SIA: Secure Information Aggregation in Sensor Networks*

Bartosz Przydatek
Carnegie Mellon University
Pittsburgh, PA 15213, USA
bartosz@cmu.edu

Dawn Song
Carnegie Mellon University
Pittsburgh, PA 15213, USA
dawnsong@cmu.edu

Adrian Perrig
Carnegie Mellon University
Pittsburgh, PA 15213, USA
perrig@cmu.edu

ABSTRACT

Sensor networks promise viable solutions to many monitoring problems. However, the practical deployment of sensor networks faces many challenges imposed by real-world demands. Sensor nodes often have limited computation and communication resources and battery power. Moreover, in many applications sensors are deployed in open environments, and hence are vulnerable to physical attacks, potentially compromising the sensor's cryptographic keys.

One of the basic and indispensable functionalities of sensor networks is the ability to answer queries over the data acquired by the sensors. The resource constraints and security issues make designing mechanisms for information aggregation in large sensor networks particularly challenging.

In this paper, we propose a novel framework for secure information aggregation in large sensor networks. In our framework certain nodes in the sensor network, called aggregators, help aggregating information requested by a query, which substantially reduces the communication overhead. By constructing efficient random sampling mechanisms and interactive proofs, we enable the user 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 the average of the measurements, for the estimation of the network size, and for finding the minimum and maximum sensor reading. Our protocols require only sublinear communication between the aggregator and the user. To the best of our knowledge, this paper is the first on secure information aggregation in sensor networks that can handle a malicious aggregator and sensor nodes.

*This research was supported in part by the Center for Computer and Communications Security at Carnegie Mellon under grant DAAD19-02-1-0389 from the Army Research Office, by NSF Aladdin Center grants CCR-0122581 and CCR-0058982, and by gifts from Bosch. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of NSF, ARO, Bosch, Carnegie Mellon University, or the U.S. Government or any of its agencies.

SenSys'03, November 5–7, 2003, Los Angeles, California, USA.

Categories and Subject Descriptors

C.2.2 [**Computer – Communication Networks**]: Distributed Systems; F.2 [**Analysis of Algorithms and Problem Complexity**]: Miscellaneous

General Terms

algorithms, reliability, security

Keywords

sensor networks, information aggregation, security, approximate interactive proofs

1. INTRODUCTION

Sensor networks are becoming increasingly popular to provide economical solutions to many challenging problems such as real-time traffic monitoring, wildfire tracking, wildlife monitoring, or building safety monitoring. In sensor networks, thousands of sensor nodes collectively monitor an area. These large sensor networks generate a substantial amount of data, yet the sensor nodes often have limited resources, such as computation power, memory, storage, communication, and most importantly, battery energy. The large scale of sensor networks and the resource constraints make it an important challenge to design and develop efficient information processing and aggregation techniques to make effective use of the data. Given a query, it may be unnecessary and inefficient to return all raw data collected from each sensor—instead, information should be processed and aggregated within the network and only processed and aggregated information is returned [13, 16]. In such a setting, certain 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 made even more challenging by the fact that the sensor nodes and aggregators deployed in hostile environments may be compromised due to physical tampering. Therefore, the processing and aggregation mechanisms need to be resilient against attacks where the aggregator and a fraction of the sensor nodes may be compromised.

Previous work in data aggregation assumes that every node is honest [21, 10, 13, 16], with the exception of [15] (cf. Sec. 1.1). In this paper, we address the problem of how to enable *secure* information aggregation, such that the user accepts the data with high probability if the aggregated result is within a desired bound, but that the user detects cheating with high probability and rejects the result if it is outside of the bound.

An attacker can perform a wide variety of attacks. For example, once the attacker compromised the base station or the aggregators,

the attacker could perform a denial-of-service attack and stop responding to any queries. Since we assume that a compromised node is under the full control of the attacker, there is nothing to prevent the attacker from mounting such denial-of-service attacks. However, in this paper we focus on another type of attack that we call *stealthy attack*. In a stealthy attack, the attacker's goal is to make the user accept false aggregation results, which are significantly different from the true results determined by the measured values, while not being detected by the user. In particular, we want to guarantee that if the user 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 compromised aggregators and/or the sensors, the user will detect the corruption and reject the reported aggregation result with high probability. We stress that in the considered model the corrupted sensors and aggregators may deviate from the protocol in an arbitrarily malicious way, and our goal is to prevent the user from accepting incorrect results.

More precisely, we propose the approach of *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.

In this paper we present the following contributions:

- We introduce the problem of *secure* information aggregation in sensor networks, analyze the attack model and security requirements.
- We propose the *aggregate-commit-prove* framework for designing secure information aggregation protocols (Section 3).
- We put forward concrete protocols for securely computing the median (Section 4), securely finding the minimum and maximum values (Section 5), securely estimating (counting) the number of distinct elements (and the network size) (Section 6), and securely computing the average of measurements (Section 7). Our protocols require only sublinear communication overhead between the aggregator and the user.
- We 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 (Section 8).

1.1 Related work

Previous work in sensor network data aggregation has mainly focused on how to aggregate information assuming every node is honest [21, 10, 13, 16]. Hu and Evans have studied the problem of information aggregation if one node is compromised [15], but their protocol may be vulnerable if a parent and a child node in their hierarchy are compromised.

Ergün *et al.* [12] 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, 18].

2. PROBLEM STATEMENT: SECURE INFORMATION AGGREGATION

2.1 Problem Setting

We consider the setting where a large number of sensors are deployed in some area distant from a *home server*. Sensors perform measurements and the home server would like to query statistics of the measured values. However, sensors are usually simple, low-powered devices which can communicate only within small range of their location, and so they cannot report the measurements directly to the distant home server [17]. Thus, a resources-enhanced *base station* is often used as an intermediary between the home server and the sensor nodes.

We assume that certain nodes in the network would perform the aggregation task. In the rest of the paper, we refer to the node that performs the aggregation task the *aggregator*. The base station is a natural candidate to perform the aggregation task, due to its enhanced computation and communication power. However, the issue of deciding which nodes are the aggregators is out of the scope of this paper and we simply assume that there exist some aggregators in the network at a given time. Moreover, some sensor networks may have multiple aggregators (For example, in TAG [21], each non-leaf node is an aggregator). For simplicity, in most of this paper, we only consider the case of a single aggregator. Nevertheless, our techniques can be extended to multiple aggregators, as we discuss in Section 9.

2.2 Key Setup And Communication Model

We assume that each sensor has a unique identifier and shares a separate secret cryptographic key with the home server and with the aggregator [25]. The keys enable message authentication, and encryption if data confidentiality is required. Note that the home server and the aggregator do not need to store $O(n)$ keys [25] — instead each of them stores simply a master key K_B and K_A (for the home server and the aggregator, respectively), and each sensor node stores the shared keys $\mathrm{MAC}_{K_B}(\text{node ID})$ and $\mathrm{MAC}_{K_A}(\text{node ID})$, where MAC is a secure message authentication code that is used here as a pseudo-random function. A specific secure instantiation of MAC is the HMAC construction by Bellare *et al.* [5]. Thus, given a node ID, the home server (or the aggregator) can compute its shared key with the sensor node by using its master key and hence authenticate the sensor node's message.

Since some sensors may be corrupted (cf. Section 2.3), the network may become partitioned by the corrupted sensors, in which case some uncorrupted sensors are able to communicate with each other and/or with the aggregator only via routes through corrupted sensors. When this happens, the corrupted sensors could always play a denial-of-service attack and cut off the communication between two partitioned sensor networks and simply claim to one partition that the other partition is not reachable. In such a case an aggregator may at best be able to compute aggregated information for one partition. Therefore, for simplicity, we assume that the uncorrupted sensors form a connected component containing the aggregator, meaning that the set of uncorrupted sensors can reach each other via paths composed of only uncorrupted sensors.

We furthermore assume that the home server and base station have a mechanism to broadcast authentic messages (e.g. queries)

into the network, such that each sensor node can verify the authenticity of the message, for example using the TESLA broadcast authentication protocol [24, 25].

2.3 Attack Model And Security Goals

We consider a setting with a polynomially bounded attacker, which can corrupt some of the sensors as well as the aggregator. Actions of a corrupted device are totally determined by the adversary. In particular, the adversary can arbitrarily change the measured values reported by a corrupted sensor. However, we assume that the adversary can corrupt at most a (small) fraction of all the sensors.

An attacker can perform a wide variety of attacks. For example, a corrupted aggregator could report some significantly biased or fictive values (possibly totally independent of the measured values), instead of the real aggregates, and so provide the home server with false information. Since in many applications the information received by the home server provides a basis for critical decisions, false information could have catastrophic implications. However, we do not want to limit ourselves to just a few *specific* selected adversarial strategies. Instead, we assume that the adversary can misbehave in any *arbitrary* way, and the only limitations we put on the adversary are its computational resources (polynomial in the security parameter) and the fraction of nodes that it can corrupt. In particular, we assume the Byzantine fault model [20] where a compromised node is under the full control of the attacker.

In this setting, we focus on *stealthy 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 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 Efficiency vs. Accuracy Tradeoff

The problems discussed in this paper have a straightforward (but unfortunately 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, hence the trivial solution of sending all the data is very inefficient, and our goal is to reduce the amount of communication between the the aggregator and the home server. 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 all the problems studied in this paper we 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, 19]. 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.

Depending on the function f being computed by the aggregator, various notions of approximations are useful. Let f be a function of $a_1, \ldots, a_n$ into real numbers, and let $y = f(a_1, \ldots, a_n)$. 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) Some sensors may be compromised and report wrong values that will affect the aggregation result. If a corrupted sensor simply reports a wrong value,[1] it may be difficult to detect the misbehavior since such a detection may require application/semantics specific knowledge. However, depending on the aggregation function, assuming that at most a certain number of sensors are compromised, we can calculate the bound on how much deviation from the correct result these corrupted sensors can cause.

(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 the above three factors can be upper bounded additively, and computing the bounds for factors (1) and (2) is relatively straightforward, in the rest of the paper we mainly focus on describing new techniques preventing the attacks of the third kind (corrupted aggregator) and compute the corresponding bounds. In particular, we propose efficient interactive proofs for verification of the accuracy of the results reported by the aggregator.

In addition to the approximation error ε, which describes the quality of a reported result, we also use 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, we call such a protocol an (ε, δ)-*approximation*, meaning that the protocol finds an ε-approximation with probability at least $1 - \delta$, and runs in time polynomial in $1/\varepsilon$ and $1/(1-\delta)$. As long as δ is bounded away from 1 by some constant fraction, the actual value of δ is not essential, since by repeating a protocol $\mathcal{O}(\log 1/\delta)$ times, we can make the probability arbitrarily small of being effectively cheated, assuming the independence of each trial.

2.5 Notation and Conventions

In the remainder of this paper, n denotes the number of sensors, $S_1, \ldots, S_n$, $\mathcal{A}$ 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 $[m] = \{1, \ldots, m\}$.

For the complexity analysis, we assume that each element and each hash value can be accessed in 1 step, and sending it costs 1

[1] However, a faulty value with a correct authentication tag, since we assume that node compromise also results in key compromise.

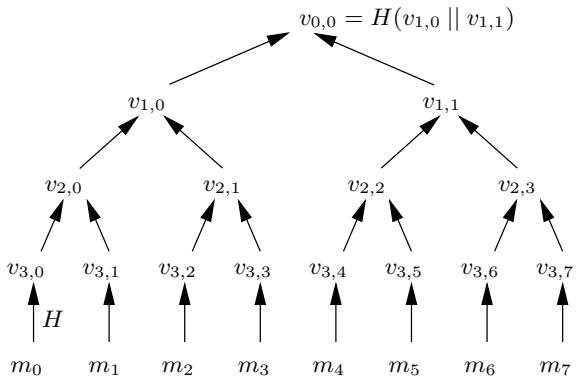

Figure 1: Merkle hash tree used to commit to a set of values. The aggregator constructs the Merkle hash tree over the sensor measurements $m_0, \ldots, m_7$. To lower the size of verification information, the aggregator first hashes the measurements with a cryptographic hash function, e.g., $v_{3,0} = H(m_0)$, assuming that the size of the hash is smaller than the size of the data. To construct the Merkle hash tree, each internal value of the Merkle hash tree is derived from its two child nodes: $v_{i,j} = H(v_{i+1,2j} \mid\mid v_{i+1,2j+1})$. The Merkle hash tree is a commitment to all the leaf nodes, and given the authentic root node $v_{0,0}$, a verifier can authenticate any leaf value by verifying that the leaf value is used to derive the root node. For example, to authenticate the measurement m_5, the aggregator sends m_5 along with $v_{3,4}, v_{2,3}, v_{1,0}$, and m_5 is authentic if the following equality holds: $v_{0,0} = H(v_{1,0} \mid\mid H(H(v_{3,4} \mid\mid H(m_5)) \mid\mid v_{2,3}))$.

unit of communication. Also, we assume that each computing a hash value can be done in $\mathcal{O}(1)$ 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.

3. OUR GENERAL APPROACH: AGGREGATE-COMMIT-PROVE

We propose a new approach, which we call *aggregate-commit-prove*: aggregators help computing aggregation of sensor nodes' raw data and reply to the home server with the aggregation 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 parts: computation of the result, committing to the collected data and report back the aggregation result, and proving the correctness of the result.

- In the first part, the aggregator collects the data from sensors and locally computes the aggregation result. As we discuss in Section 2, each sensor 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).

- In the second part, the aggregator commits to the collected data. The commitment to the input data ensures that the aggregator uses the data provided by the sensors, and that the statement to be verified by the home server about the correctness of computed results is meaningful. One efficient way of committing to the data is a Merkle hash-tree construction [22, 23]. In this construction, all the collected data is placed at the leaves of the tree, and the aggregator 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 is called the *commitment* of the collected data. Because the hash function in use is collision resistant, once the aggregator commits to the collected values, he cannot change any of the collected values. Figure 1 gives an example of a Merkle hash tree.

- In the third part, 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 steps:

  1. The home server checks that the committed data is a good representation of the true data values in the sensor network.
  2. 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. 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 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. 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 identifier of the i-th sensor.

Note that the corrupted sensor nodes can always forge their values and hence try to deviate the aggregation result from the true median. However, assuming that there are at most n' corrupted

nodes, the corrupted nodes can cause the aggregated median to deviate at most n' from the true median. If the aggregator cheats, without security mechanism built-in, the aggregator can report arbitrary value back to the home server. Therefore, in the rest of the section, we focus on designing efficient protocols to detect the aggregator cheating if the aggregator cheats more than a desired bound.

As mentioned in the introduction, the problem can be trivially solved by sending *all* the measurements to the home server. However, this solution is very inefficient in terms of communication complexity. With the low-communication requirement in mind, probably the most straightforward approach is to sample the measurements and use the median of the sample as an estimate of the true median. In the following, we first analyze this naive approach, and then propose a more efficient solution. Note that here we assume that the user knows the approximate size of the sensor network. This can be achieved, for example, with a method presented in Section 6.

4.1 Naive Approach: Median by Random Sampling

Probably the simplest method for estimating the median using sublinear communication complexity is random sampling: we take a random sample of ℓ measured values and return the median of the sample as an approximation of the median of all the measurements.

The computational cost and communication complexity of this approach are clearly determined by the number of samples (ℓ) required to achieve an estimate which with high probability is equal to some element within εn of the actual median.

THEOREM 1. *The median of a uniform sample of ℓ out of n elements $a_1, \ldots, a_n$ with probability at least $1 - (2/e^{2\ell\varepsilon^2})$ yields an element whose position in the sorted sequence $a_1, \ldots, a_n$ is within εn of $n/2$.*

Theorem 1 implies, that in order to achieve with probability close to 1 a ε-approximation of the median it is sufficient to choose the size of sample $\ell = \mathcal{O}(1/\varepsilon^2)$. However, Bar-Yossef *et al.* [3] show that $\Omega(1/\varepsilon^2)$ samples are also necessary for an ε-approximation of the median.

4.2 Our Approach for Median

The naive sampling approach presented in the previous section makes only minimal use of the capabilities of the aggregator $\mathcal{A}$—indeed, $\mathcal{A}$ only forwards messages from the sensors, without doing any processing.

In this section we consider the interactive-proof approach, in which $\mathcal{A}$ is more involved in the computation process. As we shell see, this yields significant savings in the communication complexity.

As mentioned in the general framework, we require that $\mathcal{A}$ commits the measured values using a hash-tree construction. In this case we additionally require that the sequence of the values committed to is *sorted*.

In the interactive proof, $\mathcal{B}$ obtains an alleged median a_{med} and verifies its correctness by checking (using two tests) that the sequence committed to indeed fulfills the requirements. More precisely, first $\mathcal{B}$ verifies that the committed sequence is sorted, that all the elements in the sequence are distinct, and that all the elements come from different sensors. To achieve this, we need two committed sequences, one sorted according to the measured values, and one sorted according to the sensor ids. We check that both sequences are sorted using `Sort-Check-II` spot checker from [11] with subsequent uniform sampling of *pairs* of neighboring elements (requiring $\mathcal{O}(\log n/\varepsilon)$ and $\mathcal{O}(1/\varepsilon)$ samples, respectively; cf. Section 7.1). Finally we use additional $\mathcal{O}(1/\varepsilon)$ samples to check that both lists contain the same elements. Essentially, this check picks random elements from one list, and verifies that they are present in the other list [12].

In the second test $\mathcal{B}$ checks that a_{med} is (close to) the median of committed sequence. Here $\mathcal{B}$ picks elements from random positions[2] in the committed sequence, and checks that elements picked from the left half of the sequence are smaller than the reported median, and elements from the right half are larger than the median. A pseudo-code description of this median-checking test is given below.

procedure *MedianCheck(n, a_{med}, ε)*:
  request $a_{n/2}$
  if $a_{n/2} \neq a_{\mathrm{med}}$ **then**
    return REJECT
  for $i = 1 \ldots (1/\varepsilon)$ **do**
    pick $j \in_R \{1 \ldots n\} \setminus \{n/2\}$
    request a_j
    if $j < n/2$ **and** $a_j > a_{\mathrm{med}}$ **then**
      return REJECT
    if $j > n/2$ **and** $a_j < a_{\mathrm{med}}$ **then**
      return REJECT
  return ACCEPT

THEOREM 2. *Procedure* MedianCheck(n, a_{med}, ε) *requests $1/\varepsilon$ elements a_i, runs in time $\mathcal{O}(t \cdot 1/\varepsilon)$, where t is time required to process a single request, 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

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

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

Theorem 2 together with the complexity of the first test imply that by requesting in total only $\mathcal{O}(\log n/\varepsilon)$ elements we can check whether the reported value is an ε-approximation of the median, and at the same time guarantee a constant probability of detecting a cheating aggregator. In many scenarios we have $\log n \ll 1/\varepsilon$, and in such cases the proposed procedure uses significantly fewer samples than random sampling to guarantee the same error bound.

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

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 a stand-alone primitives but also as a subprotocol 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

[2]In fact, this second test can use the samples from the first test. We describe both tests separately for better modularity.

by the sensors. Then in Section 6 we show an example application, namely how we use our secure min-discovery protocol as a building block to enable random selection of a node in the network 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 in general 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, or it is not in the interest of the adversary to report smaller values. As an example for the first case, consider the situation[3] were the input value to be used by each sensor is a MAC of the current time interval using its key shared with the home server, and the home server is interested in finding out which sensor has the smallest such MAC value. Note that assuming that the current time interval is well-defined and each sensor has a reliable way to determine the current time interval, a corrupted sensor cannot manipulate its own MAC value. An example for the second case is a scenario where the adversary tries to hide the mere existence of small measured values.

Following the "aggregate-commit-prove" method, in the first step each node authenticates its input value and sends it to the aggregator, which computes the minimum value and commits to the set of values reported by the sensor nodes. Without security considerations, we could then simply have the aggregator to report the computed minimum value back to the home server. However, this approach is insecure because the aggregator can for example hide the smallest value and not report it. Using the method for computing quantiles as described in Section 4, we could 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. Zhao et. al. proposed a similar tree-based approach for computing Min/Max value although their approach assumes nodes well behave and does not consider adverserial attacks [27].

5.1 The FindMin Protocol

The protocol works by first constructing a spanning tree in the network of sensors, such that the root of the tree holds the minimum element (procedure *MinRootedTree*), and then checking that the tree was constructed properly (procedure *FindMin*).

The construction of the tree proceeds in iterations. Assume that each sensor S_i has a value a_i. Throughout the protocol S_i maintains also a tuple of state variables $(\mathrm{p}_i, \mathrm{v}_i, \mathrm{id}_i)$, where p_i holds the ID of the current parent of S_i in the tree being constructed, v_i holds the smallest value seen so far, and id_i holds the ID of the node whose value is equal to v_i (we assume that whenever there is a tie of the values, we break the tie by considering the node ID).

Each sensor node S_i initializes its state as $\mathrm{p}_i := S_i$, $\mathrm{v}_i := a_i$, and $\mathrm{id}_i := S_i$. In each iteration S_i broadcasts $(\mathrm{v}_i, \mathrm{id}_i)$ to its neighbors, and upon receiving the analogous messages from the neighbors, S_i picks the message with the smallest value (breaking ties using the node ID). Let $(\mathrm{v}', \mathrm{id}')$ denote the message with the smallest value picked by S_i, and let S' be the sender of this message. S_i updates its state by setting $\mathrm{p}_i := S'$, $\mathrm{v}_i := \mathrm{v}'$, and $\mathrm{id}_i := \mathrm{id}'$. Then the sensors proceed to the next iteration. The construction terminates after d iterations, where d is an upper bound on the diameter of the network. A pseudo-code specification of the procedure *MinRootedTree* is given below.

[3]This scenario will be used to estimate the network size as we will show in Section 6.

procedure *MinRootedTree(d)*:
  /** *code for sensor* i **/
  $\mathrm{p}_i := S_i$ /** *current parent* **/
  $\mathrm{v}_i := a_i$ /** *current minimum* **/
  $\mathrm{id}_i := S_i$ /** *owner of current minimum* **/
  for $i = 1 \ldots d$ **do**
    send $(\mathrm{v}_i, \mathrm{id}_i)$ to all neighbors
    receive $(\mathrm{v}_j, \mathrm{id}_j)$ from neighbors
    if $\mathrm{v}_j < \mathrm{v}_i$ for some j **then**
      $\mathrm{p}_i := S_j$
      $\mathrm{v}_i := \mathrm{v}_j$
      $\mathrm{id}_i := \mathrm{id}_j$

Our assumption that the uncorrupted sensors form a connected component implies that after the construction of the tree terminates the uncorrupted sensors all have the same smallest value and they form a tree rooted at the node that is the owner of the smallest value. For the checking protocol, each node S_i authenticates its final state $(\mathrm{p}_i, \mathrm{v}_i, \mathrm{id}_i)$ from the construction protocol, using the key shared with the home server, and sends the authenticated state to the aggregator. The aggregator $\mathcal{A}$ checks the consistence of the resulting tree with the values committed to initially. If the check is successful, $\mathcal{A}$ commits to the list of all the nodes and their states, finds the root of the resulting tree and reports the root-node to the home server. Otherwise $\mathcal{A}$ reports the inconsistency and terminates the protocol. The home server then randomly picks a node in the committed list, and then traverses the path from the picked node to the root, using the information provided in the state. During the traversal the home server checks the consistency of the constructed tree, using the provided authenticated states. If all the checks are successful, then the home server $\mathcal{B}$ accepts the value reported by the aggregator as the the minimum value, otherwise $\mathcal{B}$ rejects.

procedure *FindMin(ϵ)*:
  /** *code for the home server* **/
  request construction of a tree using *MinRootedTree*
  if tree construction failed **then**
    return REJECT
  request number n of the nodes in the tree
  for $i = 1 \ldots (1/\varepsilon)$ **do**
    pick $j \in_R \{1 \ldots n\}$
    request j-th node from the tree
    follow path to the root
    if path is inconsistent **then**
      return REJECT
  return ACCEPT

Formally, we have the following theorem.

THEOREM 3. *Assuming that no more than ε fraction of the sensors are corrupted, and the minimum value in the committed values is from an uncorrupted sensor, and that all uncorrupted sensors form a connected component of diameter at most d, procedure* FindMin *requests $\mathcal{O}(d/\varepsilon)$ elements and satisfies:*

(1) If all the sensors and the aggregator follow the protocol then the home server ACCEPTs *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 REJECTS *with probability at least* $1 - \varepsilon$.

Proof. (sketch) Note first that since all the sensors are initially required to commit to their values, they cannot change the values during the protocol—any change having impact on the final result will be detected because of the data authentication. Moreover, 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. On the other hand, if the aggregator tries to cheat, the random sampling of a starting node for the tree traversal will with probability at least $1 - \varepsilon$ hit the connected component of the uncorrupted sensors, and so detect cheating. □

6. COUNTING DISTINCT ELEMENTS

In this section we study the problem of counting the number μ of distinct values in the measurements, i.e., the problem of determining the size of the set of all the measurements. Note that malicious sensor nodes can always forge their measurements and hence influence the result of counting distinct elements. Assuming there are at most n' corrupted sensor nodes, then the corrupted sensor nodes can cause the aggregation result to deviate at most n' from the true result. However, if the aggregator cheats, the aggregator can cheat arbitrarily about the result. Therefore, in the rest of the section, we focus in designing protocols where once the sensor values are committed, how we can detect cheating from the aggregator if the aggregator cheats more than certain bounds.

Ergün *et al.* [12] 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, 18]. 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 [18] imposes an additional factor of $\Omega(\log^4 n)$ per access of a single element.

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 [4, 14, 1], and on a novel technique for random selection of the nodes of the network. In the following subsections we first describe the basic tools used in our constructions, then we present the proposed protocols, and finally we discuss their applications in estimation of the network size.

6.1 Basic Tools

6.1.1 Space-Efficient Estimation of the Number of Distinct Elements

In the setting of computation on data streams Flajolet and Martin [14] proposed a space-efficient technique for estimation of the number of distinct elements in a stream. The key idea is to pick a random hash function $h : [m] \to [0 \dots 1]$, 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.* [4] further improved this method and presented a (ε, δ)-approximation for μ. The basic idea for the improvement is to maintain several ($t = \mathcal{O}(1/\varepsilon^2)$) 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.

6.1.2 Random Selection of a Node

The second basic tool needed in our constructions is a method for a selection of a node (sensor) at random. Note that even if the home server has a list of IDs 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 doesn't 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 and the current time interval. 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, the ID of the sensor and the current time interval. 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 1. *Under assumptions of Theorem 3, 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, as shown below, it is very useful for counting distinct elements and computing the network size.

6.2 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 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 from [4], by keeping t smallest hash values instead of just the single minimum. However, this improvement comes at a cost of higher communication complexity.

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

The above method for counting distinct elements depend on the random selection procedure from Section 6.1.2. In some applications the traffic in the sensor network needed by 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 efficiency gain comes at the price of relaxed accuracy guarantees. In particular, the presented methods yield approximations of lower & upper bounds, instead of approximation of the actual number of distinct elements. However, in a practical setting this is not a real disadvantage — to get an accurate estimate we require the aggregator to prove *matching* lower & upper bounds.

The proposed alternative approach to the counting of distinct elements is in the spirit of other protocols proposed in this paper: "aggregate-commit-prove". Here the aggregator first collects all the distinct values reported by the sensors, and subsequently commits to the collected values.

6.3.1 A Lower Bound on the Number of Distinct Elements

At the beginning of the protocol for estimating a lower bound on the number of distinct elements the aggregator $\mathcal{A}$ commits to the values reported by the sensors using a hash-tree construction. $\mathcal{A}$ will then run an algorithm for counting distinct elements in a stream by Bar-Yossef *et al.* [4], using hash functions specified by the home server $\mathcal{B}$.

Let $H = \{h |\ h : [m] \to [M]\}$, where $M = m^3$, be a family of pairwise independent hash functions [26], such that any function $h \in H$ has a short description ($\mathcal{O}(k)$ bits). After $\mathcal{A}$ commits to the input, $\mathcal{B}$ computes an estimate for a lower bound on 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 $i = 1 \ldots n$, 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.* [4] 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 with the median of the ℓ resulting estimators $\mu'_1, \ldots, \mu'_\ell$. 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 $j = 1 \ldots \ell$ **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 $\mu' = \text{median}(\mu'_1, \ldots, \mu'_\ell)$

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. This is a reason why the described protocol yields only a lower bound on μ, not an estimate of μ. A malicious aggregator $\mathcal{A}$ can omit some elements, or report elements which evaluate to larger values. However, such cheating results in an estimate μ' smaller than μ, and so μ' will be still a valid estimate of a lower bound.

THEOREM 4. *Procedure* DistinctLowerBound(n, m, ε, ℓ) *requests $\mathcal{O}(\ell \cdot 1/\varepsilon^2)$ elements a_i and returns a value μ' which with probability at least $1 - (1/6)^{\ell/2}$ satisfies $\mu' \leq (1+\varepsilon)\mu$.*

6.3.2 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, and additionally 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. The test works by random sampling: the verifier $\mathcal{B}$ requests random element from S, and asks $\mathcal{A}$ for an element with the same value present in S'.

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

6.3.2.1 High Number of Distinct Elements.

In the case when the number of distinct elements is a significant fraction of the number of all the elements, i.e., $\mu \geq n/c$ for some $c \geq 1$, a simple sampling through the aggregator is sufficient — pick a node at random, with uniform distribution over all nodes.

Note that this node-sampling procedure in this case is quite different from the one described in Section 6.1.2, and in particular is much more efficient. A pseudo-code description of the entire protocol based on this simple sampling is given below.

---

procedure *HighDistinctUpperBound(n, ε, ℓ)*:
  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 5. *Procedure* HighDistinctUpperBound(n, ε, ℓ) *requests ℓ elements and satisfies:*

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

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

Theorem 5 implies that by taking $\ell = c/\varepsilon$ we can detect cheating with constant probability. Therefore, if with such value of ℓ we repeat the test $\mathcal{O}(\log 1/\delta)$ times, we get confidence at least $1 - \delta$.

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 $\mathcal{B}$ [9, 3].

6.3.2.2 *Low Number of Distinct Elements.*

When the number of distinct elements is low in comparison to the total size of S the simple sampling won't work, because the omitted elements not reported in S' could be infrequent and so very hard to find by sampling. However, a modest modification of the random sampling protocol from Section 6.1.2 can fix this problem: each node report a 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(ε, ℓ)*:
  for $i = 1 \dots \ell$ **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 5.

THEOREM 6. *Procedure* LowDistinctUpperBound(ε, ℓ) *requests ℓ elements and satisfies:*

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

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

6.4 Estimating the Network Size

It is clear that the problem of estimating the size of the network is a special case of counting the distinct elements. By assumption, each sensor has a unique identifier hence the computation of the size of the network is equivalent to counting of the number of distinct elements in the set of all sensor identifiers.

7. COMPUTING THE AVERAGE

In this section, we describe how to efficiently and securely compute the average of sensor data. The sensor data can be sensor measurements such as temperature reading or light intensity, or can be information about a sensor node such as the remaining battery power or remaining storage space. Note that corrupted sensors can always report forged values and hence deviate the aggregation result from the true result. However, assuming that there are at most n' corrupted sensors and the difference between the valid maximum measurement value and the valid minimum measurement value is at most u, then the deviation caused by forged values from the corrupted sensors will be at most un'/n where n is the size of the sensor network. On the other hand, if the aggregator cheats, without a good security mechanism, the aggregation result may be arbitrarily far from the true result. Therefore, in the rest of this section, we focus on designing protocols to ensure that if the aggregator cheats more than a certain bound, the home server will detect the aggregator's cheating. Below we describe two different protocols with different efficiency tradeoffs for this purpose.

7.1 Computing the 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 generalized 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, 3].

As in the case of the computation of the median, the naïve 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.

First the aggregator $\mathcal{A}$ collects all the (value, ID)-pairs (a_i, ID_i) and commits to them. 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 using sampling. If the test complete successfully, $\mathcal{A}$ sends to $\mathcal{B}$ the occurrence-counts for each value $1 \dots m$. $\mathcal{A}$ verifies the correctness of the counts by deriving from them the positions in the committed sorted sequence, and check the values in the committed sorted sequence are well sorted using the method `Sort-Check-II` spot checker from [11]. If also this check succeeds, $\mathcal{B}$ computes the average directly from the occurrence-counts, and compares it to $\bar{a}$. Summarizing, we obtain the following theorem. The `Sort-Check-II` spot checker performs $\mathcal{O}(1/\varepsilon)$ binary searches on the committed sequence to ensure that all but a ε fraction of the elements are sorted, which results in $\mathcal{O}(\log n/\varepsilon)$ samples.

THEOREM 7. *The procedure* AverageByFrequency *requests $\mathcal{O}(\log n/\varepsilon + m)$ elements and satisfies:*

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

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

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 $[m/2^i, m/2^{i-1}]$. And we can perform a protocol similar to the above for each interval and achieve a lower bound and an upper bound for the average.

7.2 Computing the Average by Counting Distinct Elements

We can reduce the problem of computing the average to the problem of determining the number of distinct elements in a set [12]. 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 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 6, 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.

8. 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.[4] 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 on 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].

9. DISCUSSION: SECURE HIERARCHICAL AGGREGATION

If the sensor network is too large, then one aggregator may not be capable to handle the whole network. In this case, we may need to use a hierarchical aggregator to enable the aggregation. Some functions support hierarchical aggregation such as the Min/Max and average computation where each aggregator can aggregate information for a subset of the nodes in the sensor network and then the information of the aggregators can be aggregated to compute the final result. However, some other functions such as computing the median may not support hierarchical aggregation in the traditional way such as in [21]. However, if we allow each aggregator to perform random sampling over the entire network, then the median could be computed as the median of the medians computed by the aggregators.

In the case of hierarchical aggregation, we could use our approaches to ensure the security of each step of the aggregation or check the steps probabilistically.

We also plan to investigate other types of aggregation functions and see how to secure them.

[4]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 preimage x such that $f(x) = y$ given y.

10. CONCLUSION

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 concrete protocols within this framework for securely computing the median (Section 4), securely finding the minimum and maximum values (Section 5), securely estimating (counting) the number of distinct elements (and the network size) (Section 6), and securely computing the average of measurements (Section 7). 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. To the best of our knowledge, our protocols are the first ones that can handle the problem that the aggregator and/or the sensor nodes may be malicious. We anticipate that our paper introduces the problem of secure information aggregation to the community and encourages other researchers to consider this important problem.

Acknowledgments

The authors would like to thank anonymous referees and shepherd Philippe Bonnet for their valuable comments and feedback.

11. REFERENCES

[1] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proc. 28th STOC*, pages 20–29, 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] Ziv Bar-Yossef, S. Ravi Kumar, and D. Sivakumar. Sampling algorithms: lower bounds and applications. In *Proc. 33rd STOC*, pages 266–275, 2001.

[4] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. Counting distinct elements in a data stream. In *Proc. RANDOM 2002*, pages 1–10, 2002.

[5] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In *Advances in Cryptology – CRYPTO '96*, pages 1–15, 1996.

[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 *Proc. Eurocrypt'99*, 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] Moses Charikar, Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. Towards estimation error guarantees for distinct values. In *Proc. 19th PODS*, pages 268–279, 2000.

[10] Amol Deshpande, Suman Nath, Phillip B. Gibbons, and Srinivasan Seshan. Cache-and-query for wide area sensor databases. In *SIGMOD 2003*, 2003.

[11] Funda Ergün, Sampath Kannan, S. Ravi Kumar, Ronitt Rubinfeld, and Mahesh Viswanathan. Spot-checkers. *JCSS*, 60:717–751, 2000. preliminary version in *Proc. STOC'98*.

[12] Funda Ergün, Ravi Kumar, and Ronitt Rubinfeld. Fast approximate PCPs. In *Proc. 31st STOC*, pages 41–50, 1999.

[13] 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*, August 1999.

[14] P. Flajolet and G. N. Martin. Probabilistic counting. In *Proc. FOCS'83*, pages 76–82, 1983.

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

[16] C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidemann. Impact of network density on data aggregation in wireless sensor networks. In *Proceedings of International Conference on Distributed Computing Systems*, November 2001.

[17] J. M. Kahn, R. H. Katz, and K. S. Pister. Mobile networking for smart dust. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking, MobiCom 99*, Seattle, WA, August 1999.

[18] Aggelos Kiayias and Moti Yung. Secure games with polynomial expressions. In *Proc. 28th ICALP*, pages 939–950, 2001.

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

[20] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, July 1982.

[21] 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 (OSDI)*, December 2002.

[22] Ralph C. Merkle. Protocols for public key cryptosystems. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 122–134, April 1980.

[23] Ralph C. Merkle. A certified digital signature. In *Proc. Crypto'89*, pages 218–238, 1989.

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

[25] 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, September 2002.

[26] Mark N. Wegman and J. Lawrence Carter. New hash functions and their use in authentication and set equality. *JCSS*, 22:265–279, 1981.

[27] Jerry Zhao, Ramesh Govindan, and Deborah Estrin. Computing aggregates for monitoring wireless sensor networks. In *First IEEE International Workshop on Sensor Network Protocols and Applications*, May 2003.

APPENDIX

A. PROOFS

Proof. (Theorem 1) Let X_ℓ be a random variable denoting the position of the median returned by the algorithm using ℓ samples. We want to show that

$$\mathsf{Pr}\left[|X_\ell - n/2| > \varepsilon n\right] \leq e^{-2\ell\varepsilon^2} .$$

First note that by symmetry we have

$$\begin{aligned}\mathsf{Pr}\left[|X_\ell - n/2| > \varepsilon n\right] &= \\ &= \mathsf{Pr}\left[(X_\ell - n/2) > \varepsilon n\right] + \mathsf{Pr}\left[(n/2 - X_\ell) > \varepsilon n\right] \\ &\leq 2\mathsf{Pr}\left[(X_\ell - n/2) > \varepsilon n\right] .\end{aligned}$$

Now, the event that $(X_\ell - n/2) > \varepsilon n$ is equivalent to the event that more than $\ell/2$ of the sampled elements have positions greater than $n/2 + \varepsilon n$. Let for $i = 1 \ldots \ell$, Y_i denote a random variable equal to 1 if the i-th sample has position greater than $n/2 + \varepsilon n$, and equal to 0 otherwise. Since we sample with uniform distribution, we have

$$\begin{aligned}\mathsf{Pr}\left[Y_i = 1\right] &= 1/2 - \varepsilon \\ \mathsf{Pr}\left[Y_i = 0\right] &= 1/2 + \varepsilon\end{aligned}$$

Let $S_\ell = \sum_{i=1}^{\ell} Y_i$. Clearly, $\mathrm{E}[S_\ell] = \ell/2 - \ell\varepsilon$. By the above argument we get

$$\begin{aligned}\mathsf{Pr}\left[(X_\ell - n/2) > \varepsilon n\right] &= \mathsf{Pr}\left[S_\ell > \ell/2\right] \\ &= \mathsf{Pr}\left[S_\ell > \mathrm{E}[S_\ell] + \ell\varepsilon\right] \\ &\leq e^{-2\ell\varepsilon^2} ,\end{aligned}$$

where the last inequality follows from Hoeffding bound. □

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

Proof. (Theorem 4) First, and consider the special case when $\ell = 1$. Bar-Yossef *et al.* [4] bound the probability that the estimate μ_1' is significantly larger than μ:

$$\mathsf{Pr}\left[\mu_1' > (1+\varepsilon)\mu\right] < \frac{1}{6} .$$

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

$$\mathsf{Pr}\left[\mathrm{median}(y_1, \ldots, y_\ell) > (1+\varepsilon)\mu\right] < \left(\frac{1}{6}\right)^{l/2} ,$$

and the claim follows. □

Proof. (Theorem 5) 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 $\mu' < (1-\varepsilon)\mu$, then there are at least $\mu\varepsilon$ elements in S which detect cheating and lead to "REJECT". By assumption $\mu\varepsilon \geq \varepsilon n/c$ for some $c \geq 1$, so the probability that a single sample detects cheating is at least ε/c. Therefore, the probability of returning "ACCEPT" after ℓ samples is at most $(1-\varepsilon/c)^\ell \leq e^{-\ell\varepsilon/c}$, which implies the claim. □