitive is then used as follows. A binary "sampling tree" is constructed (shown in Figure 4.3), where each tree vertex represents a distinct key. Each sensor node in the network is then assigned to a distinct leaf in the tree, and is pre-loaded with all the keys represented by the path from its leaf vertex to the root. For example, in

Figure 4.3, the sensor node A is associated with the 4th leaf in the tree, and is thus preloaded with the keys $\{K19, K9, K4, K2, K1\}$. It is clear that performing a keyed predicate test on a given key K_i effectively samples for the presence of any nodes that satisfy the predicate at the leaves of the subtree rooted at K_i. For example, performing a keyed predicate test on $K4$ in Figure 4.3 will return TRUE if any of the first 4 leaf vertices are assigned to any sensor nodes that satisfy the predicate. Say a leaf vertex is "black" if its keyed predicate test returns TRUE, "white" otherwise. Then any black leaf vertex has only black parents. An example of this is shown in Figure 4.3. Initially, the colors of the vertices are unknown; the color of each vertex can be determined with a relevant keyed predicate test. The task of the algorithm is thus to estimate the total number of black leaves by probing a minimal number of vertices in the tree.

The Set Sampling algorithm is thus as follows. Let the *relative occupancy* r_i of a level i be the ratio of the number of black vertices at that level to the total number of nodes at that level, n_i. Let the *absolute occupancy* $r_i \cdot n_i$ be the actual number of black nodes at that level. First, determine a level α which has a "moderately low" relative occupancy of black nodes (relative occupancy $3/16 < r_\alpha < 5/8$, or approximately close to between $1/4$ and $1/2$ relative occupancy). Once α has been found, there are two cases: (1) If the *absolute occupancy* $r_\alpha \cdot n_\alpha$ is greater than some constant threshold c_3 (which depends on the tolerable error parameters) then with high confidence, the number of black leaf vertices can be accurately estimated. The intuition is that a high absolute occupancy coupled with a low relative occupancy implies a sufficiently fine granularity of detail into how the black vertices are distributed; for example, it is relatively unlikely that a large fraction of these black vertices have more than one black leaf vertex as a descendant. For the second case (2), the absolute occupancy is too low to immediately estimate the number of black leaf vertices. In this case, the occupancy of the next layer $\alpha + 1$ is retrieved by probing the children of every black vertex at level α (we only need to probe the children of the black vertices because all the children of the white vertices at level α are white). Since the relative occupancy can only decrease as proceeding down the tree, if the absolute occupancy of level $\alpha+1$ is now

above the threshold c_3, it reachs the condition to apply the estimation rule of case (1); if not, the process repeats for level $\alpha + 2$, and so on, with a guaranteed termination upon hitting the leaves (in which case we get an exact answer, not an approximation). This process takes no more than $O(\frac{1}{\epsilon^2} \log \frac{1}{\delta} \log n)$ probes for a (ϵ, δ) approximation (a result that is within $1 \pm \epsilon$ of the true value with probability at least $1 - \delta$).

4.4 Summary and Discussion

In this subsection we describe several variants of the basic notion of secure sampling. We focus on the problem of computing a "predicate count", or a count over the entire network of the number of nodes that satisfy a certain predicate. Each of the algorithms we have examined essentially selects a subset of the nodes and asks for a "representative member", node that signifies that the set has at least one node satisfying the predicate. The representative member then returns an authenticated report attesting to this fact. An estimate of the count of nodes satisfying the predicate can then be calculated by appropriate choice of the sampled subsets.

The algorithms presented in this section have the advantage of being fully resilient: the only way to generate a forged report is by input value falsification. Furthermore, these algorithms can be resilient against denial-of-service attacks against the verification mechanism itself: for example, in many other schemes a malicious node could repeatedly cause a verification process to fail and thus block any aggregation value from being computed. In resilient estimation schemes, since each data point is a report from a specific node identity, various countermeasures can be taken against such attacks: for example in the Set Sampling scheme, a falsified MAC can be traced back to the node that injected it and that node can then be revoked from the network.

The main drawbacks to the algorithms of this section are their lack of precision. Because the measures computed in all of these schemes are probabilistic, the querier only obtains an estimate of the true aggregate, which contains estimation error. An implementor may be tempted to use these estimates as a way to verify a precise aggregation result (e.g., by discarding any results that disagree with the resilient estimator),

but this approach is fraught with pitfalls: besides the problem of false positives from normal sampling error, the adversary may also skew the "precise" result within the acceptable error bound causing systematic overestimates or underestimates; the adversary can opportunistically inject false values for the precise results as the estimates fluctuate; and the adversary can keep reporting illegal values in the precise results, leading to persistent rejection, thus losing the property of resilience against DoS. Due to these problems it is not considered advantageous to use resilient estimation as a form of verification, and research efforts have focused on increasing the precision and efficiency of the estimator directly.

4.5 References and Further Reading

There has been significant work in the development of techniques and algorithms for efficient estimation in the no-adversary scenario, where active maliciousness is not a concern. Various schemes have been developed for approximating aggregation results while incurring minimal communication and storage overhead. In such schemes, nodes collaboratively generate a concise digest from raw readings, and by evaluating the digest, the querier obtains an approximation of the actual aggregate. Besides the FM-sketch highlighted in this section (by Flajolet and Martin [22]), researchers have developed sampling-based approximate algorithms to estimate COUNT/SUM aggregates [5, 14], to count the number of distinct items [2, 6, 26], and to find frequent items with probabilistic guarantee [16, 42]. The applications of such approximate aggregates include identifying large Internet flows for Internet traffic measurement [21], general query processing in peer-to-peer systems [4], and counting safety messages in vehicular networks [38, 48]. Other aggregation algorithms include methods for summarizing approximate order statistic such as medians and quantiles over data streams [17, 27, 29, 43, 61] and distributed data sets [15, 28, 53]. None of these algorithms are intrinsically resilient against a malicious insider, however, many of their design patterns for reducing communication and storage overhead can be leveraged in the development of secure aggregation algorithms.

The notion of "set sampling", which leverages "predicate tests" as a secure form of interactive sampling is due to Yu [60]. The paper contains a more in-depth analysis of the relative costs and benefits and possible performance bounds between the static sampling of sketches and their proposed method of set sampling.

Resilient routing is addressed in various work such as Manjhi et al. [41], and Nath et al. [46]; since this is a major research topic in network communications, a comprehensive listing of related work is beyond the scope of this article.

Computing the *complement* of any count to ensure that values are not being dropped was proposed by Garofalakis et al. [25], who also suggest modifying the FM-sketch primitive to sample approximately between k and $2k$ elements out of the total n nodes without prior knowledge of n; this is achieved by keeping authenticated MACs for *every* element inserted into the sketch (instead of just one per bit); however the elements in the more-populated (low order) buckets are progressively discarded as the higher order buckets gradually fill up.

The method of narrowing the focus of an FM-sketch to regions where the most information is being conveyed is due to Roy et al. [51], who propose this technique as a method of making FM-sketches more efficient in overhead.

5

Commitment-based Techniques

In the resilient estimation techniques of Section 4, the algorithm presents an approximation of the true value to the querier; while there may be random error due to the approximation, the approximation computation itself is usually completely resilient against data manipulation attacks. In contrast, for commitment-based techniques, the reported aggregation result is usually precise (i.e., unlike resilient estimation, the reported result is always 100% accurate in the absence of an adversary); however, to attain this precision, these techniques usually admit a small chance of not being able to detect an active data manipulation attack in any given query. In Section 5.6 we will examine a framework in which not only the result is accurate but also data manipulation is detected with certainty.

5.1 The Aggregate-Commit-Prove Technique

The general flavor for commitment-based secure aggregation was first proposed in the SIA framework by Przydatek et al. [49] and proceeds in two phases: (1) "Aggregate-Commit" and (2) "Prove". In the "aggregate-and-commit" phase, the aggregation proceeds similarly to

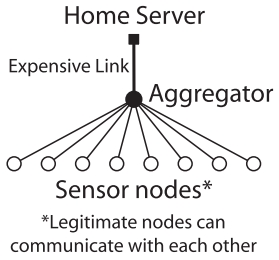


Fig. 5.1 Single Aggregator Topology in SIA.

unsecured aggregation (e.g., as in TAG [39]); however, in the process of performing the aggregation computations, aggregators *commit* to the inputs to their aggregation process by generating cryptographic hash values. The commitment data structure is forwarded to the base station along with the aggregation result. In the "proving" phase, the base station proceeds to *probe* the aggregators in a series of challenges to the aggregators to show that their aggregation operation is indeed conforming to the protocol. Since the commitment structure forces the aggregators to commit to their respective operations in the aggregation process, they cannot evade the probe depending on their a-posteriori knowledge of what is being probed, and any tampering with the aggregation process is revealed by the probe with some probability.

5.2 SIA: Secure Information Aggregation

We now examine the Secure Information Aggregation (SIA) framework in more detail. We assume a *single aggregator* topology, with all sensor nodes reporting directly to the aggregator. The aggregator node computes the aggregate and forwards it to the Querier (or Base Station). The communication metric is measured in terms of the number of bits transferred between the querier and the single aggregator node. This topology is shown in Figure 5.1.

Given access to all the raw sensor readings, the aggregator can compute the sensed aggregate result, and forward it to the base station. The aggregator then constructs a *Merkle Hash Tree* [45] (or a *Hash Tree*) over the sensor readings (see Figure 5.2). The commitment hash tree is

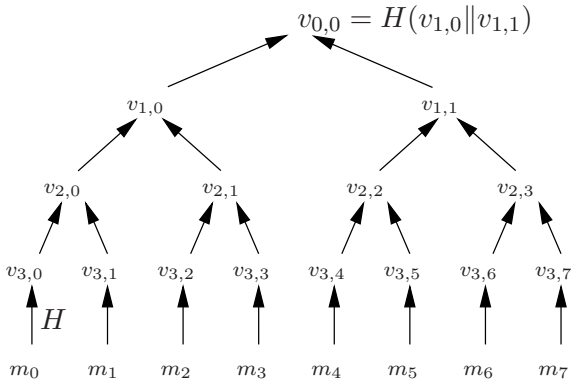


Fig. 5.2 Merkle Hash Tree [45], proposed by Merkle in 1979; arrows indicate inputs to the hash function H.

constructed in the following way: first, place the hash of each authenticated sensor reading at the leaves (this corresponds to $v_{3,i} = H(m_i)$ in Figure 5.2, where $m_i = \langle ID_i \| r_i \| \text{MAC}_{K_i}(ID_i \| N \| r_i) \rangle$; N is the per-query nonce). Then, repeatedly construct parent vertices by computing the hash function H over hash tree vertices at the previous level; e.g., in Figure 5.2, $v_{2,0} = H(v_{3,0} \| v_{3,1})$, $v_{1,0} = H(v_{2,0} \| v_{2,1})$ and so on.

A hash tree is a data structure supporting *selectively verifiable commitment*. Specifically, the hash tree root ($v_{0,0}$ in Figure 5.2) acts as a concise commitment to the *entire* tree, including all the values at the leaves. For a given value of the hash tree root vertex, it is computationally infeasible to find another tree that has the same hash tree root. This means that, once the aggregator has reported a value for the hash tree root, it is now "locked" into a specific, fixed tree topology and ordering of leaf values. Not only is this commitment non-malleable, but it is also selectively verifiable. To verify a given leaf value, instead of needing the entire tree, the querier only needs all *off-path vertices* from the verified value to the root: these are all nodes that are siblings to the vertices from the verified vertex to the root vertex. For example, to verify just the *first* value r_0 in the sequence of sensor readings in the tree of Figure 5.2, the querier requests: $r_0, v_{3,1}, v_{2,1}, v_{1,1}$. This allows the querier to compute the sequence of vertices up the path to the root, finally verifying that the recomputed root is the same as the

value reported by the aggregator. A successful verification means that the authenticated reading was included in the specific position in the tree.

There are a number of ways to use this method of selectively probing the hash tree to ensure that the reported result is correct for various aggregation functions [49]. For brevity, we describe the process of checking a result for the median aggregate, which covers the use of several of the more general property tests.

Detecting Fake Leaf Vertices. Consider the problem of computing the median of all sensor readings. To prevent cheating from the aggregator, we can require that the hash tree be sorted by sensor reading value, using node IDs to break ties such that each value is essentially distinct. In such a case, the median should be the central value in the hash tree. To "push" the median value to the right or the left (thus potentially increasing or decreasing the reported median), the adversary could attempt to insert spurious leaf vertices (which do not have the correct MAC values) in various locations. Since these leaf values do not carry correct MACs, the malicious tampering is revealed as soon as they are probed. It can be shown that using $\frac{1}{\varepsilon} \ln \frac{1}{\delta}$ probes suffices to detect if the adversary has generated ε or more fraction fake leaves, with detection probability at least $1 - \delta$.

Duplicate Detection. Instead of simply generating false leaf vertices, the adversary could instead replicate a given reading multiple times. For example, the smallest reading could be replicated many times, causing the median to decrease. SIA detects this by first (1) checking whether the committed sequence is sorted and then (2) performing probes on neighboring pairs of values to check for duplicates. The check for sortedness is by repeatedly performing binary searches for uniformly randomly selected elements. It can be shown that if there are many out-of-order elements in the committed sequence, the binary search invariant will eventually be violated in the course of one of these searches. Specifically, if there does not exist an increasing subsequence of length at least $(1 - \varepsilon_1')$ fraction of the entire sequence, then each binary search has probability at least $1 - (\varepsilon_1'/4)$ of detecting that the committed sequence was not sorted. Hence, $4/\varepsilon_1'$ searches will detect

unsortedness with probability at least $1-1/\varepsilon'$, and $\frac{4}{\varepsilon_1'}\ln\frac{1}{\delta}$ searches will reveal unsortedness with probability at least $1-\delta$. Once we have established that the sequence is mostly sorted with high probability, it is clear that, if there are a substantial number of duplicates, most of these duplicates must occur *in-place*, i.e., next to the duplicated element so that they do not increase the number of out-of-order elements. Such in-place duplicates are easily detected by unform random sampling of neighboring vertices in the hash tree. In this case, if there are ε'' or more fraction of in-place fake leaves, it be shown that $\frac{1}{\varepsilon_1''}\ln\frac{1}{\delta}$ total pairs of samples will detect a duplicate with probability at least δ.

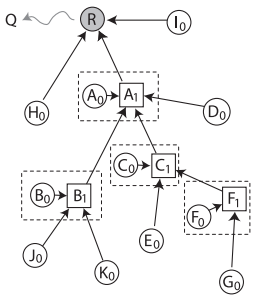
Putting it together: Median. Once we have established that the committed sequence is sorted and there are no (or negligibly few) fakes and duplicates, the median test is straightforward. First, we check that the central element is indeed the reported median. Then, to ensure that the central element is indeed central in the sorted sequence, we perform additional probes: in each probe, a random element is chosen, and compared with the alleged median; if it is to the right of the median element, then it must be larger than the median, and vice versa. If all of these checks complete successfully, then the reported median is accepted.

5.3 Hash Trees for Hierarchical Aggregators

The single aggregator SIA can be extended to support multiple hierarchical aggregators [13, 59]. The simplest extension of the hash tree structure to support multiple aggregators is to have the hash tree follow the topology of the network exactly.

An example of this is shown in Figure 5.3, depicting the construction of a topology-following hash tree for the topology of Figure 2.2. In this example, sensor node G constructs a leaf vertex consisting of its input value r_G (e.g., a sensor reading) and its node ID G. Each leaf node in the network topology transmits its leaf vertex to its parent (e.g., G sends its leaf vertex to F).

Each internal (non-leaf) sensor node i in the topology receives from each of its children a hash tree vertex. The parent node i then generates



$$R = \langle H[v_R = r_H + v_{A,1} + r_I \| H_0 \| A_1 \| I_0] \rangle$$

$$A_1 = \langle H[v_{A,1} = r_A + v_{B,1} + v_{C_1} + r_D \| A_0 \| B_1 \| C_1 \| D_0] \rangle$$

$$C_1 = \langle H[v_{C,1} = v_{F,1} + r_C + r_E \| C_0 \| E_0 \| F_1] \rangle$$

$$F_1 = \langle H[v_{F,1} = r_F + r_G \| F_0 \| G_0] \rangle$$

$$G_0 = \langle H'[ID_G \| r_G \| \mathrm{MAC}_{K_G}(ID_G \| N \| r_G)] \rangle$$

Fig. 5.3 Basic hash 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. On the right we list the labels of the vertices on the path of node G to the root.

its own leaf vertex, and creates a new parent over its leaf vertex and the vertices supplied by its children as follows:

$$u_i = \langle H[v_i \| u_1 \| u_2 \| \cdots \| u_k] \rangle$$

Where u_i is an internal vertex created by node i, and hash tree vertices $u_1, \cdots, u_k$ are received from the children of i (or generated by i itself). The value v_i is the aggregated value computed from the contributions of the values from each child. Note that different hash functions are used for the hash tree leaf vertices and the hash tree internal vertices. This ensures that leaf vertices are properly terminated and an attacker cannot attempt to "graft" another subtree onto a leaf vertex.

For example, in Figure 5.3, the sensor node A receives internal vertices B_1 and C_1 and the leaf vertex D_0 as well as its own leaf vertex A_0. Node A then creates a new parent vertex by computing a hash over the computed aggregate $v_{A,1}$ (which is simple summation in this example) and $A_0 \| B_1 \| C_1 \| D_0$. The result is a new internal vertex in the hash tree which is the parent of all the vertices received by A. Once each node i has computed all its internal hash tree vertices it transmits them to its parent, which will then construct its own internal vertex as the parent of all the vertices it receives, and so on. This also occurs at the base station, which computes the root vertex r of the hash tree.

In general, in the simple topology-following hash tree, a new internal vertex is added to the hash tree each time an aggregation takes place. The main limitation of this basic approach is that the commitment

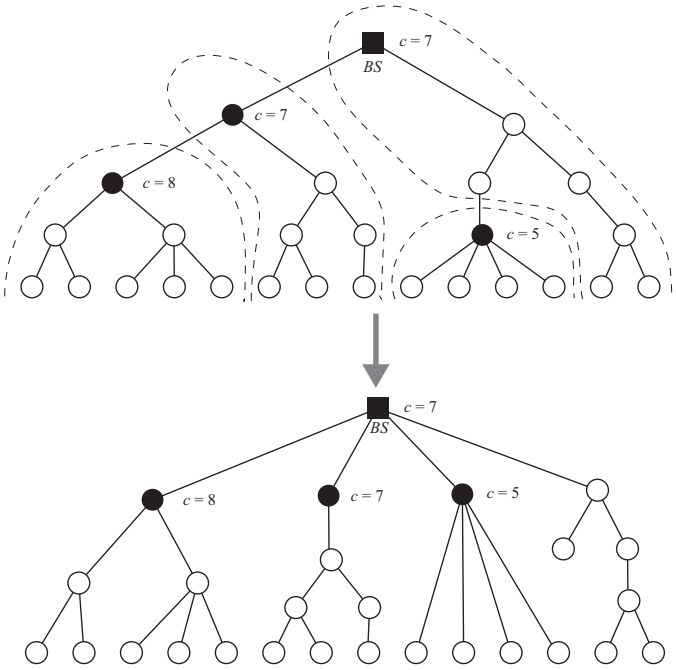


Fig. 5.4 SDAP Tree Balance via Leader Election.

tree potentially becomes as unbalanced as the network topology; for example, in a linear topology with n nodes, the communication cost of probing the farthest leaf is $n-1$ (since it is at depth n) and the cost of probing any node at random is $\Theta(n)$.

5.4 SDAP: Probing the Hierarchical Hash Tree

The SDAP algorithm [59] uses a hash tree construction that extends the basic hash tree construction of Section 5.3. SDAP consists of two separate algorithms for adapting the commitment structure to a hierarchical aggregation process.

Balancing the tree via Random Leader Election. SDAP follows the basic topology-following hash tree construction, with some differences. SDAP uses Message Authentication Codes (MAC) instead of a hash function to compute the vertices (this difference has no func-

tional impact and is omitted from our discussion, as are the particular message constructions of the protocol). A more important design distinction is that SDAP breaks the single large hash tree into a number of smaller subtrees. This decomposition is performed using a form of *random leader election*. In the random leader election, each node self-elects to become a leader based on a deterministic random process which takes as inputs the query nonce (N), its ID, and the number of currently ungrouped nodes in its subtree, c. Specifically, the hash tree is built from the leaves up as per Section 5.3; however during the construction process, a running count of the number of currently ungrouped nodes is maintained. The running count c is initialized to 1 at the leaves and added up at each internal node. A node x elects itself as a leader if $H[N\|ID_x] < F(c)$ where F is an increasing function of c. When a node elects itself as leader, it effectively splits off the subtree rooted at itself from the main topology; this causes all the nodes in its subtree to not count towards the number of ungrouped nodes at its parent. The process thus proceeds up until the root (which is always a group leader). This process is illustrated in Figure 5.4. Each group leader effectively communicates directly (using end-to-end authentication) with the base station by having other nodes forward their messages. The effect is a hash tree that is somewhat more balanced compared with the actual network topology.

Value-weighted Probing. Another change that needs to be made from the single-aggregator case is to enable a probabilistic probe of the leaf values from the base station without knowledge of the exact topology of the hash tree. As a comparison, consider probing a hash tree leaf at random in SIA: since the base station is informed of the number of leaves n, it can simply select a number in $[1, .., n]$ and probe that specific position in the hash tree. For a hash tree of arbitrary topology, simply noting the index of a node does not indicate the shape or form of the probe path. This needs to be explicitly determined in a manner such that nodes which contributed more to the result have a greater likelihood of being probed. The SDAP probe proceeds as follows: the base station selects a random probe *seed value* S_a. It then traverses the (rebalanced) hash tree from the root in the following way: at each

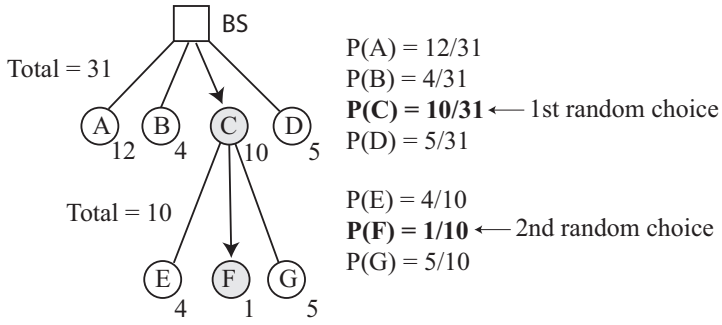


Fig. 5.5 SDAP Value-weighted Probing.

node visited by the probe, if the node corresponds to a leaf sensor node (it has no children) then the node itself is returned; otherwise, it is an internal hash tree vertex corresponding to an aggregation operation, and each child vertex of this vertex represents an input to the aggregation operation. A child vertex is chosen based on how much that input contributed to the aggregate value at that node. For example, in probing the COUNT aggregate, the probability of picking any given child is proportional to its reported sub-count from its sub-tree. Specifically, for a set of d children with counts $c_1, \ldots, c_d$, the parent node computes $\sum_{k=1}^{d} c_k \cdot H(S_a \| ID_k)$ where H is a hash function which returns a uniformly distributed real number between $0, 1$. The ith child is picked for further probing if this value falls between $[\sum_{k=1}^{i-1} c_k, \sum_{k=1}^{i} c_k]$. The process is then repeated at the selected child until termination. This process is illustrated in Figure 5.5. This probing algorithm allows the base station to issue probes based on the relative contribution of each hash tree vertex; hash tree vertices that add a greater contribution to the final result are thus more likely to be probed. The same principles of sampling that were derived in SIA (see Section 5.2) thus carry over directly, yielding a probabilistic security property: the more the attacker causes a deviation from the correct result, the more likely it is for the random probes to detect this inconsistency.

5.5 SecureDAV: Distributed Verification

SecureDAV [40] is a system for encrypting and authenticating an aggregation result for the single-aggregator case. Since this article only covers data integrity and not secrecy, we focus on the methods for ensuring integrity. The SecureDAV scheme assumes that sensor nodes have access to a *threshold signature* mechanism. In a threshold signature scheme, a group of signers G are each preloaded with specially constructed private keys, while the designated verifier (in this case, the base station) holds the group public key. The threshold signature scheme allows any t members of the group to sign a given message M such that the verifier is assured that at least t members of the group indeed attest to the authenticity of M. This signature scheme is used in verifying aggregation integrity as follows. First, the aggregator computes the aggregate result v_r, which is broadcast to all the sensor nodes. The nodes then inspect the reported aggregate to see if it is consistent with their individual readings: for example, if v_r is supposed to be an average over a normally-distributed set of readings, sensors could check if the reported average is within a certain range (e.g., one prior standard distribution) of their own reading. In a typical case, this check should pass for a large number of sensor nodes (i.e., $> t$ with high probability); these nodes will then compute their respective partial signatures on v_r and forward them to the aggregator. If there are at least t such partial signatures, then the threshold property is satisfied and the aggregator can then construct a valid group signature on v_r. The base station then only needs to check a single signature to verify that at least t nodes agreed on the reported aggregate v_r.

The main important design choice in SecureDAV is that this is the first design we have covered where the verification of the central global result is fully distributed, i.e., being performed by the sensor nodes that are contributing data values instead of by the base station. This technique will be further extended by subsequent schemes, most notably SHIA [13], which we discuss in the following section.

5.6 Secure Hierarchical In-Network Aggregation

The resilient estimation techniques of Section 4 yield approximate results that were fully resistant against tampering; the commitment-based probabilistic probing techniques of Section 5 yield exact results but can only detect tampering probabilistically. In this section, we discuss in detail a variation of the commitment-based approach that yields both exact results and are also fully resistant against malicious manipulation.

SHIA, or *Secure Hierarchical In-network Aggregation* is proposed by by Chan et al. [13]. The basic idea of SHIA is a three-phase protocol: 1) First SHIA builds a commitment tree over the aggregation topology (in a similar way to SDAP in Section 5.4, but using a different, more rigorous method of balancing the tree); 2) Subsequently, the commitment tree is then exhaustively probed *from the leaves*. This is a major difference from the base-station originated probes of SIA and SDAP; 3) Once the distributed probes are complete, a third phase aggregates the various confirmations from the network to the base station: if all the nodes have confirmed that the tree is free of inconsistencies, then the result is accepted. We now describe the protocol in more detail. For brevity, we first focus on the SUM aggregate, where the goal is to compute the total sum of all sensor readings.

Buliding a balanced commitment tree by delayed aggregation. The SHIA protocol begins with the basic topology-following hash tree described in Section 5.3 but with a balancing optimization that guarantees the generation of a binary hash tree with $\lceil \log n \rceil$ height.

In the naive commitment tree, each sensor node always computes the aggregate sum of *all* its inputs, which can be considered a strategy of *greedy aggregation*. SHIA considers instead the benefit of *delayed aggregation* at node C_1 in Figure 5.3. 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.

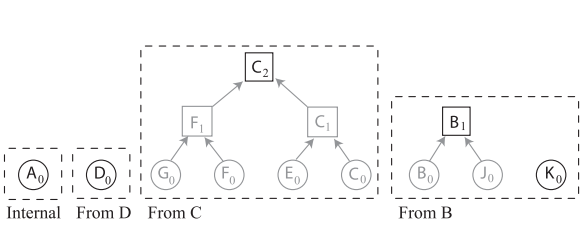
The strategy for delayed aggregation is as follows: the aggregation and hash tree construction algorithm proceeds from the leaves to the root as described in Section 5.3, except that an aggregation operation (along with the associated commit operation) is performed if and only if it results in a *complete, binary* commitment tree.

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. On the other hand, in the delayed aggregation algorithm in SHIA, 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 *perfect* binary trees (i.e., complete binary trees with 2^h leaf vertices where h is the height of the tree) and no two trees have the same height. These constraints are enforced by continually combining equal-height trees into perfect binary trees of greater height. Each combination step corresponds to an intermediate aggregation step: for example, in the case of SUM, we combine two subtrees with subsums s_1, s_2 by creating a new parent vertex over each of the subtree roots with the new sum $s_1 + s_2$.

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 perfect binary trees, the tallest tree in any commitment forest has height at most $\log n$. Moreover, because there are no two trees of the same height, any commitment forest has at most $\log n$ trees. To facilitate the construction of perfect binary trees in the forest, we add an additional field into the hash tree construction to keep track of the height of each hash subtree.

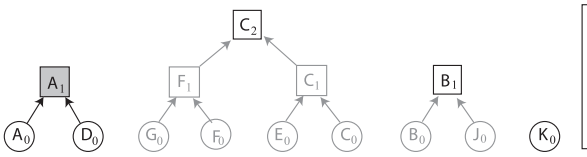
Figure 5.6 shows an example of the commitment-tree construction process for node A based on the topology in Figure 5.3. The algorithm terminates in $O(q \log n)$ steps since each step reduces the number of trees in the forest by one, and there are at most $q \log n + 1$ trees in the forest. Hence, each 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



$A_0 = \langle 1, ID_A, r_A, \text{MAC}_{K_A} \rangle$
$D_0 = \langle 1, ID_D, r_D, \text{MAC}_{K_D} \rangle$
$C_2 = \langle 4, v_{C_2}, H[N||4||v_{C_2}||F_1||C_1] \rangle$
$B_1 = \langle 2, v_{B_1}, H[N||2||v_{B_1}||B_0||J_0] \rangle$
$K_0 = \langle 1, ID_k, r_K, \text{MAC}_{K_K} \rangle$

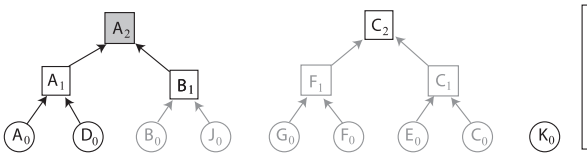
(a) Inputs: A generates A_0, and receives D_0 from D, C_2 from C, and (B_1, K_0) from B. Each dashed-line box shows the commitment forest received from a given sensor node. The solid-line box shows the vertex labels, each solid-line box below shows the labels of the new vertices.



$v_{A_1} = a_A + a_D$

$A_1 = \langle 2, v_{A_1}, H[N||2||v_{A_1}||A_0||D_0] \rangle$

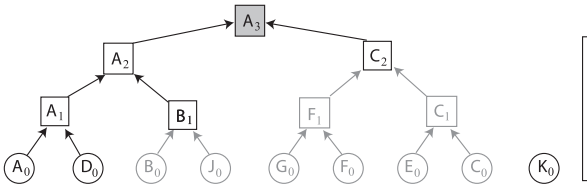
(b) First merge: Vertex A_1 created



$v_{A_2} = v_{A_1} + v_{B_1}$

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

(c) Second merge: Vertex A_2 created



$v_{A_3} = v_{A_2} + v_{C_2}$

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

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

Fig. 5.6 Process of node A (in the topology from Figure 2.2) deriving its commitment forest from the commitment forests received from its children.

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.

Note that 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 $O(\log n)$. Furthermore, 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 SHIA. This bound can be further improved to $O(\log n)$ by ensuring that all the vertices generated by a given sensor node lie on the same path to the root [23].

Distributed probing. Once the hash tree has been computed, the base station authentically broadcasts the hash tree root, and each sensor node disseminates the path to the root down to all of its children: eventually every sensor node will have the verification path in the hash tree from its leaf vertex up to the authenticated root vertex. Since the tree is balanced, this process also takes no more than $O(\log^2 n)$ congestion. Using this verification path, each sensor node can then verify that all the aggregation subcomputations involving its contributed value have been computed correctly: specifically, for the SUM operation, each subaggregate value should be the correct sum of all the inputs to that vertex, and none of the inputs should be negative.

Confirmation aggregation. 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 $\mathrm{MAC}_{K_s}(N||OK)$ where OK is a unique message identifier and K_s is the key that s shares with the querier. Each authentication code should be only computed once per value of N (e.g. N could be a sequence number that does not wrap around for the duration of the network). 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.

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:

$$\mathrm{MAC}_{K_1}(N||OK) \oplus \cdots \oplus \mathrm{MAC}_{K_n}(N||OK)$$

The querier then compares the computed authenticator with the received authenticator. If the two authenticators 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 (with a new nonce) 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 amortized over the other successful queries. In normal operation, the total congestion for the basic SHIA algorithm is $O(\log^2 n)$. The algorithm as described enables the querier to ensure a lower bound for the reported aggregate, i.e., the reported aggregate is no less than the total of the readings of the legitimate nodes. If the total number of nodes n is known, then a corresponding upper bound can be enforced by computing the complement of the aggregate: the aggregate and its complement should always sum to a fixed value dependent only on n.

It may appear surprising that a number of independent piece-meal verifications can directly compose to ensure that the overall aggregate is secure. A rigorous analysis is provided in the original paper [13]; here we provide a rough intuition of the security mechanics. Essentially, the cryptographic commitment properties of the hash tree ensure that the structure of the computations is "locked down": when the base station broadcasts the authentic root vertex value r, the adversary is unable to construct more than one version of a hash tree that hashes to that root value r. The fact that the hash tree is fixed ensures that each leaf vertex has a unique, consistent path to the root; the individual self-verifications then ensure that these computation paths of each legitimate node to the root consist only of correct aggregation operations.

As a result, the adversary is unable to perform aggregate miscomputation attacks anywhere along a computation path that has a legitimate sensor node's value as a leaf: it is thus functionally restricted to only falsifying the values coming from the nodes that it controls.

5.7 Summary and Discussion

The common design element in all commitment-based techniques is to construct an "audit structure" over the inputs and computations of the aggregation. This structure is then probed (usually probabilistically) for consistency. To support this, the audit structure must provide two properties. Firstly, any malicious tampering with the aggregate computation must cause a detectable change in the audit structure; secondly, the audit structure should not be malleable in such a way that this change can be hidden from the auditors. A secondary concern is that the audit process itself is comparable in efficiency to the aggregation process. Cryptographic hash trees are excellent structures for providing the properties of sensitivity to input and non-malleability, as well as supporting efficient probing, and hence they are often the building-blocks of choice in designing these commitment-based algorithms.

The primary advantage of commitment-based techniques is that they support the reporting of a precise (not estimated or probabilistic) aggregate value that is bound tightly to a structure that can be used for verification. Specifically, the tight binding allows us to detect that if a discrepancy occurs, it must necessarily be due to malicious tampering (this is in contrast with resilient estimators, where an estimation could deviate from the true aggregate simply due to natural errors). The converse also holds: we can show that if any malicious tampering has taken place then the the hash tree must necessarily reflect this inconsistency somewhere.

The primary limitation of commitment-based schemes is thus in detecting and locating these inconsistencies. It is not hard to see that a "full map" of all inconsistencies in the hash structure would negate the usefulness of aggregation in terms of requiring communication overhead at least linear in the size of the network. To work around this, the algorithms in this class of schemes thus must use either probabilis-

Secure aggregation protocols	
Resilient Estimation Techniques	Commitment-based Techniques
approximate result	exact result
guaranteed detection	either: (a) probabilistic detection
resilient vs DoS	or: (b) less resilient vs DoS
can be made one pass	usually multiple rounds

Table 5.1 Comparison of two general approaches for secure aggregation.

tic probing (SIA, SDAP) or distributed probing (SHIA, SecureDAV). Probabilistic probing schemes yield only probabilistic security properties, with security being a function of the amount of communication overhead invested into probing the audit structure. Distributed probing is excellent in terms of both providing full result precision and a guaranteed (optimal) security bound on the extent of undetected malicious interference; however a significant drawback is that a single round of distributed probing generally is unable to efficiently locate the exact source of any malicious tampering: due to this, such protocols are generally susceptible to denial-of-service attacks where a malicious node can cause aggregate reports to repeatedly fail while retaining relative anonymity. Table 5.1 summarizes the comparison between resilient estimation and commitment-based techniques.

5.8 References and Further Reading

Commitment-based secure aggregation was introduced in the SIA framework by Przydatek et al. [12, 49]. Besides the method for Median that was discussed in this article, the framework also contains algorithms for various other measures such as Count, Count Distinct, and Sum. SDAP was proposed by Yang et al. [59]. In addition to the basic framework for hash tree construction and probing, the paper also contains a description of a technique for increasing the probing probability for groups that are detected as outliers in the data set. SecureDAV is due to Mahimkar and Rappaport [40]. The SHIA algorithm is due to Chan et al. [13]. Frikken and Dougherty extend this scheme with a technique for rearranging the hash tree such that all vertices belonging

to a given sensor node share the same path to the root of the hash tree: this improves the communication congestion to $O(\log n)$ [23]. Manulis and Schwenk formally proved the cryptographic security of this algorithm [44]. Taban and Gligor [55] improve the resilience of the algorithm with an algorithm for identifying the specific nodes that are malicious. Chan et al. also further extend the principles of the framework to provide security for not just aggregation computations but also general communications primitives like broadcast authentication and node-to-node signatures [11].

5.9 Conclusion

We present an overview of general approaches for secure aggregation computations in distributed networks. We have focused primarily on algorithmic design rather than proof techniques. As distributed systems continue to scale up and become more pervasive, secure distributed computation techniques will become increasingly important. It is our hope that the techniques and discussion presented in this article will help both practitioners and researchers map the design space in the development of new and more resilient algorithms for distributed computing in security-critical distributed systems.

References

[1] Daniel J. Abadi, Samuel Madden, and Wolfgang Lindner. Reed: robust, efficient filtering and event detection in sensor networks. In *Proceedings of international conference on Very Large Data Bases (VLDB)*, 2005.

[2] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999.

[3] Ross Anderson, Francesco Bergadano, Bruno Crispo, Jong-Hyeon Lee, Charalampos Manifavas, and Roger Needham. A new family of authentication protocols. *ACM SIGOPS Operating Systems Review*, 32(4):9–20, 1998.

[4] Benjamin Arai, Gautam Das, Dimitrios Gunopulos, and Vana Kalogeraki. Efficient approximate query processing in peer-to-peer networks. *IEEE Transactions on Knowledge and Data Engineering*, 19(7):919–933, 2007.

[5] Brian Babcock, Surajit Chaudhuri, and Gautam Das. Dynamic sample selection for approximate query processing. In *Proceedings of ACM SIGMOD*, 2003.

[6] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. Counting distinct elements in a data stream. In *Proceedings of International Workshop on Randomization and Approximation Techniques*, 2002.

[7] V. Barnet and T. Lewis. Outliers in statistical data. *John Wiley and Sons, Inc.*, 1994.

[8] Joel Branch, Boleslaw Szymanski, Chris Giannella, Ran Wolff, and Hillol Kargupta. In-network outlier detection in wireless sensor networks. In *Proceedings of International Conference on Distributed Computing Systems (ICDCS)*, 2006.

[9] C. Castelluccia, E. Mykletun, and G. Tsudik. Efficient aggregation of encrypted data in wireless sensor networks. In *Proceedings of MobiQuitous*, 2005.

[10] Hasan Çam, Suat Özdemir, Prashant Nair, Devasenapathy Muthuavinashiappan, and H. Ozgur Sanli. Energy-efficient secure pattern based data aggregation for wireless sensor networks. *Computer Communications*, 29(4):446–455, 2006.

[11] Haowen Chan and Adrian Perrig. Efficient security primitives derived from a secure aggregation algorithm. In *Proceedings of ACM Conference on Computer and Communications Security (CCS)*, 2008.

[12] Haowen Chan, Adrian Perrig, Bartosz Przydatek, and Dawn Song. SIA: secure information aggregation in sensor networks. *Journal of Computer Security*, 15(1):69–102, 2007.

[13] Haowen Chan, Adrian Perrig, and Dawn Xiaodong Song. Secure hierarchical in-network aggregation in sensor networks. In *Proceedings of ACM Conference on Computer and Communications Security (CCS)*, 2006.

[14] Jeffrey Considine, Feifei Li, George Kollios, and John W. Byers. Approximate aggregation techniques for sensor databases. In *Proceedings of IEEE International Conference on Data Engineering (ICDE)*, 2004.

[15] Graham Cormode, Minos N. Garofalakis, S. Muthukrishnan, and Rajeev Rastogi. Holistic aggregates in a networked world: Distributed tracking of approximate quantiles. In *Proceedings of ACM SIGMOD*, 2005.

[16] Graham Cormode and Marios Hadjieleftheriou. Finding frequent items in data streams. In *Proceedings of international conference on Very Large Data Bases (VLDB)*, 2008.

[17] Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[18] Emiliano De Cristofaro, Jens-Matthias Bohli, and Dirk Westhoff. FAIR: fuzzy-based aggregation providing in-network resilience for real-time wireless sensor networks. In *Proceedings of ACM conference on Wireless network security (WiSec)*, 2009.

[19] Josep Domingo-Ferrer. A provably secure additive and multiplicative privacy homomorphism. In *Proceedings of International Conference on Information Security*, 2002.

[20] Wenliang Du, Jing Deng, Yunghsiang Han, and Pramod K. Varshney. A witness-based approach for data fusion assurance in wireless sensor networks. In *Proceedings of IEEE Global Telecommunications Conference*, 2003.

[21] Cristian Estan and George Varghese. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *ACM Transactions on Computer Systems (TOCS)*, 21(3):270–313, 2003.

[22] Philippe Flajolet and G. Nigel Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31(2):182–209, 1985.

[23] Keith B. Frikken and Joseph A. Dougherty. An efficient integrity-preserving scheme for hierarchical sensor aggregation. In *Proceedings of ACM conference on Wireless network security (WiSec)*, 2008.

[24] Saurabh Ganeriwal, Laura K. Balzano, and Mani B. Srivastava. Reputation-based framework for high integrity sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 4(3):1–37, 2008.

[25] M. Garofalakis, J.M. Hellerstein, and P. Maniatis. Proof sketches: Verifiable In-Network aggregation. In *Proceedings of IEEE International Conference on Data Engineering (ICDE)*, 2007.

[26] Phillip B. Gibbons. Distinct sampling for highly-accurate answers to distinct values queries and event reports. In *Proceedings of international conference on Very Large Data Bases (VLDB)*, 2001.

[27] Michael Greenwald and Sanjeev Khanna. Space-efficient online computation of quantile summaries. In *Proceedings of ACM SIGMOD*, 2001.

[28] Michael Greenwald and Sanjeev Khanna. Power-conserving computation of order-statistics over sensor networks. In *Proceedings of PODS*, 2004.

[29] Sudipto Guha and Andrew McGregor. Approximate quantiles and the order of the stream. In *Proceedings of PODS*, 2006.

[30] Carl Hartung, James Balasalle, and Richard Han. Node compromise in sensor networks: The need for secure systems. Technical Report Technical Report CU-CS-990-05, Department of Computer Science, University of Colorado, January 2005.

[31] Wenbo He, Xue Liu, Hoang Nguyen, Klara Nahrstedt, and Tarek F. Abdelzaher. PDA: Privacy-preserving data aggregation in wireless sensor networks. In *Proceedings of IEEE INFOCOM*, 2007.

[32] Lingxuan Hu and D. Evans. Secure aggregation for wireless networks. In *Proceedings of Symposium on Applications and the Internet Workshops*, 2003.

[33] Peter J. Huber. Robust statistics. Wiley, 1981.

[34] Ryan Huebsch, Brent N. Chun, Joseph M. Hellerstein, Boon Thau Loo, Petros Maniatis, Timothy Roscoe, Scott Shenker, Ion Stoica, and Aydan R. Yumerefendi. The architecture of PIER: an internet-scale query processor. In *Proceedings of Conference on Innovative Data Systems Research (CIDR*, 2005.

[35] Pawan Jadia and Anish Mathuria. Efficient secure aggregation in sensor networks. In *Proceedings of International Conference on High Performance Computing*, 2004.

[36] Yee Wei Law, Jeroen Doumen, and Pieter Hartel. Survey and benchmark of block ciphers for wireless sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 2(1):65–93, 2006.

[37] Donggang Liu and Peng Ning. Multilevel μTESLA: Broadcast authentication for distributed sensor networks. *ACM Transactions on Embedded Computing Systems (TECS)*, 3(4):800–836, 2004.

[38] Christian Lochert, Björn Scheuermann, and Martin Mauve. Probabilistic aggregation for data dissemination in vanets. In *Proceedings of ACM international workshop on Vehicular ad hoc networks (VANET)*, 2007.

[39] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. TAG: A tiny aggregation service for ad-hoc sensor networks. In *Proceedings of USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2002.

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

[41] Amit Manjhi, Suman Nath, and Phillip B. Gibbons. Tributaries and deltas: efficient and robust aggregation in sensor network streams. In *Proceedings of ACM SIGMOD*, 2005.

[42] Gurmeet Singh Manku and Rajeev Motwani. Approximate frequency counts over data streams. In *Proceedings of international conference on Very Large Data Bases (VLDB)*, 2002.

[43] Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G. Lindsay. Approximate medians and other quantiles in one pass and with limited memory. In *Proceedings of ACM SIGMOD*, 1998.

[44] Mark Manulis and Jorg Schwenk. Provably secure framework for information aggregation in sensor networks. In *Proceedings of the International Conference on Computational Science and Its Applications*, 2007.

[45] Ralph C. Merkle. A digital signature based on a conventional encryption function. In *Proceedings of CRYPTO*, 1987.

[46] Suman Nath, Phillip B. Gibbons, Srinivasan Seshan, and Zachary R. Anderson. Synopsis diffusion for robust aggregation in sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 4(2):7:1–7:40, 2008.

[47] Adrian Perrig, J. D. Tygar, Dawn Song, and Ran Canetti. Efficient authentication and signing of multicast streams over lossy channels. In *Proceedings of IEEE Symposium on Security and Privacy*, 2000.

[48] Fabio Picconi, Nishkam Ravi, Marco Gruteser, and Liviu Iftode. Probabilistic validation of aggregated data in vehicular ad-hoc networks. In *Proceedings of ACM international workshop on Vehicular ad hoc networks (VANET)*, 2006.

[49] Bartosz Przydatek, Dawn Xiaodong Song, and Adrian Perrig. SIA: secure information aggregation in sensor networks. In *Proceedings of ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2003.

[50] Rodrigo Roman, Cristina Alcaraz, and Javier Lopez. A survey of cryptographic primitives and implementations for hardware-constrained sensor network nodes. *Mobile Networks and Applications*, 12(4):231–244, 2007.

[51] Sankardas Roy, Sanjeev Setia, and Sushil Jajodia. Attack-resilient hierarchical data aggregation in sensor networks. In *Proceedings of ACM workshop on Security of Ad Hoc and Sensor Networks (SASN)*, 2006.

[52] Bo Sheng, Qun Li, Weizhen Mao, and Wen Jin. Outlier detection in sensor networks. In *Proceedings of ACM MobiHoc*, 2007.

[53] Nisheeth Shrivastava, Chiranjeeb Buragohain, Divyakant Agrawal, and Subhash Suri. Medians and beyond: new aggregation techniques for sensor networks. In *Proceedings of ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2004.

[54] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos. Online outlier detection in sensor data using non-parametric models. In *Proceedings of international conference on Very Large Data Bases (VLDB)*, 2006.

[55] Gelareh Taban and Virgil D. Gligor. Efficient handling of adversary attacks in aggregation applications. In *Proceedings of ESORICS*, 2008.

[56] David Wagner. Resilient aggregation in sensor networks. In *Proceedings of ACM workshop on Security of Ad Hoc and Sensor Networks (SASN)*, 2004.

[57] Dirk Westhoff, Joao Girão, and Mithun Acharya. Concealed data aggregation for reverse multicast traffic in sensor networks: Encryption, key distribution, and routing adaptation. *IEEE Transactions on Mobile Computing*, 5(10):1417–1431, 2006.

[58] Praveen Yalagandula and Michael Dahlin. A scalable distributed information management system. In *Proceedings of ACM SIGCOMM*, pages 379–390, 2004.

[59] Yi Yang, Xinran Wang, Sencun Zhu, and Guohong Cao. SDAP: a secure hop-by-hop data aggregation protocol for sensor networks. In *Proceedings of ACM MobiHoc*, 2006.

[60] Haifeng Yu. Secure and highly-available aggregation queries in large-scale sensor networks via set sampling. In *Proceedings of International Conference on Information Processing in Sensor Networks*, 2009.

[61] Ying Zhang, Xuemin Lin, Yidong Yuan, Masaru Kitsuregawa, Xiaofang Zhou, and Jeffrey Xu Yu. Summarizing order statistics over data streams with duplicates. In *Proceedings of IEEE International Conference on Data Engineering (ICDE)*, 2007.