Securely Computing an Approximate Median in Wireless Sensor Networks

Abstract— **Wireless Sensor Networks (WSNs) have proven to be useful in many applications, such as military surveillance and environment monitoring. To meet the severe energy constraints in WSNs, some researchers have proposed to use the in-network data aggregation technique (i.e., combining partial results at intermediate nodes during message routing), which significantly reduces the communication overhead. Given the lack of hardware support for tamper resistance and the unattended nature of sensor nodes, sensor network protocols need to be designed with security in mind. Recently, researchers proposed algorithms for securely computing a few aggregates, such as Sum (the sum of the sensed values), Count (number of nodes) and Average. However, to the best of our knowledge, there is no prior work which securely computes the Median, although the Median is considered to be an important aggregate. The contribution of this paper is twofold. We first propose a protocol to compute an approximate Median and verify if it has been falsified by an adversary. Then, we design an attack-resilient algorithm to compute the Median even in the presence of a few compromised nodes. We evaluate the performance and cost of our approach via both analysis and simulation. Our results show that our approach is scalable and efficient.**

I. Introduction

Wireless Sensor Networks (WSNs) are being used in many applications [16], [19], [23], such as military surveillance, wildlife habitat monitoring, forest fire prevention, etc. A WSN normally consists of a large number of sensor nodes which are self-organized into a multi-hop network.

The simplest way to collect the sensed data is to let each sensor node deliver its reading to the base station (BS). This approach, however, is wasteful since it results in excessive communication. A typical sensor node is severely constrained in communication bandwidth and energy reserve. Hence, sensor network designers have advocated alternative approaches for data collection.

An *in-network* aggregation algorithm combines partial results at intermediate nodes during message routing, which significantly reduces the amount of communication and hence the energy consumed. A typical data acquisition system [13], [7] constructs a spanning tree rooted at the BS and then performs in-network aggregation along the tree. Partial results propagate level by level up the tree, with each node awaiting messages from all of its children before sending a new partial result to its parent. Researchers [13], [7] have designed several energy-efficient algorithms to compute aggregates such as Count, Sum, Average, etc.

However, an in-network aggregation algorithm cannot cheaply compute the exact Median, where the worst case communication overhead per node is $\Omega(N)$, where N is the number of nodes in the network [13]. As a result, researchers have advocated computation of an approximate Median. In-network aggregation algorithms to compute an approximate Median are proposed in [9], [25].

Unfortunately, none of the above algorithms include any provisions for security, and hence, they cannot be used in security-sensitive applications. Given the lack of tamper-resistance and the unattended nature of many networks, we must consider the possibility that a few sensor nodes in the network might become compromised.

A compromised node in the aggregation hierarchy may attempt to change the aggregate value computed at the BS by relaying a false sub-aggregate value to its parent. This attack can be launched on most of the in-network aggregation algorithms. For example, in Greenwald et al.'s approximate Median computation algorithm [9], a compromised node in the aggregation hierarchy can corrupt the quantile summary to make the BS accept a false Median which might contain a large amount of error.

A technique to compute and verify Sum and Count aggregates has been recently proposed by Chan et al. [2]. Their scheme [2] can also verify if a given value is the true Median, but they have not proposed any solution to compute that value in the first place. To the best of our knowledge, there is no prior work which securely computes the Median using an in-network algorithm. However, researchers [26] observed that the Median is a more important aggregate than the Average.

One might suggest an approach which runs Greenwald et al.'s algorithm [9] to compute an approximate Median and then employs Chan et al.'s verification protocol [2] to verify if the computed value is indeed a valid estimate. We refer this approach as GC in the rest of the paper.The communication cost per node in this approach is $O(\frac{log^2 N}{\epsilon})$, where ϵ is the approximation error bound.

In this paper, we propose an alternative approach to compute and verify an approximate Median, which proves to be more efficient compared to the GC approach. Our approach is based on sampling—an uniform sample of sensed values is collected from the network to make a preliminary estimate of the Median, which is verified and refined later. The communication cost of our basic algorithm is $O(\frac{1}{\epsilon}\Delta \log N)$, where ϵ is the error bound and Δ is the maximum degree of the aggregation tree used by the algorithm.

Like the GC approach, our basic algorithm guarantees that an attacker cannot cause the BS to accept a Median estimate which contains an error more than the user-specified bound, ϵ. However, neither of the above approaches can guarantee the successful computation of the Median in the presence of an attacker. To address this problem, we extend the basic approach so that we can compute the Median even in the presence of a few compromised nodes. The analysis and

	Node congestion	Latency (epochs)	Verification	Attack-resilient computation
Greenwald et al.'s protocol [9]	$O(\frac{log^2 N}{\epsilon})$	2	No	No
GC approach (Section IV-A)	$O(\frac{log^2 N}{\epsilon})$	6	**Yes**	No
Our basic protocol (Section IV-C)	$O(\frac{1}{\epsilon}\Delta log N)$	6 w.h.p.	**Yes**	No
Our extended protocol (Section VI)	$O(\frac{1}{\epsilon}\Delta log N)$	6 w.h.p.	**Yes**	**Yes**

TABLE I
MEDIAN COMPUTATION PROTOCOLS: COMPARING THE PERFORMANCE AND THE SECURITY FEATURES

simulation results show that our algorithms are effective and efficient. Moreover, our algorithms can be extended to compute other quantiles besides the Median.

Table I compares our approach with other solutions on the basis of a few performance and security metrics. We report *node congestion* as a metric for communication complexity, which represents the worst case overhead on a single node. We measure the latency of the protocols in *epochs*. Similarly to the prior work [13], an epoch represents the amount of time a message takes to traverse the distance between the BS and the farthest node on the aggregation hierarchy. We observe that the latency of our protocol might increase in extreme cases; here we report the latency which our protocol incurs in most cases (i.e., with high probability (w.h.p.)). To measure the security of the protocols, we consider the following properties. We say that a protocol has *verification* property if the protocol enables the BS to verify whether the computed Median is false or not. Observe that this property does not guarantee the computation of the Median in the presence of an attack. Finally, an attack-resilient protocol is so if it guarantees the computation of the Median in the presence of a few malicious nodes.

Organization The rest of the paper is organized as follows. In Section II, we review the related work present in the literature. Section III describes the problem and the assumptions taken in this paper. In Section IV, we present our basic protocol, whose security and performance analysis is given in Section V. Section VI describes our attack-resilient protocol. We present our simulation results in Section VII, and finally, we conclude the paper in Section VIII.

II. Related Work

Several researchers [13], [7] have proposed in-network aggregation algorithms which fuse the sensed information en route to the BS to reduce the communication overhead. In particular, these algorithms are designed to compute *algebraic aggregates*, such as Sum, Count, and Average. However, Madden et al. [13] showed that in-network aggregation does not save any communication overhead in case of computing *holistic* aggregates, such as the Median.

To limit the communication complexity, researchers have advocated computing an approximate estimate instead of the exact Median [9], [25]. In particular, Greenwald et al. [9] proposed a *quantile summary* computation algorithm that exploits a concept of *delayed aggregation* so that no summary contains error more than ϵ bound. Also, Srivastava et al. [25] presented another data summarization technique called *quantile digest* to compute an approximate median, where the main idea is to compute an equi-depth histogram through in-network aggregation. There also exists a body of data stream algorithms in the literature which computes approximate quantiles [8], [14], [4]. In fact, Greenwald et al.'s algorithm [9] is an extension of [8].

Our Median computation algorithm has a sampling phase and a histogram computation phase. Sampling techniques have been previously employed for data reduction in databases [1], [22]; in particular, [1] uses a sample of a large database to obtain an approximate answer. Another work, from Munro and Paterson [17], analyzed the lower bound on storage space and number of passes of a Median computation algorithm. Jain et al. [11] proposed a centralized algorithm to compute quantiles and histograms with limited storage space. Recently, Patt-Shamir [20] designed an approximate Median computation algorithm using a synopsis diffusion framework [3], [18], which uses a multipath routing algorithm to enhance robustness against communication loss. We note that none of the above algorithms were designed with security in mind, and an attacker can inject an arbitrary amount of error in the final estimate.

Recently, a few researchers have examined security issues in aggregation algorithms. Wagner [26] addressed the problem of resilient data aggregation in the presence of malicious nodes and provided guidelines for selecting aggregation functions in a sensor network. Yang et al. [27] proposed SDAP, a secure hop-by-hop data aggregation protocol using a tree-based topology to compute the Average in the presence of a few compromised nodes. SDAP divides the network into multiple groups and employs an outlier detection algorithm to detect the corrupted groups. In our extended approach, we also use a grouping technique but without any outlier detection algorithm that would otherwise require the assumption that groups have similar data distribution. Another approach for the securely computing Count and Sum, proposed by Roy et al. [24], is designed for a synopsis diffusion framework.

Chan et al. [2] designed a verification algorithm by which the BS could detect if the computed aggregate was falsified. However, the authors did not propose any algorithm to compute the Median.Their verification algorithm is based on a novel method of distributing the verification responsibility onto the individual sensor nodes. An improvement on the communication complexity of the above algorithm has been recently proposed by Frikken et al. [6].

III. Assumptions and Problem Description

The goal of this paper is to securely compute an approximate Median of the sensor readings in a network where a few nodes might be compromised. Given a specified error bound, we return an approximate Median which is sufficiently close to the exact Median. This section describes our system model and design goals.

Network Assumptions We assume a general multihop network with a set of N sensor nodes and a single BS. The BS knows the IDs of the sensor nodes present in the network. The network user controls the BS, initiates the query and specifies the error bound ϵ. In the rest of the paper, we consider the user and the BS as a single entity. We also consider that sensor nodes are similar to the current generation of sensor nodes (e.g., Berkeley MICA2 motes [10]) in their computational and communication capabilities and power resources, while the BS is a laptop-class device supplied with long-lasting power.

We assume that the in-network aggregation is performed over an *aggregation tree* which is constructed during the query broadcast, similarly as in the TAG algorithm [13]. However, our approach does not rely on a specific tree construction algorithm. The approximation error ϵ in the estimated Median $\hat{m}$ is determined by how many position $\hat{m}$ is away from the exact Median m in the sorted list of all the sensed values. For ease of exposition, without loss of generality we assume that all the sensed values are distinct. Note that we could relax this assumption by defining an order on IDs of the nodes that have same sensed value. Also, for the ease of exposition, we assume that there is an odd number of sensed values in total so that the Median is one element of the population.

Security Model We assume that the BS cannot be compromised. The BS uses a protocol such as $\mu Tesla$ [21] to authenticate broadcast messages. We also assume that each node X shares a pairwise key, K_X with the BS, which is used to authenticate the messages it sends to BS.

In this paper, we do not address outsider attacks – we can easily prevent unauthorized nodes from launching attacks by augmenting the aggregation framework with authentication and encryption protocols [21], [28].

We consider that the adversary can compromise a few sensor nodes (i.e., insiders) without being detected. If a node is compromised, all the information it holds will also be compromised. We use a Byzantine fault model, where the adversary can inject malicious messages into the network through the compromised nodes.

We observe that a compromised node might launch multiple potential attacks against a tree-based aggregation protocol, such as corrupting the underlying routing protocol, selective dropping, or a Denial of Service attack to prevent other nodes from receiving the messages from the BS.

However, in this paper we address only false data injection attacks where the goal of the attacker is to cause the BS to accept a false aggregate. To achieve this goal in an in-network Median computation algorithm (e.g. [9]), a compromised node X could either attempt to falsify its own sensed value, v_X, or the sub-aggregate X is supposed to forward to its parent. We note that as we are computing Median, by falsifying the local value a compromised node can only deviate the final estimate by one position, i.e., the impact of the *falsified local value attack* is very limited. Moreover, it is impossible to detect the falsified local value attack without domain knowledge about what is an anomalous sensor reading. On the other hand, the *falsified sub-aggregate attack*, in which a node X does not correctly aggregate the values received from X's child nodes, poses a large threat to an in-network Median computation algorithm; a compromised node X forwards to its parent a corrupted aggregate which falsely summarizes X's descendants' sensed values. We observe that by launching this attack, a single compromised node, which is placed near the root on the aggregation hierarchy, can deviate the final estimate of the Median by a large amount (e.g., in [9]).

Problem Description We aim to compute an approximate Median against the *falsified sub-aggregate attack*. In particular, our goal is to design the following two algorithms.

- Median computation and verification algorithm: This algorithm either outputs a valid approximate Median or it detects the presence of an attack. A value, $\hat{m}$, is considered to be a valid approximate Median if it is close to the exact Median, m, within the bound specified by the user. In particular, if the user-specified relative error bound is ϵ, the BS accepts an estimate $\hat{m}$ which satisfies the following constraint:
$$|rank(\hat{m}) - \frac{N+1}{2}| \leq \epsilon N \qquad (1)$$
where $rank(x)$ denotes the position of the value x in the sorted list of all the sensed values (the population elements), and N is the size of the population.
- Attack-resilient Median computation algorithm: If the above verification fails, our further aim is to compute an approximate Median in the presence of the attack.

We finally note that by launching a *falsified local value attack*, w compromised nodes can deviate $rank(\hat{m})$ in constraint (1) above by w positions, which makes the error bound of the final estimate of the Median to be $(\epsilon + w/N)$. However, given an upper bound on w, the user could adjust his input ϵ to finally meet the required bound.

Notation A list of notations used in this paper is given in Table II.

IV. Computing and Verifying an Approximate Median

The key elements of our approach are to compute a histogram of the sensor readings and then derive an approximate Median from the histogram. We collect a sample of sensed values from the network which is used to construct the histogram bucket boundaries. Before we present our

Symbol	Meaning
N	total number of nodes (or total number of sensed values)
S	sample size
E_i	value of i-th item in the sorted sample
K_X	symmetric key shared between node X and the BS
ϵ	error bound for the approximate Median
q_i	bucket boundary in histogram
$B_i \equiv [q_i, q_{i+1}]$	i-th bucket of the histogram
c_i	count of i-th bucket
v_X	sensed value of node X
$MAC(K_X, M)$	message authentication code of message M computed using key K_X
V_X	$= (X, v_X, MAC(K_X, v_X))$
$X \rightarrow Y$	X sends a message to Y
$X \rightarrow *$	X broadcasts a message
$X \Longrightarrow Y$	X sends a message to Y via multiple paths
$a_1 \mid\mid a_2$	concatenation of string a_1 and a_2
Δ	the maximum degree of the aggregation tree
g	number of groups in the attack-resilient algorithms
w	number of compromised nodes

TABLE II
NOTATIONS

scheme, we first discuss an approach to securely compute an approximation Median whose performance will be later compared with that of our scheme. Then, we present a histogram verification algorithm and finally describe our basic scheme.

A. GC Approach

One can suggest a scheme to securely compute an approximate Median using Greenwald et al.'s Median computation algorithm [9] in conjunction with Chan et al.'s verification algorithm [2]. We briefly discuss these two algorithms in Appendix I. In the first phase of GC approach, given the approximation error bound ϵ, we can run Greenwald et al.'s algorithm to compute a quantile summary. From the quantile summary we can derive an approximate Median $\hat{m}$ which is supposed to satisfy ϵ error bound. In the next phase, we can verify the actual error present in the estimate, $\hat{m}$, which might have been falsified by an attacker in the previous phase. To verify the error, Chan et al.'s verification algorithm can be applied to count the number of nodes in the network whose value is no more than $\hat{m}$.

The communication cost per node in this approach comes from the original protocols: that is $O(\frac{log^2 N}{\epsilon})$ for Greenwald et al.'s Median computation algorithm and $O(\Delta \log N)$ for Chan et al.'s verification scheme (considering Frikken et al.'s recent improvement [6]), where N is the number of nodes in the network, ϵ is the approximation error bound and Δ is the number of neighbors of a node.

B. A Histogram Verification Algorithm

We now present an algorithm for computing and verifying a histogram of sensed values, which is adapted from Chan et al.'s scheme [2] to compute and verify Sum aggregate. In Appendix I-B, we briefly describe Chan et al.'s Sum scheme.

Formally speaking, a histogram is a list of ordered values, $\{q_0, q_1, \ldots, q_i, \ldots\}$, where each pair of consecutive values (q_i, q_{i+1}) is associated with a count c_i which represents the number of population elements, v_j, such that $q_i < v_j \leq q_{i+1}$. We refer such an interval, (q_i, q_{i+1}) as bucket B_i with boundaries q_i and q_{i+1}.

As noted in [2], the Sum scheme can be adapted to count the cardinality of a subset of nodes. Here, we apply Sum aggregate to count how many sensor readings belong to each histogram bucket. To do so, we require each node X to contribute 1 to the count of its corresponding bucket (the bucket X's sensed value, v_X, lies within) in the histogram while we compute the total count for each bucket. Like Chan et el.'s scheme, the histogram verification scheme takes four epochs to complete: query dissemination, aggregation-commit, commitment-dissemination, and result-checking.

After an aggregation tree is constructed in the query broadcast epoch, each node X's message in the aggregation-commit epoch looks like $< \beta, c_1, c_2, ..., c_b, h >$, where β is the number of nodes in X's subtree, b is the number of buckets in the histogram, each c_i represents the count for the bucket B_i, i.e $\beta = \sum_i c_i$, and h is an authentication field. Note that for each bucket count c_j all of the other bucket counts together act as a complement, i.e. $c_j + \sum_{i \neq j} c_i = \beta$. A leaf node X whose sensed value, v_X, lies within the bucket B_j sets the fields in its message as follows: $\beta = 1$, $c_j = 1$, $c_i = 0$ for all $i \neq j$, and $h = X$. If an internal node X whose value v_X lies within the bucket B_j receives messages $u_1, u_2, ..., u_t$ from its t child nodes, where $u_k =< \beta_k, c_1^k, c_2^k, ..., c_b^k, h_k >$, then X's message $< \beta, c_1, c_2, ..., c_b, h >$ is generated as follows: $\beta = \sum \beta_k + 1$, $c_1 = \sum c_1^k$, $c_2 = \sum c_2^k$, ..., $c_j = \sum c_j^k + 1$, ..., $c_b = \sum c_b^k$, and $h = H[\beta || c_1 || c_2 || ... || c_b || u_1 || u_2 || ... || u_t]$, where H is a hash function. The above messages along the aggregation hierarchy logically build a commitment tree which enables the authentication operation in the next phase. Once the base station receives the final commitment, it verifies the coherence of the final counts, c_1, c_2,..., c_b, with the number of nodes in the network, N. In particular, the BS performs the following sanity check: $\sum c_i = N$. A simplified version of the aggregation-commit phase is illustrated in Figure 1 with an example of a small network.

The commitment-dissemination epoch and the result-checking epoch are straightforward extensions of those in Chan et al.'s Sum scheme. During commitment-dissemination epoch, the final commitment is broadcast by the BS to the network. In addition, each node X receives from its parent node all of the *off-path values* up to the root relative to X's position on the commitment tree. The aim of the commitment dissemination phase is to let each single node know that its own value has been considered in the final histogram. The message containing the *off-path values* received by a node is larger compared to that in the Sum scheme because each off-path value contains b counts when a histogram with b buckets is computed. In the result-checking epoch, the BS receives a compressed authentication

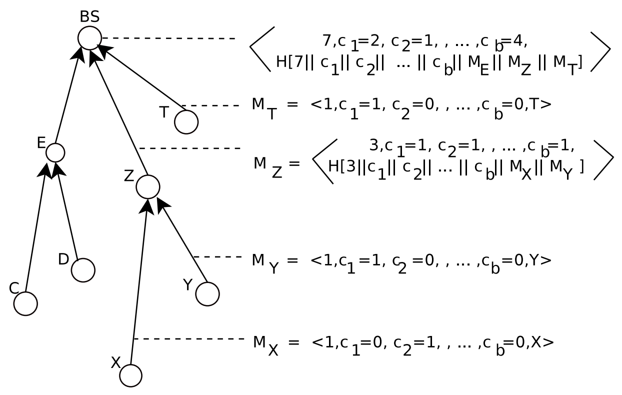

BS
T
E
Z
D
C
Y
X
7,c 1=2, c2=1, , ... ,c b=4, H[7|| c1|| c2|| ... || cb|| ME|| MZ || MT]
MT = <1,c1=1, c2=0, , ... ,c b=0,T>
MZ = 3,c1=1, c2=1, , ... ,c b=1, H[3||c1|| c2|| ... || cb|| MX|| MY]
MY = <1,c1=1, c2 =0, , ... ,cb=0,Y>
MX = <1,c1=0, c2=1, , ... ,c b=0,X>

Fig. 1. The aggregation-commit phase in histogram verification: In this example, v_X lies in bucket B_2, v_Y lies in bucket B_1, and v_Z lies in the last bucket B_b. While leaf nodes X and Y set the corresponding bucket count to 1, the internal node Z aggregates the messages M_X and M_Y and generates its message M_Z which also considers its own value v_Z. The first field in M_Z is 3, which represents the number of nodes in Z's sub-tree; the bucket counts are aggregated accordingly; the last field is the hash value for this aggregation operation. For the remaining nodes, v_T lies in bucket B_1, and v_C, v_D and v_E lie within bucket B_b.

code from all of the nodes which enables to verify if each node confirmed that its value has been considered in the final histogram.

As in Chan et al.'s Sum scheme, the main cost of this protocol is due to the dissemination of the off-path values to individual nodes. To reduce this overhead, following the recent improvement proposed by Frikken et al. [6], we use a balanced commitment tree as an overlay on the physical aggregation tree. Due to space constraint, we do not discuss the details in this paper. If a histogram with b buckets is considered, each off-path message is b times larger than that in the Sum scheme, which makes the worst case node congestion in this protocol to be $O(b\Delta \log N)$.

C. Our Basic Protocol

We now describe our basic protocol to compute and verify an approximate Median. The basic protocol has two phases: sampling phase, and histogram computation and verification phase. Below we discuss these phases in detail.

While collecting a sample of population values is highly energy-efficient compared to collecting all the values, we will later show that a sample can act as a good representative of the whole population. Also, we will show that only the sample size determines the performance of our algorithm, irrespective of the size of the population.

1) Sampling: In this phase, the BS collects a uniform sample of the sensed values from the network. To do so, the BS broadcasts the following message:

$$BS \rightarrow * : \langle SAMPLE, seed \rangle.$$

The sample request coming from the BS is broadcast in a hop-by-hop fashion and the nodes arrange themselves in a ring topology; nodes at the first hop from the BS belong to the first ring and so on. A node X considers the previous hop nodes as parents from which X has received the query message. Note that in the sampling phase, we do not use a tree topology, which is, however, used in the histogram computation and verification phase.

We assume that there is a public hash function $F : \{ID, seed\} \rightarrow \{0, 1, ..., t-1\}$, where ID represents the node id, $seed$ is the nonce broadcast during the query, and t is a positive integer which acts as a design parameter as discussed later. Each node, say X, hearing the query message applies the hash function $F(X, seed)$. If the resulting value is 0, then its sensed value, v_X, is considered to be one element in the sample. In that case, X computes $MAC(K_X, v_X)$ and sends the message $V_X = (X, v_X, MAC(K_X, v_X))$ to its parents. In addition to that, if X has child nodes, X also forwards the sample values and corresponding MACs received from the child nodes, say $V_{Z_1}, ..., V_{Z_c}$. The whole message from X looks as follows:

$$X \rightarrow Parents(X) : \langle V_X, V_{Z_1}, ..., V_{Z_c} \rangle.$$

When the BS receives all these messages, it verifies the corresponding MACs and outputs the list of values that are legal items of the sample. Note that the $seed$ is used in order to have different samples in different runs. Basically, the hash function is used to uniformly divide all of the nodes among t groups; the nodes belonging to the first group (i.e., output of the hash function is 0) are considered to constitute the sample. If the required sample size is S, one might set $t = N/S$. It is expected that this hash function uniformly maps N elements into t groups. To increase the chance that finally a sample of size no less than S will be collected, we could increase the number of groups from t to kt, and output the sample from more than k groups (e.g., $k + 1$ groups).

2) Histogram computation and verification: Once the BS obtains the sample, it sorts the items in ascending order. Then, the following steps are performed: (i) computing histogram boundaries, (ii) computing and verifying the buckets' counts, and (iii) estimating the Median.

i) Computing Histogram Boundaries

We consider the number of buckets, b, as a parameter. In Section V-B we discuss how to choose this parameter. In this step, we equally divide the sample items into b buckets. We denote the buckets as $B_i = [q_i, q_{i+1}]$, $0 \leq i \leq b-1$, where $q_0 = -\infty$, $q_i = E_{\lceil \frac{S}{b} \rceil i}$ and $q_b = +\infty$, as shown in Figure 2. E_j represents the value of j-th item in the sample sorted according to the value, with j varying from 1 to S.

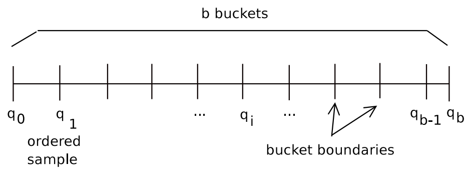


Fig. 2. Computing histogram boundaries: The histogram boundaries are computed using the sample collected in the previous phase.

ii) Computing and verifying the buckets' counts

To compute the bucket counts, the BS and the sensor nodes run the histogram verification protocol described in Section IV-B. If there is no attack present in the network, at the

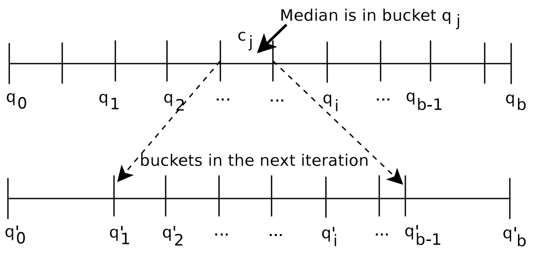


Fig. 3. Splitting the bucket: If the bucket j, which contains the Median has more than $2\epsilon N$ elements, the bucket is split in order to meet ϵ approximation error bound.

end of this step the BS knows the number of nodes that belong to each bucket in the histogram. However, an attacker node can cause this verification to fail, and in that case, the protocol terminates returning a message, "attack detected". We discuss an attack-resilient solution in Section VI.

iii) Estimating the Median

Assuming that the verification in the previous step succeeds, we have the bucket counts $c_0, \ldots, c_{b-1}$ for the corresponding buckets. Our aim is now to find the bucket which contains the Median. In particular, we find j such that the following three constraints are satisfied:

$$e_l + c_0 + c_1 + \ldots + c_{j-1} \quad < \quad \frac{N+1}{2} \tag{2}$$

$$c_0 + c_1 + \ldots + c_j \quad \geq \quad \frac{N+1}{2} \tag{3}$$

$$c_j \quad \leq \quad 2\epsilon N \tag{4}$$

where e_l is equal to 0 in the first iteration and updated as follows in other cases. We first find j such that the first two in-equalities are satisfied. Then, we check if the above j also satisfies in-equality (4). Note that if in-equality (4) is satisfied, then it is guaranteed that either q_j or q_{j+1} is ϵN away from the exact Median, which is reported as our final estimate. If the in-equality (4) is not satisfied, we further split j-th bucket equally into b sub-buckets. The new boundaries are updated as follows: $q_0' = q_0$, $q_1' = q_j$, …, $q_{b-1}' = q_{j+1}$, and $q_b' = q_b$. Bucket splitting is illustrated in Figure 3. The variable e_l is updated as $e_l = e_l + \sum_{i=0}^{j-1} c_i$. We iterate steps (ii) and (iii) until the in-equality (4) is satisfied. During the above iteration, if we reach a point where bucket j does not contain any sample items to split further, we stop after returning a message, "more sample items to be collected". We note that modifying the right-hand side of inequalities (2) and (3), other quantiles besides the Median can be approximately computed.

V. Security and Performance Analysis of Our Basic Protocol

A. Security Analysis

A node X which is selected in the sample sends an authentication code, $MAC(K_X, v_X)$, to the BS so that the BS can authenticate the sensed value v_X, where K_X is the pairwise key of X shared with the BS. An attacker node that is not legally selected by the hash function cannot inject a false value in the sample without being detected.

Moreover, because multipath routing scheme is used in the sampling phase, it is highly likely that we will be able to collect a sample, even if a few compromised nodes do not forward any messages. To establish the above observation, we consider a simplistic scenario. Let us assume that there are w compromised nodes in total and they are randomly distributed in the network. So, the probability of a randomly selected node to be compromised is w/N, where N is the total number of nodes. We also assume that each node has at least θ number of parents and the farthest node is d hops away from the BS. We assume that unless all of the parents of a node X are not compromised, X's message will reach the next hop – the probability that this happens is $(1 - (w/N)^\theta)$. So, in the presence of the dropping attack by the compromised nodes, the probability that a sample item finally reaches the BS is at least $(1 - (w/N)^\theta)^d$. As an example, with $N = 1000$, $w = 50$, $\theta = 3$, and $d = 15$, this probability is 0.998.

Like Chan et al's scheme, our histogram computation protocol is able to detect the *falsified sub-aggregate attack*, i.e., the attacker cannot modify the count of any bucket in the histogram without being detected. So, given that the verification succeeds, it is guaranteed that the final estimate is an ϵ-approximate Median.

B. Performance Analysis

In this section, we analyze the communication complexity of our basic protocol. In the first phase (i.e. during the sampling phase), the worst case node congestion occurs when a node (e.g. a node close to the BS) is required to forward all of the S samples coming from the network. So, the maximum node congestion in the sampling phase is $O(S)$. The cost of the second phase, which computes and verifies the histogram is $O(b\Delta logN)$, where b is the number of buckets, Δ is the degree of the aggregation tree, and N is number of nodes in the network. Note that our protocol iterates the second phase until the required approximation error bound is met. Our goal is to minimize the total cost of all iterations.

The second phase goes to the next iteration if the bucket b_j in which the Median lies contains more than $2\epsilon N$ population elements. We then further divide j-th bucket into b sub-buckets. We observe that further division is not possible if bucket j no longer contains a sample item, which is bound to happen within at most $\log_b S$ iterations. If bucket j still contains more than $2\epsilon N$ population elements, we cannot do anything further but collect more sample items.

To make an estimate of the sample size, S, so that we do not need to perform an extra sampling phase in most of the cases, we present the following lemma whose proof can be found in Appendix II.

Lemma 5.1: The probability that more than pN population elements lie between two consecutive items of a sorted uniform sample of size S is $\phi(S,p) = (1-p)^{S-1}$, where N is the population size.

As an example, from Lemma 5.1, we see that $\phi(S, 2\epsilon) < 2.95 \times 10^{-5}$ for $S \geq 100$ and $\epsilon \geq 0.05$. This implies that if the user requires $\epsilon \geq 0.05$ and we use $b = 10$ buckets with $S = 100$, we require at most $log_b(S) = 2$ iterations to report the Median with probability $(1 - 2.95 \times 10^{-5}) \approx 1$. It is interesting to note that this result does not depend on the population size, N.

Now, to measure the trade-off between the number of buckets, b, and the number of iterations, which together determine the total cost of the algorithm, we present the following lemma whose proof can be found in Appendix II.

Lemma 5.2: The probability that more than $\gamma p N$ ($\gamma > 1$, $0 < p < 1$, $\gamma p < 1$) population elements lie between the minimum and the maximum of pS consecutive sample items of a sorted sample of size S is

$$\xi(S, p, \gamma) = \sum_{i=0}^{pS-1} \binom{S-1}{i} (\gamma p)^i (1 - \gamma p)^{S-1-i} \tag{5}$$

where N is the population size.

1) Number of buckets vs. number of iterations: If we use $b = \frac{\gamma}{2\epsilon}$ buckets, which is of $O(\frac{1}{\epsilon})$, where γ is a constant greater than 1 and ϵ is the required error bound, then each bucket contains $\frac{2\epsilon}{\gamma} S$ sample items during the first iteration. So, the expected number of population elements in one bucket is $\frac{2\epsilon}{\gamma} N$. In Lemma 5.2, putting $p = \frac{2\epsilon}{\gamma}$, we can compute the probability that more than $\gamma \cdot \frac{2\epsilon}{\gamma} \cdot N = 2\epsilon N$ population elements fall in a bucket. As Expression (9) is a decreasing function of γ, by choosing the appropriate γ, we can make the above probability close to zero. As an example, for $\gamma = 2$, we observe that with sample size S such that $\epsilon S \geq 5$, (i.e., each bucket contains no less than 5 sample items in the first iteration) the above probability is less than 0.02 for all ϵ. That means, in this setting, our protocol ends in one iteration in 98% cases. Finally, considering the cost of the histogram verification scheme, we see that the total cost of all iterations per node, when $b = O(\frac{1}{\epsilon})$, is $O(\frac{1}{\epsilon} \Delta \log N)$, where Δ is the degree of the aggregation tree.

On the other hand, if we use $b = O(1)$ buckets and equally divide the sample items in b buckets in each iteration, then, after $\log_b(\frac{\gamma}{2\epsilon})$ iterations, each bucket will contain no more than $\frac{2\epsilon}{\gamma} S$ sample items. So, as shown above, with the appropriate γ chosen, it is almost certain that our algorithm will end at this point. Thus, considering the cost to compute and verify the histogram in each iteration, the total cost of all iterations, when $b = O(1)$, is $O(\log_b \frac{1}{\epsilon} \cdot b \cdot \Delta \log N)$, where Δ is the degree of the aggregation tree.

2) Betting on Median position: We observe that with the sorted sample items being equally divided into b buckets, the probability of a bucket containing the Median is not the same for all buckets. The Median is more likely to occur with the buckets which are in the middle of the sorted sample, compared to buckets at either end. Here we establish the above observation and exploit it to set a better trade-off between the number of buckets and the number of iterations.

Essentially, rather not dividing the whole set of sorted sample items into b buckets equally, we take a greedy approach – we divide a small fraction of sample items in the middle into $(b - 2)$ buckets and place the rest of the sample items at either end into one bucket each, as shown in Figure 4. If we are lucky, after one iteration we find that the Median lies in one of the smaller $(b - 2)$ buckets and thus our algorithm converges faster with a given number of buckets. We consider δ, $0 \leq \delta \leq 1$ as a design parameter, which represents the probability that the Median actually lies in one of the end buckets, i.e., with probability $(1 - \delta)$ the Median falls in one of the $(b - 2)$ buckets in the middle.

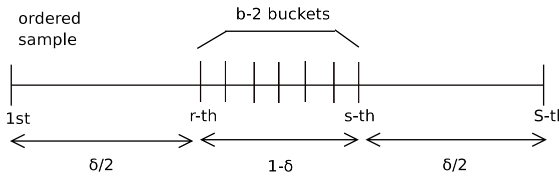


Fig. 4. We divide a small fraction of sample items in the middle into $(b - 2)$ buckets and place the rest of the sample items at either end in one bucket each.

We can compute one positive integer r so that the Median lies within r-th and $s = (S - r + 1)$-th item in the sorted sample with a high probability. In particular, for a given δ, r can be found using the formula given in [5], which is as follows:

$$1 - \delta = 2^{-S} \sum_{i=r}^{S-r} \binom{S}{i}. \tag{6}$$

Computing r using the above formula is closely related to the *sign test*, so the table by MacKinnon [12] can be used. We can also simplify the above formula considering that a binomial distribution can be approximated to a normal distribution. For $S > 10$, an approximate formula would be

$$r = \frac{S}{2} - \frac{1}{2} u_\delta \sqrt{S}, \tag{7}$$

where u_δ is the upper $\frac{1}{2}\delta$ significance point of a unit normal variate.

Finally, we construct the histogram with b buckets by dividing the sample items which are in the interval $[r, S - r + 1]$ into $(b - 2)$ buckets and adding one bucket each to both ends.

We observe that the larger the value we assign for δ, the faster we reduce the search space to find the Median (i.e., the number of sample items to consider in the next iteration), if we are lucky. Of course, if we are unlucky, we need to consider one of the larger end buckets in the next iteration. So, the question becomes what is the optimum value for δ to use, so that our algorithm converges with the fastest speed on average. Our aim here is to minimize the average search space after one iteration. If the Median does lie within one of the $b-2$ central buckets, then the search space for the next iteration is the same as the number of sample items in one central bucket, which is $\frac{u_\delta \sqrt{S}}{b-2}$. This happens with probability $1 - \delta$; otherwise, we have to consider one of the larger end buckets (i.e. the leftmost or the rightmost one) in the next iteration. The width of such an interval is $\frac{S}{2} - \frac{1}{2} u_\delta \sqrt{S}$. So, the optimization goal is to minimize the following expression, which represents the average search space after one iteration:

$$(1-\delta)(\frac{u_\delta\sqrt{S}}{b-2}) + \delta(\frac{S}{2} - \frac{1}{2}u_\delta\sqrt{S}) \quad (8)$$

Given S and b, we can numerically determine the value of δ for which the above expression attains the minimum value.

VI. Attack-resilient Median Computation

Although our basic protocol, discussed in Section IV-C, detects *falsified sub-aggregate attack*, it fails to output an estimate of the Median in the presence of the attack. To address this problem, here we propose an extended approach so that we can compute an approximate Median even in the presence of a few compromised nodes.

We design the new approach based on the *divide and conquer* principle. We divide the network into several groups of nodes, which introduces resilience against the above attack. We run the verification algorithm individually for each group, which we call *intra-group verification*. Basically, we localize the attacker nodes to specific groups, i.e. we detect which groups are corrupted and which are not. Even if a few groups are corrupted, we still compute an estimate of the Median considering the valid groups. We do not assume that the groups have similar data distribution, which is the assumption exploited in the outlier detection algorithm used in SDAP [27].

We may employ different grouping techniques based on node's geographic location or node IDs. We may also use grouping technique which is based on the nodes' positions on the aggregation tree. Once the group aggregate is computed, the group leader send it directly to the BS; to avoid having any node in the middle to drop group aggregates, we use a multipath routing mechanism. Due to space constraint, only geographical grouping technique is discussed here.

Also, we may exploit the robustness property of the Median computation to determine the maximum amount of error that can be injected by a given number of corrupted nodes, even if we do not perform the intra-group verification. In the Appendix III we estimate this error while we leave it to the network user to fix the tradeoff between the error bound and the overhead due to intra-group verification.

A. Geographical Grouping

We assume that the BS has knowledge of the location of the nodes and each node knows its own location. The network is divided into several rectangular regions, where each region is identified by a pair of geographical points. The number of regions, g, and the location of the regions are selected considering a few factors. As one criterion, the regions might be chosen in such a way that an equal number of nodes belong to each group – if a region has lower node density, it is likely that it will be of larger geographical size. In addition, if the BS expects that a part of the network is more likely to be under attack, it may prefer to form smaller regions in that area to better localize the attacker. Finally, g

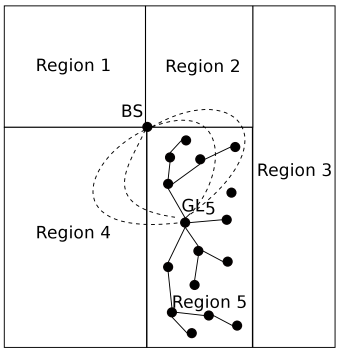


Fig. 5. Geographical grouping: The network is divided into several regions where each region has a group leader (GL_i). The GL_i sends the region aggregate to the BS by multiple paths.

rectangular regions are specified by g pairs of diametrically opposite points, $(x1_i, y1_i)$, $(x2_i, y2_i)$, where $1 \leq i \leq g$. For each group i, BS also selects a node to be the group leader, GL_i. An example of this grouping is shown in Figure 5.

Once the histogram boundaries are computed using the collected sample (as in our basic protocol), the BS initiates the histogram verification procedure. The BS sends a request to the corresponding group leaders with the necessary information to identify the regions. Receiving the request, a local aggregation tree is constructed which comprises of all of the nodes in the region with GL_i as the root. Then, the group histogram is computed locally and sent to the BS. If compromised nodes are present in a few groups, the BS will be able to identify the corrupted groups. The BS accepts aggregates from only those regions, which passed the verification. The BS may further split the region which contains an attacker node and run the protocol again in the sub-regions. Eventually, this splitting can be iterated until the attacker node is identified or the percentage of verified values satisfies the BS (e.g., when the verified groups correspond to the 95% of the nodes). Below we discuss the attack-resilient histogram computation and verification algorithm.

1) Algorithm description: The nodes in each region locally perform the histogram computation and verification protocol described in Section IV-B with the group leader acting as an agent of the BS in the corresponding group. To make the group leader GL_i an eligible agent of BS for group i, we need a few additional communication between GL_i and the BS. Below we focus on these additional messages skipping the detailed description of rest of the protocol, which can be found in Section IV-B. The messages exchanged between GL_i and the BS are authenticated using their pairwise key. To improve readability, we do not show these authentication fields in the messages below.

a) Query dissemination: BS initiates the query by sending to each group leader GL_i via multiple paths the following message which contains the coordinates of the corresponding region:

$$BS \Longrightarrow GL_i : \langle (x1_i, y1_i), (x2_i, y2_i), GL_i \rangle.$$

In each region, the group leader, GL_i, broadcasts the received query message to its neighbor nodes, which again broadcast the same message, and so on. It is a scoped broadcast, i.e., if a node whose coordinate is outside of the corresponding region receives the message, it simply drops the message. During the query broadcast, a regional aggregation tree is formed with GL_i as the root, similarly as in the TAG [13] algorithm. The query message also contains required $\mu TESLA$ information (not shown above) so that each node in the region can authenticate the query.

After the query is disseminated, the nodes in each region locally perform the histogram computation and verification protocol described in Section IV-B.

b) Aggregation-commit phase: After the group leader GL_i receives the aggregated value from the nodes in group i, it forwards the following message to the BS:

$$GL_i \Longrightarrow BS : \langle GL_i, agg_i, commit_i \rangle,$$

where agg_i represents the computed histogram of group i, and $commit_i$ is the root of the *commitment tree* of region i.

c) Commitment-dissemination phase: The BS checks if the number of nodes in the computed histogram of the group is same as the total number of nodes in that group. If yes, it sends to GL_i the $\mu TESLA$ authentication information, $\mu T(commit_i)$. So, when GL_i broadcasts $commit_i$ in group i, each node can authenticate the message.

$$BS \Longrightarrow GL_i : \langle GL_i, \mu T(commit_i) \rangle.$$

d) Result-checking phase: Each node checks if its value is incorporated in the computed histogram. If yes, node X sends a MAC over an "OK" message, $MAC(K_X, OK)$, which gets XOR-ed with other nodes' similar messages on their way to the group leader. Once GL_i receives the compressed OK message, say OK_i, from the nodes in its group, it forwards this message to the BS via multiple paths.

$$GL_i \Longrightarrow BS : \langle GL_i, OK_i \rangle.$$

As the BS knows which nodes belong to which group, it can verify OK_i messages and hence can identify valid group aggregates.

2) Security analysis: We recall from Section that the histogram computation and verification protocol, when executed on the whole network, can detect if there is any falsified sub-aggregate attack. That means, if a malicious node X fabricates the histogram of its sub-tree or if X simply does not participate in the protocol, the BS can detect the attack and flags that the computed histogram is corrupted. Our intra-group verification protocol is different from the basic one only in the following aspects: (i) the histogram of the whole network is considered as the aggregate of the group histograms and each group histogram is computed and verified individually, (ii) the group leader, GL_i exchanges a few messages with the BS, discussed in Section VI-A.1, which enable GL_i to play the role of BS in group i.

The messages exchanged between GL_i and the BS are routed via multi-paths so that they reach the destination even if an attacker node in the middle drops these messages. The communication between GL_i and the BS is also authenticated with their pairwise key. Moreover, GL_i receives from the BS the $\mu Tesla$ authentication information for the messages which are to be broadcast in the group, e.g., the query message and the $commit_i$ message. So, assuming a node X knows its location, X can securely determine to which group it belongs and the ID of the group leader, and X can also authenticate the query and the $commit_i$ message endorsed by the BS.

After the BS receives the group histogram from group i, (i.e., the agg_i message) the BS verifies if the number of nodes reflected in the group histogram is same as the number of nodes in the group. Also, after receiving the OK_i message from group i, the BS verifies if this message correctly represents, in compressed form, the OK message of all the nodes in group i. The above two checks enable the BS to correctly identify the corrupted groups, if any.

3) Performance analysis: On average, the number of nodes in one group is $N' = \frac{N}{g}$, where the network is divided into g groups. So, the worst case node congestion inside one group for running the histogram verification algorithm is $O(b \cdot \Delta \cdot \log N')$, where b is the number of buckets in the histogram and Δ is the number of neighbors of a node on the aggregation tree. Considering the analysis given in Section V-B.1, with $b = O(\frac{1}{\epsilon})$, the worst case communication overhead per node is $O(\frac{1}{\epsilon} \cdot \Delta \cdot \log N')$. In addition, a node needs to forward the messages exchanged between the group leaders and the BS, which is of $O(g)$ communication overhead in the worst case.

VII. Simulation Results

In this section, we report on a simulation study that examined the performance of our basic protocol discussed in Section IV. Recall that, in the first phase, we collect a sample of sensed values from the network, and the performance of the rest of the protocol depends on the quality of this sample. The goal of the simulation experiments reported below is to study the impact of the sample on the overall performance of the Median computation protocol. In particular, we verify the results we obtained via analysis, in Section V-B, about the inter-relationship among parameters, such as error bound ϵ, sample size S, and the number of buckets b in the histogram.

Through simulation we do not evaluate the overhead of in-network communications in our protocol. The analytical results on the communication overhead of the sampling phase and the histogram computation and verification phase are discussed in Section V-B.

A. Simulation Environment

In our basic setup, the network size is 1,000 nodes. We also vary the network size to show that it does not have a significant impact on our sampling-based approach. In our simulation, the typical value we take for the ϵ error bound varies from 5% to 15%.

Each node has one sensed value, while our goal is to compute an approximate Median. We use the method of independent replications as our simulation methodology. Each simulation experiment was repeated no less than 1000 times with different seeds.

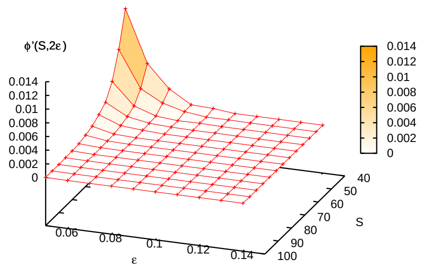
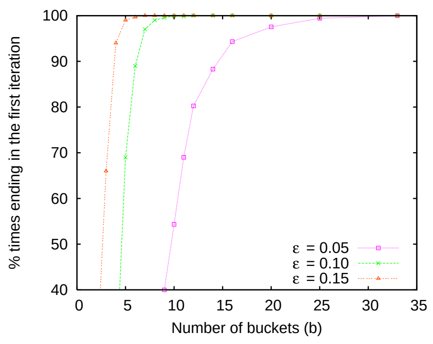
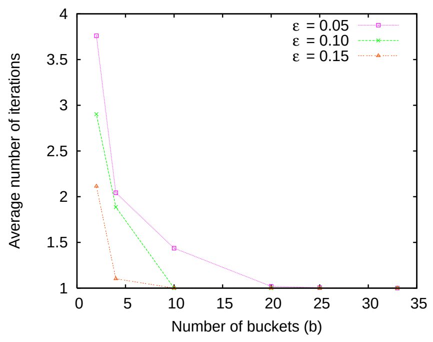


Fig. 6. Computing the chance that we need to collect more sample items: Given an ϵ, we choose a sample size so that the probability that we need to redo the sampling is close to zero.



(a) % Times ending in one iteration



(b) Average number of iterations

Fig. 7. The number of iterations vs. the number of buckets: if the number of buckets is $O(\frac{1}{\epsilon})$, it is highly likely that our algorithm ends in one iteration.

B. Results and Discussion

Here, we discuss the results obtained in our simulations. We observe that 95% confidence interval of all the quantities on the following plots are within 5% of the reported value.

What is the chance that one sampling phase is not enough? In Lemma 5.1, we analytically computes this probability which we evaluate via simulation here. For each pair (S, ϵ), we collect a sample of size S and we compute the number of time, τ there are more than $2\epsilon N$ elements between the two consecutive sample items containing the Median. The total number of runs performed is 1,000,000. The resulting $\phi'(S, 2\epsilon)$, which is the observed approximation of $\phi(S, 2\epsilon)$, is plotted in Figure 6. It is worth noticing that the value of $\phi'(S, 2\epsilon)$ is less than 4×10^{-5} for $\epsilon > 0.05$ when the sample size S is more than 95. In fact, as expected, for a given ϵ, an increase of the value of S decreases $\phi'(S, 2\epsilon)$. Finally, we verify that $\phi'(S, 2\epsilon)$ does not change significantly (not shown in the figure) even if the population size, N, is larger.

Number of buckets vs. Number of iterations. In Section V-B, we analyzed the dependence of the number of iterations of our algorithm on the number of buckets chosen, which we validate here via simulations. First, we estimate the number of buckets required to end our protocol in one iteration in most cases. Figure 7(a) illustrates the % of cases our protocol ends in the first iteration. The figure confirms our analysis that, for considering $\gamma = 2$, if we use more than $\frac{1}{\epsilon}$ buckets (i.e., 20, 10, 7 buckets for $\epsilon = 0.05, 0.10, 0.15$, respectively), it is highly likely that we need just one iteration. Finally, Figure 7(b) shows the average number of iterations required using different number of buckets, where $\epsilon = 0.05$ and $S = 100$. This validates our analysis that the average number of iterations is $O(log_b(\frac{1}{\epsilon}))$ when b buckets are used.

Betting on the Median position. In Section V-B.2 we described an optimization based on the observation that the Median lies with higher probability in the buckets that are in the center of the sorted sample. We studied how different choices of δ determines the average number of iterations for a given number of buckets. Figure 8 shows the average number of iterations for different values of δ while we use $\epsilon = 0.05$ and $S = 100$.

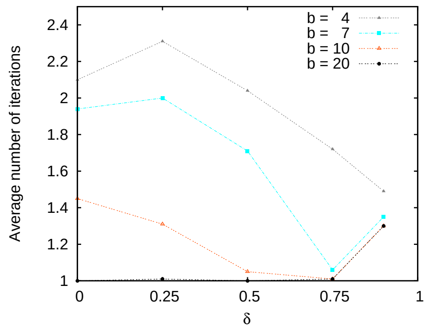


Fig. 8. Proper choice of δ reduces the number of iterations needed.

VIII. Conclusion

While researchers already addressed the problem of securely computing aggregates such as Sum, Count, and Average, to the best of our knowledge, there is no prior work on secure computation of the Median. However, it is widely considered that the Median is an important aggregate. In this paper, we proposed a protocol to compute an approximate Median and verify if it is falsified by an attack. Once the protocol is executed, the base station either possesses a valid approximate Median or it has detected an attack. Further, we proposed an attack-resilient algorithm to compute the Median even in the presence of a few compromised nodes. The evaluation via both analysis and simulation shows that our approach is efficient and secure.

References

[1] D. Barbará, W. DuMouchel, C. Faloutsos, P. J. Haas, J. M. Hellerstein, Y. E. Ioannidis, H. V. Jagadish, T. Johnson, R. T. Ng, V. Poosala, K. A. Ross, and K. C. Sevcik. The new jersey data reduction report. *IEEE Data Eng. Bull.*, 20(4):3–45, 1997.

[2] H. Chan, A. Perrig, and D. Song. Secure hierarchical in-network aggregation in sensor networks. In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, pages 278–287, 2006.

[3] J. Considine, F. Li, G. Kollios, and J. Byers. Approximate aggregation techniques for sensor databases. In *ICDE '04: Proceedings of the 20th International Conference on Data Engineering*, pages 449–460, 2004.

[4] G. Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. In *LATIN '04: Proceedings of Latin American Theoretical Informatics*, pages 29–38, 2004.

[5] H. A. David and H. N. Nagaraja. *Order Statistics, 3rd Edition*. John Wiley & Sons Inc., 2003.

[6] K. Frikken. An efficient integrity-preserving scheme for hierarchical sensor aggregation. In *WiSec '08: Proceedings of the First ACM Conference on Wireless Network Security*, to appear.

[7] W. F. Fung, D. Sun, and J. Gehrke. Cougar: the network is the database. In *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 621–621, 2002.

[8] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *SIGMOD '01: Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, pages 58–66, 2001.

[9] M. B. Greenwald and S. Khanna. Power-conserving computation of order-statistics over sensor networks. In *PODS '04: Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 275–285, 2004.

[10] http://www.xbow.com. Crossbow technology inc., 2008.

[11] R. Jain and I. Chlamtac. The P^2 algorithm for dynamic calculation of quantiles and histograms without storing observations. *Commun. ACM*, 28(10):1076–1085, October 1985.

[12] W. J. MacKinnon. Table for both the sign test and distribution-free confidence intervals of the median for sample sizes to 1,000. *Journal of the American Statistical Association*, 59(307):935–956, 1964.

[13] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: a tiny aggregation service for ad-hoc sensor networks. In *OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation*, pages 131–146, 2002.

[14] G. S. Manku, S. Rajagopalan, and B. G. Lindsay. Approximate medians and other quantiles in one pass and with limited memory. *SIGMOD Rec.*, 27(2):426–435, 1998.

[15] R. C. Merkle. A digital signature based on a conventional encryption function. In *CRYPTO '87: A Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology*, pages 369–378, 1988.

[16] J. R. Microclimate and V. R. Sensing. http://www.cens.ucla.edu.

[17] J. I. Munro and M. S. Paterson. Selection and sorting with limited storage. *Theoretical Computer Science*, (12):315–323, 1980.

[18] S. Nath, P. B. Gibbons, S. Seshan, and Z. R. Anderson. Synopsis diffusion for robust aggregation in sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 250–262, 2004.

[19] H. M. on Great Duck Island. http://www.greatduckisland.net/.

[20] B. Patt-Shamir. A note on efficient aggregate queries in sensor networks. *Theoretical Computer Science*, 370(1-3):254–264, 2007.

[21] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar. SPINS: Security protocols for sensor networks. *Wireless Networks*, 8(5):521–534, Sept. 2002.

[22] V. Poosala, P. J. Haas, Y. E. Ioannidis, and E. J. Shekita. Improved histograms for selectivity estimation of range predicates. *SIGMOD Rec.*, 25(2):294–305, 1996.

[23] T. F. Project. http://firebug.sourceforge.net.

[24] S. Roy, S. Setia, and S. Jajodia. Attack-resilient hierarchical data aggregation in sensor networks. In *SASN '06: Proceedings of the fourth ACM workshop on Security of ad hoc and sensor networks*, pages 71–82, 2006.

[25] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri. Medians and beyond: new aggregation techniques for sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 239–249, 2004.

[26] D. Wagner. Resilient aggregation in sensor networks. In *SASN '04: Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks*, pages 78–87, 2004.

[27] Y. Yang, X. Wang, S. Zhu, and G. Cao. SDAP: a secure hop-by-hop data aggregation protocol for sensor networks. In *MobiHoc '06: Proceedings of the 7th ACM international symposium on Mobile ad hoc networking and computing*, pages 356–367, 2006.

[28] S. Zhu, S. Setia, and S. Jajodia. Leap: efficient security mechanisms for large-scale distributed sensor networks. In *CCS '03: Proceedings of the 10th ACM conference on Computer and communications security*, pages 62–72, 2003.

Appendix I
Preliminaries

Here, we briefly present Greenwald et al.'s approximate Median algorithm and Chan et al.'s verification algorithm, which we often refer in our paper.

A. Greenwald et al.'s Approximate Median Algorithm

This algorithm [9] is based on a summarization technique which represents a set of sensor readings by a *quantile summary*. From a ϵ-approximate quantile summary, we can derive an arbitrary quantile of the data set satisfying ϵ-approximation error bound. In particular, an ϵ-approximate quantile summary for a data set A is an ordered set $Q = \{\alpha_1, \alpha_2, ..., \alpha_l\}$ such that (i) $\alpha_1 \leq \alpha_2 ... \leq \alpha_l$ and $\alpha_i \in A$ for $1 \leq i \leq l$, and (ii) $rank(\alpha_i + 1) - rank(\alpha_i) < 2 \cdot \epsilon \cdot |A|$.

Also, given two quantile summaries, Q_1 and Q_2, which represent two disjoint sets of sensed values, A_1 and A_2, respectively, we can aggregate them into a single quantile summary Q which represents all the values in $A = A_1 \cup A_2$. To aggregate two quantile summaries, we need two operations: *combine operation* and *prune operation*. The output of the combine operation from the quantile summaries Q_1 and Q_2 is a sorted list, Q', which is the union of Q_1 and Q_2. As a result, the size of Q' is the sum of the sizes of the original summaries Q_1 and Q_2. To keep the size of the quantile summary within limits, we apply the prune operation on Q' to determine a quantile summary Q of a constant size, say z. The prune operation introduces an additional error to that contained in the original summary. In particular, if ϵ' is the error in Q', then the error in Q will be $\epsilon' + \frac{1}{2z}$.

The aggregation of individual quantile summaries is performed over a tree structure with the BS as the root, which is formed in the query broadcast phase. A leaf node sends its quantile summary, which is simply its sensed value, to its parent. Each non-leaf node X first aggregates the quantile summaries it receives from its child nodes using the *combine* operation, and finally X applies one *prune* operation to keep the size of the summary constant. Due to the error introduced by the *prune* operation, the algorithm uses a concept of delayed aggregation, where the number of prune operations is kept within limit to satisfy the error bound ϵ in the final quantile summary. The authors design the protocol in such a way that a single sensed value experiences at most $\log N$ number of prune operations on its way to the BS. If we set the quantile size $z = \frac{\log N}{\epsilon}$, then the final error is bound to be ϵ and the worst case node congestion is $O(\frac{\log^2 N}{\epsilon})$.

B. Chan et al.'s Verification Algorithm

This scheme [2] is designed to compute and verify the Sum aggregate. The main idea behind this scheme is to move the verification responsibility from the BS to individual nodes that participated in the aggregation. Each node verifies if its own value is accounted for in the final aggregate. The algorithm consists of four operations, each of which takes one epoch to complete: (i) query dissemination, (ii) aggregation-commit, (iii) commitment-dissemination, and (iv) result-checking.

In the first epoch, the BS broadcasts an aggregation request. As the query message propagates through the network, an aggregation tree with the BS at the root is formed like in TAG algorithm [13].

During the aggregation-commit epoch, while the Sum is computed over an aggregation tree, nodes also construct a commitment structure similar to a Merkle hash tree [15] to enable the verification in the next phase. While a leaf node's message to its parent node contains its sensed value, each internal node sends the sub-aggregate it computed using the values received from its child nodes. In addition, each internal node, X, creates a commitment (a hash value) of the messages received from its child nodes. Both the sub-aggregate and the commitment are then passed to X's parent, which acts as a summary of X's sub-tree. The fields in X's message are $< \beta, v, \bar{v}, h >$, where β is the number of nodes in X's sub-tree, v is the local sum, $\bar{v}$ is the complement of the local sum (considering an upper bound v_{bound} for a sensed value), and h is an authentication field. In particular, a leaf node X sets the fields in its message as follows: $\beta = 1$, $v = v_X$, $\bar{v} = v_{bound} - v_X$, and $h = X$. If an internal node X receives messages $u_1, u_2, ..., u_t$ from its t child nodes, where $u_i = < \beta_i, v_i, \bar{v_i}, h_i >$, then X's message, $< \beta, v, \bar{v}, h >$, is generated as follows: $\beta = \sum \beta_i + 1$, $v = \sum v_i + v_X$, $v = \sum \bar{v_i} + (v_{bound} - v_X)$, and $h = H[\beta || v || \bar{v} || u_1 || u_2 || ... || u_t]$, where H is a hash function. Once the BS receives the final commitment, it verifies the coherence of the final v, $\bar{v}$ with the number of nodes in the network, N and the upper bound of sensed value, v_{bound}. In particular, the BS performs the following sanity check: $v + \bar{v} = v_{bound} \times N$. If this check succeeds, the base station initiates the next phase.

In the commitment-dissemination epoch, the final commitment C is broadcast by the BS to the network. This message is authenticated using the $\mu Tesla$ protocol [21]. The aim of the commitment dissemination phase is to let each single node know that its own value has been considered in the final aggregate. To do so, each node X should receive all of the *off-path values* up to the root node relative to X's position on the commitment tree. These values, together with the X's local commitment, allows X to compute a final commitment C'. Finally, node X checks if $C' = C$. If the check succeeds, it means that X's local value, v_X, has been included in the final Sum received by the BS.

In the last epoch, each node X that succeeded in the previous check sends an authentication code (MAC) up the aggregation tree toward the BS. These MACs are aggregated along the way with the XOR function to reduce the communication overhead. When the BS receives the XOR of all of the MACs, it can verify if all nodes confirmed that their values have been considered in the final aggregate.

The main cost of this protocol is due to the dissemination of the off-path values to individual nodes. The authors observed that this overhead is minimized if the commitment structure is balanced. They proposed to decouple the commitment structure from the physical aggregation tree, which enables the building of a balanced commitment forest as an overlay on an unbalanced aggregation tree. That results in the worst case node congestion in the protocol being $O(\Delta \log^2 N)$. To further reduce this overhead, Frikken et al. [6] modified the commitment structure, which results in a total cost of $O(\Delta \log N)$.

Finally, the authors show how the Sum computation protocol can be extended to compute the cardinality of a subset of nodes (Count) in the network. In particular, to count the elements in a given subset, we require each node to contribute 1 to the Sum aggregate if it belongs to the subset and to contribute 0 otherwise.

Appendix II
Proofs

A. Lemma 5.1

The probability that more than pN population elements lie between two consecutive items of a sorted uniform sample of size S is $\phi(S,p) = (1-p)^{S-1}$, where N is the population size.

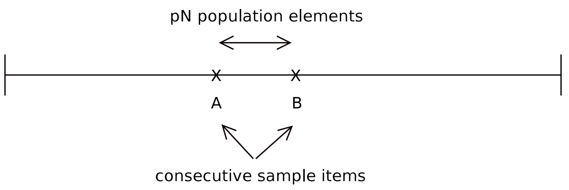


Fig. 9. How far apart are two consecutive elements in the sample?

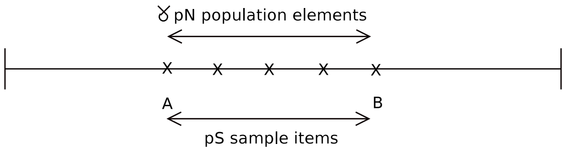


Fig. 10. What is the chance that γpN elements will fall within pS sample items, where $\gamma > 1$, $0 < p < 1$, and $\gamma p < 1$?

Proof: Let A and B be two consecutive items in the sample after the sample items are sorted (as shown in Figure 9). What we want to compute is the probability to have more than pN population elements between A and B. Once the sample item, A, is chosen, we have other $S-1$ population elements remain to be chosen for the sample. To obtain the above probability, none of these $S-1$ sample items should be chosen from the population interval which starts from A and is of length pN (i.e., the interval includes pN population elements). For each of these $S-1$ sample items, the probability to be chosen not from that interval is $(1-p)$. So, the probability that none of the $S-1$ items will be there is $(1-p)^{S-1}$. ■

B. Lemma 5.2

The probability that more than γpN ($\gamma > 1$, $0 < p < 1$, $\gamma p < 1$) population elements lie between the minimum and the maximum of pS consecutive sample items of a sorted sample of size S is

$$\xi(S,p,\gamma) = \sum_{i=0}^{pS-1} \binom{S-1}{i} (\gamma p)^i (1-\gamma p)^{S-1-i} \qquad (9)$$

where N is the population size.

Proof: Let A and B be the maximum and the minimum item among a subset of pS consecutive items in the sample while the sample items are sorted, as shown in Figure 10. So, the expected number of population elements lying between A and B is pN. We would like to compute the probability to have more than γpN population elements lying between A and B, where $\gamma > 1$ and $\gamma p < 1$. Once the sample item, A is chosen, we have other $S-1$ population elements remain to be chosen for the sample. To obtain the above probability, not more than $(pS-1)$ items of these $S-1$ sample items should be chosen from the population interval which starts from A and is of length γpN (i.e., the interval includes γpN population elements). For each of these $S-1$ sample items, the probability to be chosen from that interval is γp. So, the probability that not more than $(pS-1)$ items among the $S-1$ items will be there is

$$\sum_{i=0}^{pS-1} \binom{S-1}{i} (\gamma p)^i (1-\gamma p)^{S-1-i}.$$

■

Appendix III
Error bound without intra-group verification

Assuming that there can be at most w compromised nodes in the network, one might wish to estimate the error bound in the final estimate of the Median if intra-group verification is not performed in our attack-resilient scheme. Then, one can decide if it is worth paying the overhead for the intra-group verification to reduce the error. In this section, we compute the error bound and leave it to the user to set a tradeoff between the error and the energy overhead. Note that here we basically exploit the fact that one false value can deviate the final Median only by one position.

Let us assume that the network is divided into g groups which are of same size. To make the maximum deviation in the Median estimate, the best strategy for the attacker will be to compromise as many groups as possible – compromising one node each in w groups. We assume that no intra-group verification is performed and the group leader sends the local histogram to the BS in a authenticated way through multipath. The BS can verify these messages received from the group leaders. Also, for each group histogram, the BS verifies that no extra nodes are present in the group. This guarantees that the maximum deviation in Median that an attacker can inject by compromising one group is $\frac{N}{g}$. So, with w compromised nodes, the worst case relative error in the final estimate of the Median is $w\frac{N/g}{N} = \frac{w}{g}$.