SIA: Secure Information Aggregation in Sensor Networks

HAOWEN CHAN
Carnegie Mellon University

ADRIAN PERRIG[*]
Carnegie Mellon University

BARTOSZ PRZYDATEK[†]
ETH Zurich

DAWN SONG
Carnegie Mellon University

Abstract

In sensor networks, data aggregation is a vital primitive enabling efficient data queries. An on-site aggregator device collects data from sensor nodes and produces a condensed summary which is forwarded to the off-site querier, thus reducing the communication cost of the query. Since the aggregator is on-site, it is vulnerable to physical compromise attacks. A compromised aggregator may report false aggregation results. Hence, it is essential that techniques are available to allow the querier to verify the integrity of the result returned by the aggregator node.

We propose a novel framework for secure information aggregation in sensor networks. By constructing efficient random sampling mechanisms and interactive proofs, we enable the querier to verify that the answer given by the aggregator is a good approximation of the true value, even when the aggregator and a fraction of the sensor nodes are corrupted. In particular, we present efficient protocols for secure computation of the median and average of the measurements, for the estimation of the network size, for finding the minimum and maximum sensor reading, and for random sampling and leader election. Our protocols require only sublinear communication between the aggregator and the user.

Keywords: sensor networks, information aggregation, security, approximate interactive proofs.

1 Introduction

Sensor networks can provide economical solutions in many applications such as real-time traffic monitoring, wildfire tracking, wildlife monitoring, or building safety monitoring. In a sensor network, hundreds or thousands of wireless sensor devices called *sensor nodes* collectively monitor an area, generating a substantial amount of data. An important

[*]Contact author. Email: adrian@ece.cmu.edu. Address: Adrian Perrig, CIC 2110, 4720 Forbes Avenue, Pittsburgh, PA 15213. Telephone: 412 268 2242. Fax: 412 268 6779.

[†]Research done while this author was at Carnegie Mellon University

technical challenge is the design of efficient in-network information processing algorithms to reduce the energy cost of transmitting this large volume of data to the user [16, 22].

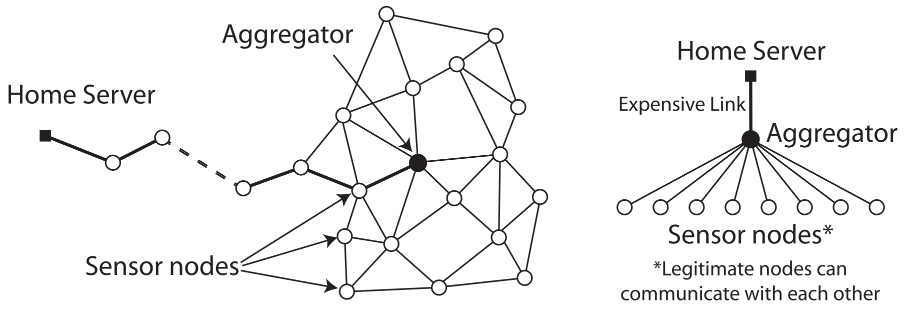
Information aggregation is an important technique which fulfills the function of efficient data collection. In an information aggregation algorithm, designated nodes in the sensor network, called *aggregators*, collect the raw information from the sensors, process it locally, and reply to the aggregate queries of a remote user. However, information aggregation in sensor networks is complicated by the fact that the sensor nodes and aggregators may be compromised due to physical tampering. A compromised aggregator may report arbitrary spurious results to the querier, thus allowing the adversary to subvert the operation of the entire network with only a few compromised nodes. To prevent this, techniques are needed to ensure that the user can still be confident of the (approximate) accuracy of the aggregated data even when the aggregator and a small subset of the sensor nodes are under the control of an adversary attempting to inject falsified data. However, the majority of current work in data aggregation assumes that every node is honest [12, 16, 22, 28].

We propose several algorithms to improve the confidence of the user in the result returned by the aggregator. Specifically, we focus on preventing an attack we call *stealthy aggregate manipulation*. In a stealthy aggregate manipulation attack, the attacker's goal is to make the user accept false aggregation results without revealing its presence to the user. Our goal is to develop countermeasures that will alert the user if the adversary causes the aggregation result to deviate significantly from the true value, thus preventing the user from accepting arbitrarily spurious values.

The framework we describe is called *aggregate-commit-prove*: in our setting, the aggregators not only perform the aggregation tasks, but also *prove* that they perform these tasks correctly. Specifically, to prevent the aggregators from cheating, we use cryptographic techniques of *commitments*, and construct efficient random sampling mechanisms and interactive proofs, which enable the user to verify that the answer given by the aggregators is a good approximation of the true value, even when the aggregators and/or a fraction of the sensor nodes may be corrupted.

1.1 Related work

Previous work in sensor network data aggregation has mainly focused on how to aggregate information assuming every node is honest [12, 16, 22, 28]. Hu and Evans [21] have studied the problem of information aggregation if one node is compromised, but their protocol may be vulnerable if a parent and a child node in their hierarchy are compromised. Wagner [36] studied what aggregation functions may be more resilient against malicious attacks. Jadia and Mathuria [23] extend the approach of Hu and Evans by incorporating privacy. Mahimkar and Rappaport [29] also propose an aggregation-verification scheme for the single-aggregator model using a secret-sharing scheme. Du *et al.* [13] propose using multiple *witness* nodes as additional aggregators to verify the integrity of the aggregator's result. Yang et al. [38] describe a secure aggregation algorithm which subdivides an aggregation tree into subtrees,



(a) Example physical topology (b) Logical Topology

Figure 1: Network Topology

each of which reports their aggregates directly to the base station. In related work, some of the authors have presented a secure aggregation scheme for the hierarchical aggregator case [9]. However, that algorithm is not practical for the single aggregator topology since it would overwhelm the aggregator with $\Omega(n \log^2 n)$ messages due to the aggregator's exceptionally high $\Omega(n)$ degree in the logical connectivity graph. A preliminary version of this paper appeared as [35].

The concept of interactive proofs was introduced by Goldwasser, Micali and Rackoff [20], and continues to be a very active research area since then. Ergün *et al.* [15] studied the problem of *approximate* interactive proofs, where a prover (the aggregator) proves to a verifier (the home server) that the input data has some property. However, in their model both the prover and the verifier can access the input data, and the task of the prover is to assist the verifier, so that the verifier doesn't have to read the entire input. Some of their solutions can be implemented directly in our model by simulating verifier's access to the input: whenever verifier should read a part of the input, he asks the prover to deliver the desired part. However, in many cases the *locations* of the desired parts should be hidden from the prover, hence a more expensive simulation is needed, e.g., using a private information retrieval protocol [7, 27].

2 Problem Statement: Secure Information Aggregation

Secure information aggregation refers to the efficient delivery of summaries of measured data from the sensor network to an off-site user in such a way that the user can have high confidence that the reported data summaries have not been manipulated by an adversary. In this section, we describe the specific parameters, metrics and assumptions of the problem.

2.1 Problem Setting

The secure aggregation problem arises in the following scenario, an example of which is depicted in Figure 1(a). Consider a large wireless ad-hoc sensor network deployed over an extensive area, with a subset of sensor nodes covering a particular area of interest. A remote *home server* wishes to perform an aggregate query (such as getting the median or average of the sensor readings) on the nodes in the area of interest. The home server could be, for example, a dedicated data-storage server or a user-driven program running on a desktop computer. Forwarding each of the individual sensor readings from each sensor node to the home server would be impractical for the energy-constrained sensor nodes [24]. Since only aggregate statistics are required, an *aggregator* device (which could be a sensor node itself, or a dedicated base station device) collects the readings from all the sensor nodes in the area of interest. The aggregator then computes the desired aggregation result, and transmits this condensed report to the home server. We assume that, due to its great length, the link between the home server and the aggregator is very costly in terms of energy overhead, and thus our goal is to minimize traffic between the aggregator and the home server. This assumption is true, for example, if the aggregation report is transmitted using peer-to-peer routing via the nodes of the sensor network, but some situations are exceptions; for example, if the aggregator was a resource-rich base station and was linked directly to the home server via a wired network. Such special cases are out of the scope of this paper since in-network aggregation is not particularly useful in such sensor networks. Some sensor networks may have multiple aggregators (for example, in TAG [28], each non-leaf node is an aggregator). In this paper, we only consider the case of a single aggregator.

We assume that the aggregator and some subset of the data nodes may be compromised by a malicious attacker (cf. Section 2.3). We assume the compromised nodes are unable to partition the network. Partitioning occurs when there are two or more subsets of legitimate nodes which are unable to communicate with each other except through one or more compromised nodes. Specifically, our assumption of non-partitioning implies that any legitimate data node is able to communicate with any other legitimate data node, and the adversary is unable to prevent this.

The problem topology is illustrated in Figure 1(b). The single aggregator collects data from the data nodes (which are free to communicate with each other) and reports the result to the home server, over the expensive home server-aggregator link. Our goal is to verify the reported result with as little traffic between the home server and the aggregator as possible.

2.1.1 Additional Assumptions

If we assume that (1) the home server has completely no information about which sensor nodes are active and uncompromised in the network and (2) the home server cannot communicate with the sensor nodes in the network except through the aggregator, then it is impossible to prevent a malicious aggregator from undetectably omitting the readings of any set of legitimate nodes: for example, the aggregator can falsely claim that some legitimate node A is dead, and there would be no way to verify this. Hence, one of the above conditions has to be weakened. We consider algorithms for both cases.

ID-knowledge scenario. This scenario assumes the home server has knowledge of the IDs of all the deployed sensor nodes and the adversary can only compromise a small fraction of these. Clearly, the home server always knows the universe of all possible node IDs since it shares cryptographic keys with every possible node ID. In this scenario, we further assume that the home server has knowledge of which IDs are deployed and alive (i.e., not battery exhausted, and connected to the network) in the area of interest. ID-knowledge could be maintained, for example, by randomly sampling sensor nodes in the network. We omit the details of such ID-knowledge maintainence algorithms and assume perfect ID-knowledge — adaptation of our algorithms for cases of partial ID-knowledge is straightforward.

Alarm channel scenario. This scenario assumes there exists a reliable multihop communication channel that sensor nodes can use to alert the home server of the presence of the adversary, and its latency bound is known. In certain applications, tracking the set of active node IDs in a specific area of interest may be difficult, especially considering natural node death and the possible disruptive presence of malicious nodes. In particular if the set of queried nodes is unknown to the home server but is defined via a logical predicate, ID-knowledge presents a technical challenge in itself. As an alternative, we consider the availability of a method for sensor nodes to (reliably) communicate with the home server without using the aggregator. In the most naive setting, this may simply be a network-wide flooding broadcast from the node, which guarantees transmission if the attacker has not successfully partitioned the node away from the network. Multipath routing may also be used; many such routing schemes are known. A few examples are braided multipath [18], meshed multipath [11], split multipath [26], SMLDR [3] and AOMDV [32]. It is assumed that this alarm channel is more expensive even than the link between the aggregator and the home server; however, since it is not used unless an adversary is detected, its high cost is not a factor under normal operation. Under this scenario, we also do not assume any limits on the ability of the adversary to perform node compromise.

In this paper, if the specific scenario is not specified, then the described techniques are applicable to *both* scenarios. Techniques specific to each scenario will be labeled as such in the section headings.

2.2 Key Setup

We assume that each sensor has a unique identifier and shares a separate secret cryptographic key with the home server and with the aggregator [34]. The keys enable message authentication, and encryption if data confidentiality is required. We assume that the home server has a mechanism to broadcast authentic messages into the network, such that each sensor node can verify the authenticity of the message, for example using the TESLA broadcast authentication protocol [33, 34].

2.3 Attack Model And Security Goals

We consider a setting with a polynomially-bounded attacker, which can compromise some of the sensors as well as the aggregator. Actions of a corrupted device are totally determined by the adversary (i.e., a compromised node or aggregator is *Byzantine* in its behavior). In particular, the adversary can arbitrarily change the measured values reported by a corrupted sensor. We assume that the adversary can only compromise a (small) fraction of the sensor nodes.

In this setting, we focus on *stealthy aggregate manipulation attacks*, where the attacker's goal is to make the home server accept false aggregation results, which are significantly different from the true results determined by the measured values, while not being detected by the home server. In this context, denial-of-service attacks such as not responding to queries clearly indicates to the home server that something is wrong and therefore is not a stealthy attack.

Our security goal is to prevent stealthy aggregate manipulation attacks. In particular, we want to guarantee that if the home server accepts a reported aggregation result from the aggregators, then the reported result is "close" to the true aggregation value with high probability; otherwise, if the reported value is significantly different from the true value due to the misbehavior of the corrupted aggregators and/or the sensors, the home server will detect the corruption and reject the reported aggregation result with high probability.

2.4 Notation and Conventions

In the remainder of this paper, n denotes the number of sensors, $S_1, \dots, S_n$, $\mathcal{A}$ denotes the aggregator, and $\mathcal{B}$ the home server. We consider scenarios where the values measured by the sensors are from some totally ordered set, and we denote by a_i the value reported by sensor S_i. In fact, without loss of generality, we assume that the values a_i are integers from $[m] = \{1, \dots, m\}$.

For the complexity analysis, we assume that each element and each hash value can be accessed in 1 step, and sending it costs 1 unit of communication. Also, we assume that each hash value can be computed in $O(1)$ steps. Assuming that all measurements may be different, in "real life" each element is actually at least $\lceil \log m \rceil$ bits long.

Finally, we assume that the verifier knows the number of sensors reporting the measurements (or a good estimate of this number). This number can be given as a system parameter, or can be estimated using our secure counting protocol, which we describe in Section 6.

2.5 Efficiency vs. Accuracy Tradeoff

The problems discussed in this paper have a straightforward (but very inefficient) solution: the aggregator forwards to the home server all data and authentication information from each sensor. Given all the data, the home server can verify authenticity of each data item, and answer all the statistical queries locally.

However, we assume that the communication between the aggregator and the home server is expensive, which makes this approach impractical. On the other hand, communicating just the result of a query is in many cases (e.g., for count, min/max, average, or median queries) very efficient, but it does not give the guarantee of correctness.

Moreover, for many of the problems studied in this paper one can show that in order to prove that the reported aggregation result is exact (with zero probability of error), we need at least linear communication complexity (linear in the size of the network), i.e., we cannot do much better than sending all the data to the aggregator. If we are willing to accept (a small) non-zero probability of error, then theoretically general methods based on PCP techniques could be applied [2, 25]. However, such methods would be very inefficient in practice. Hence, in order to achieve practical sublinear communication complexity, we need to relax the accuracy requirements and accept approximative results, and we need new, efficient techniques that bound the attacker's ability to tamper with the aggregation result.

Let the aggregation result be $y = f(a_1, \ldots, a_n)$ where $f : [m]^n \rightarrow \mathbb{R}$ is the aggregation function. We say that $\tilde{y}$ is a *multiplicative* ε*-approximation* of y (or just ε*-approximation*) if $(1-\varepsilon)y \leq \tilde{y} \leq (1+\varepsilon)y$. We say that $\tilde{y}$ is an *additive* ε*-approximation* of y if $y - \varepsilon \leq \tilde{y} \leq y + \varepsilon$.

The difference between y and $\tilde{y}$ can be caused by various factors:

(1) One or more corrupted data nodes is reporting a value different from its actual sensor reading. It may be difficult to detect this misbehavior since anomaly detection requires application-specific knowledge.

(2) In some scenarios, when the aggregator uses sampling techniques to calculate the aggregation result, the sampling technique will introduce some estimation error. We can bound the estimation error by adjusting the number of required samples.

(3) The aggregator may be compromised, and may try to cheat by reporting wrong aggregation values. Without security mechanisms, a corrupted aggregator can lie about the aggregation result and report wrong values that are very far from the true result.

Because the errors caused by factors (1) and (2) can be upper bounded, they do not represent a serious threat to data accuracy if simple countermeasures are taken, such as using only resilient aggregation functions like the median, or bounding the range of allowable sensor readings. Wagner [36] presents an in-depth analysis of the countermeasures against this class of attacks. In the rest of the paper we focus on describing new techniques preventing the attacks of the third kind (corrupted aggregator). To keep our techniques as general as possible, we do not assume application-specific knowledge (such as knowing the general statistical distribution of sensor readings). Without application-specific knowledge, no algorithm can extract any information about the true readings of compromised nodes. Hence, we will not consider the falsification of sensor data by compromised nodes to be an attack; when we speak of the "true" aggregate result, it is assumed that the aggregate considers the (possibly false) claimed reading of a compromised node as a legal input.

In addition to the approximation error ε, which describes the quality of a reported result, we introduce a parameter δ, which upper bounds the probability of not detecting a cheating aggregator (i.e., an aggregator reporting a result not within ε bounds). Formally, a (multiplicative/additive) (ε,δ)-*approximation* protocol finds a (multiplicative/additive) ε-approximation with probability at least $1-\delta$, and runs in time polynomial in $1/\varepsilon$ and $\log(1/\delta)$. Adopting this convention, our goal is the following: a (multiplicative/additive) (ε,δ)-*secure* protocol runs in time polynomial in $1/\varepsilon$ and $\log(1/\delta)$, and always returns the true aggregation result in the absence of an adversary. In the presence of an adversary, the protocol either (1) returns an aggregation result: the result is a (multiplicative/additive) ε-approximation of the true result with probability at least $1-\delta$, or (2) returns REJECT: if so, it is able to prove the presence of an adversary in the system. Intuitively, this means that if the protocol returns an aggregation result, there is only a small possibility δ that the returned result is not close to the true result. Hence, an adversary faced with an (ε,δ)-secure protocol may either report a result that is ε-approximate to the true result, or report a result that is not ε-approximate, but run the high $1-\delta$ probability of detection.

2.6 Tools and Techniques

In our constructions we make use of numerous tools from cryptography and distributed computing. Here we briefly describe the main techniques and concepts, and give references for more detailed expositions.

2.6.1 Committing to Data

The concept of *commitments* denotes the functionality which allows one party to commit to some data, so that the data cannot be changed once committed, but the data remains secret[1] until the commitment is *opened*. An important feature of (computationally binding) commitment schemes is that the size of the commitment can be substantially smaller than the size of the data committed to. For example in the *Merkle-tree* construction [30, 31] all the collected data is placed at the leaves of the tree, and the committing party then computes a binary hash tree starting from the leaf nodes: each internal node in the hash tree is computed as the hash value of the concatenation of the two child nodes. The root of the tree represents the commitment. Because the hash function in use is collision resistant, once the party commits

[1]This secrecy property is not essential for our applications.

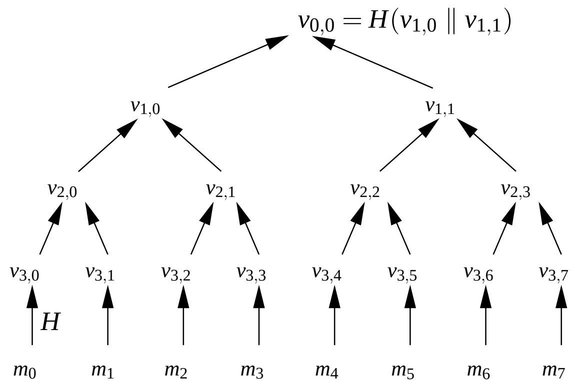


Figure 2: Merkle hash tree used to commit to a set of values.

to the collected values, she cannot change any of the collected values. On the other hand, the party can construct a concise proof of correctness for the value at any leaf by providing all hash values (with siblings) on the path from the leaf to the root. Note that the size of the proof of correctness is in this case only logarithmic in the size of the data. Figure 2 gives an example of a Merkle hash tree.

2.6.2 Interactive Proofs

Intuitively, an *interactive proof* is a two-party protocol which allows one party (the prover) to convince some other party (the verifier) about the truth of some statement. This concept was introduced by Goldwasser, Micali, and Rackoff [20]. Interactive proofs have numerous applications in cryptography, distributed computing, and in complexity theory (see for example a survey by Goldreich [19]).

2.6.3 Computation on Data Streams

In the setting of computation on data streams Flajolet and Martin [17] proposed a space-efficient technique for estimation of the number of distinct elements (μ) in a stream. The key idea is to pick a random hash function $h : [m] \rightarrow [0 \ldots 1]$, apply it to all the elements a_i and keep the value $v = \min_{i=1}^{n} h(a_i)$. Finally, the number of distinct elements is estimated by the value $\mu' = 1/v$.

Alon *et al.* [1] have shown that in this algorithm pairwise independent hash functions are sufficient to achieve with probability $2/c$ an estimate μ' satisfying $\mu/c \leq \mu' \leq c\mu$, for any $c > 2$. Bar-Yossef *et al.* [5] further improved this method and presented a (ε, δ)-approximation for μ. The basic idea for the improvement is to maintain several ($t = O(1/\varepsilon^2)$) elements a_i on which a randomly picked hash function h evaluates to the t smallest values. This significantly improves the accuracy for the cost of increased space complexity.

3 Our General Approach: Aggregate-Commit-Prove

We propose a new approach, which we call *aggregate-commit-prove*: aggregators collect the sensor nodes' raw data, compute the desired function of the data, and reply to the home server with the computation result together with a *commitment* to the collection of data; the home server and the aggregators then perform efficient *interactive proofs* such that the home server will be able to verify the correctness of the results (or detect cheating with high probability).

We show that such an approach improves both security and efficiency. By letting the aggregator perform the aggregation, the raw data does not need to be sent back to the home server, but only the aggregation result together with a (small) proof of correctness is transfered over the expensive long-distance communication link. By engaging with the aggregator in an interactive proof phase, the home server will detect with high probability if the aggregator or some sensor nodes are cheating.

More precisely, the solutions proposed in this paper consist of three steps: computation of the result, committing to the collected data and reporting the aggregation result, and proving the correctness of the result.

1. In the first step, the aggregator collects the data from sensors and locally computes the aggregation result. As we discuss in Section 2, each data node shares a key with the aggregator, such that the aggregator can verify the authenticity of each sensor reading (preventing sensor impersonation, but not flawed data from a corrupt sensor).

2. In the second step, the aggregator commits to the data it received from the data nodes. The commitment includes the authentication codes which allow the home server to verify that the data was indeed delivered from an authentic data node (and not, for example, simply fabricated by the aggregator). The commitment to the input data further ensures that the aggregator cannot be change the received data after the commitment step. Once the data is committed to, it can be used multiple times, for computation of various aggregates. Hence, it can be viewed as a pre-processing step, and its cost can be amortized over multiple aggregates.

3. In the third step, the aggregator and the home server engage in a protocol, in which the aggregator communicates the aggregation result and the commitment to the server, and proves to the server that the reported results are correct using interactive proof protocols. The interactive proof usually contains two phases:

   (a) The home server checks that the committed data is a good representation of the true data values in the sensor network.

   (b) The home server checks if the aggregator is cheating, in the sense that the aggregation result is not (close to) the correct result aggregated from the committed data values.

Functions Covered in Our Framework. It follows immediately that *any* function approximable by naive uniform sampling of the input values can be approximated securely in the proposed framework, since the combination of commitments with authentication enables reliable uniform sampling. In other words, our framework in a natural way enables communication-efficient and *robust* approximation of a general class of functions. However, as we show in the sequel, for some problems uniform sampling does not yield a good approximation, or is still too expensive in terms of communication. In a few such cases we propose solutions which are significantly better than the uniform sampling. Our techniques employ more involved actions of the aggregator and/or sampling over specially constructed probability spaces.

4 Common Primitives for Aggregate-Commit-Prove

We now describe in detail the phases of the aggregate-commit-prove framework which are common for most of the aggregation functions we will consider in this paper.

The home server initiates a query by sending it to the aggregator, which then disseminates the query message to the sensor nodes. The query message is broadcast-authenticated such that each sensor node can verify that it is from the home server. The message also contains a query nonce N, which uniquely identifies the query; for example, it could be the home server's unique identifier and a timestamp.

Each sensor node S_i then constructs a message:

$$M_i = ID_i \| a_i \| N \| \mathrm{MAC}_{K_{ia}}(ID_i \| a_i \| N) \| \mathrm{MAC}_{K_{is}}(ID_i \| a_i \| N)$$

The message contains the sensor node's unique identifier (ID_i), its reported data value a_i, the query nonce N, and two message authentication codes (MACs) of the data with the two keys that it shares with the aggregator and the home server respectively. This message is forwarded to the aggregator. While it is collecting the data, the aggregator checks the first authentication code ($\mathrm{MAC}_{K_{ia}}(ID_i \| a_i \| N)$) to verify that this data has a valid origin.

When the aggregator has received all the data messages from the sensor nodes, it cryptographically commits to the inputs it has received. This commitment binds the aggregator to a particular ordered set of inputs: after commitment, the aggregator cannot claim that its input set was anything other than the committed set, since any discrepancy would result in an inconsistent commitment value. The specific mechanics of the commitment are as follows. The aggregator first arranges the received messages (i.e. the various M_is) in a specific order (for example, sorted by the node IDs). Using these messages as leaves, the aggregator then constructs a Merkle hash tree. The root value of the hash tree is the commitment value, which is reported to the home server along with the number of leaves of the hash tree (i.e.

number of responding nodes) and the evaluated value of the aggregation function.

Once the home server has received the aggregator's response, it may then begin probing the aggregator to verify the correctness of the aggregated result. The adversary's may take any or all of the following actions:

1. Add some fabricated data values that were not reported by any sensor node at all.

2. Duplicate some existing legitimate sensor value so it has a disproportionately large impact on the aggregation result, or allow a compromised sensor node to report more than one data value.

3. Ignore some legitimate data values and not include them in the committed set of data readings.

4. Report a aggregation result that is inconsistent with the committed input data.

The countermeasures against attacks 1 and 2 are common to most of our algorithms regardless of the aggregation function, and we discuss them in Sections 4.1 and 4.2. Attack 3 can be prevented by calculating an estimate of the size of the set of data values, and we show how to do this in Section 6. Countermeasures against attack 4 involve tests specific to each aggregation function, and we discuss how this is done for each aggregation function in Sections 7, 8 and 9.

4.1 Detecting Falsified Inputs

An attacker performs a falsified input attack when it places an non-legitimate, fabricated data value in one of the leaves of its commitment hash tree. Clearly, if the attacker was allowed to do this without bound, it could arbitrarily affect the result of the aggregation function.

The algorithm detects the presence of falsified inputs by requesting random leaves of the commitment hash tree. To make a request, the home server chooses a leaf position in the Merkle hash tree at random, and prompts the aggregator to return the specified leaf as well as its path verification information. Since the position and contents of each leaf is fixed by the structure of the Merkle hash tree, the aggregator is unable to hide the presence of a fabricated data value if it happens to be randomly requested. Since each leaf in the Merkle hash tree contains an authentication code ($\mathrm{MAC}_{K_{is}}(ID_i \| a_i \| N)$) that the home server can verify, the aggregator is also unable to fabricate illegal values that appear to be from legitimate sensor nodes in the network.

The malicious aggregator is thus only able to undetectably falsify values if the home server happens to not request any falsified leaves. By requesting sufficiently many random samples of the leaves of the commitment hash tree, we can bound the likelihood of this event.

Theorem 4.1. *Suppose the home server requests $\frac{1}{\varepsilon} \ln \frac{1}{\delta}$ samples. Then, if the aggregator has falsified at least ε data values, the probability of detecting the falsification is at least $1-\delta$.*

Proof. If there are ε falsified leaves in the commitment hash tree, then each random request has probability $(1-\varepsilon)$ of completing successfully. The probability that all requests complete successfully is $((1-\varepsilon)^{1/\varepsilon})^{\ln\frac{1}{\delta}} < (1/e)^{\ln\frac{1}{\delta}} = \delta$. Hence, the probability of detection is at least $1-\delta$. □

4.2 Detecting Duplicated IDs

The detection algorithm of Section 4.1 works only for data values with inauthentic MACs. However, an attacker can add illegal leaves with authentic MACs to the commitment hash tree in two ways: first, it may be in control of a compromised node, which it can then use to add multiple illegal data values with authentic MACs; second, it may include the data value of a single legitimate node multiple times. Both these attacks are *duplicated ID* attacks, where a single node ID appears more than once in the input data set.

To detect duplicate IDs, the algorithm requires the aggregator to commit to the sequence of measured values *sorted* according to the *sensor IDs*. Subsequently, we verify that the sequence is properly sorted, using the `Sort-Check-II` spot checker proposed by Ergün *et al.* [14]. The `Sort-Check-II` spot checker performs binary searches for randomly selected elements in the committed sequence. Ergün *et al.* showed that 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 $1-(\varepsilon_1'/4)$ probability of detecting that the committed sequence was not sorted. Hence, $4/\varepsilon_1'$ searches will detect unsortedness with probability at least $1-1/e$, and $\frac{4}{\varepsilon_1'}\ln\frac{1}{\delta}$ searches will reveal unsortedness with probability at least $1-\delta$.

Consider an adversary attempting to include two (possibly identical) readings from the same sensor S_i with identifier ID_i. Without loss of generality call one reading the "original" and the other reading the "duplicate". The adversary may choose to place the duplicate reading in sorted order in the committed sequence, in which case the duplicate reading will be adjacent to the original. Alternatively, the adversary may choose to place the duplicate somewhere else, in which case it will be out of order. In the previous stage we established that the total fraction of out of order elements is at most ε_1'.

To detect the presence of duplicates which appear in sorted order in the committed sequence, the algorithm performs uniform sampling of *pairs* of neighboring elements. If ε_1'' or more fraction of the elements are in fact duplicates which appear in sorted order in the sequence, each sample has probability at least ε_1'' of detecting a duplicate. Hence, $\frac{1}{\varepsilon_1''}\ln\frac{1}{\delta}$ total pairs of samples will detect a duplicate with probability at least δ.

The entire duplicate-checking procedure is given in pseudo-code below.

In practice, the samples for duplicate detection and for detecting inauthentic values should be combined. Specifically, we no longer need to specially request any elements to check for authenticity. Instead, the MAC of every leaf element sampled in the duplicate-checking procedure should be checked for authenticity. When these tests are

```
procedure CheckInput(n, ε):
    /** run Sort-Check-II: **/
    for $i = 1 \ldots (4 \ln \frac{1}{\delta})/\varepsilon_1'$ do
        pick $j \in_R \{1 \ldots n\}$
        request $a_j$
        perform a binary search for $a_j$
        if search failed then
            return REJECT
    /** check for duplicates & multiples: **/
    for $i = 1 \ldots (\ln \frac{1}{\delta})/\varepsilon_1''$ do
        pick $j \in_R \{1 \ldots (n-1)\}$
        request $a_j$, $a_{j+1}$
        if $a_j$, $a_{j+1}$ invalid or stem from the same sensor then
            return REJECT
    return ACCEPT
```

combined, it is clear that the following property holds:

Theorem 4.2. *The duplicate-detecting algorithm requests $O\left((\frac{1}{\varepsilon_1'} + \frac{1}{\varepsilon_1''}) \ln \frac{1}{\delta}\right)$ elements, and ensures that either there are less than a total of $\varepsilon_1 = \varepsilon_1' + \varepsilon_1''$ non-legitimate values in the committed sequence, or the adversary's presence will be detected with probability at least $1 - \delta$.*

Proof. The `Sort-Check-II` spot checker detects cheating with probability $1-\delta$ if at least ε_1' of the elements in the committed sequence are duplicates placed out of node-ID order, or are values with inauthentic MACs. The second phase of duplicate detection detects cheating with probability $1-\delta$ if at least ε_1'' of the elements in committed sequence are duplicates placed in node-ID order or are values with inauthentic MACs. If the adversary has less than $1-\delta$ probability of detection, then it must be that the total number of falsified and duplicated values is less than $\varepsilon_1' + \varepsilon_1''$. □

5 Secure Computation of Min/Max

The problem of finding the minimum (or the maximum) value of the measurements is a fundamental task in monitoring applications, and protocols for solving it are useful not only as stand-alone primitives but also as subprotocols for more complex aggregates. In this section, we describe a secure min-discovery protocol that enables the home server to find the minimum of the values reported by the sensors. Then in Section 8 we show example applications, namely how we use our secure min-discovery protocol as a building block to enable random selection of a node in the network, leader election, and secure counting the number of distinct elements and estimating the network size.

Recall that in our setting some sensors may be corrupted, and a corrupted sensor could always report a forged value which is smaller than the smallest true value, which renders the problem of finding the minimum value meaningless.

Therefore, here we focus on the scenarios where either a corrupted sensor cannot lie about its value (for example, if the value is a deterministic function of a node's ID or keying information), or it is not in the interest of the adversary to report smaller values (for example, if the adversary is trying to hide exceptionally small or large values, such as for a triggered burglar or fire alarm).

One approach to preventing cheating is to use the method for computing quantiles as described in Section 7, to ensure that the value the aggregator reported is among the εn smallest with high probability. Below we propose a new protocol, *FindMin*, for finding the minimum value, which achieves a better bound. Assuming that an uncorrupted sensor node holds the minimum value in the committed values mentioned above, the *FindMin* protocol will enable the home server to find the minimum value in the committed values with high probability.

5.1 *FindMin* for the ID-Knowledge Scenario

The *FindMin* protocol involves having each (legitimate) sensor node perform a peer-to-peer *multihop* broadcast of its value (and, associated with its value, its own ID) to all the nodes in the area of interest. Assuming that the legitimate sensor nodes form a connected component, eventually the true minimum value v_{min} and the ID of the originator of the minimum ID_{min} will reach each of the legitimate nodes (assume ties for the minimum value are broken by ID). Each of the legitimate nodes will then construct an authenticated message indicating it believes the minimum is v_{min} and it comes from node ID_{min}. The structure of the messages is identical to that described in Section 4, with $a_i = v_{min} \| ID_{min}$. Each of these messages are relayed to the aggregator, which then commits to these messages in the method described in Section 4. The aggregator reports v_{min} and the commitment to the home server.

The home server must now verify that v_{min} was indeed the minimum value. Assuming that at most κ fraction of the nodes in the area of interest are malicious, the home server issues $O(\log \delta / \log \kappa)$ requests for uniformly random selected node IDs from the known list of IDs in the area of interest (recall that the home server has knowledge of these IDs by the ID-knowledge assumption). If each requested node ID is present in the commitment of the aggregator, and all agree on v_{min} and ID_{min}, the home server then requests an authenticated message from the node ID_{min} verifying that its value was indeed v_{min}. If so, then it accepts v_{min}.

Formally, we have the following theorem.

Theorem 5.1. *Assuming that no more than κ fraction of the sensors are corrupted, that the minimum value in the committed values is from an uncorrupted sensor, and that all uncorrupted sensors form a connected component, procedure* FindMin *requests $O(\frac{\log \delta}{\log \kappa})$ elements and satisfies:*

(1) If all the sensors and the aggregator follow the protocol then the home server ACCEPT*s the result, which is equal to the minimum of the values committed to initially.*

(2) If the value reported by the aggregator is not equal to the minimum of the values committed to initially, then the home server REJECT*s with probability at least* $1-\delta$.

Proof. (sketch) The assumption that the uncorrupted sensors form a connected component implies that the adversary cannot stop the propagation of the minimum value in this component. If the aggregator tries to cheat, each random sample of a node will with probability at least $1-\kappa$ hit an uncorrupted sensor, and detect cheating. Hence, $\frac{\log\delta}{\log\kappa}$ such samples suffice to detect cheating with probability at least $1-\delta$. □

5.2 *FindMin* for the Alarm Channel Scenario

If an alarm channel is available, the home server simply broadcasts the minimum value reported by the aggregator. Any sensor node that has a smaller value than the minimum raises an alarm using the reliable communication channel with the home server. If the home server does not receive an alarm within a given latency bound, then it assumes that no sensor node has raised an alarm and accepts the proposed minimum. Unlike the algorithm described in Section 5.1, this algorithm detects any forged result with certainty and involves no probes (or, in fact, any form of cryptography) at all.

5.3 Applications: Random Selection of a Node & Leader Election

A basic tool needed in many applications is a method for a selection of a sensor node at random. Note that even if the home server has a list of *ID*s of the sensor nodes in the network, a mere selection of a node from the list uniformly at random does not solve the problem — the aggregator might be corrupted and deny contact to the picked sensor by claiming that the picked node does not respond. In such a case the home server has no way of deciding what is faulty, the sensor or the aggregator.

We propose a new mechanism which enables the home server to perform a random sampling in the sensor network and does not suffer from the above drawback. The main idea of the proposed *RandomSample* procedure is as follows. The home server picks a random hash function h and sends it to the aggregator. The aggregator is then supposed to broadcast h with the sampling request. Each sensor node then computes the hash value of its *ID*. Then the whole network performs a MIN-discovery protocol to discover the node with the smallest hash value. If a corrupted sensor node happens to have the smallest hash value, it could choose not to report its own value. However, a corrupted sensor cannot report any fake value, since the value to be reported by each sensor is uniquely determined by h and the *ID* of the sensor. Moreover, if the smallest hash value is computed by an uncorrupted sensor node, the attacker cannot stop the uncorrupted sensor node to become the winner and be discovered and reported back to the home server. Thus, because any uncorrupted sensor node has equal probability of computing the smallest hash value, this method enables the home server to sample uniformly at random from the uncorrupted sensor nodes.

Corollary 5.2. *Under assumptions of Theorem 5.1, with h denoting a function picked uniformly at random from a family of pairwise independent hash functions, the procedure* RandomSample(h) *satisfies*

(1) If all the sensors and the aggregator follow the protocol then the home server ACCEPTs.

(2) If the home server ACCEPTs*, then with probability at least $(1-\varepsilon)$, for every honest sensor node S, the probability of picking S as the sample is within $1/n$ and $1/(n(1-\varepsilon))$.*

This random sampling technique has many applications. In particular, it can be used to pick a leader among the sensors, and as shown below, it is very useful for counting distinct elements and computing the network size.

6 The COUNT Aggregate

The COUNT aggregate computes the cardinality of a subset of the sensor nodes. Let the subset to be counted be called the *counted set*. The exact membership of the counted set is unknown to the querier, but may be defined using a logical predicate. For example, we may wish to count the number of sensors which are registering a reading above 20° C. We can visualize the COUNT aggregate as performing a summation aggregation operation over the data values of either 1 (fulfills the predicate) or 0 (does not fulfill the predicate). As a special case, the COUNT aggregate may be used to determine the total number of data nodes in the system.

The algorithm proceeds as follows. First, the sensor nodes that fulfill the predicate send their authenticated values (i.e., $a_i = 1$) to the aggregator. The aggregator then commits to the received values using the preprocessing step in Section 4. The aggregator also computes the count aggregate $\overline{a}$ and returns both the commitment and the computed aggregate to the querier.

The querier now has the count aggregate $\overline{a}$ and a commitment hash value which should be the root of a Merkle hash tree of height $\lceil \log \overline{a} \rceil$ with exactly $\overline{a}$ leaves each of which is an authenticated message from some node S_i indicating its data value of $a_i = 1$. The querier must now verify the authenticity of the result $\overline{a}$. Let the correct count be a. There are two kinds of attacks to consider: either $\overline{a} > a$ (inflation of the reported result) or $\overline{a} < a$ (deflation of the reported result).

6.1 Resisting Result Inflation

To verify that the result $\overline{a}$ has not been inflated, the querier needs to check that each one of the $\overline{a}$ leaves of the commitment hash tree consists of an authentic message from a distinct node S_i. The preprocessing step of Section 4 ensures that less than a small fraction ε of the leaf nodes may be duplicate or falsified messages, otherwise the adversary is detected with probability $1-\delta$. Hence, no additional work needs to be done to prevent result inflation.

6.2 Resisting Result Deflation - ID-knowledge Scenario

The adversary may attempt to reduce the reported count by ignoring the reported messages of some of the sensor nodes. The home server can counter this by probing the commitment structure to verify that all sensor nodes that should be counted are indeed counted. To select a node for probing, the home server uses the random node selection algorithm of Section 5.3 to randomly choose a node from among the counted set. The home server then requests the aggregator to prove the selected node's presence in the committed set.

Theorem 6.1. *If the adversary reports $\overline{a} < (1-\varepsilon)a$ where a is the true count, then $O(\frac{1}{\varepsilon}\ln\frac{1}{\delta})$ requests suffice to detect this with probability at least $1-\delta$.*

Proof. (Theorem 6.1) If $\overline{a} < (1-\varepsilon)a$, then at least ε fraction of the counted set is omitted (not present in the committed set). Each request fails to detect the manipulation only if an omitted element was not requested; this happens with probability $1-\varepsilon$. By an argument as in the proof of Theorem 4.1, the probability of detection is at least $1-\delta$. □

Each application of *FindMin* uses $O(\log 1/\delta)$ requests, hence the overall request complexity is $O(\frac{1}{\varepsilon}\log^2\frac{1}{\delta})$.

6.3 Resisting Result Deflation - Alarm Channel Scenario

The algorithm of Section 6.2 requires multiple rounds of *FindMin*, which involve multiple rounds of broadcast and many requests per application of *FindMin*. If an alarm channel exists, we have a more efficient algorithm which does not involve significant broadcast overhead, and only requires $O(\frac{1}{\varepsilon}\log\frac{1}{\delta})$ requests. The algorithm uses a probabilistic "roll-call" mechanism where data nodes randomly verify that they are indeed included in the aggregate.

The home server authentically broadcasts a check-request message containing the computed COUNT aggregate $\overline{a}$, and the commitment of the aggregator, to every sensor node. Each sensor node then randomly decides whether to check that it was included in the commitment. Each sensor node S_i, which belongs to the counted subset, repeats the following trial $O(1/\varepsilon)$ times: it flips a coin with probability $1/\overline{a}$ of success, where $\overline{a}$ is the authenticated COUNT aggregate sent to it by the base station in the query. If any of these trials succeed, the sensor node then performs a *probe*. The probe asks the aggregator to prove the sensor node's presence in the commitment tree. The sensor node is able to verify the aggregator's response because it knows the commitment hash value that was sent to the home server. If the aggregator does not respond satisfactorily, the sensor node may then use the reliable communication channel to raise an alarm to the home server.

```
**procedure** *CountDeflateCheck(n, $\overline{a}$, ε)*:
    **receive** $\overline{a}$ and commitment value $h$
    $c \leftarrow 0$
    **for** $i = 1 \ldots \left\lceil \frac{\ln 1/\delta}{\varepsilon - \frac{1}{2}\varepsilon^2} \right\rceil$ **do**
        **if random**$([0,1)) < 1/\overline{a}$ **then** $c \leftarrow 1$
    **if** $c = 1$ **then**
        **challenge** aggregator to verify this sensor's inclusion in commitment
        **if** challenge not satisfied **then**
            **raise alarm**
```

For simplicity, we analyze a variant of the participation-checking algorithm where, instead of having each node immediately perform all the coin flips, we consider the case where we have several *rounds*, and in each round, every node flips a coin with probability $1/\overline{a}$ of success, where $\overline{a}$ is the authenticated COUNT aggregate sent to it by the base station in the query. Clearly, the original single-round algorithm is equivalent to this multi-round algorithm, except that in the single-round algorithm, nodes do not wait for a round to complete before making their next coin flip, and do not repeat their probes if they receive more than one successful coin flip.

Lemma 6.2. *If an adversary reduces the aggregation result by a factor of* $\varepsilon' \geq \varepsilon$*, then the probability of it being undetected in a single round of coin flips is less than* $1 - (\varepsilon - \frac{1}{2}\varepsilon^2)$.

Proof. Suppose an adversary reports a low result $\overline{a} < (1 - \varepsilon')a$ where a is the true COUNT aggregate. Since the number of leaves (and the shape) of the commitment hash tree is fully determined by the reported COUNT aggregate value, at most $1 - \varepsilon'$ of the nodes which should be in the counted subset can actually have data values as leaves in the commitment tree. This implies that at least ε' fraction of the sensor nodes have no leaves in the commitment tree.

The adversary's tampering is detected whenever a sensor node that has no leaf in the commitment tree performs a probe, since the aggregator will be unable to produce the leaf vertex that corresponds to the sensor node. Let a be the true count, and let $m = \varepsilon' a$ be the number of nodes without leaves in the commitment tree.

$$\begin{aligned}
&\Pr[\text{Adversary is undetected in any single round}] = \\
&= (1-\frac{1}{\overline{a}})^m \\
&< (1-\frac{1}{a})^m \\
&< 1-\frac{m}{a}+\binom{m}{2}\frac{1}{a^2} \quad (\text{Since } \tfrac{m}{a} < 1\text{, the absolute value of each succeeding term decreases}) \\
&= 1-\varepsilon'+(\frac{1}{2}-\frac{1}{2m})\varepsilon'^2 \\
&< 1-(\varepsilon'-\frac{1}{2}\varepsilon'^2) \\
&\leq 1-(\varepsilon-\frac{1}{2}\varepsilon^2) \quad (\text{Since } x-\tfrac{1}{2}x^2 \text{ is monotonically increasing for } 0 \leq x \leq 1)
\end{aligned}$$

Theorem 6.3. *Let each sensor node in the counted subset perform* $\left\lceil \frac{\ln 1/\delta}{\varepsilon-\frac{1}{2}\varepsilon^2} \right\rceil$ *coin tosses. Then, if an adversary reduces the aggregation result by a factor of* $\varepsilon' \geq \varepsilon$*, this cheating will be detected with probability at least* $1-\delta$*, and the expected number of requests issued by the base station when the adversary is absent is* $O(\frac{1}{\varepsilon}\log\frac{1}{\delta})$*.*

Proof. From Lemma 6.2, the probability that the adversary remains undetected in $\left\lceil \frac{1}{\varepsilon-\frac{1}{2}\varepsilon^2} \right\rceil$ rounds is less than $(1-(\varepsilon-\frac{1}{2}\varepsilon^2))^{\frac{1}{\varepsilon-\frac{1}{2}\varepsilon^2}} < \frac{1}{e}$. Hence, if we repeat this $\ln\frac{1}{\delta}$ times (for a total of $\left\lceil (\ln\frac{1}{\delta})\frac{1}{\varepsilon-\frac{1}{2}\varepsilon^2} \right\rceil$ rounds), the probability of non-detection is less than δ.

By linearity of expectations, the expected number of probe requests in each round is $\sum_i^n \frac{1}{\overline{a}} = \frac{1}{1-\varepsilon'}$. Hence, the expected number of probe requests in the multi-round protocol is $\frac{1}{1-\varepsilon'}\left\lceil (\ln\frac{1}{\delta})\frac{1}{\varepsilon(1-\frac{1}{2}\varepsilon)} \right\rceil$. When the adversary is not present, $\overline{a} = a = n$, and so the expected number of probes in the network is $\frac{1}{n}\cdot n\cdot\left\lceil (\ln\frac{1}{\delta})\frac{1}{\varepsilon(1-\frac{1}{2}\varepsilon)} \right\rceil = O(\frac{1}{\varepsilon}\log\frac{1}{\delta})$. □

Since the single-round protocol makes no more probe requests than the multi-round protocol, the number of probes in the actual protocol is also $O(\frac{1}{\varepsilon}\log\frac{1}{\delta})$. When the adversary is in the network, a malicious aggregator can easily increase the probe complexity (e.g. by reporting $\overline{a} = 0$, causing every node to initiate a probe and fail), but this would immediately reveal the adversary's presence. Causing every node in the network to use the expensive alarm channel may be an effective denial-of-service (DoS) attack; however DoS is outside the scope of this paper. An attacker in complete control of the aggregator is capable of far more insidious damage than simple DoS if it can cause the home server to accept false results.

7 Computing The Median

In this section we study the problem of computing the median of the measured values. Without loss of generality we assume that all values a_i are distinct—if they are not distinct, we can run the protocol on the (distinct) pairs (a_i, ID_i), where ID_i is a unique ID of the i-th sensor.

The most straightforward approach is to sample the measurements and use the median of the sample as an estimate of the true median. Bar-Yossef *et al.* [4] showed that $\Omega(1/\varepsilon^2)$ samples are necessary to achieve a ε-approximation with high probability [35]. We show how to achieve more efficient solutions with only $O(\log n/\varepsilon)$ element requests. Note that here we assume that the user knows the approximate size of the sensor network. This can be achieved, for example, with the methods presented in Section 6.

7.1 Algorithm for MEDIAN

The aggregate-commit-prove approach for MEDIAN builds on common steps described in Section 4 with some additional steps to verify the integrity of the proposed median.

The common steps in Section 4 describe how the aggregator commits to a sequence of node readings sorted in order of the node IDs. For MEDIAN, we additionally require the aggregator to provide a commitment to a sequence which is *median-separated*, or pivoted around the median element. Specifically, the median element is in position $n/2$, while every reading that is smaller than the median lies to its left in the commitment tree, and every reading that is larger than the median lies to the right of the middle. To verify that the two committed sequences are the same, the home server requests $\frac{1}{\varepsilon_2} \ln \frac{1}{\delta}$ random elements from each sequence and verifies that these elements are present in the other sequence. This ensures (with probability $1-\delta$) that the two sequences have at least a factor of $1-\varepsilon_2$ overlap; the proof structure is essentially identical to the proof of Theorem 4.1.

Subsequently, the home server $\mathcal{B}$ obtains an alleged median a_{med} and verifies its correctness in an interactive proof. Specifically, to check that a_{med} is (close to) the median of the committed sequence, $\mathcal{B}$ picks elements from random positions in the committed median-separated sequence and checks that elements picked from the left half of the sequence are less than the reported median, and elements from the right half are greater than the median.

A pseudo-code description of this median-checking test is given below.

Theorem 7.1. *Procedure* MedianCheck(n, a_{med}, ε_3) *requests* $1/\varepsilon_3$ *elements* a_i*, and satisfies:*

(1) if the measurements sequence is median-separated and a_{med} *is equal to* $a_{n/2}$*, then the result is "*ACCEPT*"*

(2) if a_{med} *is not present in the sequence, or its position* p *in the sorted sequence satisfies*

$$|p - n/2| > \varepsilon_3 n ,$$

```
procedure MedianCheck($n$, $a_{\text{med}}$, $\varepsilon_3$):
    request $a_{n/2}$
    if $a_{n/2} \neq a_{\text{med}}$ then
        return REJECT
    for $i = 1 \dots \frac{1}{\varepsilon_3}$ do
        pick $j \in_R \{1 \dots n\} \setminus \{n/2\}$
        request $a_j$ from median-separated sequence
        if $j < n/2$ and $a_j > a_{\text{med}}$ then
            return REJECT
        if $j > n/2$ and $a_j < a_{\text{med}}$ then
            return REJECT
    return ACCEPT
```

then with probability at least $1 - 1/e > 1/2$ *the result is* "REJECT"

Proof. The number of requests and property (1) follow immediately. For property (2), notice that if $|p - n/2| > \varepsilon_3 n$, then there are at least $\varepsilon_3 n$ values of j, which yield REJECT. Hence with probability at most $(1-\varepsilon_3)^{1/\varepsilon_3} \leq 1/e$ the **for**-loop completes without rejection, i.e., the algorithm rejects with probability at least $1 - 1/e$. □

Theorem 7.1 implies that by requesting in total $\frac{1}{\varepsilon_3} \ln \frac{1}{\delta}$ elements, we can provide a $1-\delta$ probability of detecting if the reported value is not an ε_3-approximation of the median in the committed *median-separated* sequence.

When all the stages are combined, we have the following analysis. We wish to derive an (ε, δ)-secure MEDIAN; in such an algorithm, an attacker is either limited to reporting a result that is ε-close to the true value or it must be detected with probability $1-\delta$. We prove that if the attacker does not encounter the $1-\delta$ probability of detection at any stage, then its reported result must be ε-close for some ε. Suppose the attacker has a greater than δ probability of evading detection. Then the reported median must be a ε_3-approximation to the median for the median-separated sequence. The median-separated sequence is ε_2-close to the sequence sorted by node ID, which has at most ε_1 elements which were duplicated, falsified, or dropped. Setting $\varepsilon = \varepsilon_1 + \varepsilon_2 + \varepsilon_3$, it is clear that the combined algorithm for MEDIAN is a (ε, δ)-secure aggregation algorithm.

NOTE: Clearly, using the same techniques we can compute with low communication complexity not only the median, but also arbitrary quantiles.

7.2 MEDIAN for the Alarm Channel Scenario

In the alarm channel scenario, the algorithm for COUNT is significantly more efficient than in the ID-knowledge scenario (cf. Section 6). Hence, an alternative algorithm for MEDIAN is to reduce it to COUNT. The aggregator collects the data values and computes the median, which it then forwards to the home server. Note that no commitments are necessary at this stage. After the home server has received the proposed median, it then performs a COUNT of the total number of sensor nodes that have a reading less than the reported median. If the number is not half the total number of nodes queried, then the proposed median is rejected. The fact that this is an (ε, δ)-secure algorithm for MEDIAN follows directly from the property that the algorithm for COUNT is (ε, δ)-secure. This algorithm can also be adapted for arbitrary quantiles.

8 Counting Distinct Elements

COUNT-DISTINCT is the problem of counting the number μ of distinct values in the measurements, i.e., the problem of determining the cardinality of the set of all the measurements.

Ergün *et al.* [15] give a very efficient protocol for proving a lower bound on the size of a set. While it is possible to use their solution in our context, a direct implementation would require application of PIR protocols [7, 27]. The reason for this requirement is the fact that in the protocol proposed by Ergün *et al.* it is essential that the prover does not know the positions of randomly selected elements. The application of PIR significantly increases the communication complexity of the solution — the currently most efficient PIR protocol [27] imposes an additional factor of $\Omega(\log^2 n)$ per access of a single bit.

We propose two different protocols for estimating the number of distinct elements. Our solutions are based on algorithms for space-efficient approximation of the number of distinct elements in a data stream [5, 17, 1], and on a novel technique for random selection of the nodes of the network (cf. Section 5.3).

8.1 Method I: Counting Distinct Elements by Distributed MIN-Computation

The described approach to the estimation of the number of distinct elements in a data stream can be viewed as a process of finding a minimum, in which the same computation is performed for each element: compute the hash-value and save it if it is smaller than the current minimum. This observation immediately suggests that the algorithm for the data stream [1] can be easily implemented in a distributed way, and the estimation of the number of distinct elements can be reduced to the problem of finding the minimum. First the home station picks at random a hash function h from an appropriate family, and through the aggregator announces h to each sensor. Each sensor locally computes the hash value of its element, and then participates in a protocol for finding the minimum hash value (cf. Section 5).

If we want to improve the accuracy, we can implement the algorithm of Bar-Yossef et al. [5], by keeping t smallest hash values instead of just the single minimum. However, this improvement comes at a cost of higher communication complexity.

8.2 Method II: Proving Bounds on the Number of Distinct Elements

The above method for counting distinct elements depends on the random selection procedure from Section 5.3. In some applications the communication overhead of this selection procedure may be too high. In this section we present alternative, more efficient methods for estimation of the number of distinct elements.

The method requires the aggregator to show evidence that the proposed distinct-count is not lower than a certain lower bound, and is not higher than a certain upper bound. Since we require that the lower and upper bounds are closely matched, this implies that the aggregator is not able to either inflate or deflate the count.

8.2.1 A Lower Bound on the Number of Distinct Elements

At the beginning of the protocol for showing a lower bound on the number of distinct elements the aggregator $\mathcal{A}$ commits to the values reported by the sensors using the hash-tree construction of Section 4, and reports the commitment (and the proposed COUNT-DISTINCT aggregate $\overline{\mu}$) to the home server B. $\mathcal{A}$ then runs an algorithm for counting distinct elements in a stream by Bar-Yossef *et al.*[5], using hash functions specified by the home server $\mathcal{B}$.

Let $H = \{h|\ h : [m] \rightarrow [M]\}$, where $M = m^3$, be a family of pairwise independent hash functions [37], such that any function $h \in H$ has a short description. After $\mathcal{A}$ commits to the input, $\mathcal{B}$ computes an estimate for a lower bound on the true aggregate result μ as follows. $\mathcal{B}$ picks at random a hash function h from the family H and sends it to $\mathcal{A}$. Then $\mathcal{A}$ computes $h(a_i)$ for all $i = 1 \dots n$, and sends back to $\mathcal{B}$ t elements (for appropriately chosen t), on which h evaluates to the t smallest values. Then $\mathcal{B}$ checks the correctness of the received elements and computes an estimate of μ as $\mu' = tM/v$, where v is the t-th smallest value to which h maps the received elements.

Bar-Yossef *et al.* [5] show that with high probability μ' is a good approximation of the number of distinct elements. We can further amplify the accuracy by repeating the protocol ℓ times and estimating μ with the median of the ℓ resulting estimators $\mu'_1, \dots, \mu'_\ell$. A pseudo-code description of the entire protocol is given below.

```
procedure DistinctLowerBound(n, m, ε, ℓ):
    $t := \lceil 96/\varepsilon'^2 \rceil$
    $M := m^3$
    for $j = 1 \dots \ell$ do
        pick $h_j \in_R H$
        send description of $h_j$ to $\mathcal{A}$
        request $t$ elements $a_i$ on which $h_j$ evaluates to the $t$ smallest values
        let $v$ be the $t$-th smallest such value
        set $\mu'_j = tM/v$
    return $\mu' = \text{median}(\mu'_1, \dots, \mu'_\ell)$
```

Theorem 8.1. *Procedure* DistinctLowerBound(n, m, ε', ℓ) *requests* $O(\ell \cdot 1/\varepsilon'^2)$ *elements* a_i *and returns a value* μ' *which satisfies the following:*

- $\mu' > (1+\varepsilon')\mu$ *with probability less than* $(1/6)^{\ell/2}$.

- $\mu' < (1-\varepsilon')\mu$ *with probability less than* $(1/6)^{\ell/2}$ *if the attacker is not present.*

Proof. First consider the special case when $\ell = 1$. Bar-Yossef *et al.* [5] bound the probability that the estimate μ'_1 is significantly larger than μ: $\Pr[\mu'_1 > (1+\varepsilon)\mu] < \frac{1}{6}$. For $\ell > 1$, the median of ℓ values $\mu'_1, \ldots, \mu'_\ell$ exceeds the bound $(1+\varepsilon')\mu$ if at least half of the estimates exceed the bound, hence $\Pr[\text{median}(y_1, \ldots, y_\ell) > (1+\varepsilon)\mu] < \left(\frac{1}{6}\right)^{l/2}$. A symmetric argument bounds the probability of μ' being significantly lower than μ, hence the result follows. $\square$

A naive approach would be to use μ' as the reported aggregate value. However, the estimator is probabilistic, which means that even if the adversary is absent, there exists a chance that it may return an estimate that is far from the true count. To ensure complete accuracy in the absence of an attacker, we use the verified estimator value μ' as a proof that there are (probably) at least μ' distinct values in the set, and compare it against the precise proposed distinct count $\overline{\mu}$.

Let $\varepsilon' = \varepsilon/2$. If $\mu' \geq (1-\varepsilon')\overline{\mu}$ then the home server accepts the proposed result $\overline{\mu}$ as consistent with the verified estimate μ'. However, if $\mu' < (1-\varepsilon')\overline{\mu}$ then the reported aggregate $\overline{\mu}$ is suspiciously high (i.e. the proven lower bound μ' appears too weak). To determine if this discrepancy was due to a false positive (e.g. the statistical estimator just happened to get a high error), or if it was due to malicious activity on the aggregator's part, the home server requests a *full dump* of all the data values committed to by the aggregator so that its correctness can be checked with certainty. Since this request for all n values is clearly expensive, we choose l such that the probability of such a request is low. Specifically, we add the condition that $l > \frac{2}{\log 6} \log n$. This ensures that, if the attacker is not present, the probability that the estimator raised a false positive is less than $1/n$, hence the average cost of the full dump of data values is constant. We also need to bound the probability of a false negative, which happens if $\mu' > (1+\varepsilon')\mu$, thus enabling the adversary to claim a value for $\overline{\mu}$ which is up to $\frac{1+\varepsilon'}{1-\varepsilon'} \approx 1+2\varepsilon' = 1+\varepsilon$ factor larger than μ. By enforcing the requirement that $l > \frac{2}{\log 6} \log \frac{1}{\delta}$, we provide the $1-\delta$ probability of catching the attacker in such a case. The final request complexity is $O(\frac{1}{\varepsilon^2}(\log \frac{1}{\delta} + \log n))$.

Note that in the protocol $\mathcal{B}$ has no means to check that $\mathcal{A}$ has evaluated the hash function on all the elements, and that the reported elements evaluate indeed to the t smallest values. A malicious aggregator $\mathcal{A}$ can omit some elements, or report elements which evaluate to larger values. However, such cheating results in a smaller estimate μ', which does not help the cheater pass the requirement that $\mu' \geq (1-\varepsilon)\overline{\mu}$ unless it also reports a correspondingly low $\overline{\mu}$. In the next section we show how a low $\overline{\mu}$ can be detected.

8.2.2 An Upper Bound on the Number of Distinct Elements

Consider the following sampling-based test: first, $\mathcal{A}$ commits to the multi-set S of all the elements. Additionally, $\mathcal{A}$ commits to a subset S' containing all *distinct* elements (without repetitions). $\mathcal{A}$ reports $\mu' = |S'|$ to $\mathcal{B}$, and $\mathcal{B}$ verifies

$\mathcal{A}$'s claim by checking that all the distinct elements from S are present in S'. In other words, the test checks that μ' is an upper bound on μ.

Now, depending on the assumption scenario, and on the ratio of μ' to the total size of multi-set S, two different approaches can be used for random sampling from S.

High Number of Distinct Elements Suppose the number of distinct elements is a significant fraction of the number of all the elements, i.e., $\mu \geq n/c$ for some $c \geq 1$. Then for the ID-knowledge scenario, a simple sampling through the aggregator is sufficient. The home server picks a node at random, with uniform distribution over all nodes. It requests the chosen node's reading from S, and checks to see if the reading is also in S'.

This technique can also be adapted for the alarm-channel scenario by applying the techniques of Section 6 to ensure that S is a good representation of the data values of the sensor nodes. Specifically, the techniques of Section 6 will ensure that at most ε_1 of the elements in S are illegal, and furthermore at most ε_1 of the legitimate nodes are not represented in S. Then, the home server may select a random reading in S, and check to see if the reading is also in S', without having any knowledge of the set of node IDs being queried.

Note that this node-sampling procedure is quite different from the one described in Section 5.3, and in particular is much more efficient since the home server does not require the FINDMIN primitive. A pseudo-code description of the entire protocol based on this simple sampling is given below.

```
procedure HighDistinctUpperBound($n$, $\varepsilon_2$, $\ell$):
    for $i = 1 \ldots \ell$ do
        pick $j \in_R \{1 \ldots n\}$
        request $a_j$
        request an element from $S'$ equal to $a_j$
        if any of the requests failed then
            return REJECT
    return ACCEPT
```

Theorem 8.2. *Procedure* HighDistinctUpperBound(n, ε_2, ℓ) *requests ℓ elements and satisfies:*

*(1) if S' contains all the distinct elements from the input the result is "*ACCEPT*".*

*(2) if $\mu' < (1-\varepsilon_2)\mu$ then with probability at least $1 - e^{-\ell\varepsilon_2/c}$ the result is "*REJECT*", assuming that $\mu \geq n/c$ for some constant $c \geq 1$.*

Proof. Claim (1) is obvious: if all distinct elements are present in S', the prover $\mathcal{A}$ will always be able to return requested elements. For claim (2) notice that if $\mu' < (1-\varepsilon_2)\mu$, then there are at least $\mu\varepsilon_2$ elements in S which detect cheating and lead to "REJECT". By assumption $\mu\varepsilon_2 \geq \varepsilon_2 n/c$ for some $c \geq 1$, so the probability that a single sample detects cheating is at least ε_2/c. Therefore, the probability of returning "ACCEPT" after ℓ samples is at most $(1-\varepsilon_2/c)^\ell \leq e^{-\ell\varepsilon_2/c}$, which implies the claim. $\square$

Theorem 8.2 implies that by taking $\ell = c/\varepsilon_2$ we can detect cheating with constant probability $1-1/e$. Therefore, if with such value of ℓ we repeat the test $\ln 1/\delta$ times, we get confidence at least $1-\delta$. Setting $\varepsilon = \varepsilon_2$ concludes the analysis for the ID-knowledge case. For the alarm-channel case, combining this ε_2 bound with the $2\varepsilon_1$ bound on the adversary's ability to drop legitimate values from S or add false values to S (cf. Section 4), we get an (ε,δ)-secure algorithm for COUNT-DISTINCT if we set $\varepsilon = 2\varepsilon_1 + \varepsilon_2$.

Note that this method yields a significantly better estimate than the approximation by sampling with a "trivial aggregator" $\mathcal{A}$, which only forwards the measurements collected from the sensors selected by $\mathcal{B}$ [10, 4].

Low Number of Distinct Elements. When the number of distinct elements is low in comparison to the total size of S the simple sampling will not give us the desired bounds, because the omitted elements not reported in S' could be infrequent and so very hard to find by sampling. However, a small modification of the random sampling protocol from Section 5.3 can fix this problem: each node reports the hash value based on the value measured by the node, not on its *ID*. This results in a distribution uniform on *distinct values*, not uniform on *nodes*. In other words, different nodes with the same measured value will report the same hash values, and by the properties of the hash function, each measured value will be equally likely to be the minimum. The pseudo-code of this sampling approach is given below.

```
procedure LowDistinctUpperBound($\varepsilon_2$, $\ell$):
    for $i = 1 \ldots \ell$ do
        apply RandomSample on measured values
        request an element from $S'$ equal to the sampled value
        if any of the requests failed then
            return REJECT
    return ACCEPT
```

The following theorem can be proved analogously to the Theorem 8.2.

Theorem 8.3. *Procedure* LowDistinctUpperBound(ε_2, ℓ) *requests ℓ elements and satisfies:*

*(1) if S' contains all the distinct elements from the input the result is "*ACCEPT*".*

*(2) if $\mu' < (1-\varepsilon_2)\mu$ then with probability at least $1-e^{-\ell\varepsilon_2}$ the result is "*REJECT*".*

9 Computing the Average

In this section, we describe how to efficiently and securely compute the average of sensor data. We describe three different protocols with different efficiency tradeoffs for this purpose.

9.1 Computing AVERAGE by Counting Frequencies

In this subsection, we describe how to compute the average by counting frequencies of the sensor values. We first describe a special case and then show that the special case can be extended to the general case.

In the special case when $m \ll n$, or more generally when the range of the sensor values is small in comparison with the size of the sensor network (e.g., poly-logarithmic in n), we can estimate the average quite accurately by taking a small sample of all the elements and returning the average of the sample. However, the number of the samples needed to assure with constant probability a good estimate for the average depends on the the underlying distribution of the values, and in general is lower-bounded by $\Omega(1/\varepsilon^2)$, where $m\varepsilon$ is the desired additive error bound [8, 4].

As in the case of the computation of the median, the naive sampling only minimally uses capabilities of the aggregator. Below we present an alternative method for computing the average, in which the aggregator is more involved. As we shall see, this approach, which we call *AverageByFrequency*, is in some cases significantly more efficient.

We assume that the home server knows how many sensor nodes contribute data readings to the AVERAGE computation: this can be computed using the COUNT primitive (cf. Section 6). Now the aggregator $\mathcal{A}$ collects all the (value, *ID*)-pairs (a_i, ID_i) and commits to them in the manner described in Section 4. Then $\mathcal{A}$ computes an average $\bar{a}$ and reports it to $\mathcal{B}$. In order to prove the correctness of $\bar{a}$, $\mathcal{A}$ sorts all the pairs using the value as the main sorting key and the node *ID* as a secondary sorting key, and commits also to the sorted sequence. Then $\mathcal{B}$ tests whether the two committed sequences contain the same elements by requesting $\frac{1}{\varepsilon_2}\ln\frac{1}{\delta}$ samples (this is identical to the process for MEDIAN in Section 7). If the test completes successfully, $\mathcal{A}$ sends to $\mathcal{B}$ the occurrence-counts for each value $1 \ldots m$. $\mathcal{B}$ verifies the correctness of the counts by deriving from them the positions in the committed sorted sequence, and checks that the values in the committed sorted sequence are well sorted using the method `Sort-Check-II` spot checker [14] (cf. Section 4). If this check also succeeds, $\mathcal{B}$ computes the average directly from the occurrence-counts, and compares it to $\bar{a}$. Summarizing, we obtain the following theorem.

Theorem 9.1. *Let $a_1, \ldots, a_n$ denote the values committed to by the aggregator. The procedure* AverageByFrequency *requests $O(\log n/\varepsilon_3 + m)$ elements and satisfies:*

(1) If $\bar{a}$ is equal to the average of the values $a_1, \ldots, a_n$, and the aggregator follows the protocol, then $\mathcal{B}$ always ACCEPT*s.*

(2) If $|\bar{a} - \text{avg}(a_1, \ldots, a_n)| > \varepsilon_3 m$, then $\mathcal{B}$ REJECT*s with probability $\geq 3/4$ (for a suitable choice of constant parameters).*

Proof. (sketch) Claim (1) is obvious, since when the aggregator follows the protocol, $\mathcal{B}$ receives the exact counts of each value and so computes the exact average. On the other hand, if the average of the values committed to is at least $\varepsilon_3 m$ away from $\bar{a}$, then at least ε_3 fraction of all values are not in order (not sorted), and so the test will fail. $\square$

Theorem 9.1 implies that, since duplicates and dropped values are limited to a total of $2\varepsilon_1$ of the number of legitimate values by the process in Section 4, by setting $\varepsilon = 2\varepsilon_1 + \varepsilon_2 + \varepsilon_3$, we will have (ε, δ)-secure algorithm.

Note that in many scenarios we have typically $\log n \ll 1/\varepsilon$ — in such cases the procedure *AverageByFrequency* uses significantly fewer samples than random sampling to guarantee the same error bound.

For the general case where m is large, we could split the range $[1, m]$ in intervals of exponential scale where the i-th interval is $[m/2^i, m/2^{i-1}]$. Subsequently, we can perform a protocol similar to the above for each interval and achieve an estimate of the average. This involves $\log m$ invocations of the COUNT primitive. A related approach also with $\log m$ invocations of COUNT is presented in Section 9.3.

9.2 AVERAGE by reduction to COUNT-DISTINCT

We can reduce the problem of computing the average to the problem of determining the number of distinct elements in a set [15]. Without loss of generality, assume that the sensor values are integers. In particular, consider the set $\Psi = \{(i, j) | 1 \leq i \leq n, 1 \leq j \leq a_j\}$ (if $a_j = 0$ then there will be no j such that $(i, j) \in \Psi$). Thus, Ψ contains only distinct elements and the cardinality of Ψ equals to $\sum_{i=1}^{n} a_i$. By using our protocol for counting the number of distinct elements in Section 8, we could obtain a protocol to compute the average. Note that the communication efficiency for the protocol to compute the average is the same as the communication efficiency for the protocol to compute the number of distinct elements.

9.3 AVERAGE by reduction to COUNT

Clearly, if we have the sum of all the sensor readings, and the total number of sensor readings, we can compute the average as the sum divided by the number of readings. We can get an approximation of the number of readings using the COUNT aggregate described in Section 6.

To get an approximation of the SUM aggregate, we reduce SUM to COUNT. Specifically, observe that COUNT is a special case of SUM with the range of allowable data values in $\{0, 1\}$. Since the data readings are integers in $[m]$, we can represent them with $\lceil \log m \rceil$ bits. To compute SUM, we simply call the algorithm for COUNT once for each of the $\lceil \log m \rceil$ bit positions. To compute the sum, the algorithm multiplies the each count by the significance of the bit counted. For example, if 18 nodes had bit 0 set, 23 nodes had bit 1 set, and 10 nodes had bit 2 set, then the sum is $18 \cdot (2^0) + 23 \cdot (2^1) + 10 \cdot (2^2) = 104$. Since, for each bit position, the COUNT algorithm provides a multiplicative ε-approximation for the total number of nodes whose data reading has a 1-bit at that position, the final computation of the resultant SUM must also be a multiplicative ε-approximation.

The resultant algorithm requests $O(\frac{1}{\varepsilon}\log m \log n \log\frac{1}{\delta})$ elements, which is less than the request complexity of the method described in Section 9.2 for typical values of m. Furthermore, it is requires fewer intermediate steps than the algorithm of Section 9.1 and so is more efficient (by some constant factor, but not asymptotically).

10 Forward Secure Authentication

Consider the challenge of securely querying past data. For example, an innocuous event in the past that later became interesting and we may still want to place a query on that event. We could use the same mechanisms we proposed in previous sections to run queries on past data. However, we need to solve some additional security issues to securely query past data. In particular, if a sensor is compromised at a certain time, the attacker should not be able to alter the data collected in the past before the sensor was compromised. We call this property *forward secure authentication*. We propose an efficient mechanism to enable forward secure authentication.

As we described before, each sensor shares a key with the home station. We assume that each sensor node and the home station are loosely time synchronized and the time is divided into constant time intervals. The length of the time interval can be minutes or hours depending on the security requirements. To enable forward secure authentication, each sensor updates its key shared with the home station at the beginning of each time interval using a one-way function and uses the updated key to compute the MAC on the sensing data during that time interval.[2] Thus, even when an attacker compromises the sensor node in a later time interval, because of the property of the one-way function, the attacker is unable to compute the MAC key for the previous time interval, and hence will not be able to alter the sensing data for previous time intervals.

A challenge of this approach is how to efficiently store the past data and authenticator, as well as the challenge that the verifier either needs to compute many one-way functions for deriving the current key of a node or that the verifier needs to store one key per node.

Similar techniques have been used to achieve forward secure encryption [6].

11 Conclusions

It is a challenging task to securely aggregate information in large sensor networks when the aggregators and some sensors may be malicious. We propose the *aggregate-commit-prove* framework for designing secure data aggregation protocols (Section 3). We propose specific protocols within this framework for securely finding the minimum and

[2]A one-way function f is a function that it is easy to compute $f(x)$ given x, but it is difficult to compute a pre-image x such that $f(x) = y$ given y.

maximum values, secure random sampling and leader election (Section 5), securely computing the cardinality of a subset of nodes (Section 6), the median (Section 7), securely estimating the number of distinct elements (Section 8), and securely computing the average of measurements (Section 9). Our protocols require only sublinear communication between the aggregator and the user. We also propose the approach of *forward secure authentication* to ensure that even if an attacker corrupts a sensor node at a point in time, it will not be able to change any previous readings the sensor has recorded locally.

References

[1] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the 28th Symposium on the Theory of Computing*, 1996.

[2] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *Proc. 23rd ACM STOC*, pages 21–32, 1991.

[3] Chandramouli Balasubramanian and J. J. Garcia-Luna-Aceves. Shortest Multipath Routing using Labeled Distances. In *Proceedings of the IEEE International Conference on Mobile Ad hoc and Sensor Systems*, 2004.

[4] Ziv Bar-Yossef, S. Ravi Kumar, and D. Sivakumar. Sampling algorithms: lower bounds and applications. In *Proceedings of the 33rd Symposium on the Theory of Computing*, 2001.

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

[6] Mihir Bellare and Bennet Yee. Forward security in private key cryptography. Report 2001/035, Cryptology ePrint Archive, 2001.

[7] Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In *Advances in Cryptology — EUROCRYPT'99: Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques*, pages 402–414, 1999.

[8] Ran Canetti, Guy Even, and Oded Goldreich. Lower bounds for sampling algorithms for estimating the average. *Information Processing Letters*, 53(1):17–25, 1995.

[9] Haowen Chan, Adrian Perrig, and Dawn Song. Secure hierarchical in-network aggregation for sensor networks. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, 2006.

[10] Moses Charikar, Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. Towards estimation error guarantees for distinct values. In *Proceedings of the 19th Conference on the Principles of Database Systems*, 2000.

[11] Swades De, Chunming Qiao, and Hongyi Wu. Meshed multipath routing with selective forwarding: an efficient strategy in wireless sensor networks. *Comput. Networks*, 43(4):481–497, 2003.

[12] Amol Deshpande, Suman Nath, Phillip B. Gibbons, and Srinivasan Seshan. Cache-and-query for wide area sensor databases. In *Proceedings of the International Conference on the Management of Data*, 2003.

[13] 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 the IEEE Global Telecommunications Conference*, 2003.

[14] Funda Ergün, Sampath Kannan, S. Ravi Kumar, Ronitt Rubinfeld, and Mahesh Viswanathan. Spot-checkers. *Journal of Computer and System Sciences*, 60:717–751, 2000.

[15] Funda Ergün, Ravi Kumar, and Ronitt Rubinfeld. Fast approximate PCPs. In *Proceedings of the 31st Symposium on the Theory of Computing*, 1999.

[16] Deborah Estrin, Ramesh Govindan, John Heidemann, and Satish Kumar. Next century challenges: Scalable coordination in sensor networks. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking, MobiCom 99*, 1999.

[17] Philippe Flajolet and G. Nigel Martin. Probabilistic counting. In *Proceedings of the Symposium on the Foundations of Computer Science*, 1983.

[18] Deepak Ganesan, Ramesh Govindan, Scott Shenker, and Deborah Estrin. Highly-resilient, energy-efficient multipath routing in wireless sensor networks. *SIGMOBILE Mob. Comput. Commun. Rev.*, 5(4):11–25, 2001.

[19] Oded Goldreich. Probabilistic proof systems - a survey. In *Symposium on Theoretical Aspects of Computer Science*, 1997.

[20] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the 17th Symposium on the Theory of Computing*, 1985.

[21] Lingxuan Hu and David Evans. Secure aggregation for wireless networks. In *Workshop on Security and Assurance in Ad hoc Networks*, 2003.

[22] Chalermek Intanagonwiwat, Deborah Estrin, Ramesh Govindan, and John Heidemann. Impact of network density on data aggregation in wireless sensor networks. In *Proceedings of International Conference on Distributed Computing Systems*, 2001.

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

[24] Joseph Kahn, Randy Katz, and Kristofer Pister. Mobile networking for smart dust. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking*, 1999.

[25] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *Proc. 24th ACM STOC*, pages 723–732, 1992.

[26] Sung-Ju Lee and Mario Gerla. Split Multipath Routing with Maximally Disjoint Paths in Ad hoc Networks. In *Proceedings of the IEEE International Conference on Communications*, 2001.

[27] Helger Lipmaa. An oblivious transfer protocol with log-squared communication. In *Proceedings of the 8th Information Security Conference*, 2005.

[28] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. TAG: a Tiny AGgregation service for ad-hoc sensor networks. In *Proceedings of the Fith Annual Symposium on Operating Systems Design and Implementation*, 2002.

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

[30] Ralph C. Merkle. Protocols for public key cryptosystems. In *Proceedings of the IEEE Symposium on Security and Privacy*, 1980.

[31] Ralph C. Merkle. A certified digital signature. In *Proceedings on Advances in Cryptology*, 1989.

[32] Asis Nasipuri and Samir R. Das. On-Demand Multipath Routing for Mobile Ad Hoc Networks. In *Proceedings of the IEEE International Conference on Computer Communication and Networks*, 1999.

[33] Adrian Perrig, Ran Canetti, J.D. Tygar, and Dawn Song. The TESLA broadcast authentication protocol. *RSA CryptoBytes*, 5(Summer), 2002.

[34] Adrian Perrig, Robert Szewczyk, Victor Wen, David Culler, and J.D. Tygar. SPINS: Security protocols for sensor networks. *Wireless Networks Journal (WINET)*, 8(5):521–534, 2002.

[35] Bartosz Przydatek, Dawn Song, and Adrian Perrig. SIA: Secure information aggregation in sensor networks. In *Proceedings of the 1st ACM Conference on Embedded Networked Sensor Systems*, 2003.

[36] David Wagner. Resilient aggregation in sensor networks. In *Proceedings of the ACM Workshop on Security of Ad Hoc and Sensor Networks*, 2004.

[37] Mark N. Wegman and J. Lawrence Carter. New hash functions and their use in authentication and set equality. *Journal on Computer and System Sciences*, 22:265–279, 1981.

[38] Yi Yang, Xinran Wang, Sencun Zhu, and Guohong Cao. SDAP: A secure hop-by-hop data aggregation protocol for sensor networks. In *Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 2006.