SECURE DATA AGGREGATION IN WIRELESS SENSOR NETWORKS

by

Sankardas Roy

A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Information Technology

Committee:

_____ Dr. Sushil Jajodia, Dissertation Co-director

_____ Dr. Sanjeev Setia, Dissertation Co-director

_____ Dr. Robert Simon, Committee Member

_____ Dr. Songqing Chen, Committee Member

_____ Dr. Daniel Menascé, Senior Associate Dean

_____ Lloyd J. Griffiths, Dean, The Volgenau School of Information Technology and Engineering

Date: _____ Fall Semester 2008
George Mason University
Fairfax, VA

Secure Data Aggregation in Wireless Sensor Networks

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Sankardas Roy
Master of Science
Indian Statistical Institute, 2001
Bachelor of Science
Bengal Engineering College, 1997

Co-director: Dr. Sushil Jajodia, Professor
Center for Secure Information Systems
Co-director: Dr. Sanjeev Setia, Associate Professor
Department of Computer Science

Fall Semester 2008
George Mason University
Fairfax, VA

# Dedication

I dedicate this dissertation to my parents for their guidance and love throughout my life and to my wife Tamali, who keeps on encouraging me.

# Acknowledgments

I would like to thank my advisors, Professor Sushil Jajodia and Professor Sanjeev Setia, for their supervision, help, and encouragement in my successful completion of the work described in this dissertation. Without any doubt, they deserve all the credit for cultivating my interest in the sensor network security research field. I would also like to thank my other Ph.D. committee members for their guidance and fruitful discussions.

Further, I would like to thank all my colleagues in the George Mason University, with whom I enjoyed working and who have helped me with courses and research. Special thanks go to Mauro Conti, Venkata Addada, Lei Zhang, Dr. Bo Zhu, Dr. Chao Yao, Dr. Shiping Chen, and Dr. Sencun Zhu for valuable technical feedback, suggestions and proofreading many drafts of my papers.

Finally, I would like to thank my family. Without their cooperation, this dissertation would never have existed. I am deeply indebted to my parents, who were very supportive and encouraging in everything I undertook. My son Ritodeep and wife Tamali have tolerated many countless nights and weekends while I was doing research; their patience, love, and sacrifice have turned this Ph.D. from a dream to a reality.

# Table of Contents

# List of Tables

# List of Figures

# Abstract

SECURE DATA AGGREGATION IN WIRELESS SENSOR NETWORKS
Sankardas Roy, PhD
George Mason University, 2008
Dissertation Co-director: Dr. Sushil Jajodia
Dissertation Co-director: Dr. Sanjeev Setia

Wireless sensor networks have proved to be useful in several applications, such as environment monitoring and perimeter surveillance. In a large sensor network, *in-network data aggregation* (i.e., combining partial results at intermediate nodes during message routing) significantly reduces the amount of communication and energy consumption. Recently, the research community has proposed a robust aggregation framework called *synopsis diffusion* which combines multi-path routing schemes with duplicate-insensitive algorithms to accurately compute aggregates (e.g., Count, Sum) in spite of message losses resulting from node and transmission failures. However, this aggregation framework does not address the problem of false sub-aggregate values contributed by compromised nodes resulting in large errors in the aggregate computed at the base station, which is the root node in the aggregation hierarchy. This is an important problem since sensor networks are highly vulnerable to node compromises due to the unattended nature of sensor nodes and the lack of tamper-resistant hardware.

In this dissertation, we make the synopsis diffusion approach secure against attacks in

which compromised nodes contribute false sub-aggregate values. In particular, we present two classes of algorithms to securely compute Count or Sum. First, we propose a lightweight *verification algorithm* which enables the base station to determine if the computed aggregate includes any false contribution. Second, we present *attack-resilient computation algorithms* which can be used to compute the true aggregate by filtering out the contributions of compromised nodes in the aggregation hierarchy. Thorough theoretical analysis and extensive simulation study show that our algorithms outperform other existing approaches.

This dissertation also addresses the security issues of in-network computation of Median and presents verification algorithms and attack-resilient computation algorithms to securely compute an approximate estimate of this aggregate. To the best of our knowledge, prior to this dissertation there was no other work related to the security of in-network computation of Median. We evaluate the performance and cost of our algorithms via both analysis and simulation. The results show that our approach is scalable and efficient.

# Chapter 1: Introduction

Wireless sensor networks (WSNs) are increasingly used in several applications [1–3], such as wild habitat monitoring, forest fire detection, and military surveillance. After being deployed in the field of interest, sensor nodes organize themselves into a multi-hop network with the base station as the central point of control. Typically, a sensor node is severely constrained in terms of communication bandwidth, computation capability, and energy reserves. A straightforward method to collect the sensed information from the network is to allow each sensor node's reading to be forwarded to the base station, possibly via other intermediate nodes, before the base station processes the received data. However, this method is prohibitively expensive in terms of communication overhead, which prompted active research to design an energy-efficient mechanism.

In large WSNs, computing aggregates *in-network* (i.e., combining partial results at intermediate nodes during message routing) significantly reduces the amount of communication and hence the energy consumed. An approach used by several data acquisition systems for WSNs [4, 5] is to construct a spanning tree rooted at the base station, and then perform in-network aggregation along the tree. Partial results propagate level-by-level up the tree, with each node awaiting messages from all its children before sending a new partial result to its parent.

The most important aggregates considered by the research community include Count, Sum, Uniform Sample, and Median. We present the definitions of these aggregates in Table 1, where $v_i$ denotes the sensed value of the $i$-th sensor node. A Count computation algorithm can be extended to compute the cardinality of a subset of nodes in the network. As an example, upon receiving a request from the base station to count the number of nodes with sensed values lying between 100 and 200 units, if only the relevant nodes contribute to the aggregate, the base station will receive the correct count. We can compute Average from

Table 1.1: Definition of Aggregates

| Aggregate | Definition |
|---|---|
| Count | $N = \|\{v_i\}\|$, where $\{v_i\}$ is the set of sensed values. |
| Sum | $S = \sum_{i=1}^{N} v_i$ |
| Uniform Sample | $U = \{v_{i_1}, v_{i_2}, \ldots, v_{i_k}\}$, where these $k$ values are randomly selected from the population set $\{v_i\}$. |
| Median | $M = \begin{cases} v_i \ \ \textbf{s.t.}\ rank(v_i) = \frac{N+1}{2}, & \text{N is odd} \\ \frac{v_i+v_j}{2} \ \ \textbf{s.t.}\ rank(v_i) = \frac{N}{2} \text{ and } rank(v_j) = \frac{N}{2}+1, & \text{otherwise} \end{cases}$ |

Count and Sum. A Sum algorithm can be also extended to compute Standard Deviation and Statistical Moment of any order. We also note that a Uniform Sample can be used to have an approximate estimate of several aggregates, including Most Frequent Items (i.e., a subset of sensed values which are most common among all of the nodes).

Tree-based aggregation approaches are not resilient to communication losses resulting from node and transmission failures, which are relatively common in WSNs [4–6]. Because each communication failure loses an entire subtree of readings, a large fraction of sensor readings is potentially unaccounted for at the querying node, leading to a significant error in the query answer. To address this problem, the research community has proposed the use of multi-path routing techniques for forwarding sub-aggregates [4]. For aggregates such as Min and Max, which are duplicate-insensitive, this approach provides a fault-tolerant solution. However, for duplicate-sensitive aggregates, such as Count and Sum, multi-path routing leads to double-counting of sensor readings, resulting in an incorrect aggregate being computed.

Recently, several researchers [7–9] have presented clever algorithms to solve the double-counting problem associated with multi-path approaches. A robust and scalable aggregation framework called *synopsis diffusion* has been proposed for computing duplicate-sensitive

aggregates, such as Count and Sum. There are two primary elements of this approach. First, a ring topology is used instead of a tree topology for organizing the nodes in the aggregation hierarchy. In a ring topology, a node may have multiple parents in the aggregation hierarchy, unlike the tree topology. Second, each sensed value or sub-aggregate is represented by a duplicate-insensitive bitmap called *synopsis*. The function to generate a synopsis or to merge multiple synopses is based on Flajolet and Martin's algorithm for counting distinct elements in a multi-set [10].

The research community also observed that in-network computation of Median is a harder problem than computation of Count or Sum. Using the Count algorithm within a synopsis diffusion framework as a building block, Patt-Shamir [11] proposed a probabilistic algorithm to compute an approximate Median. On the other hand, Greenwald et al. [12] and Shrivastava et al. [13] proposed tree-based algorithms to compute an approximate Median. Unfortunately, neither of the above algorithms included any provisions for security, which constitutes the problem considered in this dissertation.

## 1.1 Problem Statement

The objective of this dissertation is to address the security issues of in-network data aggregation in WSNs where a fraction of nodes may become compromised. Our goal is to design protocols to securely compute the basic aggregates, such as Count, Sum, and Median.

Most of the existing in-network data aggregation algorithms [4,7,9,14] have no provisions for security. As a result, an attacker can eavesdrop on the wireless communication in the network and inject false data during the aggregation process, which results in a bogus aggregate being computed at the base station with no alarm being raised.

A straightforward solution for preventing an unauthorized node from injecting false messages is to augment the existing aggregation algorithms with the standard encryption and authentication schemes [15–17]. However, this solution cannot counter attacks launched by an insider node that has been compromised by the adversary, because an insider node possesses the cryptographic secrets used in the encryption and authentication schemes. It

is important to design WSN protocols that are resilient to insider attacks because a fraction of nodes may become compromised due to the unattended nature of WSNs and the lack of tamper-resistant hardware.

A compromised node might attempt to thwart the aggregation process by launching several attacks, such as eavesdropping, jamming, message dropping, message fabrication, and so on. This dissertation focuses on a subclass of these attacks in which the goal of the adversary is to cause the base station to derive an incorrect aggregate. By relaying a false sub-aggregate to the parent node, a compromised node attempts to contribute a large amount of error to the aggregate. As an example, during the Sum computation algorithm [4,7,9,14], a compromised node $X$ can inject an arbitrary amount of error in the final estimate of Sum by falsifying $X$'s own sub-aggregate. We refer to this attack as the *falsified sub-aggregate attack*.

In this dissertation, we aim to design algorithms to securely compute aggregates despite the falsified sub-aggregate attack launched by compromised nodes. We have two goals: (i) to enable the base station to verify if the computed aggregate is valid, and (ii) to empower the base station to filter out the false contributions of the compromised nodes from the aggregate. We call the algorithms which can achieve these goals the *verification algorithms* and *attack-resilient computation algorithms*, respectively. In particular, for each of the aggregates, Count, Sum, and Median, we will design verification algorithms and attack-resilient computation algorithms. While meeting the above goals, we also will show how to minimize the cost (i.e., the communication overhead and latency).

## 1.2 Summary of Contributions

The research presented in this dissertation addresses the problems discussed in Section 1.1. We propose secure in-network aggregation protocols which can tolerate compromised nodes present in the network.

To incorporate attack-resiliency into in-network aggregation algorithms, we use the standard security infrastructure widely accepted by the research community. In particular, we

assume that each node shares a key with the base station, and any two neighboring sensor nodes possibly share a pairwise key. The sensor nodes have the ability to compute a collision-resistant cryptographic hash function. We also assume that the basic security services are available (e.g., $\mu Tesla$ [18]) for authentication of messages broadcast from the base station.

We design verification algorithms and attack-resilient computation algorithms for the basic aggregates, Count, Sum, and Median. Our verification algorithms enable the base station to verify if the computed aggregate is valid; however, in the presence of the attacks launched by compromised nodes, these algorithms do not guarantee the successful computation of the aggregate. To compute the aggregates in the presence of compromised nodes, we design attack-resilient computation algorithms which incur comparatively larger overhead.

Below we summarize our contributions.

**Securely Computing Count and Sum** We analyze the vulnerabilities of the existing in-network aggregation algorithms designed within the synopsis diffusion framework in the presence of compromised nodes. Then, as the countermeasures, we design verification algorithms and attack-resilient computation algorithms to securely compute Count and Sum. The key observation behind the design of our algorithms is that to verify the correctness of the final synopsis, which represents the Count or Sum of the whole network, the base station does not need to receive authentication messages from all of the nodes. The main challenge is to minimize the number of authentication messages transmitted (which is the main source of the communication overhead), while guaranteeing the correctness of the computed aggregate. The above challenge is addressed in this dissertation. The analytical and simulation results show that our algorithms are effective and efficient. We show that the per-node communication overhead in our verification algorithm is $O(1)$ irrespective of the network size, while that of the least expensive existing algorithm [19] is $O(\log S)$, where $S$ is the value of the aggregate, Count or Sum. The communication complexity of our attack-resilient computation algorithm is $O(t)$ if $t$ compromised nodes are present in the network, and this complexity does not grow with the network size. This algorithm costs much less

than the worst case overhead of the existing attack-resilient computation algorithm [20], which is $O(N)$, where $N$ is the network size. We also show that the communication overhead of our attack-resilient computation algorithm can be further reduced at the expense of latency.

**Securely Computing Median** We design verification algorithms and attack-resilient computation algorithms to compute an approximate Median, which are based on the tree topology. The main idea employed in the proposed algorithms is to collect a sample of sensed values from the network and then to find the value which is closest to the Median. We design an iterative histogram verification algorithm which fine-tunes the search for an approximate Median within the collected sample. The key challenge is to minimize the number of times we need to execute the histogram verification algorithm (over the network) while guaranteeing the desired accuracy in the final estimate of the Median. We show that our algorithms take $O(1)$ iterations to complete and cost $O(\frac{1}{\epsilon} \cdot \log N)$ per-node communication overhead, where $\epsilon$ is the desired error bound of the final estimate. We evaluate the proposed algorithms via thorough theoretical analysis and extensive simulation study.

The above algorithms can be extended to compute other order-statistics besides Median. The Uniform Sample of sensor readings collected from the network during our Median algorithms is an important aggregate on its own and can be used to compute many other aggregates, such as Most Frequent Items.

## 1.3 Organization of the Dissertation

The rest of this dissertation is organized as follows. Chapter 2 reviews the prior work related to this dissertation, and provides a background on data aggregation protocols and the state-of-the-art secure solutions. The contributions documented in this dissertation are primarily grouped into two parts, either of which is dedicated to the security issues of computing one particular class of aggregates. The first part discusses secure computation of Count and Sum. Chapter 3 analyzes the vulnerabilities of synopsis diffusion algorithms and

proposes secure protocols to compute the aggregate. The second part, presented in Chapter 4, focuses on how to securely compute Median. Finally, Chapter 5 presents conclusions and suggestions for further research.

# Chapter 2: Background and Related Work

Several researchers have studied problems related to data aggregation in wireless sensor networks (WSNs). However, most of the existing work is driven by the performance efficiency issues without considering the possibility of the presence of attacks. Only recently, have several papers been published which address security issues of the aggregation protocols.

In Section 2.1, we discuss the security issues of WSNs in general. Then, in the rest of this chapter, we present summaries of the existing body of work related to data aggregation in WSNs. In Section 2.2, we review the tree-based aggregation protocols and analyze their vulnerabilities in the presence of compromised nodes. Section 2.3 reviews the ring-based aggregation protocols and analyzes their vulnerabilities against compromised nodes. In Section 2.4, we discuss the existing algorithms which partially solve the security problems in data aggregation. Section 2.5 presents the other body of work which also has some relation to the security issues in data aggregation. Finally, in Section 2.6, we summarize how our work differs from that of other researchers.

## 2.1 Security in WSNs

In general, an adversary may attempt to launch several attacks to thwart the normal functionality of a WSN. Here, we briefly discuss the security issues of WSNs and the state-of-the-art solutions. For more details, the reader can refer to the article [21] by Perrig et al. and the book, 'Wireless Sensor Network Security' [22], edited by Lopez et al.

A WSN is especially vulnerable to outsider and insider attacks due its unique characteristics, as follows. Typically, sensor nodes are severely constrained in terms of memory, computation capability, communication bandwidth, and energy resources. Also, it is easy

for the attacker to physically access the sensor nodes because they are not built with tamper-resistant hardware for cost-effectiveness and they must be deployed near the source of the events. Moreover, the attacker can access the information exchanged among nodes because the communication channel is wireless.

Due to the use of a wireless communication channel, an attacker node can launch multiple attacks in a WSN, such as eavesdropping, message modification, message relay, and message injection. We can prevent an unauthorized node from injecting false messages by employing the standard encryption and authentication schemes [15–17], but this solution cannot counter attacks launched by a compromised node.

A WSN is also vulnerable to Denial-of-Service (DoS) attacks, which may take several forms. The DoS attacker may attempt to prevent a particular section of the network from receiving any broadcast message from the base station. It may also try to jam the communication channel of a certain node to stop that node from sending or receiving any message. Also, the attacker may repeatedly request packets from a benign node to deplete its energy resource. We can partially defend against the jamming attack by employing spread-spectrum communication [23]. A jamming-resistant network [24] may also be built to counter this attack.

Karlof et al. [25] studied the vulnerabilities of routing protocols in a WSN in the presence of compromised nodes. They demonstrated that to hinder normal message routing in a WSN, several complex attacks can be designed, such as selective forwarding, wormhole attack, sinkhole attack, hello flood attack, and acknowledgement spoofing. The scientific community is involved in active research [26] with the goal of mitigating these attacks.

Furthermore, after compromising a few nodes, the adversary can exploit these nodes' identity to launch the node replication attack. Mounting this impersonation attack on a WSN, the adversary can paralyze some of the core protocols, such as routing. Recently, a couple of schemes [27, 28] have been proposed to detect the node replication attack.

This dissertation addresses insider attacks where the goal of the attacker is to cause the base station to compute a false aggregate. To this end, our attacker node does not correctly

follow the data aggregation protocol, but it behaves normally in other ways.

## 2.2 Aggregation using Tree Topology

Several researchers, including Madden et al. [4] and Greenwald et al. [12], designed energy-efficient in-network algorithms using a tree topology to compute aggregates, such as Count, Sum, and Median. Typically, these algorithms construct a spanning tree rooted at the base station and then perform in-network aggregation along the tree.

In a tree topology, nodes in the network form a tree with the base station as the root during the aggregation request dissemination. Typically, the following tree construction algorithm is used: the base station's aggregation request is broadcast hop-by-hop, and each node $Y$ on any hop selects one node $Z$ on the previous hop from which $Y$ has received the aggregation request, and node $Y$ considers node $Z$ as its parent. Once the aggregation request reaches all of the nodes in the network, a spanning tree is formed with the base station as its root.

In the aggregation phase, partial results propagate level-by-level up the tree, with each node awaiting messages from all of its child nodes before sending a new partial result to its parent. Finally, the base station computes the aggregate of interest from the messages it receives from its child nodes on the tree.

Below we discuss the tree-based in-network aggregation algorithms present in the literature.

### 2.2.1 Computing Count and Sum

**Madden et al.'s Algorithm [4]** Madden et al. proposed a Tiny Aggregation Service (TAG) to compute aggregates, such as Count and Sum. To compute Count, each node adds the partial counts received from its child nodes, increments the result by one, and sends this sub-aggregate to its parent node, which represents the total number of nodes present in its sub-tree, and so on. In the example depicted in Figure 2.1, node $Z$'s sub-aggregate is 3, which indicates that there are 3 nodes in total in $Z$'s sub-tree. The Count computation

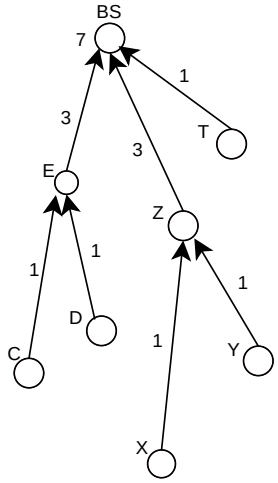



Figure 2.1: Aggregation Tree Computing Count—Partial counts propagate level-by-level up the tree before reaching the base station (BS).

algorithm can be easily extended to compute Sum, where each node contributes its sensed value to the aggregate. In these algorithms, each node needs to send only a single message containing the partial Count or Sum. We note that Yao et al. [14] and Zhao et al. [5] proposed similar in-network algorithms to compute Count and Sum.

### 2.2.2 Computing Median

Madden et al. [4] showed that an in-network aggregation algorithm does not reduce communication in case of computing holistic aggregates, such as Median (order-statistics, in general), compared with the simple approach of forwarding all of the sensor readings directly to the base station. To reduce the communication overhead, the research community advocated computing an approximate estimate of holistic aggregates.

Recently, Greenwald et al. [12] and Shrivastava et al. [13] designed in-network aggregation algorithms to compute an approximate order-statistic (i.e., quantile) with low communication overhead. Given a positive number $\epsilon$, $0 < \epsilon < 1$, as user input, these algorithms

return an approximate estimate, say $\hat{y}$, so that $\hat{y}$ does not differ from the exact order-statistic of interest, $y$, by more than $\epsilon N$ positions on the sorted list of all of the sensor readings, where $N$ is the total number of sensor readings. We call $\hat{y}$ an $\epsilon$-approximate estimate of the order statistic.

**Greenwald et al.'s Algorithm [12]** Greenwald et al. defined *$\epsilon$-approximate quantile summary* as a subset of sensed values from which we can derive an $\epsilon$-approximate estimate of any quantile. In Greenwald et al.'s algorithm, every node $X$ sends an $\epsilon$-approximate quantile summary to its parent, which represents the sensed values of $X$ and its descendant nodes. Once a node $Y$ receives the quantile summaries from its child nodes, $Y$ does not necessarily merge all of the received summaries. Arbitrary merging may result in the merged summary not remaining an $\epsilon$-approximate one. After determining which of the received summaries can be merged without violating the $\epsilon$-approximation criterion, node $Y$ aggregates them by a merging algorithm, while the other summaries of $Y$ remain unmodified. $Y$ then forwards both the merged summary and the other ones to its parent node, and so on. After the base station obtains the summaries from its child nodes, it computes the final quantile summary, which can derive any order-statistic satisfying the $\epsilon$ error bound. In this scheme, it is guaranteed that a node needs to transmit at most $O(log^2 N/\epsilon)$ data values irrespective of the network topology.

**Shrivastava et al.'s Algorithm [13]** Shrivastava et al. presented a distributed data summarization technique called *quantile-digest* for computing approximate order-statistics. In this algorithm, each node $X$ sends a quantile-digest to its parent, which is a summary of the data values of $X$ and its descendant nodes. The key element of this technique is to accurately preserve information about high frequency sensor readings while compressing information about low frequency readings. If $l$ is the size of the digest used, it is shown that the error in the final estimate is at most $O(\log \sigma/l)$, where the sensed values are in the range $[1, \sigma]$.

**Cox et al.'s Algorithm [29]** Cox et al. proposed a lightweight algorithm to persistently update the base station's estimate over an order-statistic of the sensor readings, assuming

the sensor readings may change with time. In this scheme, the order-statistic of interest is computed first, and later a validation protocol is periodically run to determine whether the initial estimate is still valid. If not, a binary search is efficiently used to compute the new estimate. This scheme uses in-network aggregation and transmission suppression to reduce the communication overhead.

### 2.2.3 Attacks

The above algorithms do not include any provisions for security. A compromised node can launch multiple attacks which can potentially corrupt the final result of the aggregation query. Below we discuss these attacks in the context of Sum aggregate, which can be extended to other aggregates.

A compromised node $X$ can corrupt the aggregate value computed at the base station in three ways. First, $X$ can simply drop aggregation messages that it is supposed to forward to its parent. If $X$ is located at a relatively high position in the aggregation hierarchy, this has the effect of omitting a large fraction of the set of sensor readings from being considered. Second, $X$ can falsify its own sensor reading with the goal of influencing the aggregate value. We refer to this attack as the *falsified local value attack*. Third, $X$ can falsify the sub-aggregate which $X$ is supposed to compute based on the messages received from $X$'s child nodes. We refer to this attack as the *falsified sub-aggregate attack*.

Moreover, because each communication failure loses an entire subtree of readings, these tree-based algorithms may incur large amounts of error in lossy network environments (i.e., they are inherently vulnerable to communication loss).

## 2.3 Aggregation using Ring Topology

Since packet losses and node failures are relatively common in WSNs, several studies have investigated the design of robust aggregation algorithms. To this end, several researchers [4, 7, 9] proposed to use a ring topology instead of a tree topology, where a node

on the aggregation hierarchy is allowed to have multiple parents. In a ring-based aggregation algorithm, an individual node's contribution to the aggregate reaches the base station possibly via multiple paths through the intermediate nodes.

Madden et al. [4] proposed a scheme to compute aggregates, such as Min and Max, where each node forwards its sub-aggregate to multiple parents. For such duplicate-insensitive aggregates, this approach provides a fault-tolerant solution. However, for duplicate-sensitive aggregates, such as Count, Sum, and Median, this approach cannot be applied as it would result in obtaining an incorrect estimate due to double counting of individual sensor values.

Below we discuss the ring-based algorithms present in the literature to compute aggregates, such as Count, Sum and Median.

### 2.3.1 Computing Count and Sum

**Nath et al.'s Algorithm [9] and Considine et al.'s Algorithm [7]** Nath et al. designed a robust and scalable aggregation framework called *synopsis diffusion*. This framework combines the use of multi-path routing with clever algorithms that avoid double-counting of sensor readings. Considine et al. independently proposed very similar algorithms to compute Count and Sum. There are two primary elements of the synopsis diffusion approach—the use of a ring topology for organizing the nodes in the aggregation hierarchy, and the use of duplicate-insensitive algorithms for computing aggregates based on Flajolet and Martin's algorithm for counting distinct elements in a multi-set [10].

In the synopsis diffusion approach, nodes are classified into multiple rings determined by their hop counts from the base station, illustrated in Figure 2.2. During the query distribution phase, the base station's aggregation request is broadcast hop-by-hop, and each node $X$ on any hop keeps track of the nodes on the previous hop from which $X$ has received the aggregation request, and node $X$ considers all of them as its parents. Once the aggregation request reaches all of the nodes in the network, ring construction is complete.

In the subsequent query aggregation phase, starting in the outermost ring, each node generates a local synopsis (i.e., a summary bitmap) relevant to the query, and broadcasts it

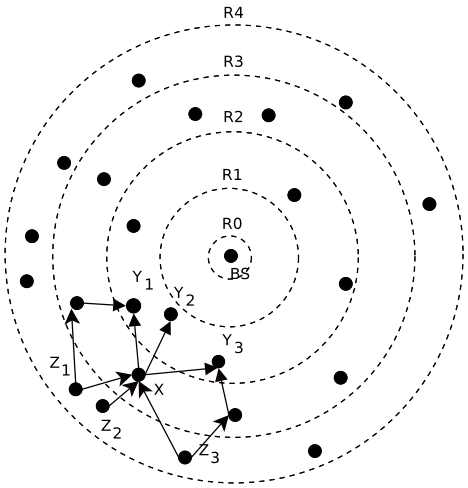



Figure 2.2: Synopsis Diffusion over a Ring Topology—Rings $R_i$ are constructed during the query broadcast from the BS. A node may have multiple parents, e.g. $X$ has 3 parents, $Y_1, Y_2, Y_3$. During the subsequent aggregation phase, each node $X$ forwards its synopsis to all of the parents, i.e. $X$'s contribution reaches the BS via multiple paths.

to its neighbors. A node in ring $R_i$ will receive broadcasts from all of the nodes in its range in ring $R_{i+1}$. It will then aggregate its own local synopsis with the synopses received from its children, and then broadcast the updated synopsis. Thus, the fused synopses propagate level-by-level until they reach the base station, which combines the received synopses to derive the final aggregate. This approach is robust against communication loss because each node's contribution to the aggregate reaches the base station via multiple paths. One limitation of the synopsis diffusion algorithms is that they output only an approximate estimate of the aggregate.

**Majhi et al.'s Algorithm [8]** To reduce the approximation error present in the output of the synopsis diffusion algorithms, Majhi et al. proposed a hybrid aggregation hierarchy, called Tributary and Delta, which forms the tree topology in some parts of the network while constructing the ring topology in other regions. The Tributary and Delta approach aims to combine the advantages of the tree topology and ring topology in an efficient way. For the nodes which are far away from the base station, trees are used for their zero approximation error. For nodes close to the base station whose sub-aggregates account for

many nodes' values, the multi-path approach is used due to its robustness property. The challenge, addressed in [8], is how to dynamically adapt this hybrid aggregation hierarchy to the current message loss rate.

### 2.3.2 Computing Median

**Patt-Shamir's Algorithm [11]** Patt-Shamir proposed a probabilistic algorithm to compute an approximate Median within the synopsis diffusion framework. The design of this algorithm is based on a new definition of an $(\alpha, \beta)$-approximate Median where parameter $\alpha$ represents the desired error bound in terms of rank, and parameter $\beta$ controls the allowed error in terms of value.

Patt-Shamir's algorithm performs a binary search over the whole range of sensor values until it finds a value $\hat{y}$ such that $\hat{y}$ satisfies the two error bounds, $\alpha$ and $\beta$, with a high probability. In each iteration of the binary search, the number of values smaller than $\hat{y}$ is computed using the Count algorithm, discussed in Section 2.3.1, in which only relevant nodes participate.

Given a failure probability $\delta$, this algorithm must invoke the Count algorithm $O(\log^2 N/\delta)$ times, which indicates that the communication overhead of this algorithm can become prohibitively expensive. The communication complexity becomes even higher if the difference between the maximum and minimum sensed value is of the order higher than $O(N)$.

The research community also proposed algorithms to compute aggregates such as Uniform Sample [9] and Most Frequent Items [8] within the synopsis diffusion framework.

### 2.3.3 Attacks

The synopsis diffusion aggregation framework does not include any provisions for security; as a result, it is vulnerable to many attacks that can be launched by compromised nodes. These attacks include insider attacks which can potentially corrupt the final result of the aggregation query. In fact, all of the three attacks relevant to tree-based algorithms, discussed in Section 2.2.3, can be launched to corrupt the aggregate value computed in these

ring-based algorithms. However, the impact of these attacks on the ring topology can be different, which we describe below in the context of Sum aggregate.

The attack in which a compromised node $X$ drops aggregation messages that it is supposed to relay does not have a significant impact on the synopsis diffusion approach, because multi-path routing is used. However, a compromised node $X$ can launch the falsified local value attack and the falsified sub-aggregate attack to corrupt the aggregate computed at the base station. We observe that even a single node can launch the second and third attack with a high rate of success because the use of multi-path routing in the synopsis diffusion approach makes it highly likely that the falsified synopsis will propagate to the base station.

## 2.4 Secure Aggregation Techniques

Several secure aggregation algorithms have been proposed assuming that the base station is the only aggregator node in the network, i.e., no in-network aggregation. Only recently, the research community has paid attention to the security issues of hierarchical aggregation where the presence of intermediate aggregators is considered.

### 2.4.1 Single Aggregator Algorithms

**Wagner's Algorithm [30]** For the single-aggregator model, Wagner addressed the issue of measuring and bounding malicious nodes' contribution to the final aggregation result. Wagner's paper measures how much damage an attacker can inflict by taking control of a number of nodes and using them solely to inject erroneous data values. Wagner also provides guidelines for selecting resilient aggregation functions. As an example, in scenarios where we expect that a few nodes might become compromised, Median is a preferred aggregate over Average because the possible error in Median is bounded, while Average can contain an arbitrary amount of error. Wagner also proposed a few outlier filtering techniques such as *truncation* and *trimming* for achieving resilient aggregation.

**Buttyan et al.'s Algorithm [31,32] and Mahimkar et al.'s Algorithm [33]** Buttyan

et al. [31] proposed a technique of attack-resilient aggregation, where the aggregator analyzes the received sensor readings before applying the aggregation function. They show that, for computing Average, if the attacker wants to remain undetected, the maximum error injected by the attacker in the final estimate of the aggregate has an upper bound.

Buttyan et al. also proposed another technique called RANBAR [32] for filtering outliers. RANBAR is based on the use of RANSAC [34], a well-known algorithm used to estimate the parameters of a mathematical model from a set of observed data which contains outliers. Buttyan et al. show that compared to other resilient aggregation functions such as the trimmed average and the Median, RANBAR results in smaller distortion, especially when a large fraction of nodes is compromised.

Mahimkar and Rappaport [33] proposed an aggregation-verification scheme for the single-aggregator architecture using a threshold signature scheme which ensures that at least $t$ of the nodes agree with the aggregation result.

**Przydatek et al.'s Algorithm [35]** In the schemes described above, it is assumed that the aggregator is not compromised. Przydatek et al. [35] relaxed this assumption in the design of their secure aggregation framework named Secure Information Aggregation (SIA). SIA considers a WSN in which a large number of sensors are deployed in an area distant from a home server (i.e., a user) and a base station is used as an intermediary between the home server and the sensor nodes. After sensor nodes send their readings to the base station, the base station performs the aggregation task and forwards the result to the home server. The base station is the only aggregator node in the network.

SIA enables the user to verify that the answer given by the aggregator is a good approximation of the true value, where the aggregator and a fraction of the sensor nodes might become corrupted. In particular, SIA aims to guarantee that if the user accepts an aggregation result reported by the aggregator, 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 aggregator, the user will detect the attack and reject the reported aggregate with high probability. SIA achieves

this goal by constructing efficient random sampling mechanisms and interactive proofs. Aggregates such as Min, Max, Count, Average, and Median can be securely computed in this approach.

### 2.4.2 Hierarchical Data Aggregation

**Hu et al.'s Algorithm [36]** As Wagner [30] observed, securing the hierarchical data aggregation algorithms is a harder problem because the intermediate aggregator nodes may falsify their sub-aggregates. The first attack-resilient hierarchical data aggregation protocol was designed by Hu et al. [36]. This protocol secures in-network aggregation against a single Byzantine adversary. In this protocol, each aggregator node $X$ forwards its inputs (received from $X$'s child nodes) to its parent nodes in the aggregation tree so that parent nodes can verify if $X$ has performed the aggregation correctly. This scheme is secure unless more than one malicious nodes is present.

**Secure Data Aggregation Protocol (SDAP) [20]** Yang et al. [20] proposed SDAP to compute Average, which can tolerate more than one compromised node. SDAP uses a novel probabilistic grouping technique to dynamically partition the nodes in a tree topology into multiple logical groups (subtrees) of similar sizes. An aggregate is computed within each logical group using a standard hop-by-hop aggregation protocol. The leader of each logical group transmits the group's aggregate to the base station, along with a commitment that is generated based on the contributions of each node in the group. After the base station has collected all of the group aggregates, it uses an outlier detection algorithm to identify groups whose contributions are potentially false. Finally, each group under suspicion participates in an attestation process to prove the correctness of its group aggregate. The base station discards any suspicious aggregates from groups that fail the attestation procedure. The final aggregate is calculated over all of the group aggregates that have either passed the outlier detection algorithm or the attestation procedure. As SDAP is a tree-based protocol, it is vulnerable to link loss and node failures, which are relatively common in WSNs.

**Chan et al.'s Algorithm [37] and Frikken et al.'s Algorithm [38]** Chan et al. [37]

designed a verification algorithm by which the base station can detect if the final aggregate, Count or Sum, is falsified. The algorithm is based on a novel method of distributing the verification responsibility onto the individual sensor nodes. During the aggregation process, the nodes construct a logical commitment structure (for authentication purposes) as an overlay on the physical aggregation tree. A clever technique is used to guarantee that a balanced commitment structure is formed, which ensures sub-linear congestion bounds. The algorithm induces $O(\Delta \cdot \log^2 N)$ node congestion, where $\Delta$ is the maximum number of neighbors of a node.

Recently, Frikken et al. [38] proposed a modification to Chan et al.'s [37] scheme that reduces the maximum communication per node from $O(\Delta \cdot \log^2 N)$ to $O(\Delta \cdot \log N)$. The modification mainly involves designing a new commitment structure for authentication, which is used with the other components of Chan et al.'s scheme [37]. The idea behind the communication overhead reduction in this new approach is that a few dummy logical nodes are introduced to prevent nodes close in the physical aggregation tree from being scattered throughout the logical commitment structure. This proves to be a significant gain in performance for a large WSN. For example, in a 1000-node network this approach would require a tenth of the communication of the scheme of [37] (ignoring constants).

Although Chan et al.'s [37] algorithm as well as Frikken et al.'s [38] algorithm prevent the base station from accepting a false aggregate, they do not guarantee the successful computation of the aggregate in the presence of the attack, which can be launched by a single compromised node.

**Garofalakis et al.'s Algorithm [19]** Garofalakis et al. designed a verification algorithm for computing Count and Sum within the synopsis diffusion approach. The main idea underlying their algorithm is that to verify the final synopsis the base station needs to receive only one authentication message (Message Authentication Code, also called MAC in short) for each bit in the synopsis even if multiple nodes contribute to one particular bit. Hence, it is sufficient for each node in the aggregation hierarchy to forward only one MAC corresponding to each bit in the synopsis. As the length of the synopsis is $O(\log S)$ where

$S$ is the upper bound of Count or Sum, the communication overhead per node is O(log $S$).

## 2.5 Other Related Work

We now discuss the other body of work present in the literature that also has some relation to this dissertation. We briefly present some of the existing schemes related to gossip-based aggregation, privacy-preserving aggregation, and false data injection prevention.

Jelasity et al. [39] proposed a robust gossip-based protocol for computing aggregates over network components in a fully decentralized fashion. They assumed that nodes form an overlay network where any pair of nodes can be considered to be neighbors, which makes this protocol impractical for WSNs.

Several researchers also have studied the problem of privacy-preserving aggregation, where we aim to keep the individual sensor readings secret from the intermediate aggregators. Such algorithms have been proposed by Girao et al. [40], Castelluccia et al. [41], and Cam et al. [42]. The main ideas of these algorithms are to maintain the end-to-end confidentiality between sensors and the base station, and to perform in-network aggregation of encrypted data. To allow the computation of the correct aggregate from the encrypted data, in [40], a *privacy homomorphism* (PH) [43] technique is employed.

This dissertation is also related to the general problem of preventing false data injection. Several solutions [44–46] have been proposed to prevent false data injection attacks in WSNs. In a false data injection attack, an adversary injects false data into the network with the goal of deceiving the base station or depleting the energy resources of the relaying nodes. The Hop-by-Hop Authentication scheme proposed by Zhu et al. [46] counters this attack in the following way assuming that the number of compromised nodes is below a threshold number, $t$. First, the base station accepts a data report only if it is endorsed by at least $t+1$ different sensor nodes. Second, every node en route to the base station participates in an interleaved hop-by-hop authentication scheme to filter out the false report, if any.

Du et al. [47] proposed a mechanism that allows the base station to check the aggregated

values submitted by several designated aggregators, based on the endorsements provided by a certain number of witness nodes around the aggregators. Their scheme, however, does not consider hop-by-hop aggregation.

## 2.6 Difference between this Dissertation and Prior Work

We now summarize how the prior work is different from this dissertation. First, the original synopsis diffusion framework is robust to communication loss, but there is no provision for security. This dissertation incorporates security into the synopsis diffusion framework, and our algorithms are robust to communication loss as well as secure against attacks launched by compromised nodes. Second, the verification protocols proposed by Chan et al. [37], Frikken et al. [38], and by Garofalakis et al. [19] prevent the base station from accepting a false aggregate, but they cannot compute the correct aggregate in the presence of the attack. In addition to verification protocols, we designed attack-resilient computation protocols which guarantee the successful computation of the aggregates even in the presence of the attack. Third, to the best of our knowledge, we are the first to address the security issues in an in-network computation of Median. More specifically, we design verification algorithms and attack-resilient computation algorithms for Median. In addition to security, our algorithms also gain in terms of performance efficiency compared to Greenwald et al.'s [12] non-secure Median algorithms. Fourth, Hu et al.'s [36] aggregation technique has limited provision for security – it cannot tolerate more than one compromised node. In contrast, most of our algorithms can tolerate multiple compromised nodes.

We compare the security features of our schemes with those of the existing schemes in Table 2.6. For each scheme, we indicate which aggregates the scheme computes and how many compromised nodes against which it is secure. We also state if the scheme has provisions to verify the computed aggregate and to compute the aggregate in the presence of compromised nodes.

Table 2.1: Comparing this Dissertation with the Prior Schemes

| Algorithms | Aggregates considered | number of compromised nodes | Verification | Attack-resilient computation |
|---|---|---|---|---|
| TAG [4] | Count, Sum | 0 | None | None |
| Synopsis Diffusion [7, 9] | Count, Sum, Median | 0 | None | None |
| Greenwald et al. [12] | Median | 0 | None | None |
| Hu et al. [36] | Count, Sum, Median | 1 | Count, Sum, Median | None |
| Chan et al. [37] | Count, Sum | $\geq 1$ | Count, Sum | None |
| Garofalakis et al. [19] | Count, Sum | $\geq 1$ | Count, Sum | None |
| SDAP [20] | Count, Sum | $\geq 1$ | Count, Sum | Count, Sum |
| This dissertation | Count, Sum, Median | $\geq 1$ | Count, Sum, Median | Count, Sum, Median |

# Chapter 3: Securely Computing Count and Sum

This chapter discusses the security issues of in-network aggregation algorithms to compute Count and Sum when a fraction of nodes in the network have been compromised. Prior research has addressed some of these security issues, but most of the existing schemes are limited to tree-based aggregation. We analyze the vulnerabilities of the ring-based aggregation within the synopsis diffusion framework. We also propose solutions for securely computing Count and Sum within this framework. In particular, two secure algorithms, a verification algorithm and an attack-resilient computation algorithm, are proposed in this chapter. The goal of the verification algorithm is to detect whether an attack has been launched, whereas the goal of the attack-resilient computation algorithm is to compute the aggregate despite the presence of the attack.

**Organization.** This chapter is organized as follows. In Section 3.1, we present an overview of the synopsis diffusion approach, which serves as the background. Then, we discuss the vulnerabilities of the synopsis diffusion approach in the presence of compromised nodes in Section 3.2. Section 3.3 describes the problem statement and the assumptions made in our work. In Section 3.4, we discuss our verification protocol, while Section 3.5 describes our attack-resilient protocol. We present simulation results in Section 3.6. Finally, we conclude this chapter in Section 3.7.

## 3.1 Preliminaries: Synopsis Diffusion

We presented an overview of the synopsis diffusion approach in Section 2.3. In this section, we provide more details about the synopsis diffusion algorithms, which build a background for the present chapter.

As discussed in Section 2.3, the synopsis diffusion framework uses a ring topology. During the query distribution phase, nodes form a set of rings around the base station (BS) based on their distance in terms of hops from BS (as illustrated in Figure 2.2). By $T_i$ we denote the ring consisting of the nodes which are $i$ hops away from BS. In the subsequent aggregation period, starting in the outermost ring, each node generates and broadcasts a local synopsis $SG(v)$, where $SG()$ is the synopsis generation function and $v$ is the sensor value relevant to the query. A node in ring $T_i$ will receive broadcasts from all of the nodes in its communication range in ring $T_{i+1}$. It will then combine its own local synopsis with the synopses received from its children using a synopsis fusion function $SF()$ and then broadcast the updated synopsis. Thus, the fused synopses propagate level-by-level until they reach BS, which first combines the received synopses using $SF()$ and then uses the synopsis evaluation function $SE()$ to translate the final synopsis to the answer to the query.

The functions $SG()$, $SF()$, and $SE()$ depend upon the target aggregation function, e.g. Count, Sum, etc. We now describe the duplicate-insensitive synopsis diffusion algorithms for Count and Sum. These algorithms are based on Flajolet and Martin's probabilistic algorithm for counting the number of distinct elements in a multi-set [10].

### 3.1.1 Count

In this algorithm, each node $X$ generates a local synopsis $Q^X$ which is a bit vector of length $\eta > \log N'$, where $N'$ is the upper bound on Count. To generate $Q^X$, node $X$ executes the function $CT(X, \eta)$ given below (Algorithm 1), where $X$ is the node's identifier. $CT()$ can be interpreted as a coin-tossing experiment (with a cryptographic hash function $h()$, modeled as a random oracle whose output is 0 or 1, simulating a fair coin-toss), which returns the number of coin tosses, say $i$, until the first head occurs or $\eta + 1$ if $\eta$ tosses have occurred with no heads occurring. In the synopsis generation function $SG_{count}$, the $i$-th bit of $Q^X$ is set to '1' while all other bits are '0'. Thus, $Q^X$ is a bit vector of the form $0^{(i-1)}10^{(\eta-i)}$ with probability $2^{-i}$.

The synopsis fusion function $SF()$ is the bitwise Boolean OR of the synopses being

**Algorithm 1** $CT(X, \eta)$

```
i=1;
while i < η + 1 AND h(X, i) = 0 do
   i = i + 1;
end while
return i;
```

combined. Each node $X$ fuses its local synopsis $Q^X$ with the synopses it receives from its children.

Let $B$ denote the final synopsis computed by BS by combining all of the synopses received from its child nodes. We observe that $B$ will be a bit vector of length $\eta$ of the form $1^{z-1}0[0,1]^{\eta-z}$, where $z$ is the lowest-order bit in $B$ that is 0. BS can estimate Count from $B$ via the synopsis evaluation function $SE()$: The count of nodes in the network is $2^{z-1}/0.7735$ [9]. The synopsis evaluation function $SE()$ is based on Property 2 below. Intuitively, the number of sensor nodes is proportional to $2^{z-1}$ since no node has set the $z$-th bit while computing $CT(X, \eta)$. We now present a definition which we repeatedly use in this chapter.

**Definition:** The *fused synopsis* of a node $X$, $B^X$, is recursively defined as follows. If $X$ is a leaf node (i.e., $X$ is in the outermost ring), $B^X$ is its local synopsis $Q^X$. If $X$ is a non-leaf node, $B^X$ is the logical OR of $X$'s local synopsis $Q^X$ with $X$'s children's *fused synopses*.

If node $X$ receives synopses $B^{X_1}, B^{X_1}, \ldots, B^{X_d}$ from $d$ child nodes $X_1, X_2, \ldots, X_d$, respectively, then $X$ computes $B^X$ as follows:

$$B^X = Q^X || B^{X_1} || B^{X_1} || \ldots || B^{X_d}, \tag{3.1}$$

where $||$ denotes the bitwise OR operator. Note that $B^X$ represents the sub-aggregate of node $X$, including its descendant nodes. We note that $B^{BS}$ is same as the final synopsis $B$.

Below we present a few important properties of the final synopsis $B$ computed at BS. The first three properties have been derived in [7,10], while Property 4 is documented from our observation. Let $B[i], 1 \leq i \leq \eta$ denote the $i$-th bit of $B$, where bits are numbered

starting from the left. Also, $N$ is the number of nodes present in the network.

**Property 1.** For $i < \log_2 N - 2\log_2\log_2 N$, $B[i] = 1$ with probability $\approx 1$. For $i \geq \frac{3}{2}\log_2 N$, $B[i] = 0$ with probability $\approx 1$.

This result implies that for a network of $N$ nodes, we expect that $B$ has an initial prefix of all ones and a suffix of all zeros, while only the bits around $B[\log_2 N]$ exhibit much variation. This provides an estimate of the number of bits, $\eta$, required for a node's local synopsis. In practice, $\eta = \log_2 N' + 4$ bits are sufficient to represent $B$ with high probability [10], where $N'$ is the upper bound of Count. This result also indicates that the length of the prefix of all ones in $B$ can be used to estimate $N$. Let $z = \min\{i|B[i] = 0\}$, i.e., $z$ is the location of the leftmost zero in $B$. Then $R = z - 1$ is a random variable representing the length of the prefix of all ones in the synopsis. The following results hold for $R$.

**Property 2.** The expected value of $R$, $E(R) \approx \log_2(\phi N)$, where the constant $\phi$ is approximately 0.7735.

This result implies that $R$ can be used as an unbiased estimator of $\log_2(\phi N)$, and it is the basis for the synopsis evaluation function $SE()$, which estimates $N$ as $2^R/\phi$.

**Property 3.** The standard deviation of $R, \sigma_R \approx 1.1213$.

This property implies that estimates of $N$ derived from $R$ will often be off by a factor of two or more in either direction. To reduce the standard deviation of $R$, Flajolet et al. [10] proposed an algorithm named PCSA, where $m$ synopses are computed in parallel. The single synopsis computation algorithm is extended to the PCSA algorithm [10] as follows: In synopsis generation function $SG_{count}$, one synopsis out of $m$ synopses is randomly chosen before $CT()$ is invoked and then, only the chosen synopsis is updated. The synopsis fusion function SF() for each synopsis is bitwise Boolean OR as in the original algorithm. In synopsis evaluation function SE(), the new estimator is the average of all individual $R$'s of these synopses. Flajolet et al. [10] showed that for PCSA, the standard error in the estimate of $N$, i.e., $\sigma_N/N$, is equal to $0.78/\sqrt{m}$.

**Property 4.** If $N$ nodes participate in Count algorithm, the expected number of nodes that will contribute a '1' to the $i$-th bit of the final synopsis $B$ is $N/2^i$. We call these nodes *contributing nodes* for bit $i$ of $B$.

This property is derived from the observation that each node $X$ sets the $i$-th bit of its local synopsis $Q^X$ with probability $2^{-i}$. As an example, for bit $r = E(R) = \log_2(\phi N)$, the expected number of contributing nodes is $1/\phi \approx 1.29$. This result also implies that the expected number of nodes that contribute a '1' to the bits right to the $i$-th bit (i.e., bits $j$, where $i < j \leq \eta$) is approximately $N/2^i$. As an example, the expected number of contributing nodes for bits $j \geq r + 1$ is approximately $1/\phi$.

### 3.1.2 Sum

Considine et al. [7] extended the Count algorithm, described in Section 3.1.1, for computing Sum. The synopsis generation function $SG()$ for Sum is a modification of that for Count, while the fusion function $SF()$ and the evaluation function $SE()$ for Sum are identical to those for Count.

To generate the local synopsis $Q^X$ to represent its sensed value $v_X$, node $X$ invokes the function $CT()$, used for Count synopsis generation, $v_X$ times[1]. In the $i$-th, $1 \leq i \leq v_X$ invocation, node $X$ executes the function $CT(X_i, \eta)$ where $X_i$ is constructed by concatenating its ID and integer $i$ (i.e., $X_i =< X, i >$), and $\eta$ is the synopsis length. The value of $\eta$ is taken as $\log_2 S' + 4$, where $S'$ is an upper bound on the value of Sum aggregate. Note that unlike the local synopsis of a node for Count, more than one bit in the local synopsis of a node for Sum may be equal to '1'. The pseudo code of the synopsis generation function, $SG_{sum}(X, v_X, \eta)$, is presented below (Algorithm 2).

Note that Count can be considered as a special case of Sum where each node's sensor reading is equal to one unit. Considine et al. [7] showed that Properties 1, 2, 3, and 4 described above for Count synopsis also hold for Sum synopsis, with appropriate modifications. Below we present these properties of Sum synopsis, which we will find useful in the

[1]Without loss of generality, each sensor reading is assumed to be an integer.

**Algorithm 2** $SG_{sum}(\mathrm{X}, v_X, \eta)$

$Q^X[j] = 0 \ \ \forall j \ \ 1 \leq j \leq \eta$;
i=1;
**while** $i \leq v_X$ **do**
  $X_i =< X, i >$;
  $j = CT(X_i, \eta)$;
  $Q^X[j] = 1$;
  $i = i + 1$;
**end while**
return $Q^X$;

rest of this chapter. Let $B[i], 1 \leq i \leq \eta$ denote the $i$-th bit of the final synopsis $B$, where bits are numbered starting from the left. Furthermore, $S$ is the Sum of the sensed values of the nodes present in the network.

**Property 1.** For $i < \log_2 S - 2\log_2\log_2 S$, $B[i] = 1$ with probability $\approx 1$. For $i \geq \frac{3}{2}\log_2 S$, $B[i] = 0$ with probability $\approx 1$.

**Property 2.** Let $R$ represent the length of the prefix of all ones in $B$, i.e., $R = z - 1$ where $z = \min\{i|B[i] = 0\}$. The expected value of $R$, $E(R) \approx \log_2(\phi S)$, where the constant $\phi$ is approximately 0.7735.

**Property 3.** The standard deviation of $R, \sigma_R \approx 1.1213$.

Unlike the above properties, Property 4 is not a straightforward extension of its counterpart for Count synopsis. From the construction of the synopsis generation function, $SG_{sum}()$ (Algorithm 2), we observe that if the Sum is $S$, then the function $CT()$ is invoked $S$ times in total considering synopsis generation of all nodes. Each node $X$ gets a chance to set the $i$-th bit of $Q^X$, its local synopsis, $v_X$ times—each time with probability $2^{-i}$. So, the expected number of contributing nodes for the $i$-th bit of $B$ not only depends on the total number of nodes $N$ and the value of $i$ but also on the distribution of sensor readings.

**Property 4.** The expected number of invocations of $CT()$ that will contribute a '1' to the $i$-th bit of the final synopsis $B$ is $S/2^i$, where $S$ is the value of Sum.

As an example, with $r = E(R) = \log_2(\phi S)$, the expected number of invocations of $CT()$ which set the $r$-th to '1' is $1/\phi \approx 1.29$. This result also implies that the expected number of contributing nodes for bit $r$ is less than $1/\phi$. Furthermore, the expected number

of invocations of $CT()$ that contribute a '1' to the bits right to the $i$-th bit (i.e., bits $j$, where $i < j \leq \eta$) is approximately $S/2^i$. As an example, the expected number of invocations of $CT()$ that contribute a '1' to the bits right to the $r$-th bit is approximately $1/\phi$, which implies that the expected number of contributing nodes for the bits to the right of the $r$-th bit is less than $1/\phi$.

Similarly, as in the case of Count, the PCSA algorithm [10] can be used to reduce the standard deviation of the estimate for Sum, where $m$ synopses are computed in parallel. The single synopsis computation algorithm is extended to the PCSA algorithm [10] as follows: In synopsis generation function $SG_{sum}$, one synopsis out of $m$ synopses is randomly chosen before each invocation of $CT()$ and only the chosen synopsis is updated after that particular invocation of $CT()$.

## 3.2 Attacks on Synopsis Diffusion

As discussed before (ref. Section 2.3), the synopsis diffusion aggregation framework is vulnerable to many attacks that can be launched by a compromised node. These attacks include jamming at the physical or link layers, route disruption, message flooding, message dropping, message modification, false data injection, and many others. In this dissertation, we aim to defend against an important subclass of the insider attacks which can potentially corrupt the final result of the aggregation query. In particular, the focus of this chapter is on the falsified sub-aggregate attack against Count or Sum algorithm, in which a compromised node falsifies the aggregate value it is relaying to its parents in the aggregation hierarchy. Below we describe how this attack can be launched and its impact on the final result.

**The Falsified Sub-Aggregate Attack.** Since BS estimates the aggregate based on the lowest-order bit $z$ that is '0' in the final synopsis, a compromised node $C$ would need to falsify its fused synopsis $B^C$ such that it would affect the value of $z$. It can accomplish this quite easily by simply inserting '1's in one or more bits in positions $j$, where $z \leq j \leq \eta$, in $B^C$ which it broadcasts to its parents. Let $\hat{B^C}$ denote the synopsis finally broadcast by

node $C$. Note that the compromised node $C$ does not need to know the true value of $z$; it can simply set some higher-order bits to '1' with the expectation that this will affect the value of $z$ computed by BS.

Since the synopsis fusion function is a bitwise Boolean OR, the fused synopsis computed at any node which is at the higher level than node $C$ on the aggregation hierarchy will contain the false contributions of node $C$. We observe that when a node $X$ computes the fused synopsis $\hat{B^X}$, $X$ is not sure if $\hat{B^X}$ contains any false '1's contributed by a compromised node lower in the hierarchy. The observation is true is for BS when it computes the final synopsis $\hat{B}$. Note that $\hat{B^X}$ is the same as $X$'s true fused synopsis, $B^X$ if there is no false '1' injection attack.

Let $\hat{R} = \hat{z} - 1$, where $\hat{z}$ be the lowest-order bit that is '0' in the received final synopsis $\hat{B}$. Also, let $R = z - 1$, where $z$ is the lowest-order bit that is '0' in the correct final synopsis $B$. Then BS's estimate of the aggregate will be larger than the correct estimate by a factor of $2^{\hat{R}-R}$. It is easy to see that, with the above technique, the compromised node can inject a large amount of error in the final estimate of BS.

We also observe that even a single node can launch this attack with a high rate of success because the use of multi-path routing in the synopsis diffusion approach makes it highly likely that the falsified synopsis will be propagated to BS. If $\alpha$ is the packet loss rate and if each node has $f$ parents in the aggregation hierarchy, then the probability of success for this attack is $(1 - \alpha^f)^g$, if the compromised node is $g$ hops away from BS. As an example, if $\alpha = 0.1$, $f = 3$, and $g = 5$, then the probability that the attack will succeed is 99.5%.

On the other hand, it is very hard to launch an attack which results in the aggregate estimated at BS being lower than the true estimate. This is because a compromised node $C$'s changing bit $j$ in its fused synopsis, $B^C$ from '1' to '0' has no effect if there is another node $X$ that contributes a '1' to bit $j$ in its local synopsis $Q^X$ and hence to bit $j$ in the final synopsis $B$. To make this attack a success, the attacker must compromise all of the possible paths from node $X$ to BS so that $X$'s '1' cannot reach BS, which is hard to achieve. If there

is more than one node which contributes to the same bit, then it is even harder. As an example, in Count algorithm, half of the nodes are likely to contribute to the leftmost bit of the synopsis, one-fourth of the nodes contribute to the second bit, and so on. We note that there are bits in the synopsis to which only one or two nodes contribute. However, it is very hard to predict in advance which nodes will be contributing to these particular bits if BS broadcasts, along with the query request, a random seed to be used with the hash function in the synopsis generation function. Hence, we can safely assume that this attack is extremely difficult to launch. In the rest of this chapter, we do not further discuss this second type of attack (changing bit '1' to '0'). We restrict our discussion to the first type of attack (changing bit '0' to '1'), which we call the *false '1' injection attack*. From now on, by 'falsified sub-aggregate attack' we will mean the 'false '1' injection attack'. That means the goal of our attacker is only to increase the estimate of the aggregate.

## 3.3 Problem Statement and Assumptions

### 3.3.1 Problem Description

In a sensor network where a fraction of the nodes are potentially compromised, there are three sources that contribute to the error in the estimate of the aggregate: (i) error due to packets loss, (ii) error due to the approximation algorithm used, e.g., Flajolet and Martin's probabilistic counting algorithm [10], and (iii) error injected by compromised nodes.

The first two types of error are already addressed in the synopsis diffusion aggregation framework. Our contribution is complementary to the previous work; our objective is to detect and then filter out the third type of error.

In particular, our goal is two fold: (a) to detect the falsified sub-aggregate attack, and (b) to enable BS to obtain the 'true' estimate of the aggregate even in the presence of the falsified sub-aggregate attack. By 'true' estimate, we mean the estimate of the aggregate which BS would compute if there were no compromised nodes. More specifically, goal (a) is to detect if $\hat{B}$, the synopsis received at BS is the same as the 'true' final synopsis $B$, and

goal (b) is to compute $B$ from $\hat{B}$.

Without loss of generality, we present our algorithms in the context of Sum aggregate. As Count is a special case of Sum, where each node reports a unit value, these algorithms are readily applicable to Count aggregate also.

### 3.3.2 Assumptions

We here discuss the system assumptions and the security model assumed in this chapter.

**System Assumptions** We assume that the sensor nodes form a multi-hop network with BS as the central point of control. We also assume that sensor nodes are similar to the current generation of sensor nodes, e.g., Mica2 motes [48], in their computational and communication capabilities and power resources, while BS is a laptop class device supplied with long-lasting power.

**Security Assumptions** We assume that BS cannot be compromised and it uses a protocol such as $\mu$Tesla [15] to authenticate its broadcast messages to the network nodes. We also assume each node shares a pair-wise key with BS. Let the key of the node with ID $X$ be denoted as $K_X$. To authenticate a message to BS, a node $X$ sends a MAC (Message Authentication Code) generated using the key $K_X$. Some of our countermeasures (e.g., the attack-resilient protocol in Section 3.5) further assume that each pair of neighboring nodes has a pairwise key to authenticate its mutual communication.

We assume that if a node is compromised, all of the information it holds will also be compromised. We conservatively assume that all malicious nodes can collude or can be under the control of a single attacker. We use a Byzantine fault model, where the adversary can inject any message through the compromised nodes. Compromised nodes may behave in arbitrarily malicious ways, which means that the *sub-aggregate* of a compromised node can be arbitrarily generated. The goal of the attacker is to launch a stealthy attack to cause the querier to accept a false aggregate while remaining undetected as in [37].

**Notation** A list of notations used in this chapter is given in Table 3.1.

Table 3.1: Notations Used in Describing the Secure Sum Protocols

| Symbol | Meaning |
|---|---|
| $N$ | the total number of nodes |
| $N'$ | an upper bound on the total number of nodes |
| $v_X$ | sensed value of node X |
| $S$ | the value of Sum aggregate |
| $S'$ | an upper bound on the value of Sum aggregate |
| $K_X$ | symmetric key shared between node $X$ and the BS |
| $MAC(K_X, M)$ | message authentication code of message $M$ computed using key $K_X$ |
| $X \rightarrow Y$ | X sends a message to Y |
| $X \rightarrow *$ | X broadcasts a message to one hop neighbors |
| $X \rightarrow\rightarrow *$ | X broadcasts a message to the network |
| $< a_1 \ , \ a_2 >$ | concatenation of string $a_1$ and $a_2$ |
| $\|\|$ | the bitwise OR operator |
| $t$ | number of compromised nodes |
| $\eta$ | the length of the synopsis |
| $Q^X$ | the local synopsis of node $X$ |
| $B^X$ | the fused synopsis of node $X$ if no attack is in the network |
| $\hat{B^X}$ | the fused synopsis actually computed by node $X$ |
| $B$ | the final synopsis at $BS$ if no attack is in the network |
| $\hat{B}$ | the final synopsis actually computed by $BS$ |
| $R$ | length of the prefix of all '1's in $B$ |
| $\hat{R}$ | length of the prefix of all '1's in $\hat{B}$ |

## 3.4 Verification Algorithm

Section 3.2 discussed how an adversary can launch the falsified sub-aggregate attack, in which the adversary falsely sets one or more bits in the synopsis to increase BS's estimate of Sum. Here, we present an algorithm which enables BS to verify the received synopsis. First, we present a naive solution that serves as the background but turns out to be prohibitively

expensive. Then, we present our verification algorithm.

### 3.4.1 Background

A straightforward solution to detect the falsified sub-aggregate attack is as follows. In the query dissemination phase, BS broadcasts an aggregation query message which includes a random value, *Seed*, associated to the current query. In the subsequent aggregation phase, along with the fused synopsis $\hat{B^X}$, each node $X$ also sends a MAC towards BS authenticating its sensed value $v_X$. Node $X$ uses *Seed* and its own ID to compute its MAC. As a result, BS is able to detect any false '1' bits inserted in the final synopsis $B$ by simulating the execution of the synopsis generation function for each node $X$.

In particular, if node $X$ contributes to bits $i_1, i_2, \ldots, i_\zeta$ in its local synopsis $Q^X$, it generates a MAC, $M = MAC(K_X, L)$, where the format of $L$ is $[X|v_X|i_1, i_2, \ldots, i_\zeta|Seed]$ and $K_X$ is the key that node $X$ shares with BS. We observe that this approach requires each node's MAC to be forwarded to BS, which incurs $O(N)$ per-node communication overhead, and hence, this approach is not suitable for a sensor network. Our verification algorithm presented below also uses similar MACs but reduces the total number of them.

In the following discussion, we present an overview of our verification protocol followed by a description of the protocol operation. Then, we discuss the correctness of this protocol, analyze its performance and security, and compare it with other existing protocols.

### 3.4.2 Protocol Overview

We observe that, in general, BS can verify the final synopsis if it receives one valid MAC for each '1' bit in the synopsis. In fact, to verify a particular '1' bit, say bit $i$, BS does not need to receive authentication messages from all of the nodes which contribute to bit $i$. As an example, more than half of the nodes are likely to contribute to the leftmost bit of the synopsis (Property 4 of Sum synopsis), while to verify this bit, BS needs to receive a MAC only from one of these nodes. Hence, it is sufficient for each node in the aggregation hierarchy to forward only one MAC corresponding to each '1' bit in the synopsis.

Our verification algorithm further reduces the communication overhead per node. In particular, each node forwards one MAC each for at most $k$ bits in the synopsis, where $k$ is a small constant (e.g., 5). This ensures, as shown later, that BS will be able to authenticate the rightmost $k$ '1' bits in the final synopsis. Then, as proven later, BS can securely compute $R$ with very high probability, where $R$ is the length of the prefix of consecutive '1's in the final synopsis $B$. We remind the reader that $R$ determines the value of the final aggregate. The higher the value of $k$, the greater the probability that our scheme will detect a false '1' bit in the final synopsis.

Our proposed protocol is executed simultaneously with the original synopsis diffusion algorithm [7,9], i.e., BS can compute and verify the synopsis at the same time. In particular, along with the fused synopsis, computed following the original synopsis diffusion algorithm, each node in the aggregation hierarchy forwards $k$ MACs.

### 3.4.3 Protocol Operation

The verification protocol runs concurrently with the original synopsis diffusion protocol [7,9] as described below. We remind the reader that in the original protocol, $m \geq 1$ synopses are computed. However, for ease of exposition, we describe our verification protocol with respect to one single synopsis. Each synopsis can be verified independently and hence our algorithm is readily applicable for computing multiple synopses.

#### Query dissemination

In the query dissemination phase, BS broadcasts the name of the aggregate to compute and verify a random number ($Seed$) and the chosen value of $k$. The query that BS broadcasts is as follows:

$$BS \rightarrow\rightarrow * : \langle\ F_{agg}, Seed, k\ \rangle,$$

where $F_{agg}$ is the name of the aggregate (e.g., 'Sum'). During this phase, nodes form a set of rings around BS based on their distance in hops from BS, as in the original synopsis diffusion algorithm [7, 9].

**Aggregation phase**

Each node executes the aggregation phase of the original synopsis diffusion protocol [7, 9] along with sending some additional authentication messages. We recall from Section 3.2 that in the presence of the falsified sub-aggregate attack the fused synopsis, $\hat{B^X}$ computed at a node $X$ can be different from node $X$'s true fused synopsis $B^X$.

**Algorithm 3** $Verifiable Aggregation(X, Q^X, k)$

*receive* $\{< B^{\hat{X}_1}, \mathfrak{M}^{X_1} >, < B^{\hat{X}_2}, \mathfrak{M}^{X_2} >, \ldots, < B^{\hat{X}_d}, \mathfrak{M}^{X_d} >\}$ *from d child nodes*;
$\hat{B^X} = Q^X || B^{\hat{X}_1} || B^{\hat{X}_2} || \ldots || B^{\hat{X}_d}$; // aggregate received synopses with local one
$I_j^X =$ *the index of the* $j-$*th rightmost '1' bit in* $\hat{B^X}$, *for* $1 \leq j \leq k$;
*generate one MAC for each bit* $I_j^X$ *in* $Q^X$, *for* $1 \leq j \leq k$;
*construct the union* $\mathfrak{M}$ *of the received MACs and the self*$-$*generated ones*;
*randomly select* $\mathfrak{M}^X = \{M_{I_1^X}, M_{I_1^X}, \ldots, M_{I_k^X}\}$ *from* $\mathfrak{M}$;
*broadcast* $< \hat{B^X}, \mathfrak{M}^X >$ *to parents*;

We start the description of this phase by introducing the following notations. With $M_i^X$ we denote the MAC, generated by node $X$, authenticating the $i$-th bit of its local synopsis $Q^X$. Note that $M_i^X$ is required to be generated only if $Q^X[i] = 1$, i.e. there are no MAC for '0' bits. Furthermore, with $M_i$ we indicate one arbitrary element of the following set: $\{M_i^X |\ Q^X[i] = 1\}$. As an example, if two nodes $X_1$ and $X_2$ set bit $i$ to be '1' in their local synopses, then $M_i$ corresponds to either $M_i^{X_1}$ or $M_i^{X_2}$.

When a node $X$ broadcasts $\hat{B^X}$ to its parents, it also forwards one MAC for each of the rightmost $k$ '1's in $\hat{B^X}$ synopsis[2]. The corresponding message is as follows:

$$X \rightarrow * : \langle\ \hat{B^X}, \mathfrak{M}^X\ \rangle,$$

where $\mathfrak{M}^X$ represents a set of $k$ MACs, $\{\ M_{I_1^X}, M_{I_2^X}, \ldots, M_{I_k^X}\}$ with $I_j^X$ denoting the index of the $j$-th rightmost '1' bit in $\hat{B^X}$. We remind the reader that the bits in the synopsis are

[2]We note that to reduce the message size, a source node generates one single MAC to authenticate all of the bits to which it contributes, say, bit $i$ and bit $j$. However, to help the exposition, our illustrations list these MACs separately as $M_i$ and $M_j$.

numbered from left to right. As an example, if $\hat{B^X} = 1111110101000$, $I_1^X = 10$ (the index of the rightmost '1' bit is 10), $I_2^X = 8$ (the index of the second rightmost '1' bit is 8), and so on.

It is worth noting that all of the $k$ MACs in $\mathfrak{M}^X$ are not necessarily generated by node $X$. In fact, $X$ randomly selects these $k$ MACs from the pool of MACs received from its child nodes or generated by itself. In general, node $X$ might have more than one MAC (received from its child nodes or generated by itself) for one particular '1' bit in $\hat{B^X}$. However, for each of the rightmost $k$ '1' bits, node $X$ forwards just one of these MACs (i.e., $k$ MACs in total). Later, we will see that $k$ acts as a parameter which trades between the communication overhead and the level of security. The pseudo code run by each node $X$ is presented below as the procedure $VerifiableAggregation()$ (Algorithm 3).

Finally, after receiving the messages from its child nodes, BS computes the final synopsis $\hat{B}$ and verifies the received MACs. If it has received one valid MAC for each of the rightmost $k$ '1's present in $\hat{B}$, the verification succeeds and $\hat{B}$ is accepted. Otherwise, the verification fails.

**An Example** Figure 3.1 illustrates an example of the protocol operation where $k = 5$. Assume that node $P$ is in ring $i$ and nodes $X$, $Y$ and $Z$ are in ring $i + 1$. Nodes $X$, $Y$ and $Z$ send to $P$ their fused synopses, $\hat{B^X} = 111111010100$, $\hat{B^Y} = 111111100000$, and $\hat{B^Z} = 111110001000$, respectively. Node $X$ also forwards one MAC each for the 4th, 5th, 6th, 8th and 10th bit, which are denoted as $M_4$, $M_5$, $M_6$, $M_8$, and $M_{10}$, respectively. Similarly, $P$ receives MACs $M_3$, $M_4$, $M_5$,$M_6$, and $M_7$ from node $Y$, and $M_2$, $M_3$, $M_4$, $M_5$, and $M_9$ from node $Z$. We note that the MACs received by $P$ for the 4th bit may be different depending on the MAC's original source nodes. The same consideration applies for other bits. Let the local synopsis of node $P$, $Q^P$ be 00100000000. $P$ fuses all of the received synopses ($\hat{B^X}$, $\hat{B^Y}$, and $\hat{B^Z}$), including its local synopsis ($Q^P$) to compute its fused synopsis ($\hat{B^P}$), and sends it to the parent nodes in ring $i - 1$. In this example, $\hat{B^P} = 111111111100$. $P$ also forwards the MACs for the five rightmost '1' bits ($M_6$, $M_7$, $M_8$, $M_9$, and $M_{10}$) to its

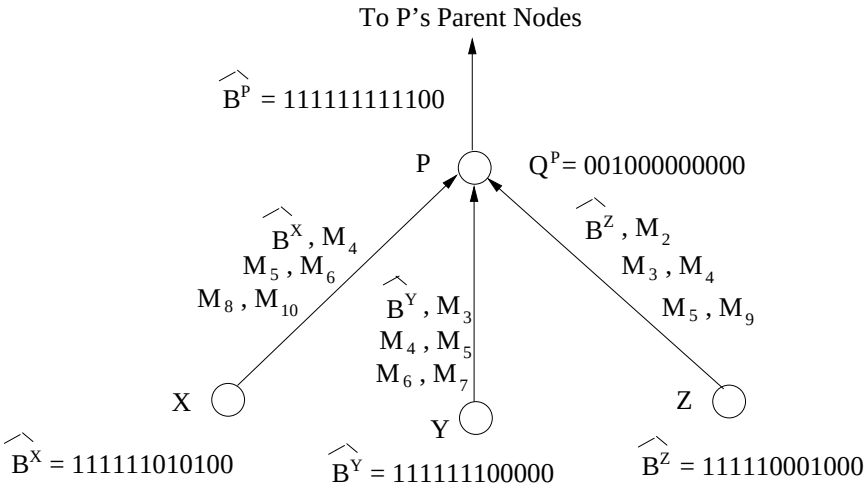



Figure 3.1: Aggregation Phase of the Verification Algorithm. Along with the fused synopsis, nodes X, Y, and Z forward a set of MACs to parent node P. Node X forwards one MAC for each of the five rightmost '1's in its fused synopsis, so do nodes Y and Z. After fusing the received synopses with the local synopsis, node P similarly forwards 5 MACs and drops the others.

parent nodes.

### 3.4.4 Correctness

To prove the correctness of the above verification protocol, we need to answer the following questions.

- If no attacker is present, does the verification process end with a 'success'?
- If the false '1's injection attack is launched, does this protocol detect it?

To answer the first question, we recall that in the absence of the attack, by definition each node $X$'s fused synopsis $\hat{B^X}$ is the same as $B^X$, and BS receives the true final synopsis $B$. That means each node $X$ in the aggregation hierarchy forwards one MAC for each of the rightmost $k$ '1's in its fused synopsis $B^X$. To see if this ensures that BS will receive at least one MAC for each of the rightmost $k$ '1's in the received final synopsis $B$, we present Claim 3.4.1 below.

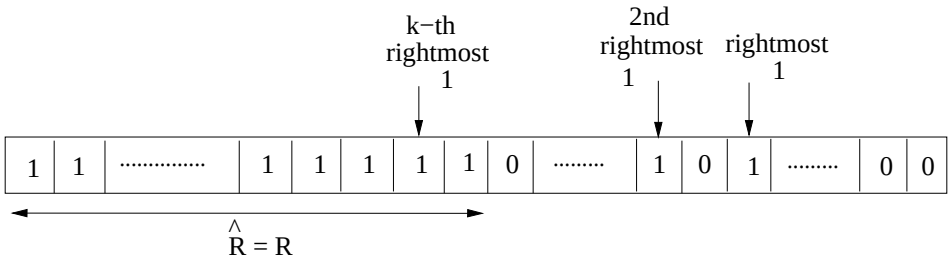
**Claim 3.4.1.** *Let no attacker be present in the network. Let $\hat{B^X}$ denote the fused synopsis of node $X$. Let $I_j^X$ denote the bit index of the $j$-th rightmost '1' in $\hat{B^X}$ and $I_j$ denote the bit index of the $j$-th rightmost '1' in $\hat{B}$, $j \geq 1$. For any node $X$ which has one or more '1' bits in $\hat{B^X}$, the following inequality holds: $I_j^X \leq I_j$.*

*Proof (by contradiction).* As no attack is launched, by definition $\hat{B^X}$ is same as $B^X$ and $\hat{B}$ is same as $B$. Assume that there is one node $X$ for which this claim does not hold. That means there exists one bit in $B^X$, say the $i$-th bit, which is the $j$-th rightmost '1' in $B^X$, $j \geq 1$, and $i = I_j^X > I_j$. This implies that the number of '1's to the right of the $(i-1)$-th bit in $B^X$ is $j$, but that in $B$ is less than $j$. That means there is at least one '1' bit in $B^X$ which is reset to '0' in $B$. This contradicts the fact that the synopsis fusion function, $SF()$, is a bitwise Boolean OR, i.e. $B = SF(B^X, B^{Y_1}, B^{Y_2}, \ldots, B^{Y_n}) = B^X \mid\mid B^{Y_1} \mid\mid B^{Y_2} \mid\mid \ldots \mid\mid B^{Y_n}$. □

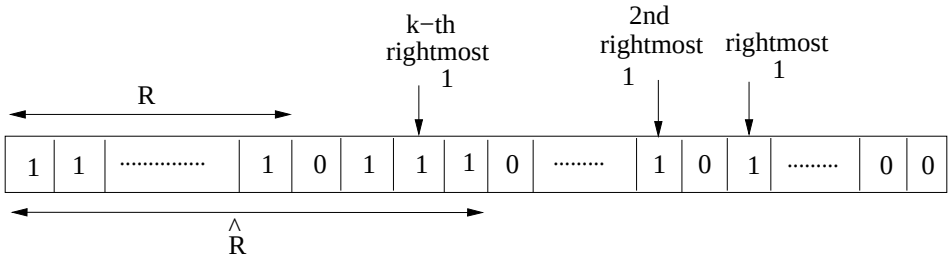
We assume that a node $X$'s message to one of its parents, $P$, can be lost due to communication failure but it cannot be partially or wrongly received—node-to-node authentication and acknowledgement mechanisms can be used to enforce this property. It implies that if $B^X$ reach $P$, all of the $k$ MACs sent by $X$ also reaches $P$.

In our verification protocol, each node $X$ in the aggregation hierarchy forwards one MAC for each of the rightmost $k$ '1's in its fused synopsis. Claim 3.4.1 implies that any other node $P$, BS included, in the higher level of the hierarchy will receive at least one MAC for each of the rightmost $k$ '1's in $P$'s fused synopsis. This means that in the absence of the attack, this protocol will end with a 'success'. In the example illustrated in Figure 3.1, nodes $X$, $Y$, and $Z$ forward one MAC for each of the rightmost five '1's in the corresponding fused synopsis. This ensures that node $P$ receives at least one MAC for each of the rightmost five '1's in $P$'s fused synopsis.

To answer the second question we recall that only the rightmost $k$ '1's in the final synopsis $\hat{B}$ are verified, i.e., BS does not check the validity of other '1's in $\hat{B}$. We need to see whether the above check is sufficient for the BS to verify if there is any false '1' bit in




(a) No '0' occurs to the left of the $k$-th rightmost '1' in the final synopsis $B$.




(b) A '0' occurs to the left of the $k$-th rightmost '1' bit in the final synopsis $B$.

Figure 3.2: Two Possibilities with respect to Event $E^k$

the final synopsis $\hat{B}$.

Now, we introduce $E^k$, which denotes the following event: A '0' bit appears to the left of the $k$-th rightmost '1' bit in $B$. Below we discuss the possibility of a false '1' bit in $\hat{B}$ not being detected considering both of the cases: (a) event $E^k$ does not occur in synopsis $B$, and (b) event $E^k$ occurs in synopsis $B$. We discuss these two cases with an example illustrated in Figure 3.2.

Case (a): No '0' appears to the left of the $k$-th rightmost '1', say bit $i$, in $B$. In this case, the attacker can manage to change a bit from '0' to '1' only on the right of the $k$-th rightmost '1' bit in $B$. Then, the number of '1's to the right of bit $i$ increases from $B$ to $\hat{B}$. Because BS will check the MACs of the $k$ rightmost '1' bits in $\hat{B}$, the attacker will be able to detect the injected false '1'. In fact, the attacker is not able to produce a valid MAC corresponding to that bit. That means that, in this case, the attacker cannot falsely increase the prefix length $\hat{R}$ in $\hat{B}$ from the true prefix length $R$ in $B$. In the example shown in Fig. 3.2(a), $\hat{R} = R = I_k + 1$ where $I_j$ represents the index of $j$-th rightmost '1'.

Case (b): A '0', say $j$-th bit, appears to the left of the $k$-th rightmost '1', say bit $i$, in $B$. From case (a) we know that the attack can be detected if it changes a bit (from '0' to '1') on the right of $i$. However, in this case the attacker can manage to change a '0' to '1' on the left of the $i$-th bit. Because BS will just check the MACs of the $k$ rightmost '1' bits in $\hat{B}$, this will result in the attack not being detected. In the example shown in Fig. 3.2(b), bit $(I_k - 2)$ is '0' where $I_j$ represents the index of $j$-th rightmost '1'. If the attacker falsely injects a '1' at bit $(I_k - 2)$, the false '1' would not be detected in our verification protocol. As a result, BS overestimates the value of $R$: BS's estimate would be $\hat{R} = I_k + 1$, while $R = I_k - 3$.

We remind the reader that the aim of the attacker is to inject false '1's in $B$ while being undetected. We are now interested in computing the probability for the attacker to succeed. From cases (a) and (b), we observed that this can happen only if event $E^k$ occurs. So, the probability that the attacker can succeed is the same as the probability of event $E^k$ to occur. Below we study the probability of event $E^k$ to occur.

To compute $E^k$, we will use the following Lemmas, Lemma 3.4.2 and Lemma 3.4.3.

**Lemma 3.4.2.** *Let the value of Sum be $S$ and $r$ be the expected value of $R$ in $B$. The probability that $B[i] = 0$, with $r - a \leq i \leq r + a$ and $a \geq 0$, is:*

$$Pr[B[i] = 0] \approx e^{-\frac{(1/\phi)}{(2^{i-r})}},$$

*where $\phi = 0.7735$.*

*Proof.* From Property 2 of Sum synopsis (ref. Section 3.1.2), we see that $r$ is $\log_2(\phi S)$. As observed in Section 3.1.2, the function $CT()$ in $SG_{sum}$ algorithm is invoked $S$ times in total considering synopsis generation of all nodes. A bit $j$ in the final synopsis $B$ is '0' only if none of the above $S$ invocations of $CT()$ returns $j$. So, we see that:

$$Pr[B[j] = 0] = (1 - \frac{1}{2^j})^S.$$

To calculate the probability of $r$-th bit[3] being '0', we substitute $j$ with $r = \log_2(\phi S)$ and we get:

$$Pr[B[r] = 0] = (1 - \frac{1}{2^r})^S = (1 - \frac{1}{\phi S})^S \approx e^{-\frac{1}{\phi}}.$$

In general, for the $i$-th bit where $r - a \leq i \leq r + a, a \geq 0$, we get:

$$Pr[B[i] = 0] = (1 - \frac{1}{2^i})^S = (1 - \frac{1}{2^{i-r}2^r})^S = (1 - \frac{1}{2^{i-r}\phi S})^S \approx e^{-\frac{(1/\phi)}{(2^{i-r})}}.$$

□

We observe from Lemma 3.4.2: The probability that $B[j] = 0$ is determined by only the distance of the $j$-th bit from the $r$-th bit, where the value of $r$ is $\log_2(\phi S)$. Furthermore, in Lemma 3.4.3 below, we observe that the bits close (left or right) to bit $r$ or far to the right of bit $r$ can be considered as independent.

**Lemma 3.4.3.** *Let the value of Sum be $S$ and $r$ be the expected value of $R$ in $B$. The value ('0' or '1') of any two bits $j$ and $j'$ in $B$, with $j \geq (r - b)$, $j' \geq (r - b)$, $j \neq j'$, $b << log_2 S$ are independent.*

*Proof.* By construction of $SG_{sum}$ algorithm (ref. Section 3.1.2), we get the following two relations:

$$Pr[B[j] = 0] = (1 - \frac{1}{2^j})^S \tag{3.2}$$

$$Pr[B[j] = 0, B[j'] = 0] = (1 - \frac{1}{2^j} - \frac{1}{2^{j'}})^S \tag{3.3}$$

[3]Strictly speaking, as $r$ is a real number, '$r$-th bit' is not a well-defined bit. However, as we are making an average case analysis, we refer to 'bit $r$' for ease of exposition.

Then, by basic rules of probability, we have:

$$Pr[B[j] = 0 \mid B[j'] = 0] = \frac{Pr[B[j] = 0, B[j'] = 0]}{Pr[B[j'] = 0]} = \frac{(1 - \frac{1}{2^j} - \frac{1}{2^{j'}})^S}{(1 - \frac{1}{2^{j'}})^S} \tag{3.4}$$

From Relation 3.2 and Relation 3.4, we get:

$$\frac{Pr[B[j] = 0|B[j'] = 0]}{Pr[B[j] = 0]} = \frac{(1 - \frac{1}{2^j} - \frac{1}{2^{j'}})^S}{(1 - \frac{1}{2^{j'}})^S(1 - \frac{1}{2^j})^S} = (1 - \frac{1}{(2^j - 1)(2^{j'} - 1)})^S \tag{3.5}$$

So, we have:

$$\frac{Pr[B[j] = 0|B[j'] = 0]}{Pr[B[j] = 0]} \approx (1 - \frac{1}{2^j 2^{j'}})^S \approx e^{-\frac{S}{2^j 2^{j'}}} \leq e^{-\frac{S}{(\frac{\phi S}{2^b})(\frac{\phi S}{2^b})}} = e^{-\frac{2^{2b}}{\phi^2 S}} \approx 1 \tag{3.6}$$

□

Using Lemma 3.4.2 and Lemma 3.4.3, we establish the following Claim.

**Claim 3.4.4.** *$Pr[E^k] < 0.001$, for $k \geq 5$.*

*Proof.* The main idea behind this proof is as follows. Lemma 3.4.2 shows that the probability of a bit in the synopsis being '0' (or '1') depends on its closeness to the $r$-th bit: It rapidly decreases (increases) for the bits to the left of the $r$-th bit and rapidly increases (decreases) for the bits to the right of the $r$-th bit. That means it is very unlikely to find a '0' far to the left of bit $r$ and a '1' far to the right of bit $r$. However, for event $E^k$ to occur, one '0' has to occur somewhere, say at bit $j$, and $k$ '1's have to occur to the right of bit $j$. Intuitively, the most likely 'place' where the bit pattern associated to event $E^k$ may occur is close to $r$. We exploit the above intuition to establish the claim.

Let $F_j^k$ represent the event that $k$ '1's appear to the right of bit $j$ in $B$, $j \geq 1$. Moreover, let $E_j^k$ represent the event that bit $j$ is '0' and at the same time $F_j^k$ occurs. We can express

the probability of event $E_j^k$ as follows.

$$Pr[E_j^k] = Pr[B[j] = 0] \cdot Pr[F_j^k \mid B[j] = 0] \tag{3.7}$$

From Lemma 3.4.2, we see that for bits $j < (r-2)$, $Pr[B[j] = 0] \approx 0$, so $Pr[E_j^k] \approx 0$. So, we need to evaluate Expression 3.7 only for bits $j \geq (r-2)$.

Thanks to Lemma 3.4.3, for $j \geq r-2$ Expression 3.7 becomes:

$$Pr[E_j^k] = Pr[B[j] = 0] \cdot Pr[F_j^k]. \tag{3.8}$$

Let $p_j$ denote $Pr[B[j] = 1]$. For a particular bit $j$, let us define the following groups of bits, where $\eta$ is the length of the synopsis:

$$g_1 = (j+1, j+k+1, j+k+2, \ldots, \eta);$$

$$g_2 = (j+2, j+k+1, j+k+2, \ldots, \eta);$$

$$\ldots\ldots$$

$$g_k = (j+k, j+k+1, j+k+2, \ldots, \eta).$$

We observe that for event $F_j^k$ to occur at least one bit in each group $g_i, 1 \leq i \leq k$ has to be '1'. Let $G_i$ denote the event that at least one bit in group $g_i$ is '1'. By Boole's inequality[4], we get that for any $i$, $1 \leq i \leq k$,

$$Pr[G_i] \leq (p_{j+i} + p_{j+k+1} + p_{j+k+2} + \ldots + p_\eta). \tag{3.9}$$

[4]Boole's inequality says that for any finite or countable set of events, the probability that at least one of the events happens is no greater than the sum of the probabilities of the individual events.

Hence, by basic rules of probability we get the following inequality.

$$Pr[F_j^k] \leq (p_{j+1} + p_{j+k+1} + p_{j+k+2} + \ldots + p_\eta) \cdot (p_{j+2} + p_{j+k+1} + p_{j+k+2} + \ldots + p_\eta) \cdot$$

$$\ldots \cdot (p_{j+k-1} + p_{j+k+1} + p_{j+k+2} + \ldots + p_\eta) \cdot (p_{j+k} + p_{j+k+1} + p_{j+k+2} + \ldots + p_\eta) \quad (3.10)$$

Substituting Expression 3.10 for $\Pr[F_j^k]$ in Expression 3.8, we get an upper bound for $\Pr[E_j^k]$.

Also, by Boole's inequality we get the following:

$$Pr[E^k] \leq \sum_{j=r-2}^{\eta-k} Pr[E_j^k], \quad (3.11)$$

where $\eta$ is the length of the synopsis. Finally, substituting $\Pr[E_j^k]$ in Inequality 3.11 by the upper bound of $\Pr[E_j^k]$, we get an upper bound of $Pr[E^k]$.

Now let $k$ be 5. Using Lemma 3.4.2 and Expression 3.10 with $j = r - 2$, we get $F_{r-2}^5 \leq 0.074$, and then, from Expression 3.8, using Lemma 3.4.2, we get $Pr[E_{r-2}^5] \leq 0.00042$. Similarly, we get $Pr[E_{r-1}^5] \leq 0.00040$ and $Pr[E_r^5] \leq 0.00008$. We also find that $Pr[E_j^5] \approx 0$ for $j > r$. Substituting the above values of $\Pr[E_j^5]$ in the Inequality 3.11, we get $Pr[E^5] < 0.001$. By definition of $E^k$, if $k' > k$, then $Pr[E^{k'}] \leq Pr[E^k]$. Hence, $Pr[E^k] < 0.001$ for $k \geq 5$.

□

Furthermore, similarly as $k = 5$ in Claim 3.4.4, we computed an upper bound of $Pr[E^k]$ for $k = 4, 6, 7$, which are as follows: $Pr[E^4] < 0.0085$, $Pr[E^6] < 5.2 \times 10^{-5}$, and $Pr[E^7] < 1.5 \times 10^{-6}$.

### 3.4.5 Protocol Analysis and Comparison

Here, we analyze the performance and the security issues of our verification algorithm and compare them with other existing algorithms. To the best of our knowledge, only three other verification algorithms have been proposed: (i) by Chan et al. [37], (ii) by Yang et al. [20], and (iii) by Garofalakis et al. [19].

Table 3.2 compares these four algorithms over latency, communication overhead, approximation error, robustness to communication loss, and security. We also discuss the table entries for each of the considered features.

Table 3.2: Comparing Our Verification Algorithm with Others

| Algorithms | Latency | Communication overhead | Approx. error in Sum estimate | Robust to communication loss | Deterministic attack detection |
|---|---|---|---|---|---|
| Our Verification Algo | 1 epoch | $O(mk)$ | Yes | Yes | No |
| Algo by Chan et al. [37] | 2 epochs | $O(\Delta \log^2 N)$ | No | No | Yes |
| Algo by Yang et al. [20][5] | 1 epoch | $O(n_g)$ | No | No | No |
| Algo by Garofalakis et al. [19] | 1 epoch | $O(m \log S)$ | Yes | Yes | Yes |

- Latency: Our verification protocol completes within one epoch[6], simultaneously with the original synopsis diffusion algorithm. Chan et al.'s algorithm takes two epochs to complete the verification, while Yang et al.'s and Garofalakis et al.'s algorithms take one epoch.

- Communication Overhead: During our protocol in the worst case each node has to forward $k$ MACs for each synopsis. If $m$ synopses are computed, then the worst case

[6] As defined in the prior work [4], an epoch represents the amount of time a message takes to reach BS from the farthest node on the aggregation hierarchy.

per-node communication overhead is $O(mk)$. We note that while for ease of exposition we presented our protocol to compute just one synopsis, our protocol computes multiple synopses in practice. In Chan et al.'s algorithm, the worst case node congestion is $\mathrm{O}(\Delta \log^2 N)$ hash values, where $\Delta$ is the number of neighbors of a node and $N$ is the total number of nodes in the network. Recently, Frikken et al. [38] proposed a modification to Chan et al.'s scheme that reduces the maximum communication per node to $\mathrm{O}(\Delta \log N)$. In Yang et al.'s algorithm, a node needs to forward $n_g$ MACs in the worst case, where $n_g$ is the number of groups formed in the the network. The worst case node congestion in Garofalakis et al.'s algorithm is $O(m \log S)$, where $m$ is the number of synopses used and $S$ is the value of the aggregate.

- Approximation Error: Our algorithm and Garofalakis et al.'s algorithm produce an approximate estimate of the aggregate, where the amount of error is reduced if the number of synopses used, $m$, is increased. On the other hand, Chan et al.'s and Yang et al.'s algorithms return the exact estimate if no aggregation message is lost before reaching BS.

- Robustness to Message Loss: Our algorithm and Garofalakis et al.'s algorithm are robust against message loss because they use multi-path routing schemes to forward nodes' authentication messages to BS. In contrast, Chan et al.'s algorithm is very sensitive to communication loss, and for the verification to succeed BS has to receive the authentication message from each and every node in the network. As nodes construct an aggregation tree, communication loss over any link may paralyze this algorithm. As a tree-based topology is used for message routing, Yang et al.'s algorithm is also not robust to communication loss which is a common phenomenon in a sensor network.

- Security: Theoretically, there is a chance that our algorithm may not detect the falsified sub-aggregate attack, but we can make that probability approximately 0 by properly choosing the value of $k$ (Claim 3.4.4). Furthermore, if the attacker does succeed to stealthily inject some '1's in a synopsis, we have a further level of defense.

In fact, while for ease of exposition we presented the protocol to compute just one synopsis, multiple synopses are computed in practice (ref. Section 3.4.3). The $R$ value of these synopses are highly correlated [10]. So, if the $R$ value of one synopsis appears to be an outlier compared to the others, that synopsis can be rejected.

Chan et al.'s algorithm and Garofalakis et al.'s algorithm deterministically detect the falsified sub-aggregate attack, while Yang et al.'s algorithm achieves probabilistic detection.

We also note that a compromised node might falsely increase its own sensed value, which would cause some error in the final estimate of the aggregate. While countering this type of attack is out of the scope of our verification protocol, we observe the following: If there are $t$ compromised nodes, the maximum injected error is upper bounded by $tv_m$, where $v_m$ is the upper bound of an individual sensed value. For Chan et al.'s algorithm and Garofalakis et al's algorithm, the same amount of error can be present while remaining undetected in the final estimate of the aggregate. In contrast, in Yang et al.'s algorithm, the amount of error injected by $t$ compromised nodes which falsify their own sensed value could be as high as $tgv_m$, where $g$ is the number of sensor nodes present in one group.

## 3.5 Computing Count and Sum Despite Attacks

The verification protocol presented above guarantees that the adversary cannot fool BS to accept a corrupted synopsis; however, this protocol cannot filter out the correct aggregate in the presence of the attack when a compromised node inserts false '1's and fabricated MACs corresponding to these false '1's. In this section, we propose an attack-resilient protocol which enables BS to compute the aggregate despite the presence of the attack. First, we give an overview of the protocol, then we discuss the protocol operation in detail, and finally we present the performance and security analysis.

### 3.5.1 Protocol Overview

We recall that if BS receives one valid MAC from a source node for each '1' bit in the final synopsis, it is able to correctly compute the aggregate. Before presenting our protocol, we describe a simpler protocol in which each node $X$ executes the previous verification protocol (ref. Section 3.4) with $k = \eta$, where $\eta$ is the length of the synopsis. That is, each node $X$ forwards one MAC for each of the '1' bits in $\hat{B^X}$. So, BS will verify all of the '1's in the received final synopsis $\hat{B}$. Let there be a compromised node $C$ which falsely injects a few '1's in its fused synopsis $\hat{B^C}$ and sends a false MAC for each of these false '1' bit. Then, with some probability, these false MACs may get selected at each hop before reaching BS.

If for a bit in final synopsis $B$, say bit $j$, BS does not receive a valid MAC but only false MACs, then BS cannot determine the real state of bit $j$. In fact, this can be the consequence of either of the following two scenarios: (i) $B[j] = 0$ and a false MAC has been generated; (ii) a source node has sent a valid MAC for bit $j$ ($B[j] = 1$, indeed), but this MAC lost the race to false MACs in the random selection procedure.

However, we observe that the probability of this 'undecidability' problem to arise is not the same for all of the bits. In fact, a false MAC is not equally likely to get selected for all of the bits because the number of source nodes that contribute to a bit (hence, the number of valid MACs) varies with the bit position. Property 4 of Sum synopsis (discussed in Section 3.1.2) says that the number of source nodes increases exponentially from the right to the left. As an example, approximately, $1/\phi$ =1.27 nodes are expected to contribute to bit $r$, $2/\phi$ =2.54 nodes are expected to contribute to bit $(r-1)$, and so on, where $r$ is the expected length of the prefix of consecutive '1's in the final synopsis $B$. So, if the number of compromised nodes, $t$, is small compared to the total number of nodes, $N$, we expect that BS will receive a valid MAC for the leftmost bits far from bit $r$, but may not receive a valid MAC for the other bits.

Considering the above observation, we design an attack-resilient protocol having two phases as follows:

- In phase one, we run the simple protocol described above. That is, each node $X$ forwards one randomly selected MAC for each '1' bit in $\hat{B^X}$. At the end of this phase, BS verifies the received MACs. The '1's in $\hat{B}$ for which no valid MACs have been received by BS are reset to '0'. Let $\bar{B}$ represent the final synopsis at BS after the above filtering process is performed. Analyzing $\bar{B}$, we make an estimate, $r'$ of the expected prefix length, $r$ of $B$.

- In phase two, nodes which contribute to bit $r'$ or to the bits to the right of bit $r'$ send a MAC to BS. In this phase, no random selection technique is employed in forwarding MACs—each node forwards all of the received MACs toward BS.

We will show later that, given a deviation of $r'$ from $r$ to the left, the number of MACs required in the second phase is exponential of this deviation. The main challenge is how to get a good estimate, $r'$ in the first phase. We will show that in the presence of $t$ compromised nodes, the deviation can be kept within $\mathrm{O}(\log_2 t)$. In this case, the number of MACs transmitted in the second phase will be $\mathrm{O}(t)$, i.e. proportional to the number of compromised nodes, $t$.

### 3.5.2 Protocol Operation

Our attack-resilient protocol exploits the fact that in the synopsis diffusion approach [7, 9], $m$ synopses are computed in parallel. Table 3.3 introduces a few additional notations which we use to describe this protocol. In the synopsis diffusion approach, multiple synopses $B_i$, $1 \leq i \leq m$ are computed to reduce the approximation error in the estimate of the aggregate, where a typical value of $m$ is 20.

We recall that the single synopsis computation algorithm is extended to the PCSA algorithm [10] as follows: In synopsis generation function $SG_{sum}$, one synopsis out of $m$ synopses is randomly chosen before each invocation of $CT()$ and only the chosen synopsis is updated after that particular invocation of $CT()$. So, if the value of Sum is $S$, we expect $S/m$ invocations of $CT()$ to update each synopsis $B_i$. Let $r$ be the expected value of $R_i$ in

the synopsis $B_i$. Then, $r$ is $\log_2(\frac{\phi S}{m})$ [10] (note that $r = E(R_1) = E(R_2) = \ldots = E(R_m)$). We also recall that the random variables $R_i$, $1 \leq i \leq m$, may differ for these $m$ synopses. Flajolet et al. [10] showed that $R_i$ values are highly correlated with one another, and with more than 95% probability, $R_i$ values are within $\pm 2$ of $r$.

In the query dissemination phase, BS broadcasts the name of the aggregate (i.e., 'Sum'), the chosen value of $m$, and another integer $m' \leq m$. With this, BS broadcasts a flag bit which indicates that the *attack-resilient computation protocol* is to be followed. As in the original synopsis diffusion algorithm, during the query distribution phase nodes arrange themselves into a ring topology around BS. Upon receiving the query message, the nodes aggregate their local synopses with their child nodes' synopses and send some authentication messages to BS in the following two phases.

**Phase One**

In phase one, nodes execute the aggregation algorithm of the original synopsis diffusion approach, involving $m$ synopsis, with additional transmission of some MACs. To reduce the communication overhead, we propose to collect MACs only for a subset of synopses, $B_1, B_2, \ldots, B_{m'}$ in this phase. From the query message broadcast from BS, each node becomes aware of which subset of synopses to be considered. Our analysis and simulation results will show that $m' = 4$ synopses are sufficient to get a preliminary estimate of $r$.

Each node $X$ randomly selects one MAC for each '1' bit in synopses $\hat{B_1^X}, \hat{B_2^X}, \ldots, \hat{B_{m'}^X}$ from the MACs received from its child nodes (possibly including $X$'s own MAC). $X$ forwards the selected MACs to its parents. The message broadcast by $X$ to its parent nodes is as follows:

$$X \rightarrow * : \langle\ \hat{B_1^X}, \hat{B_2^X}, \ldots, \hat{B_m^X},\ \{M_{i,j} \mid \hat{B_i^X}[j] = 1, 1 \leq i \leq m', 1 \leq j \leq \eta\}\ \rangle,$$

where $\hat{B_i^X}$ represents the $i$-th fused synopsis at node $X$, $M_{i,j}$ represents a MAC corresponding to $\hat{B_i^X}[j]$, and $\eta$ is the length of each synopsis.

Table 3.3: Notations Used in Describing the Attack-resilient Computation Protocol

| Symbol | Meaning |
|---|---|
| $\eta$ | the length of each synopsis |
| $m$ | the number of the synopses computed parallelly using PCSA |
| $B_i^X$ | the $i$-th fused synopsis of node $X$ if no attack is launched |
| $\hat{B_i^X}$ | the $i$-th fused synopsis computed by node $X$ |
| $B_i$ | the $i$-th final synopsis if no attack is launched |
| $B_{i,j}$ | the $j$-th bit in $B_i$ |
| $B_i[j]$ | the value of the $j$-th bit in $B_i$ |
| $R_i$ | the length of the prefix of all '1's in $B_i$ |
| $r$ | the expected value of $R_i$ |
| $r'$ | BS's estimate of $r$ after phase one |
| $\hat{B}_i$ | the $i$-th final synopsis received by $BS$ in phase one |
| $\hat{R}_i$ | the length of the prefix of all '1's in $\hat{B}_i$ |
| $\bar{B}_i$ | the $i$-th final synopsis at $BS$ after false MACs, if any are filtered in phase one |
| $\bar{R}_i$ | the length of the prefix of all '1's in $\bar{B}_i$ |
| $B_i'$ | the $i$-th final synopsis at $BS$ after phase two terminates |
| $R_i'$ | the length of the prefix of all '1's in $B_i'$ |

We require a restriction on the number of MACs that a node can forward. In fact, if node $X$ sends an aggregation message (synopses $\hat{B_i^X}$ and corresponding MACs) to its parent node $Y$, $Y$ does not accept more than one MAC for each '1' bit in $B_i^X$. This assumption

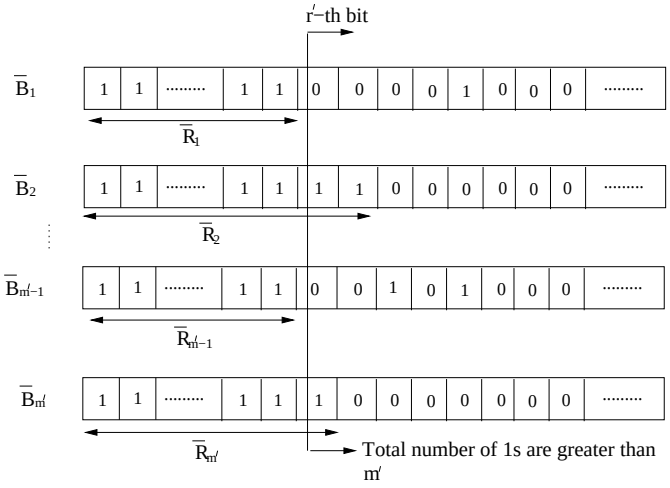



Figure 3.3: Getting a Preliminary Estimate of $r$—BS scans synopses $B_1, B_2, \ldots, B'_m$ from the right and reports the bit position $j$ as the estimate $r'$ at which total number of '1's already scanned goes above $m'$.

can be enforced by employing authentication techniques in the communication procedure among neighboring nodes.

After all of the MACs have been received by BS, for any '1' bit, say bit $\hat{B_{i,j}}$, in the synopses $\hat{B_1}, \hat{B_2}, \ldots, \hat{B_{m'}}$ for which no valid MAC has been received, BS resets $\hat{B_{i,j}}$ to '0'. The resulting set of synopses after this filtering process has been performed are denoted by $\bar{B_1}, \bar{B_2}, \ldots, \bar{B_{m'}}$, respectively. Now, BS makes an estimate of the expected length of prefix of all '1's, $r$ using $\bar{B_1}, \bar{B_2}, \ldots, \bar{B_{m'}}$. First, we recall from Section 3.1 that approximately one '1' bit is expected, on average, to appear to the right of bit $r$. This observation is exploited to design the algorithm used by BS to estimate $r$ (Algorithm 4). This algorithm outputs $j$ as the estimate of $r$, where $j$ is the rightmost position for which the count of all of the '1's to the right of the $j$-th bit in the $m'$ synopses is more than $m'$. We denote this estimate of $r$ as $r'$. Figure 3.3 illustrates Algorithm 4 with an example.

During phase one, along with the MACs for the $m'$ synopses, each node also forwards all of the $m$ synopses following the original synopsis diffusion protocol. However, the rest of the synopses, $(m - m')$ synopses, for which no MACs are computed in phase one are not

**Algorithm 4** Estimate-Expected-Prefix-Length $(\bar{B_1}, \bar{B_2}, \ldots, \bar{B_{m'}})$

```
int j = η;   // η is the length of each synopsis
int α = 0;   // α is the total number of '1's to the right of B̄_{i,j}, 1 ≤ i ≤ m'
while α ≤ m' and j > 1 do
   j = j − 1;
   α = α + B̄_1[j] + B̄_2[j] + ... + B̄_{m'}[j];
end while
return j;
```

validated until phase two.

We observe that there are two factors which could possibly deviate the estimate $r'$, which is derived only using $m'$ synopses, from $r$: (i) a pseudo random hash function is used to generate the synopses; (ii) injection of false MACs by the adversary—which can cause BS not receiving any valid MAC for a few '1' bits near bit $r$ in synopses $B_i$, $1 \leq i \leq m'$. We observe that the first factor could cause the estimate to deviate toward either side; the second factor could contribute to a deviation to the left only, as shown later. In Claim 3.5.7, we study the deviation of $r'$ from $r$, considering $t$ compromised nodes: The deviation on the right side is independent of $t$, while the deviation on the left side is at most $\log_2 \phi t + 1$, with high probability.

### Phase Two

In phase two, BS requests the nodes which contribute to bits $j$, $j \geq r'$, in all of the $m$ synopses to send back the corresponding MACs. The message sent by BS is as follows:

$$BS \rightarrow\rightarrow * : \langle \text{“}PhaseTwo\text{”}, r' \rangle,$$

where “$PhaseTwo$” is a flag indicating that phase two is going to begin.

After receiving the request from BS, each node $X$ broadcasts to its parents the MACs, $\{M_{i,j} \mid 1 \leq i \leq m, r' \leq j \leq \eta\}$. Unlike the first phase, now no MAC is dropped by the intermediate nodes, i.e, each node $X$ forwards to $X$'s parents all of the MACs $X$ received from its child nodes.

After BS receives the MACs, any bit $B_{i,j}$, $1 \leq i \leq m'$, $j \geq r'$ for which a valid MAC is received is set to '1'. Also, any bit $\hat{B}_{i,j}$, $m' + 1 \leq i \leq m$, $j \geq r'$ for which no valid MAC is received is reset to '0'. The resulting synopses are denoted by $\bar{\bar{B}}_i$, $1 \leq i \leq m$, respectively. Then, depending on whether bit $\bar{\bar{B}}_i[r'] = 1$ for all the synopses $i$ (*termination criteria*), BS proceeds as follows:

- If it is so, BS assumes that all of the bits to the left of $r'$ are also '1' in all synopses, i.e. assumes that the position of bit $r$ is to the right of bit $r'$. We remind the reader that all of the bits $j$, $j \geq r'$ are now verified, indeed. So, BS knows the values of all of the bits of every synopsis. Eventually, to get the final estimate of the aggregate, BS applies on the resulting synopses the synopsis evaluation function, $SE()$ ( ref. Section 3.1.2).

- Otherwise, if $\bar{\bar{B}}_i[r'] = 0$ for some $i$, $r$ might be smaller than $r'$. That is, BS needs to further verify a few more bits to the left of bit $r'$. BS does this in an iterative way using a *sliding window* of $w$ bits. In each iteration, BS broadcasts a request for the MACs of the next $w$ bits to the left of the leftmost bit already verified (i.e., $\bar{\bar{B}}_i[j]$, $1 \leq i \leq m$, $r' - w \leq j < r'$, in the first iteration). Nodes reply with MACs of the requested bits.

### 3.5.3 Correctness

In this section, we prove the correctness of our protocol. In particular, we prove that phase two in Section 3.5.2 always terminates. Also, we show that when phase two terminates, i.e., when the sliding window satisfies the termination criterion, with very high probability BS can correctly infer the values of all of the bits in every synopsis.

First, we present Lemma 3.5.1 and Lemma 3.5.2, which we will use to obtain the subsequent results. Lemma 3.5.1 and Lemma 3.5.2 are the counterparts of Lemma 3.4.2 and Lemma 3.4.3, respectively, when $m$ synopses are computed instead of a single synopsis.

**Lemma 3.5.1.** *Let $m$ synopses be computed using the PCSA algorithm [10], where $S$ is the value of the Sum. Let $r$ be the expected value of $R_i$ in the $i$-th synopsis $B_i$ at BS. The probability that $B_i[j] = 0$, with $r - a \leq i \leq r + a$ and $a \geq 0$, is:*

$$Pr[B_i[j] = 0] \approx e^{-\frac{(1/\phi)}{(2^{i-r})}},$$

*where $\phi = 0.7735$.*

*Proof.* We recall that for PCSA, in the synopsis generation function $SG_{sum}$, one synopsis out of $m$ synopses is randomly chosen before each invocation of $CT()$ and only the chosen synopsis is updated after that particular invocation of $CT()$. So, for each invocation of $CT()$, the probability that the bit $B_{i,j}$ will be set to '1' is $\frac{1}{m2^j}$. As a result, after all of the $S$ invocations of $CT()$, we have that:

$$Pr[B_i[j] = 0] = (1 - \frac{1}{m2^j})^S.$$

To calculate the probability of $r$-th bit being '0', we substitute $j$ with $r = \log_2(\frac{\phi S}{m})$ and we get:

$$Pr[B[r] = 0] = (1 - \frac{1}{m2^r})^S = (1 - \frac{1}{\phi S})^S \approx e^{-\frac{1}{\phi}}.$$

In general, for the $i$-th bit where $r - a \leq i \leq r + a, a \geq 0$, we get:

$$Pr[B[i] = 0] = (1 - \frac{1}{m2^i})^S = (1 - \frac{1}{m2^{i-r}2^r})^S = (1 - \frac{1}{2^{i-r}\phi S})^S \approx e^{-\frac{(1/\phi)}{(2^{i-r})}}.$$

□

We observe from Lemma 3.5.1: The probability that $B[j] = 0$ is determined by only the distance of the $j$-th bit from the $r$-th bit, where the value of $r$ is $\log_2(\frac{\phi S}{m})$. Furthermore, in Lemma 3.5.2 below, we observe that the bits in all of the synopses close to bit $r$ (to the

left or to the right of it) or far to the right of bit $r$ can be considered as independent.

**Lemma 3.5.2.** *Let $m$ synopses be computed using the PCSA algorithm [10], where $S$ is the value of the Sum. Let $r$ be the expected value of $R_i$ in the $i$-th synopsis $B_i$ at BS. The value ('0' or '1') of any two bits $B_{i,j}$ and $B_{i',j'}$, with $1 \leq i \leq m$, $1 \leq i' \leq m$, $j \geq (r-b)$, $j' \geq (r-b)$, $(i,j) \neq (i',j')$, $b << log_2 S$ are independent.*

*Proof.* By the construction of Algorithm 2 extended with PCSA, we get the following two relations:

$$Pr[B_i[j] = 0] = (1 - \frac{1}{m2^j})^S, \text{ for } 1 \leq i \leq m, 1 \leq j \leq \eta \tag{3.12}$$

$$Pr[B_i[j] = 0, B_{i'}[j'] = 0] = (1 - \frac{1}{m2^j} - \frac{1}{m2^{j'}})^S,$$

$$\text{for } 1 \leq i \leq m, 1 \leq j \leq \eta, 1 \leq i' \leq m, 1 \leq j' \leq \eta, (i,j) \neq (i',j') \tag{3.13}$$

So, by basic rules of probability we have:

$$Pr[B_i[j] = 0 \mid B_{i'}[j'] = 0] = \frac{Pr[B_i[j] = 0, B_{i'}[j'] = 0]}{Pr[B_{i'}[j'] = 0]} = \frac{(1 - \frac{1}{m2^j} - \frac{1}{m2^{j'}})^S}{(1 - \frac{1}{m2^{j'}})^S} \tag{3.14}$$

From Relation 3.12 and Relation 3.14, we get:

$$\frac{Pr[B_i[j] = 0 | B_{i'}[j'] = 0]}{Pr[B_i[j] = 0]} = \frac{(1 - \frac{1}{m2^j} - \frac{1}{m2^{j'}})^S}{(1 - \frac{1}{m2^{j'}})^S (1 - \frac{1}{m2^j})^S} = (1 - \frac{1}{(m2^j - 1)(m2^{j'} - 1)})^S \tag{3.15}$$

So, we have:

$$\frac{Pr[B_i[j] = 0 | B_{i'}[j'] = 0]}{Pr[B_i[j] = 0]} \approx (1 - \frac{1}{m^2 2^j 2^{j'}})^S \approx e^{-\frac{S}{m^2 2^j 2^{j'}}} \leq e^{-\frac{S}{(\frac{\phi S}{2^b})(\frac{\phi S}{2^b})}} = e^{-\frac{2^{2b}}{\phi^2 S}} \approx 1 \tag{3.16}$$

□

Using the above Lemmas we now show that phase two ends. This requires us to prove that we can get a bit index $u$ at which the termination criterion of phase two is satisfied. We do it using the following claim.

**Claim 3.5.3.** *For any $m$ there exists a bit index $u$ such that*

$$Pr[B_i[u] = 1 \ \forall i, 1 \leq i \leq m] \approx 1.$$

*Proof.* Using Lemma 3.5.1 and Lemma 3.5.2 we get that the probability of bit $B_i[j] = 1$ in all of the $m$ synopses is:

$$\begin{aligned} p_{j,m} &\approx (1 - e^{-\frac{(1/\phi)}{(2^{j-r})}})^m \\ &= (1 - e^{-\frac{2^{r-j}}{\phi}})^m \end{aligned} \tag{3.17}$$

From Relation 3.17, for $j < r$ we get that $p_{j,m} \approx (1 - me^{-\frac{2^{r-j}}{\phi}})$. We observe that $p_{j,m}$ increases if $j$ is decreased. For any $m$, we can find a $u$ such that $p_{u,m} \approx 1$.

□

We further observe that for $m = 20$, $Pr[B_i[r-2] = 1 \forall i, 1 \leq i \leq m] = 0.9$, and $Pr[B_i[r-3] = 1 \forall i, 1 \leq i \leq m] \approx 1$. That means if 20 synopses are used, phase two will stop at bit $r-2$ in 90% of the cases and at bit $r-3$ in the rest of them.

Also, we remind the reader that if the termination criterion of phase two is satisfied at bit $u$, BS already verified all of the bits $B_i[j]$, $j \geq u$ for every synopsis. For the bits $B_i[j]$, $j < u$, BS exploits one of our assumptions, which is as follows: If bit $B_i[u] = 1$ for all $i$, then $B_i[j] = 1$ for all $i$ and all $j < u$. We now show that this assumption holds with high probability. We introduce $\mathcal{E}^m$, which denotes the following event: For some bit $j$, $1 \leq j \leq \eta$

at which $B_i[j] = 1$ for all $i$, $1 \leq i \leq m$, a '0' bit appears in one or more synopses to the left of the $j$-th bit. We now show that $Pr[\mathcal{E}^m]$ is very small for a practical value of $m$. Recall that the typical value of $m$ is 20 [7, 9].

**Claim 3.5.4.** *$Pr[\mathcal{E}^m] < 0.02$, for $m \geq 20$.*

*Proof.* The main idea behind this proof is as follows. Lemma 3.5.1 shows that the probability of a bit in the synopsis being '0' (or '1') depends on its closeness to the $r$-th bit: It rapidly decreases (increases) for the bits to the left of $r$-th bit and rapidly increases (decreases) for the bits to the right of the $r$-th bit. That means it is very unlikely to find a '0' far to the left of bit $r$ and a '1' far to the right of bit $r$. However, for event $\mathcal{E}^m$ to occur, one '0' has to occur somewhere, say bit $j$, and simultaneously $B_i[j'] = 1$ for one $j' > j$ and for all $i$, $1 \leq i \leq m$. Intuitively, the most likely 'place' where the bit pattern associated to event $\mathcal{E}^m$ may occur is close to $r$.

Let $\mathcal{E}^m_{j,j'}$, $j < j'$ denote the following event: $B_i[j'] = 1$ for all $i$, $1 \leq i \leq m$ and '0' bit appears in one or more synopses at the $j$-th bit.

Using Lemma 3.5.1 and Lemma 3.5.2, we get that the probability that bit $B_i[j] = 1$ in all of the synopses is:

$$p_{j,m} \approx (1 - e^{-\frac{(1/\phi)}{(2^{j-r})}})^m, \tag{3.18}$$

and the probability that bit $B_i[j] = 0$ in one or more synopses is:

$$q_j \approx 1 - (1 - e^{-\frac{(1/\phi)}{(2^{j-r})}})^m. \tag{3.19}$$

As a result, we get the following:

$$Pr[\mathcal{E}^m_{j,j'}] \approx (1 - (1 - e^{-\frac{(1/\phi)}{(2^{j-r})}})^m) \times (1 - e^{-\frac{(1/\phi)}{(2^{j'-r})}})^m \tag{3.20}$$

From Relation 3.20, for $m \geq 20$ we observe that $Pr[\mathcal{E}^m_{j,j'}] \approx 0$ for all combinations of $j$

and $j'$ except $j = r-2$ and $j' = r-1$. So, we have:

$$Pr[\mathcal{E}^m] \approx Pr[\mathcal{E}^m_{r-2,r-1}] \approx me^{-\frac{4}{\phi}}(1-e^{-\frac{2}{\phi}})^m \tag{3.21}$$

Taking $m = 20$, we get $Pr[\mathcal{E}^m] = 0.02$. From Relation 3.21, we also observe that for higher values of $m$, the value of $Pr[\mathcal{E}^m]$ decreases.

□

We recall that if BS's final estimate of $R_i$ in $m$ synopses are $R_1', R_2', \ldots, R_m'$, BS uses $R' = (R_1' + R_2' + \ldots + R_m')/m$ to evaluate the final Sum as $2^{R'}/\phi$. From Claim 3.5.4, we observe the following for $m = 20$: If the termination criterion of phase two is satisfied at bit $u$, then our assumption that all of the bits $B_i[j]$, $1 \leq i \leq 20$, $j < u$ are '1' holds in 98% of the cases. In these cases, BS gets the 'true' Sum. In the other 2% of the cases, BS overestimates some $R_i$ by 3 bits on average ($R_i = r-3$ is overestimated as $r$). Hence, BS overestimates the final Sum. We observe that for $j < u$ the probability that $B_i[j] = 0$ for more than one synopses is negligible and hence the error in the final Sum is $(2^{3/20} - 1) \times 100\% \approx 10\%$. We also observe that for higher values of $m$, this error is reduced.

### 3.5.4 Performance Analysis

In this section, we evaluate our attack-resilient computation protocol in terms of communication overhead and latency.

The communication overhead of phase one does not depend on the number of compromised nodes. The worst case per-node communication burden is to forward $m'l$ MACs, where $l$ is the maximum number of '1's in any synopsis. From property 1 of Sum synopsis (Section 3.1.2), we know that $l$ is approximately $\log_2 S$, $S$ being the Sum. That means the communication overhead per node is $O(m' \log_2 S)$. Also, phase one takes one epoch to complete.

On the other hand, the communication overhead and latency of phase two are determined

by how close the estimate $r'$, obtained in phase one, is to the real value of $r$. In the following discussion, first we assume no attack and we examine the possible deviation in this estimate $r'$ in Claim 3.5.5 and Claim 3.5.6. In particular, Claim 3.5.5 determines the deviation of $r'$ to the right of $r$, while Claim 3.5.6 determines the deviation of $r'$ to the left of $r$. Lemma 3.5.1 and Lemma 3.5.2 serve as the basis for these two Claims. Furthermore, we remind the reader that as observed in Section 3.5.2 a practical attack can only move the estimate $r'$ to the left. So, Claim 3.5.5 also describes the deviation to the right in case of attack. Instead, we present Claim 3.5.7 to describe the deviation to the left in case of attack.

Now, by Lemma 3.5.1 we know that $Pr\left[B_i\left[j\right]=1\right]$ is the same for each synopsis $B_i$ and $Pr\left[B_i\left[j\right]=1\right] \approx 1-e^{-\frac{(1/\phi)}{(2^{j-r})}}$. Let $p_j$ denote this probability. Then, thanks to Lemma 3.5.2, for any particular bit $j$, the random variable $Z_j = \sum_{i=1}^{m'} B_i[j]$ follows a binomial distribution: $Z_j \sim \mathbf{B}(m', p_j)$.

Let $Y_j$ denote the total number of '1's which appear to the right of bit $j$ in all of the synopses $B_1, B_2, \ldots, B_{m'}$, i.e., $Y_j = Z_{j+1} + Z_{j+2} + \ldots + Z_\eta$, where $\eta$ is the length of each synopsis. We also observe that thanks to Lemma 3.5.2, $Z_{j+j'}$, $1 \leq j' \leq \eta - j$ variables are independent for $j \geq (r-b)$, where $b << \log_2 S$.

**Claim 3.5.5.** *Let there be no false MAC injection attack in the network and $r'$ be the output of Algorithm 4, which is an estimate of the position of bit $r$. Let $G_j$ represent the event that $r' > r + j$, for $j > 0$. We have that: (i) $Pr[G_j]$ does not depend on $r$; (ii) $Pr[G_j]$ decreases while $j$ increases; (iii) We can find a $\gamma > 0$ such that $Pr[G_\gamma] \approx 0$.*

*Proof.* If event $G_j$ occurs, it follows from the construction of Algorithm 4 that $Y_{r+j}$ is at least $m' + 1$. So, the probability of the event $G_j$ is as follows:

$$Pr[G_j] = Pr[Y_{r+j} \geq m' + 1] = Pr[(Z_{r+j+1} + Z_{r+j+2} + \ldots + Z_\eta) \geq (m' + 1)] \qquad (3.22)$$

Note that $p_{r+j+j'} \approx 1 - e^{-\frac{(1/\phi)}{(2^{r+j+j'-r})}} = 1 - e^{-\frac{(1/\phi)}{(2^{j+j'})}}$. So, the probability distribution of a

general random variable, $Z_{r+j+j'}$ in Expression 3.22 follows $\mathbf{B}(m', 1 - e^{-\frac{(1/\phi)}{(2^{j+j'})}})$. That is, the probability distribution of $Z_{r+j+j'}$ is independent from $r$. So, also the probability for the event $G_j$ to occur is independent from $r$, that is the point (i) of the claim.

From the definition of $G_j$, it follows that if $j_1 < j_2$, then $Pr[G_{j_1}] > Pr[G_{j_2}]$, that is point (ii) of the claim.

The above expression of $p_{r+j+j'}$ can be approximated as follows:

$$p_{r+j+j'} \approx 1 - e^{-\frac{(1/\phi)}{(2^{j+j'})}} \approx 1 - (1 - \frac{1/\phi}{2^{j+j'}}) = \frac{1/\phi}{2^{j+j'}} \tag{3.23}$$

We observe that the probability $p_{r+j+j'}$ becomes one half if $(j + j')$ increases by one (i.e., if we move one bit to the right), and eventually becomes negligible for high value of $j$. Also, from the property of binomial distribution we get that $E(Z_{r+j+j'}) = m' \cdot p_{r+j+j'}$, and $Var(Z_{r+j+j'}) = m' \cdot p_{r+j+j'} \cdot (1 - p_{r+j+j'})$.

From the definition of $Y_j$, we get the expected value and variance of $Y_{r+j}$ are as follows:

$$E(Y_{r+j}) = E(Z_{r+j+1}) + E(Z_{r+j+2}) + \ldots + E(Z_\eta) \approx 2 \cdot E(Z_{r+j+1}) \tag{3.24}$$

$$Var(Y_{r+j}) = Var(Z_{r+j+1}) + Var(Z_{r+j+2}) + \ldots + Var(Z_\eta) \approx 2 \cdot Var(Z_{r+j+1}) \tag{3.25}$$

We observe that both $E(Y_{r+j})$ and $Var(Y_{r+j})$ decrease while $j$ increases and eventually become 0 for the high value of $j$. So, from Chebyshev's theorem, we get that there is a $\gamma$ for which $Pr[Y_{r+\gamma} > m' + 1] \leq \epsilon$ for any $\epsilon > 0$. That is, point (iii) of the claim is proved.

□

Further, we observe that for $m' \geq 4$ and $j \geq 2$, $Pr[G_j] \approx 0$. As an example, let us compute the probability of $G_j$ for $m' = 4$ and $j = 2$. It is easier to compute the probability of $G_2$, if we first compute the probability of the complementary events. We compute that $\Pr[Z_{r+3} + Z_{r+4} + Z_{r+5} + Z_{r+6} + \ldots = 0] \approx 0.2977$, $\Pr[Z_{r+3} + Z_{r+4} + Z_{r+5} + Z_{r+6} + \ldots = 1] \approx$

0.3823, $\Pr[Z_{r+3}+Z_{r+4}+Z_{r+5}+Z_{r+6}+\ldots=2] \approx 0.2218$, $\Pr[Z_{r+3}+Z_{r+4}+Z_{r+5}+Z_{r+6}+\ldots=3] \approx 0.0770$, $\Pr[Z_{r+3}+Z_{r+4}+Z_{r+5}+Z_{r+6}+\ldots=4] \approx 0.0178$. From the above probabilities, finally we get the probability of event $G_2$ is 0.0034.

As the false MAC injection attack could only cause $r'$ to deviate to the left, we do not need to consider this attack in the estimation of the upper bound on the deviation of $r'$ to the right of $r$. The upper bound, $\gamma$ computed assuming no false MAC injection attack is still an upper bound under the presence of the attack.

**Claim 3.5.6.** *Let there be no false MAC injection attack in the network and $r'$ be the output of Algorithm 4, which is an estimate of the position of bit $r$. Let $F_j$ represent the event that $r' < r - j$, for $j > 0$. We have that: (i) $Pr[F_j]$ does not depend on $r$. (ii) We can find a $\delta > 0$ such that $Pr[F_\delta] \approx 0$*

*Proof.* Let $p_j = Pr[B_i[j] = 1]$, where by Lemma 3.5.1 we know $p_j$ being the same for each synopsis, $p_j \approx 1 - e^{-\frac{(2^{r-j})}{\phi}}$. If event $F_j$ occurs, it follows from the construction of Algorithm 4 that $Y_{r-j}$ is at most $m'$. So, the probability of the event $F_j$ is as follows:

$$Pr[F_j] = Pr[Y_{r-j} \leq m'] = Pr[(Z_{r-j+1} + Z_{r-j+2} + \ldots + Z_\eta) \leq m']. \tag{3.26}$$

Note that $p_{r-j+j'} \approx 1 - e^{-\frac{(2^{r-r+j-j'})}{\phi}} = 1 - e^{-\frac{(2^{j-j'})}{\phi}}$. So, the probability distribution of a general random variable, $Z_{r-j+j'}$ in Expression 3.26 follows $\mathbf{B}(m', 1 - e^{-\frac{(2^{j-j'})}{\phi}})$. That is, the probability distribution of $Z_{r-j+j'}$ is independent from $r$. So, also the probability for the event $F_j$ to occur is independent from $r$, that is the point (i) of the claim.

From the definition of $Y_j$, it follows that for $j > 2$,

$$Pr[Y_{r-j} \leq m'] < Pr[(Z_{r-j+1} + Z_{r-j+2} + \ldots + Z_{r-2}) \leq m']. \tag{3.27}$$

From the above expression of $p_{r-j+j'}$, we get that the probability $p_{r-j+j'} \approx 1$ for $j - j' > 2$.

So, for $j - j' > 2$, we get that $E(Z_{r-j+j'}) = m' \cdot p_{r-j+j'} \approx m'$, and $Var(Z_{r-j+j'}) = m' \cdot p_{r-j+j'} \cdot (1 - p_{r-j+j'}) \approx 0$. Also, we have

$$E(Z_{r-j+1} + Z_{r-j+2} + \ldots + Z_{r-2}) = E(Z_{r-j+1}) + E(Z_{r-j+2}) + \ldots + E(Z_{r-2}) \approx m' \cdot (j-2), \tag{3.28}$$

and

$$Var(Z_{r-j+1} + Z_{r-j+2} + \ldots + Z_{r-2}) = Var(Z_{r-j+1}) + Var(Z_{r-j+2}) + \ldots + Var(Z_{r-2}) \approx 0. \tag{3.29}$$

We observe that with the increment in $j$, the difference between $E(Z_{r-j+1} + Z_{r-j+2} + \ldots + Z_{r-2})$ and $m'$ increases. So, from Chebyshev's theorem, we get that there is a $\delta$ for which $Pr[Y_{r-\delta} \leq m'] \leq \epsilon$ for any $\epsilon > 0$. That is, point (ii) of the claim is proved.

□

**Claim 3.5.7.** *Let $r'$ be the output of Algorithm 4, which is an estimate of the position of bit $r$, and let $t$ denote the number of compromised nodes in the network. There is a $\delta_a$ such that $Pr[r' < (r - \delta_a)] \approx 0$.*

*Proof.* We now consider the attack in which compromised nodes inject false MACs in phase one. Recall that we enforce a restriction on the number of MACs that a node can inject into the network. In fact, if node $X$ sends an aggregation message (synopses $\hat{B_i^X}$ and corresponding MACs) to node $Y$, $Y$ does not accept more than one MAC for each '1' bit in $B_i^X$.

Without loss of generality, we focus on one particular '1' bit, say $B_{i,j}$ i.e., the $j$-th bit in synopsis $B_i$. Let us assume that there are $s$ nodes in the network which contribute to this bit and hence each of these $s$ nodes sends a MAC toward BS. On the other hand, if there are $t$ compromised nodes present in the network, then there can be at most $t$ injected false MACs for this bit. We can consider that these $s$ valid MACs and $t$ false MACs compete with one another in the random selection procedure at the intermediate hops, and finally, only

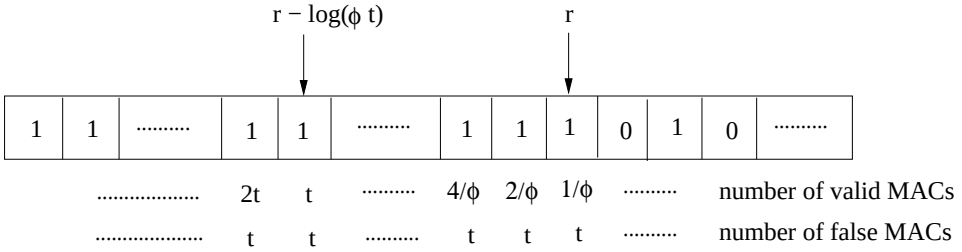



Figure 3.4: A High-level View of the Random Selection Procedure in MACs Forwarding—The number of contributing nodes and hence the number of valid MACs varies with the bit position, while the number of false MACs is at most $t$ for any '1' bit in the presence of $t$ compromised nodes.

one MAC reaches BS. We assume that these $s$ good nodes and $t$ bad nodes are randomly distributed in the network. So, we can consider that a valid MAC finally reaches BS with probability $s/(s+t)$.

Recall that the number of contributing nodes varies exponentially with the bit position—for the $r$-th bit, the expected number of contributing nodes is $1/\phi$, for the $(r-1)$-th bit $2/\phi$, and so on. Let $r_a = r - \log(\phi t)$. For bit $r_a - j$, the expected number of contributing nodes is $2^j t$ (as illustrated in Figure 3.4 ). Hence, the probability that a valid MAC reaches BS for this bit is $p_{r_a-j} = (2^j t)/(2^j t + t) = (2^j)/(2^j + 1)$. We observe that this probability is approximately 1 while $j > 4$.

Let $Z_j = \sum_{i=1}^{m'} B_i[j]$ and $Y_j = Z_{j+1} + Z_{j+2} + \ldots + Z_\eta$, where $\eta$ is the length of each synopsis. From the definition of $Y_j$ it follows that for $j > 4$,

$$Pr[Y_{r_a-j} \leq m'] < Pr[(Z_{r_a-j+1} + Z_{r_a-j+2} + \ldots + Z_{r_a-4}) \leq m']. \tag{3.30}$$

For $j > 4$, $j' \geq 1$ and $j - j' \geq 4$, we also have that

$$E(Z_{r_a-j+j'}) = m' \cdot p_{r_a-j+j'} \approx m', \text{ and}$$

$$Var(Z_{r_a-j+j'}) = m' \cdot p_{r_a-j+j'} \cdot (1 - p_{r_a-j+j'}) \approx 0.$$

So, we get that

$$E(Z_{r_a-j+1} + \ldots + Z_{r_a-4}) = E(Z_{r_a-j+1}) + \ldots + E(Z_{r_a-4}) \approx m' \cdot (j-4), \tag{3.31}$$

and

$$Var(Z_{r_a-j+1} + \ldots + Z_{r_a-4}) = Var(Z_{r_a-j+1}) + \ldots + Var(Z_{r_a-4}) \approx 0. \tag{3.32}$$

We observe that with the increment in $j$, the difference between $E(Z_{r_a-j+1} + \ldots + Z_{r_a-4})$ and $m'$ increases. So, from Chebyshev's theorem, we get that there is a $\delta'$ for which

$$Pr[Y_{r_a-\delta'} \leq m'] \approx 0. \tag{3.33}$$

That means, there is a $\delta_a = \log(\phi t) + \delta'$ for which

$$Pr[Y_{r-\delta_a} \leq m'] \approx 0. \tag{3.34}$$

□

As a special case, we observe that for $\delta_a = \log \phi t + 1$, the value of the probability in Expression 3.34 is less than 0.01. That is, even in the presence of the attack the estimate $r'$ of phase one is no less than $r - \log \phi t + 1$ with probability more than 0.99.

To make the performance analysis of phase two, we discuss the two opposite situations that can occur in phase one: (i) $r'$ is to the left of $r$; (ii) $r'$ is to the right of $r$. In case (i), we observed that with high probability (more than 0.99), the leftmost position is $r - log_2 \phi t - 1$. Recall that the termination criterion of phase two is likely to be satisfied for any bit $j \leq r-2$ (ref. Section 3.5.3). It implies that if $log_2 \phi t \geq 1$, phase two is likely to complete within one epoch and by Property 4 of Sum synopsis (ref. Section 3.1.2) at most $4t$ MACs are expected to be transmitted for each synopsis. If $log_2 \phi t < 1$, with the sliding window width $w$, phase two is likely to complete within $2/w$ epochs and at most $8/\phi = 10.34$ MACs are

expected to be transmitted for each synopsis.

In case (ii), we observed that with high probability (more than 0.99) the rightmost position of $r'$ is $r + 2$. In that case, BS will find that bit $r'$ is '0' in some of the synopses. This means that phase two will not end but will require more iterations. First, we note that in the presence of an attack in phase one, this case is very unlikely to occur. However, if this case does occur, BS will iterate phase two for another round during which MACs for $w$ (width of the sliding window) more bits to the left of bit $r'$ are collected. In the worst case, we will need $(r' - r + 2)/w$ more rounds. The average number of MACs transmitted per synopsis will be $8/\phi = 10.34$. As a special case, we observe that if $w = 2$, then with high probability phase two completes within 3 epochs.

### 3.5.5 Security Analysis

As each node randomly selects the MACs to forward in phase one of this protocol, there is a possibility that no valid MAC reaches BS corresponding to a few bits near bit $r$ in the synopses. So, there exists some room for the adversary to shift the estimate, $r'$ from the true value of $r$. However, phase two examines this deviation by further validating the bits near bit $r'$ to securely obtain the position of bit $r$. In phase two, nodes do not employ a random selection procedure in forwarding MACs; each node forwards to its parent nodes all of the received MACs. We assume that due to the use of multi-path routing, at least one MAC for each bit will reach BS.

Below we observe that the above assumption holds in a practical network. Let $\alpha$ be the packet loss rate and each node have at least $f$ parents in the aggregation hierarchy. Then, the probability of node $X$'s MAC to reach BS is greater than $(1 - \alpha^f)^g$, where node $X$ is $g$ hops away from BS. As an example, if $\alpha = 0.1$, $f = 3$, and $g = 5$, then this probability is more than 99.5%. Furthermore, a compromised node $C$'s dropping a MAC generated by node $X$ which contributes a '1' to bit $B_{i,j}$ has no effect if there is another node $Y$ which also contributes a '1' to bit $B_{i,j}$ and hence sends its own MAC. To stop all of the MACs corresponding to bit $B_{i,j}$ from reaching BS, the attacker has to compromise all of the

possible paths from all of these contributing nodes to BS, which is hard to achieve. We note that there are bits in the synopses to which only one or two nodes contribute. However, it is very hard for the attacker to predict in advance which nodes will be contributing to these particular bits. As a result, our protocol is secure against the falsified sub-aggregate attack.

A compromised node can inject errors in the final estimate of the aggregate only by falsely increasing its own sensed value. If there are $t$ compromised nodes, then the injected error is at most $tv_m$, where $v_m$ is the upper bound of an individual sensed value.

### 3.5.6 A Variant Protocol

We observed that the worst case communication overhead of the attack-resilient computation protocol described above is proportional to the number of compromised nodes, $t$. To keep the overhead within reasonable limits even when there are more compromised nodes, we design a variant protocol which differs from the basic attack-resilient computation protocol only in phase two.

Phase two in the variant protocol trades off communication overhead at the cost of latency. In this protocol, phase two may take multiple epochs depending on the deviation in the estimate $r'$ from $r$. Claim 3.5.7 says that the left bound deviation is likely to be less than $\log_2(\phi t) + \delta' = \Delta$ bits, where $\delta'$ is a small constant. In the basic attack-resilient computation protocol, BS verifies the $r'$-th bit and all of the bits to the right of the $r'$-th bit in one epoch. In contrast, this protocol verifies those bits using a sliding window of appropriate width $w'$ bits in multiple epochs. A wider window results in fewer epochs but higher communication overhead. In the first epoch in phase two, bits right to bits $j$, $j \geq r' + \Delta$ are checked; in the next epoch, bits $j$, $r' + \Delta - w' + 1 \leq j < r' + \Delta$, and so on. As soon as BS observes that bit $r$ is reached (i.e., the termination criterion is satisfied), it broadcasts a STOP message to terminate the protocol. It is assumed that BS has an estimate of $t$ (i.e., how many nodes may be compromised at most), and this estimate is used to compute $\Delta$.

We observe that during phase two of the variant protocol, no MACs for the bits to the left of bit $r-2$ are likely to be transmitted. As a result, the worst case communication overhead on a node in phase two is equal to the expected number of nodes which contribute to the bit $r-2$ or to its right. Using Property 4 of Sum synopsis, we get that $8/\phi \approx 10.54$ MACs per synopsis will be transmitted. The latency is at most $\Delta/w' = (\log_2(\phi t)+1)/w'$ epochs with high probability.

### 3.5.7 Comparing with Existing Approaches

In this section, we compare our attack-resilient protocols with the existing algorithms. We note that the attestation phase in Yang et al.'s [20] SDAP algorithm can be used to filter out the false sub-aggregates injected by the compromised nodes from the final aggregate. To the best of our knowledge, this is the only work present in the literature which addresses the problem of computing aggregates despite the attack.

Here, we present a comparative study of our basic attack-resilient computation protocol (Section 3.5.2 ), our variant protocol (Section 3.5.6 ), and SDAP. Table 3.4 compares these three protocols over latency, communication overhead, approximation error, and robustness to communication loss. We explain the table entries for each of the considered features. We also discuss the security issues of these three protocols.

Table 3.4: Comparing the Attack-resilient Protocols

| Protocols | Latency | Communication overhead | Approximate estimate in presence of attack | Robust to message loss |
|---|---|---|---|---|
| Our Basic Attack-resilient Protocol | 2 epochs (w.h.p.) | $O(mt)$ | Yes | Yes |
| Our Variant Attack-resilient Protocol | $O(\log t)$ epochs (w.h.p.) | $O(m)$ | Yes | Yes |
| SDAP by Yang et al. [20] | 2 epochs | $O(N)$ (worst case) | Yes | No |

- Latency: With the attestation phase included, SDAP takes 2 epochs to compute the final aggregate, while our basic attack-resilient protocol takes 2 epochs in most of the cases. The worst case latency incurred in our variant protocol is $O(\log t)$, where $t$ is the number of compromised nodes.

- Communication Overhead: In SDAP, the communication overhead of the attestation phase depends on the topology of the network; for an irregular network, the worst case node congestion is $O(N)$ hash values, where $N$ is the network size. In our basic protocol, a node needs to forward $O(mt)$ MACs in the worst case, where $t$ is the number of compromised nodes in the the network and $m$ is the number of synopses used. The worst case node congestion in our variant protocol is $O(m)$.

- Error in estimate: Our algorithm returns an approximate estimate of the aggregate where the amount of error is reduced if multiple synopses are used. On the other hand, Yang et al.'s SDAP returns an exact estimate if there is no attack and no message is lost in the network. However, an attacker can inject some amount of error in the estimate while remaining undetected.

- Robustness to Message Loss: As a tree-based topology is used for routing, communication loss over any link may disrupt Yang et al.'s algorithm. In contrast, our protocols are robust against loss because they use multi-path routing schemes to forward nodes' authentication messages to BS.

- Security: All of these algorithms are secure against the falsified sub-aggregate attack, but they are not equally resilient to the falsified local value attack. For our algorithms, the amount of error which can be present yet remain undetected in the final estimate of the aggregate is upper bounded by $tv_m$, where $t$ is the number of compromised nodes and $v_m$ is the upper bound of an individual sensed value. In contrast, in Yang et al.'s algorithm, the amount of error injected by $t$ compromised nodes which falsify their own sensed value could be as high as $tgv_m$, where $g$ is the number of sensor nodes present in one group. We note that nodes classify themselves into several groups in

Yang et al.'s scheme.

## 3.6 Simulation Results

In this section, we report on a detailed simulation study that examined the performance and security of our verification algorithm and the attack-resilient computation algorithms discussed in Sections 3.4 and 3.5, respectively. The simulation experiments examined our schemes in terms of several metrics, such as false negative rate, communication overhead, and latency.

### 3.6.1 Simulation Environment

Our simulations were written based on the TAG simulator developed by Madden et al. [4]. In particular, we added the security functionality to the source code provided by Considine et al. [7], which simulates their multi-path aggregation algorithms in the TAG simulator environment.

For our basic experimental network topology, we used a regular $30 \times 30$ grid with 900 sensor nodes, where one sensor is placed at each grid point and BS is at the center of the grid, as in [7]. The communication radius of each node is $\sqrt{2}$ unit, allowing the nearest eight grid neighbors to be reached. We assigned a unique ID to each sensor, and each sensor reading was a random integer uniformly distributed in the range of 0 to 250 units. We used the method of independent replications as our simulation methodology. If not mentioned otherwise, each simulation experiment was repeated 200 times with a different seed. We computed the 95% confidence intervals; unless shown in the reported plot, the confidence intervals are within $\pm 2\%$ of the reported value.

We considered packet losses, which are relatively frequent in sensor networks. We used a simple packet loss model in which packets are dropped with a fixed probability; if not mentioned otherwise, the loss rate is assumed to be 10%.

### 3.6.2 Results and Discussion

We now present the results we obtained for the verification protocol followed by those for the attack-resilient protocol. As Count can be considered as a special case of Sum, here we discuss only the results related to Sum aggregate.

#### Verification Protocol

For the verification protocol, we studied the false negative rate, which is the complement of the attack detection rate. We did not, however, study the false positive rate, where a false positive occurs if the following holds: Although no attack is launched, BS does not receive at least one valid MAC for each of the $k$ rightmost '1's in the final synopsis $\hat{B}$. Recall that using integrity checks in node-to-node communication, synopses and MACs in each node's aggregation message to its parents are bound in such a way that the following is ensured: If no attack is launched, BS will receive at least one MAC for each of the $k$ rightmost '1's in the final synopsis $\hat{B}$. In fact, a corrupted MAC that is a consequence of something besides an attack (e.g., communication error) can reach the BS. However, we observe that this problem is not protocol-dependent and it is out of the scope of our work. Finally, we observe that the verification protocol completes in one epoch irrespective of the final result being 'a success' or 'a failure'. So, we did not further study the latency in our simulation.

Recall that we presented our verification protocol for a single synopsis which can be extended for multiple synopses. Furthermore, this protocol works independently for each of the synopses eventually used. So, we present the following results for a single synopsis.

**False negative rate.** In our simulation we considered the worst case attack scenario: The attacker knows the network topology and the synopsis computed by each node. That is, the attacker can compute the final synopsis received by BS. So, the attacker is able to check if the following event, $E^k$ (ref. Section 3.4.4), occurs in the final synopsis: $k$ '1's are present to the right of a '0' bit, say bit $j$. We remind the reader that the aim of the attacker is to increase the value of Sum as much as possible while remaining undetected. So, the attacker takes the following strategy: If $E^k$ occurs, it changes all '0's at positions $\leq j$ to

'1's; otherwise, it does nothing. In fact, if the attacker modifies a bit after the $j$-th bit, that would be detected—the protocol verifies the MACs of the $k$ rightmost '1's. On the other hand, the attacker knows that no bit to the left of $j$ will be verified: For each '0' there, the attacker will change it to '1'.

Considering this worst case attack scenario, we assume that an attack is not detected each time an event $E^k$ occurs. In our simulation, we experimentally computed the probability for this event to occur, which we analytically studied in Section 3.4.4.

We extensively simulated the verification protocol for different values of network size ($20 \times 20$, $30 \times 30$, $40 \times 40$, $50 \times 50$ and $60 \times 60$ grid sizes) and value of the parameter $k$ (4, 5 and 6). For each combination of these parameters, we simulated the verification protocol $b = 100,000,000$ times.

Figure 3.5 reports the ratio $c/b$, where $c$ is the number of cases in which event $E^k$ occurred, i.e., the false negative rate. We observe that the network size does not affect the attack detection rate. As we chose the sensed value uniformly distributed between 0 to 250 units, the expected value of Sum is 125 multiplied with the network size. So, from Figure 3.5, it follows that also Sum does not affect the detection rate. Furthermore, the probability of $E^k$ to occur decreases while $k$ increases, as expected. For example, for $k = 4$ the false negative rate is about 0.007 while it is about $4.5 \times 10^{-5}$ for $k = 6$.

**Communication overhead.** We compare the communication overhead of the verification protocol to that of the original synopsis diffusion (SD) approach [9]. We recall that the original synopsis diffusion approach does not have any provision for security, hence no authentication message is transmitted. In contrast, while running our verification protocol, each node needs to forward the synopsis along with at most $k$ MACs.

Figure 3.6 plots the number of bytes a node transmits on average during the verification protocol considering different network sizes. This figure also illustrates the per-node byte overhead of the original synopsis diffusion approach. We assume that the size of a MAC is 4 bytes as used in [46] and the size of each synopsis is 2 bytes (compressed using run-length coding as used in [9]). We observe that the verification protocol costs roughly $4k$ bytes

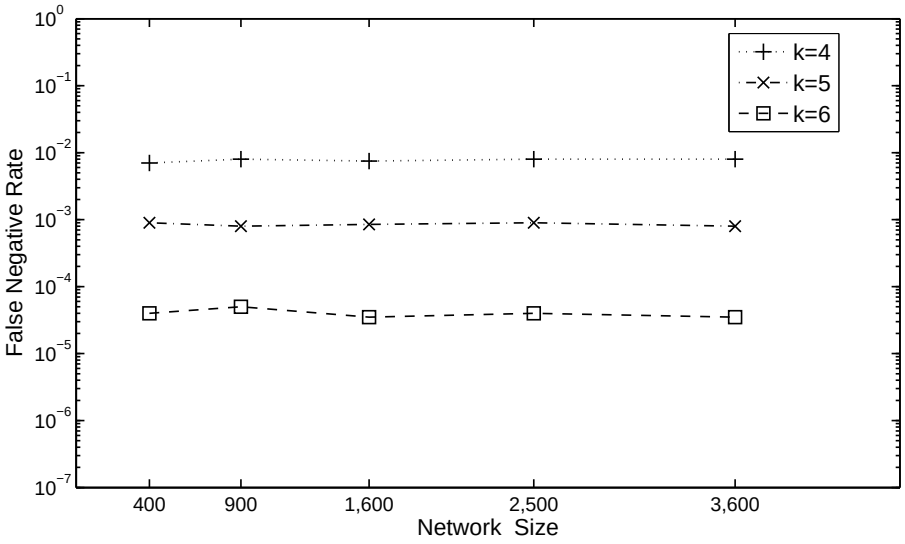



Figure 3.5: False Negative Rate of the Verification Protocol

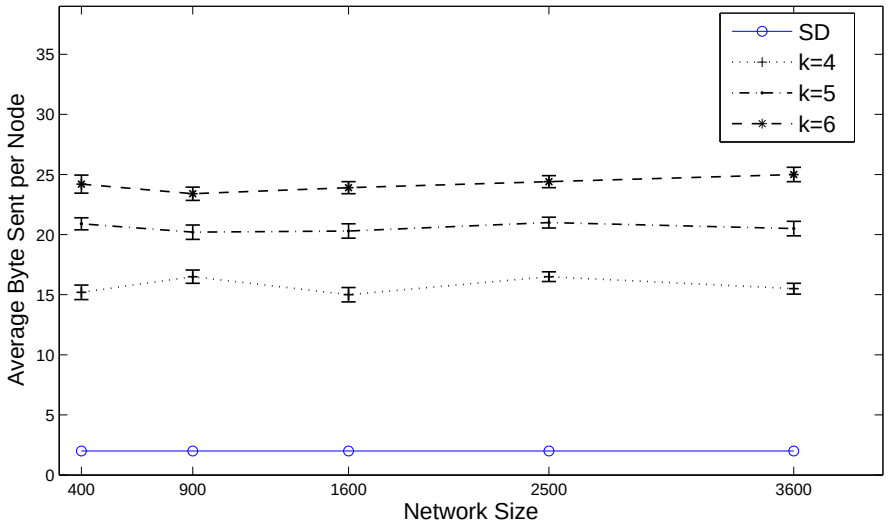



Figure 3.6: Sent Byte Overhead per node in the Verification Protocol

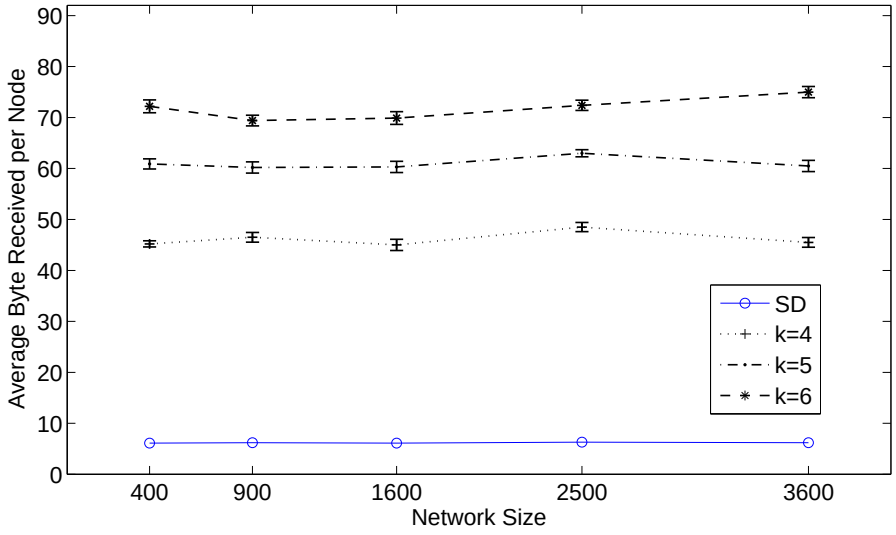



Figure 3.7: Received Byte Overhead per node in the Verification Protocol

of extra overhead for each node compared with the original synopsis diffusion approach. We also observe that the byte overhead does not increase with the network size, which illustrates the scalability of our approach.

As for the number of bytes received, we observe that this is proportional to the number of bytes sent. This is because each parent node has more than one child node and each of them sends at most $k$ MACs. For example, if node $X$ has $c$ child nodes, then $X$ receives at most $ck$ MACs. Figure 3.7 shows the average per-node byte overhead received.

### Attack-resilient Protocol

We recall that the attack-resilient protocol exploits the correlation among the different synopses eventually used. We present the following results for $m = 20$ synopses as used in previous work [7, 9]. We computed these 20 synopses in parallel using the PCSA algorithm as in the experiments reported in [7, 9, 10].

We recall from Section 3.5 that the performance of the attack-resilient protocol primarily depends on the looseness of the estimate, $r'$ obtained in phase one. Below we first study the

accuracy of the estimate $r'$ in different scenarios, and then report on the other performance metrics, such as communication overhead and latency.

**Error in estimate $r'$.** We recall that in phase one MACs are generated only for $m'$ synopses. As discussed in Section 3.5, for a given $m'$, the maximum error in estimate $r'$ depends on how many compromised nodes participate in the false MAC injection attack during phase one. The analysis in Section 3.5.4 predicts that with $m' = 4$, the following inequality holds with more than 99% probability, where $t$ is the number of compromised nodes, $ALB$ stands for Approximate Lower Bound, and $AUB$ stands for Approximate Upper Bound of $r'$.

$$ALB = r - \log_2 \phi t - 1 \leq r' \leq r + 2 = AUB \tag{3.35}$$

To verify the lower bound of $r'$, we used $m' = 4$ and varied $t$ over 2,4,6,8, and 10. For any particular value of $t$, we simulated the false MAC injection attack during phase one 1,000,000 times and reported the percentage of times $r'$ was lower than $r - \log_2 \phi t - 1$. We observed that for each of $t$, $r'$ was lower than the corresponding ALB fewer than 0.5% of the times.

We note that a false MAC injection attack can only lower the estimate $r'$, so we verified the upper bound of $r'$ in the absence of the attack. We simulated phase one of the attack-resilient protocol in the absence of the false MAC injection attack 1,000,000 times. We observed that for each value of $t$, in fewer than 0.4% of times $r'$ was higher than the $r + 2$, which confirms our analysis.

**Communication overhead.** For the communication overhead of phase one, we recall that during this phase each node needs to forward at most $\eta$ MACs per synopsis, where $\eta$ is the length of each synopsis. The communication overhead on a node in phase two depends upon how many nodes contribute to bits to the right of bit $r'$ in the synopses because each of these nodes send a MAC to BS. Our analysis in Section 3.5.4 shows that for each synopsis, the total number of MACs sent in the network during phase two is likely to be less than $4t$, when $t$ compromised nodes participate in the attack.

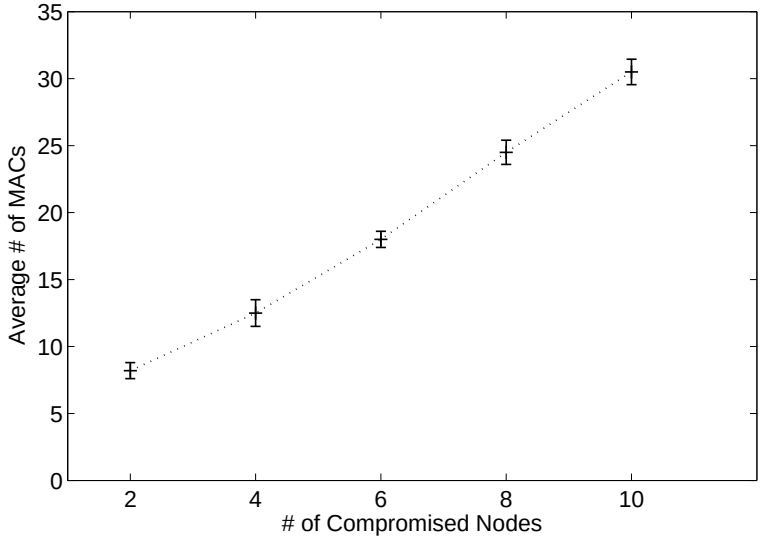



Figure 3.8: Average Number of MACs Forwarded by a Node in Phase Two of the Attack-resilient Computation Protocol

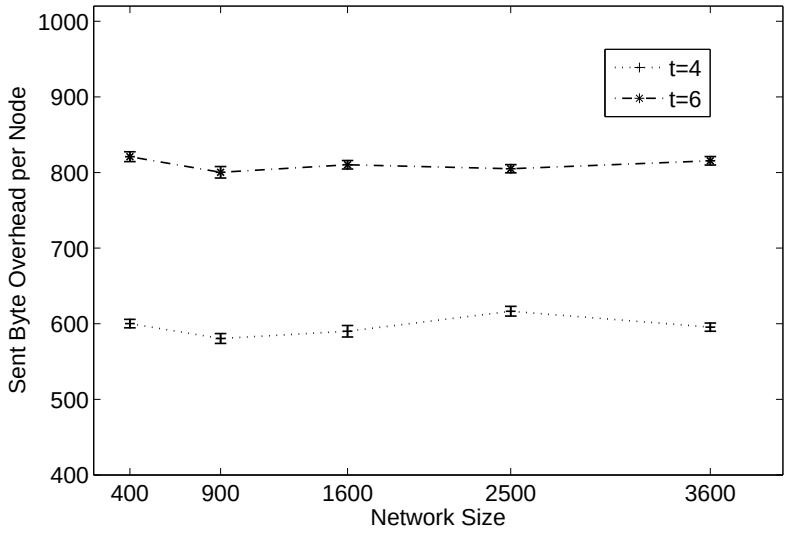



Figure 3.9: Impact of Network Size on Sent Byte Overhead per Node in the Attack-resilient Computation Protocol

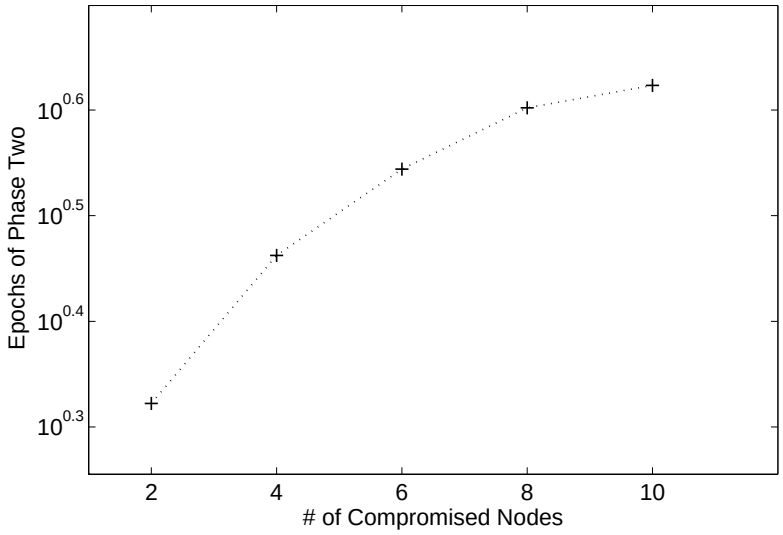



Figure 3.10: Latency of Phase Two of the Variant Attack-resilient Computation Protocol

Figure 3.8 plots the number of MACs sent per synopsis during phase two as a function of the number of compromised nodes, $t$. In the above experiment, we used $30 \times 30$ grid as the network. We observe that the number of MACs increases linearly with $t$, which confirms our analysis.

**Effect of network size.** In this experiment, we studied the impact of the network size on the communication overhead of the attack-resilient protocol. Recall from Section 3.5 that the number of MACs sent in phase two depends on the number of compromised nodes, $t$ present in the network, but not on the network size. Figure 3.9 confirms our analysis; we observe that for a particular value of $t$ (4 or 6), the average sent byte overhead per node is more or less constant as the network size increases. This figure thus illustrates the scalability of our approach for attack-resilient aggregation.

**Variant protocol.** As discussed in Section 3.6.2, the worst case latency of phase two of the variant protocol is determined by the upper bound of the number of compromised nodes, $t$. It is interesting to note that the latency improves if the attack is successful to deviate the estimate $r'$ in phase one, i.e., the protocol incurs the worst case latency when there is

no attack during phase one. Our simulation confirms this observation. So, in the following we report the latency in the worst case scenario for the latency—a no-attack condition. Figure 3.10 plots the number of epochs taken by phase two of the variant protocol as a function of the upper bound of the number of compromised nodes with sliding window width $w' = 2$. The figure shows that the number of epochs increases at logarithmic scale with the upper bound of the number of compromised nodes. However, we observed that the byte overhead of phase two does not increase with $t$.

## 3.7 Summary

We discussed the security issues of in-network aggregation algorithms to compute two basic aggregates, Count and Sum. We presented a verification algorithm which would enable the base station (BS) to verify whether the computed aggregate, Count or Sum, was valid, and we also proposed an attack-resilient computation algorithm which would guarantee the successful computation of the aggregate even if a few nodes in the network were compromised.

Prior to this dissertation, researchers proposed a few secure aggregation algorithms to compute Count and Sum, but these proposals were limited to tree-based aggregation. To the best of our knowledge, most of the ring-based aggregation algorithms are vulnerable to simple attacks, such as algorithms within the synopsis diffusion framework, which is the focus of this chapter. We analyzed the vulnerabilities of the synopsis diffusion framework in the presence of compromised nodes and proposed countermeasures.

# Chapter 4: Securely Computing Median

This chapter discusses the security issues of in-network aggregation algorithms to compute Median, which is an important aggregate besides Count and Sum. We design a verification algorithm which enables the base station (BS) to verify whether the computed aggregate, Median, is valid. Furthermore, we propose an attack-resilient computation algorithm which guarantees the successful computation of the aggregate even if a fraction of nodes in the network are compromised. To design these algorithms, we use the tree topology. Our algorithms can be extended to compute other quantiles besides Median.

**Organization.** The rest of this chapter is organized as follows. In Section 4.1, we provide the background for this chapter. Section 4.2 describes the problem statement and the assumptions taken in this chapter. In Section 4.3, we present our basic protocol, whose security and performance analysis is given in Section 4.4. Section 4.5 describes our attack-resilient protocol and its security and performance analysis. We present the simulation results in Section 4.6 and compare our proposed solutions with others in Section 4.7. Finally, we conclude the chapter in Section 4.8 by presenting a summary.

## 4.1 Preliminaries

We observe that an in-network aggregation algorithm cannot cheaply compute the exact Median, while the worst case communication overhead per node is $\Omega(N)$, where $N$ is the number of nodes in the network [4]. As a result, several researchers have advocated computation of an approximate Median. Greenwald et al. [12] and Shrivastava et al.[13] proposed in-network aggregation algorithms to compute an approximate Median, which we briefly discussed in Section 2.2.2.

Like the in-network Count or Sum computation algorithms, the above in-network Median algorithms [12, 13] are vulnerable to attacks. As discussed in Section 2.2.3, 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. For example, in Greenwald et al.'s approximate Median computation algorithm [12], 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. [37], which we briefly presented in Section 2.4.2. Their scheme [37] 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 Median using an in-network algorithm. However, researchers [30] observed that Median is an important aggregate.

One might suggest an approach which runs Greenwald et al.'s algorithm [12] to compute an approximate Median and then employs Chan et al.'s verification protocol [37] to verify if the computed value is indeed a valid estimate. We refer to this approach as GC in the rest of this chapter. Later we will compare our algorithms with the GC approach.

To provide a background for the rest of this chapter, we now discuss Greenwald et al.'s Median computation algorithm and Chan et al.'s Sum verification algorithm. Next, we present the 'GC approach'.

### 4.1.1 Greenwald et al.'s Approximate Median Algorithm

This algorithm [12] is based on a summarization technique which represents a set of sensor readings by a *quantile summary*. From a $\epsilon$-approximate quantile summary, we can derive an arbitrary quantile of the data set satisfying $\epsilon$-approximation error bound. In particular, an $\epsilon$-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 $\epsilon'$ 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 $\epsilon$ 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 $\epsilon$ and the worst case node congestion is $O(\frac{\log^2 N}{\epsilon})$.

### 4.1.2 Chan et al.'s Verification Algorithm

This scheme [37] 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[1] 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 [4].

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 [49] 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 $\beta$ 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 the 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 BS 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 [18]. 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

[1]Similar to the prior work [4], 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.

*off-path values* up to the root node relative to $X$'s position on the commitment tree. These values, together with $X$'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 had 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. [50] 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.

### 4.1.3 GC Approach

One can suggest a scheme to securely compute an approximate Median using Greenwald et al.'s Median computation algorithm [12] in conjunction with Chan et al.'s verification algorithm [37], presented above. In the first phase of the GC approach, given the approximation error bound $\epsilon$, 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 $\epsilon$ 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 [50]), where $N$ is the number of nodes in the network, $\epsilon$ is the approximation error bound, and $\Delta$ is the number of neighbors of a node.

Later in this chapter, we propose an alternative approach to compute and verify an approximate Median, which proves to be more efficient compared to the GC approach.

## 4.2 Assumptions and Problem Description

The goal of this chapter 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 multi-hop 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 $\epsilon$. In the rest of the chapter, 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 [51]) 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 [4]. However, our approach does not rely on a specific tree construction algorithm. The approximation error $\epsilon$ in the estimated Median $\hat{m}$ is determined by how many positions $\hat{m}$ is away from

the exact Median $m$ in the sorted list of all of the sensed values. For ease of exposition, without loss of generality we assume that all of the sensed values are distinct. Note that we could relax this assumption by defining an order on IDs of the nodes that have the same sensed value. Also, for the ease of exposition, we assume that there is an odd number of sensed values in total so that Median is one element of the population.

**Security Model** We assume that the BS cannot be compromised. The BS uses a protocol such as $\mu Tesla$ [18] 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 chapter, 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 [16, 18].

We consider that the adversary can compromise a few sensor nodes (i.e., insiders) without being detected. If a node is compromised, all of 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 chapter 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. [12]), 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 Median by a large amount (e.g., in [12]).

**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 $\epsilon$, the BS accepts an estimate $\hat{m}$ which satisfies the following constraint:

$$|rank(\hat{m}) - \frac{N+1}{2}| \leq \epsilon N \tag{4.1}$$

  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 Median to be $(\epsilon + w/N)$. However, given an upper bound on $w$, the user could adjust his input $\epsilon$ to finally meet the required bound.

**Notation** A list of notations used in this chapter is given in Table 4.1.

Table 4.1: Notations Used in Describing the Secure Median Protocols

| 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 |
| $\epsilon$ | 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 $a_2$ |
| $\Delta$ | the maximum degree of the aggregation tree |
| $g$ | number of groups in the attack-resilient algorithms |
| $w$ | number of compromised nodes |

## 4.3 Computing and Verifying an Approximate Median

Our approach is based on sampling—a uniform sample of sensed values is collected from the network to make a preliminary estimate of Median, which is verified and refined later. In particular, the key elements of our approach are to compute a histogram of the sensor readings and then derive an approximate Median from the histogram. The collected sample of sensed values from the network is used to construct the histogram bucket boundaries.

We first present a histogram verification algorithm and then describe our basic scheme.

### 4.3.1 A Histogram Verification Algorithm

Our algorithm for computing and verifying a histogram of sensed values is adapted from Chan et al.'s scheme [37] to compute and verify Sum aggregate.

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 to such an interval, $(q_i, q_{i+1})$ as bucket $B_i$ with boundaries $q_i$ and $q_{i+1}$.

As noted in [37], 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 al.'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 $\beta$ 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

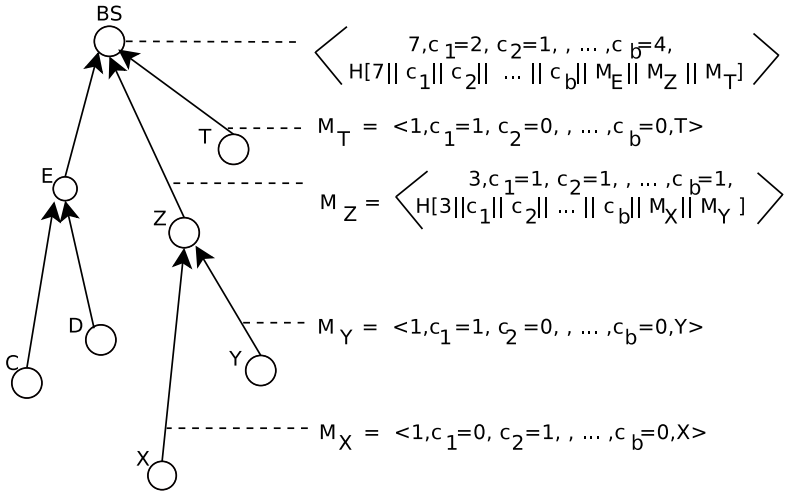



Figure 4.1: The Aggregation-commit Phase in Histogram Verification: In this example, $v_X$ lies in bucket $B_2$, $v_Y$ lies in bucket $B_1$, and $v_Z$ lies in the last bucket $B_b$. While leaf nodes $X$ and $Y$ set the corresponding bucket count to 1, the internal node $Z$ aggregates the messages $M_X$ 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; and 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 $B_b$.

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 4.1 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 the 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

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. [50], we use a balanced commitment tree as an overlay on the physical aggregation tree. 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 $O(b\Delta \log N)$.

### 4.3.2 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 of 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.

#### 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 identifier, *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 of 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).

**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 Median.

**i) Computing histogram boundaries**

We consider the number of buckets, $b$, as a parameter. In Section 4.4.2 we discuss how

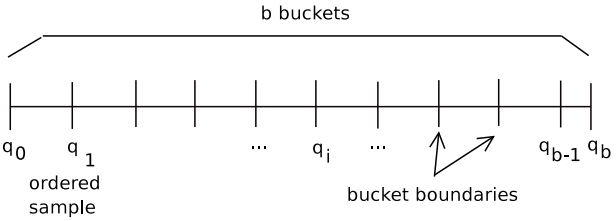



Figure 4.2: Computing Histogram Boundaries: The histogram boundaries are computed using the sample collected in the previous phase.

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 4.2. $E_j$ represents the value of $j$-th item in the sample sorted according to the value, with $j$ varying from 1 to $S$.

**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 4.3.1. If there is no attack present in the network, at the 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 4.5.

**iii) Estimating 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:

$$e_l + c_0 + c_1 + \ldots + c_{j-1} < \frac{N+1}{2} \tag{4.2}$$

$$c_0 + c_1 + \ldots + c_j \geq \frac{N+1}{2} \tag{4.3}$$

$$c_j \leq 2\epsilon N \tag{4.4}$$

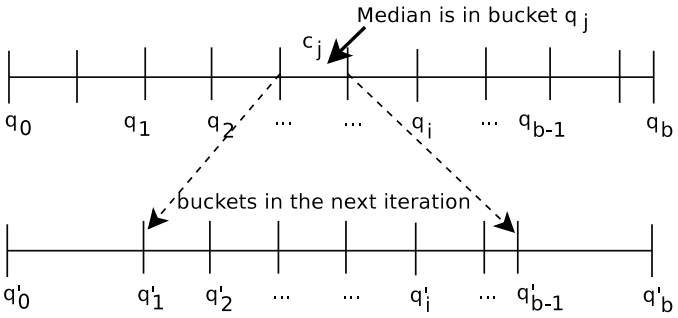



Figure 4.3: Splitting the Bucket: If the bucket $j$, which contains Median, has more than $2\epsilon N$ elements, the bucket is split in order to meet $\epsilon$ approximation error bound.

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 inequalities are satisfied. Then, we check if the above $j$ also satisfies inequality (4.4). Note that if inequality (4.4) is satisfied, then it is guaranteed that either $q_j$ or $q_{j+1}$ is $\epsilon N$ away from the exact Median, which is reported as our final estimate. If the inequality (4.4) 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 4.3. 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 inequality (4.4) 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 when modifying the right-hand side of inequalities (2) and (3), other quantiles besides Median can be approximately computed.

## 4.4 Security and Performance Analysis of Our Basic Protocol

### 4.4.1 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 the multi-path 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 $\theta$ 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 $\epsilon$-approximate Median.

### 4.4.2 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, $\Delta$ 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.

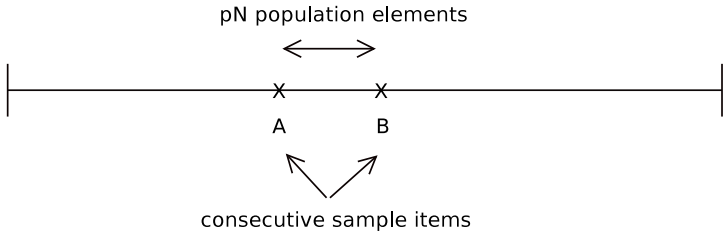



Figure 4.4: How Far Apart are Two Consecutive Elements in the Sample?

The second phase goes to the next iteration if the bucket $b_j$ in which 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.

**Lemma 4.4.1.** *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.*

*Proof.* Let $A$ and $B$ be two consecutive items in the sample after the sample items are sorted (as shown in Figure 4.4). What we want to compute is the probability of having more than $pN$ population elements between $A$ and $B$. Once the sample item, $A$, is chosen, we have other $S-1$ population elements that 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 $(1-p)^{S-1}$. □

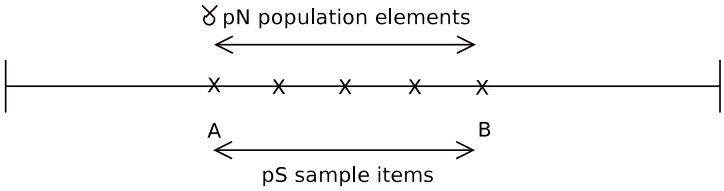



Figure 4.5: What is the Chance that $\gamma pN$ Elements will Fall within $pS$ Sample Items, where $\gamma > 1$, $0 < p < 1$, and $\gamma p < 1$?

As an example, from Lemma 4.4.1, 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 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.

**Lemma 4.4.2.** *The probability that more than $\gamma pN$ ($\gamma > 1$, $0 < p < 1$, $\gamma p < 1$) population elements lie between the minimum and the maximum of $pS$ consecutive sample items of a sorted sample of size $S$ is*

$$\xi(S, p, \gamma) = \sum_{i=0}^{pS-1} \binom{S-1}{i} (\gamma p)^i (1 - \gamma p)^{S-1-i} \tag{4.5}$$

*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 4.5. So, the expected number of population elements lying between $A$ and $B$ is $pN$. We would like to compute the probability of having more than $\gamma pN$ 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 $\gamma pN$ (i.e., the interval includes $\gamma pN$ population elements). For each of these $S-1$ sample items, the probability to be chosen from that interval is $\gamma p$. So, the probability that not more than $(pS-1)$ items among the $S-1$ items will be:

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

□

**Number of buckets vs. number of iterations**

If we use $b = \frac{\gamma}{2\epsilon}$ buckets, which is of $O(\frac{1}{\epsilon})$, where $\gamma$ is a constant greater than 1 and $\epsilon$ 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 4.4.2, 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 (4.5) is a decreasing function of $\gamma$, by choosing the appropriate $\gamma$, 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 $\epsilon$. That means, in this setting, our protocol ends in one iteration in 98% of the 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 $\Delta$ 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

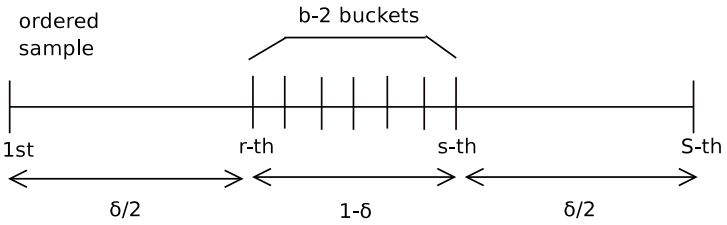



Figure 4.6: Betting on Median position—We divide a small fraction of sample items in the middle into $(b-2)$ buckets and place the rest of the sample items at either end in one bucket each.

than $\frac{2\epsilon}{\gamma}S$ sample items. So, as shown above, with the appropriate $\gamma$ 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 $\Delta$ is the degree of the aggregation tree.

**Betting on Median position**

We observe that with the sorted sample items being equally divided into $b$ buckets, the probability of a bucket containing Median is not the same for all buckets. 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 4.6. If we are lucky, after one iteration we find that Median lies in one of the smaller $(b-2)$ buckets and thus our algorithm converges faster with a given number of buckets. We consider $\delta$, $0 \leq \delta \leq 1$ as a design parameter, which represents the probability that Median actually lies in one of the end buckets, i.e., with probability $(1-\delta)$ that the Median falls in one of the $(b-2)$ buckets in the middle.

We can compute one positive integer $r$ so that 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 $\delta$, $r$ can be found using the formula given in [52], which is as follows:

$$1 - \delta = 2^{-S} \sum_{i=r}^{S-r} \binom{S}{i}. \tag{4.6}$$

Computing $r$ using the above formula is closely related to the *sign test*, so the table by MacKinnon [53] 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

$$r = \frac{S}{2} - \frac{1}{2} u_\delta \sqrt{S}, \tag{4.7}$$

where $u_\delta$ 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 $\delta$, the faster we reduce the search space to find 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 $\delta$ 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 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:

$$(1-\delta)(\frac{u_\delta\sqrt{S}}{b-2}) + \delta(\frac{S}{2} - \frac{1}{2}u_\delta\sqrt{S}) \tag{4.8}$$

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

## 4.5 Attack-resilient Median Computation

Although our basic protocol, discussed in Section 4.3.2, detects *falsified sub-aggregate attack*, it fails to output an estimate of 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 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 [20].

We may employ different grouping techniques based on node's geographic location or node IDs. We may also use a grouping technique which is based on the nodes' positions on the aggregation tree. Once the group aggregate is computed, the group leader sends it directly to the BS; to avoid having any node in the middle to drop group aggregates, we use a multi-path routing mechanism.

Also, we may exploit the robustness property of 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 Section 4.5.4, 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.

### 4.5.1 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, $g$ 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 4.7.

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 95% of the nodes). Below we discuss the attack-resilient histogram computation and verification algorithm.

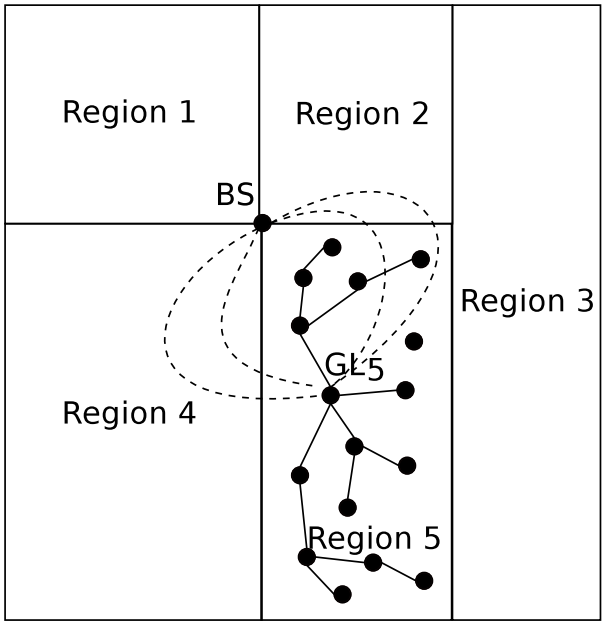



Figure 4.7: Geographical Grouping: The network is divided into several regions where each region has a group leader ($GL_i$). The $GL_i$ sends the region aggregate to the BS by multiple paths.

**Algorithm description**

The nodes in each region locally perform the histogram computation and verification protocol described in Section 4.3.1 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 communications between $GL_i$ and the BS. Below we focus on these additional messages, skipping the detailed description of the rest of the protocol, which can be found in Section 4.3.1. 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.

**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 [4] 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 4.3.1.

**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 $i$.

**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.

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

**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.

**Security analysis**

We recall from Section 4.4.1 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 flag 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, and (ii) the group leader, $GL_i$ exchanges a few messages with the BS, discussed in Section 4.5.1, 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.

**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 $\Delta$ is the number of neighbors of a node on the aggregation tree. Considering the analysis given in Section 4.4.2, 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.

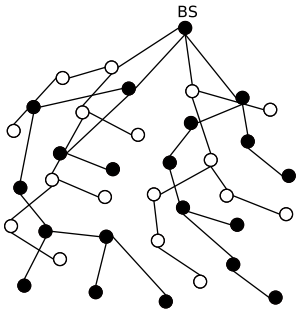



Figure 4.8: ID-based grouping: The network is divided into several groups based on node ID, e.g., the odd ID nodes [filled circles] form one group and the even IDs [empty circles] form another. The aggregation is performed separately in each group.

### 4.5.2 ID-based Grouping

We now propose a different grouping technique which is based on the node's ID instead of the node's location. In this scheme, no location information is needed for the nodes or for the BS. The main idea is that the BS divides the set of node IDs into several subsets, and the nodes belonging to a subset form an aggregation group. This technique assumes that nodes in each subset are connected. The limitation of this scheme is that reducing the size of a subset increases the probability that these nodes are not physically connected; so, in that case, we cannot form a group which is connected by itself. We can address the above problem by giving an overlay structure to a group, where two nodes in a group can be connected via multiple paths which may possibly go through a few non-group nodes. Except for the grouping criteria, this scheme works similarly as the geographical grouping scheme described above. The level of security and the performance of the two schemes are similar.

### 4.5.3 Dynamic Grouping

We may also design a dynamic grouping scheme which does not use pre-defined groups. All of the nodes in the network perform the basic histogram verification algorithm described in Section 4.3.2 with storing some additional information—each node $X$ stores the aggregate

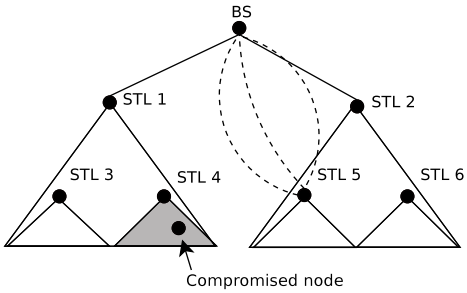



Figure 4.9: Dynamic Grouping: A single aggregation tree is constructed which covers all of the network nodes. If the verification fails, the BS dynamically selects a few sub-trees. The local aggregates are verified, where the root of the sub-tree ($STL_i$) acts as the group leader.

of its sub-tree and the compressed OK string which $X$ has forwarded to the parent node. We assume that the BS has the knowledge of the topology of the aggregation tree. If the BS successfully verifies the OK message, no further action is taken. Otherwise, the BS identifies some nodes on the aggregation tree and requests these nodes to send their stored information (the aggregate and OK string). In this way, the BS can localize the attacker node. Further verification can be performed using different aggregation points. Like geographical grouping, the refinement can be achieved until the attacker node is identified or the percentage of verified values satisfies the BS.

### 4.5.4 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 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 the 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 multi-path. 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 Median is $w\frac{N/g}{N} = \frac{w}{g}$.

## 4.6 Simulation Results

In this section, we report on a simulation study that examined the performance of our basic protocol discussed in Section 4.3. 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 4.4.2, about the inter-relationship among parameters, such as error bound $\epsilon$, 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 of the communication overhead of the sampling phase and the histogram computation and verification phase are discussed in Section 4.4.2.

### 4.6.1 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 $\epsilon$ error bound varies from 5% to 15%.

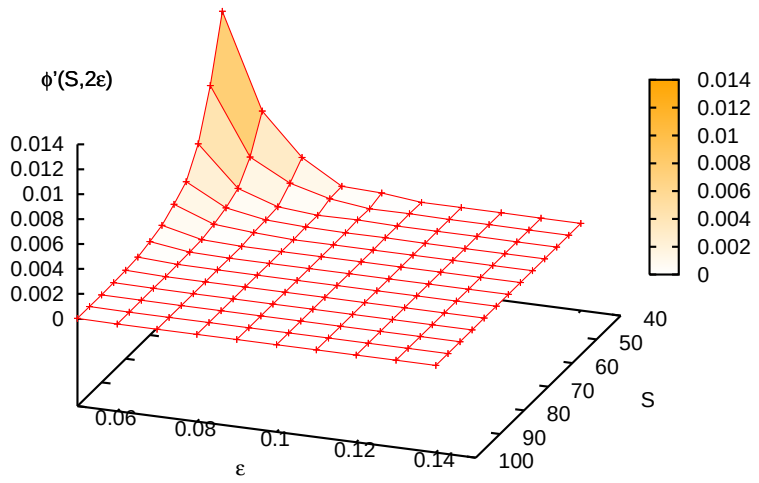



Figure 4.10: Computing the Chance that We need to Collect more Sample Items: Given an $\epsilon$, we choose a sample size so that the probability that we need to redo the sampling is close to zero.

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.

### 4.6.2 Results and Discussion

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

**What is the chance that one sampling phase is not enough?** In Lemma 4.4.1, we analytically compute this probability which we evaluate via simulation here. For each pair $(S, \epsilon)$, we collect a sample of size $S$ and we compute the number of times, $\tau$ there are more than $2\epsilon N$ elements between the two consecutive sample items containing 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 4.10. It is worth noticing that the value of

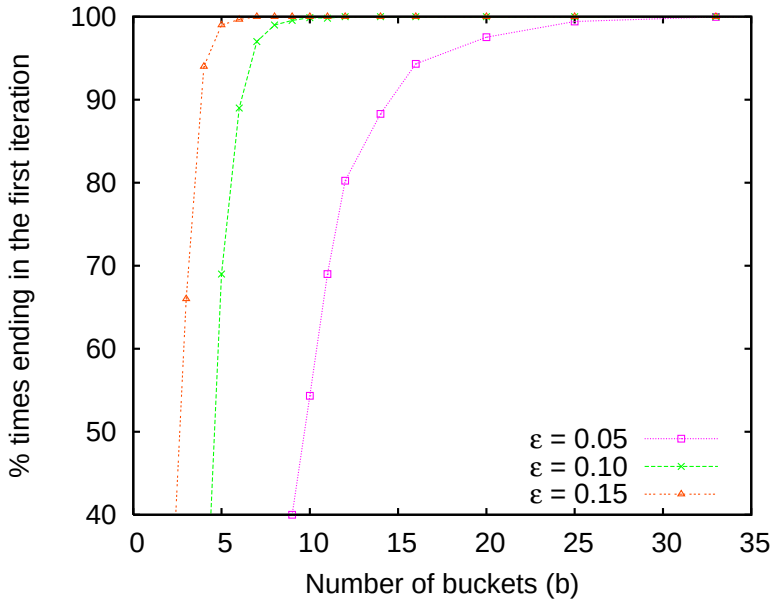



Figure 4.11: How the Chance of Our Algorithm Ending in One Iteration Varies with Different Numbers of Buckets

$\phi'(S, 2\epsilon)$ is less than $4 \times 10^{-5}$ for $\epsilon > 0.05$ when the sample size $S$ is more than 95. In fact, as expected, for a given $\epsilon$, 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 4.4.2, we analyzed the dependence of the number of iterations of our algorithm on the number of buckets chosen, which we validate here via simulation. First, we estimate the number of buckets required to end our protocol in one iteration in most cases. Figure 4.11 illustrates the percent of cases our protocol ends in the first iteration. The figure confirms our analysis that, for $\gamma = 2$, if we use more than $\frac{1}{\epsilon}$ buckets (i.e., 20, 10, and 7 buckets for $\epsilon = 0.05, 0.10, 0.15$, respectively), it is highly likely that we need just one iteration. Finally, Figure 4.12 shows the average number of iterations required using different number of buckets, where $S = 100$. This validates our analysis that the average number of iterations is $O(log_b(\frac{1}{\epsilon}))$ when $b$ buckets

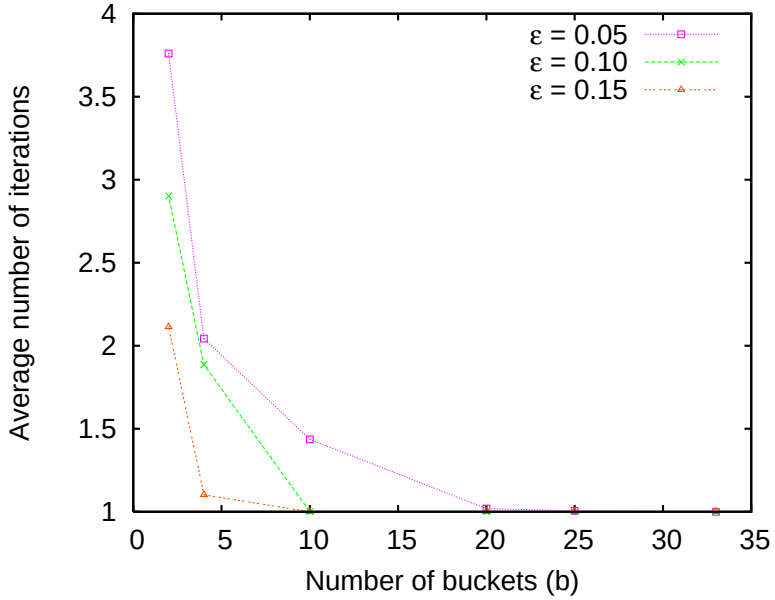



Figure 4.12: The Number of Iterations vs. the Number of Buckets: if the number of buckets is $O(\frac{1}{\epsilon})$, it is highly likely that our algorithm ends in one iteration.

are used.

**Betting on the Median position.** In Section 4.4.2, we described an optimization technique based on the observation that Median lies with higher probability in the buckets that are in the center of the sorted sample. We studied how different choices of $\delta$ determines the average number of iterations for a given number of buckets. Figure 4.13 shows the average number of iterations for different values of $\delta$ while we use $\epsilon = 0.05$ and $S = 100$.

## 4.7 Comparing Our Algorithms with Others

To the best of our knowledge, there is no prior work which securely computes Median using an in-network algorithm. We compare our algorithms with Greenwald et al.'s insecure algorithm and the GC approach presented in Section 4.1.3.

Table 4.2 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,

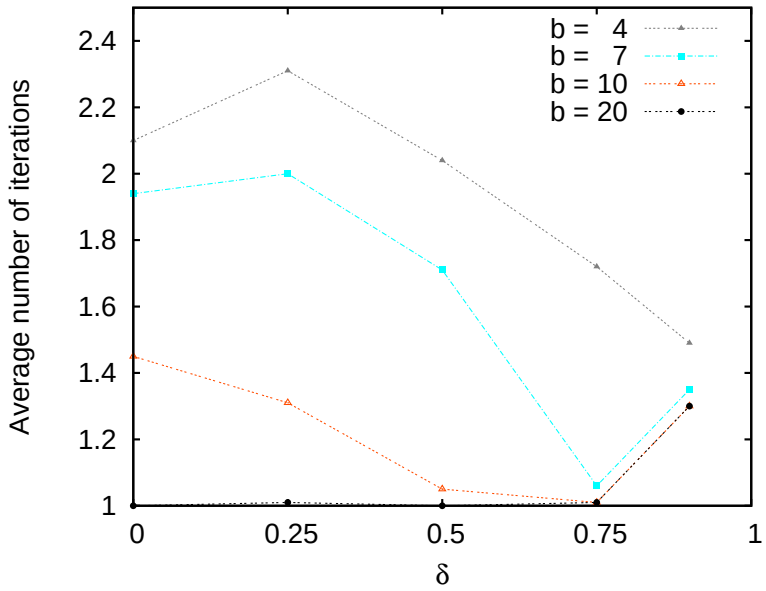



Figure 4.13: Proper Choice of $\delta$ Reduces the Number of Iterations Needed

Table 4.2: Comparing the Median Computation Protocols

| | Node congestion | Latency (epochs) | Verification | Attack-resilient computation |
|---|---|---|---|---|
| Greenwald et al.'s protocol [12] | $O(\frac{log^2 N}{\epsilon})$ | 2 | No | No |
| GC approach (Section 4.1.3) | $O(\frac{log^2 N}{\epsilon})$ | 6 | **Yes** | No |
| Our basic protocol (Section 4.3.2) | $O(\frac{1}{\epsilon}\Delta logN)$ | 6 w.h.p. | **Yes** | No |
| Our extended protocol (Section 4.5) | $O(\frac{1}{\epsilon}\Delta logN)$ | 6 w.h.p. | **Yes** | **Yes** |

which represents the worst case overhead on a single node. We measure the latency of the protocols in *epochs*. 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 a *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 Median in the presence of an attack. Finally, an attack-resilient protocol is so if it guarantees the computation of Median in the presence of a few malicious nodes.

## 4.8 Summary

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 Median. However, it is widely considered that Median is an important aggregate. In this chapter, 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 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. Moreover, our algorithms can be extended to compute other quantiles besides Median.

# Chapter 5: Conclusions

## 5.1 Summary

In this dissertation, we presented secure in-network aggregation algorithms for wireless sensor networks considering the possibility that a fraction of nodes might become compromised. In particular, we designed verification algorithms and attack-resilient computation algorithms to compute basic aggregates, such as Count, Sum and Median. Using a verification algorithm, the base station can verify the correctness of the computed aggregate, while an attack-resilient computation algorithm guarantees the successful computation of the aggregate despite the presence of attacks.

Our secure Count and Sum algorithms are designed within the synopsis diffusion framework, which uses the ring topology. In the synopsis diffusion framework, the aggregate (Count or Sum) is represented by a bitmap called *synopsis*. The proposed secure algorithms exploit our observation that to verify the final *synopsis*, the base station does not need to receive authentication messages from all of the nodes in the network. The per-node communication overhead of our verification algorithm is $O(1)$, which is more efficient than $O(\log S)$ complexity of the existing verification algorithm [19] , where $S$ is the value of the final aggregate (Count or Sum). Like the existing algorithm, the latency incurred in our verification algorithm is 2 *epochs*.

To the best of our knowledge, there is no algorithm proposed in the literature to compute Count and Sum within the synopsis diffusion framework in the presence of a compromised node, which is achieved by our attack-resilient computation algorithm. The per-node communication overhead incurred in this algorithm is $O(t)$, where $t$ compromised nodes are present in the network, and the latency is $O(1)$ epochs. We also proposed a variant algorithm which costs $O(1)$ communication overhead per node but at the expense of greater

latency, which is $O(\log t)$.

Our secure Median algorithms are designed on the tree topology. We believe there is no other work, prior to this dissertation, which addresses the security issues of in-network computation of Median. Our Median verification algorithm costs $O(\frac{1}{\epsilon} \cdot \log N)$ node congestion while taking 6 epochs to complete, where $N$ is the network size and $\epsilon$ is the desired error bound in the estimate of Median. To compute an approximate Median in the presence of compromised nodes, our attack-resilient computation algorithm exploits the principle of divide and conquer.

## 5.2 Future Work

There are some open issues related to this dissertation, which may be addressed in future research. Below we discuss two of them.

### 5.2.1 Secure Median Computation over the Ring Topology

We recall that our Median algorithm, described in Chapter 4, is designed for the tree topology. In each iteration of our algorithm, a histogram of sensor readings is computed and verified. Also, the bucket in the histogram which contains the Median is refined more and more over the iterations until the desired error bound is achieved. As a future research topic, this scheme may be extended to the synopsis diffusion framework, which is based on the ring topology.

As multi-path routing is used in the synopsis diffusion framework, the extended scheme will become robust to the message loss problem. However, the extended scheme needs to use a duplicate-insensitive counting algorithm, such as our secure Count algorithm, discussed in Chapter 3, to compute the bucket counts in the histogram.

The remaining challenge is as follows. Like the original synopsis diffusion Count algorithm, our secure Count algorithm outputs an approximate estimate. As a result, in any iteration of the histogram computation algorithm, we cannot decide which bucket contains the Median for sure—we can only predict which bucket is the most likely one. This requires

including a backtracking provision during the iterations of the histogram computation algorithm. Also, the final estimate of the Median will be probabilistic in nature. The tradeoff between the amount of error in the estimate of the Median and the communication overhead may be studied in the future research.

### 5.2.2 In-network Filtering of False Data

In the proposed schemes, discussed in Chapters 3 and 4, authentication messages (MACs) sent by individual nodes are propagated all the way to the base station before being verified (and filtered, if false). In large-scale networks, it may be advantageous to do in-network processing of these messages before they reach the base station. As an example, a parent node $X$ can easily detect if any of its child nodes, say $Y$, are falsifying the local value, because sensor readings are correlated with location. To detect if any child node $Y$ is forwarding a false sub-aggregate, a parent node $X$ may demand multiple descendant nodes of $Y$ to send an authentication message to $X$. The attack-resilient property of this kind of schemes may be derived from an assumption that no more than $k$ nodes among the one-hop neighbors of a node can be compromised. The main challenge is to find the tradeoff among several factors, such as attack-resiliency (the value of $k$), communication overhead, and storage.

We may also explore a few other directions to achieve in-network filtration of false data. For a large sensor network, we may study an alternative architecture which assumes multiple trusted nodes deployed throughout the network, instead of just one trusted node as the base station. In this architecture, only trusted nodes act as the aggregators, and other nodes send their local values directly to the nearest trusted node. Each aggregator node securely forwards its sub-aggregate to the aggregator in the next level in the hierarchy, and so on.

# Bibliography

# Bibliography


[1] Habitat Monitoring on Great Duck Island, http://www.greatduckisland.net/.

[2] The Firebug Project, http://firebug.sourceforge.net.

[3] James Reserve Microclimate and Video Remote Sensing, http://www.cens.ucla.edu.

[4] S. Madden, M. J. Franklin, J. Hellerstein, and W. Hong, "TAG: A tiny aggregation service for ad hoc sensor networks," in *Proc. of 5th USENIX Symposium on Operating Systems Design and Implementation*, 2002.

[5] J. Zhao, R. Govindan, and D. Estrin, "Computing aggregates for monitoring sensor networks," in *Proc. of the 2nd IEEE International Workshop on Sensor Network Protocols and Applications*, 2003.

[6] J. Zhao and R. Govindan, "Understanding packet delivery performance in dense wireless sensor networks," in *Proc. of the 1st international conference on Embedded networked sensor systems (SenSys)*, 2003.

[7] J. Considine, F. Li, G. Kollios, and J. Byers, "Approximate aggregation techniques for sensor databases," in *Proc. of IEEE Int'l Conf. on Data Engineering (ICDE)*, 2004.

[8] A. Manjhi, S. Nath, and P. Gibbons, "Tributeries and deltas : Efficient and robust aggregation in sensor network streams," in *Proc. of ACM International Conference on Management of Data (SIGMOD)*, 2005.

[9] S. Nath, P. B. Gibbons, S. Seshan, and Z. Anderson, "Synopsis diffusion for robust aggregation in sensor networks," in *Proc. of the 2nd international conference on Embedded networked sensor systems (SenSys)*, 2004.

[10] P. Flajolet and G. N. Martin, "Probabilistic counting algorithms for data base applications," *Journal of Computer and System Sciences*, vol. 31, no. 2, pp. 182–209, 1985.

[11] B. Patt-Shamir, "A note on efficient aggregate queries in sensor networks." in *PODC*, 2004, pp. 283–289.

[12] M. B. Greenwald and S. Khanna, "Power-conservative computation of order-statistics over sensor networks," in *PODS*, 2004.

[13] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri, "Medians and beyond : New aggregation techniques for sensor networks," in *Proc. of the 2nd international conference on Embedded networked sensor systems (SenSys)*, 2004.

[14] Y. Yao and J. E. Gehrke, "The cougar approach to in-network query processing in sensor networks," *ACM SIGMOD Record*, vol. 31, no. 2, pp. 9–18, Sep. 2002.

[15] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar, "SPINS: Security protocols for sensor networks," in *Seventh Annual International Conference on Mobile Computing and Networks (MobiCOM)*, 2001.

[16] S. Zhu, S. Setia, and S. Jajodia, "LEAP: Efficient security mechanisms for large-scale distributed sensor networks," in *Proc. of the 10th ACM Conference on Computer and Communications Security (CCS '03)*, 2003.

[17] C. Karlof, N. Sastry, and D. Wagner, "Tinysec: a link layer security architecture for wireless sensor networks," in *Proc. of the 2nd international conference on Embedded networked sensor systems (SenSys)*, 2004, pp. 162–175.

[18] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar, "SPINS: Security protocols for sensor networks," *Wireless Networks*, vol. 8, no. 5, pp. 521–534, Sep. 2002.

[19] M. Garofalakis, J. M. Hellerstein, and P. Maniatis, "Proof sketches: Verifiable in-network aggregation," in *Proceedings of the 23rd International Conference on Data Engineering (ICDE '07)*.

[20] Y. Yang, X. Wang, S. Zhu, and G. Cao, "SDAP: A secure hop-by-hop data aggregation protocol for sensor networks," in *Proc. of ACM MOBIHOC*, 2006.

[21] A. Perrig, J. Stankovic, and D. Wagner, "Security in wireless sensor networks," in *Communications of the ACM*, vol. 47, no. 6, June 2004, pp. 53–57.

[22] J. Lopez and J. Z. (Eds.), *Wireless Sensor Network Security*. IOS Press., 2008.

[23] D. Adamy, *A First Course in Electronic Warfare*. Artech House Publishers, Norwood, MA, 2001.

[24] A. D. Wood, J. A. Stankovic, and S. H. Son, "Jam: A jammed-area mapping service for sensor networks," in *RTSS '03: Proceedings of the 24th IEEE International Real-Time Systems Symposium*. IEEE Computer Society, 2003, p. 286.

[25] C. Karlof and D. Wagner, "Secure routing in wireless sensor networks: Attacks and countermeasure," in *Ad-Hoc Networks*, vol. 1, no. 2-3. Elsevier, September 2003, pp. 293–315.

[26] Y.-C. Hu, A. Perrig, and D. Johnson, "Packet leashes: a defense against wormhole attacks in wireless networks," in *Proceedings of the INFOCOM 2003*, vol. 3, pp. 1976–1986.

[27] B. Parno, A. Perrig, and V. Gligor, "Distributed detection of node replication attacks in sensor networks," in *SP '05: Proceedings of the 2005 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2005, pp. 49–63.

[28] B. Zhu, V. G. K. Addada, S. Setia, S. Jajodia, and S. Roy, "Efficient distributed detection of node replication attacks in sensor networks," in *Proceedings of the 2007 ACSAC*, pp. 257–267.

[29] L. P. Cox, M. Castro, and A. Rowstron, "POS: practical order statistics for wireless sensor networks," in *Proceedings of the 26th International Conference on Distributed Computing Systems*, 2006.

[30] D. Wagner, "Resilient aggregation in sensor networks," in *Proc. of ACM Workshop on Security of Sensor and Adhoc Networks (SASN)*, 2004.

[31] L. Buttyan, P. Schaffer, and I. Vajda, "Resilient aggregation with attack detection in sensor networks," in *Proc. of 2nd IEEE Workshop on Sensor Networks and Systems for Pervasive Computing*, 2006.

[32] L. Buttyán, P. Schaffer, and I. Vajda, "Ranbar: Ransac-based resilient aggregation in sensor networks," in *SASN*, 2006, pp. 83–90.

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

[34] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," pp. 726–740, 1987.

[35] B. Przydatek, D. Song, and A. Perrig, "SIA: Secure information aggregation in sensor networks," in *Proc. of the 1st international conference on Embedded networked sensor systems (SenSys)*, 2003.

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

[37] H. Chan, A. Perrig, and D. Song, "Secure hierarchical in-network aggregation in sensor networks," in *Proceedings of ACM Conference on Computer and Communications Security (CCS)*, 2006.

[38] K. B. Frikken and J. A. Dougherty, "An efficient integrity-preserving scheme for sensor aggregation," in *Proc. of Wisec*, 2008.

[39] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-based aggregation in large dynamic networks," *ACM Transactions on Computer Systems*, vol. 23, no. 3, pp. 219–252, 2005.

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

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

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

[43] J. Domingo-Ferrer, "A provably secure additive and multiplicative privacy homomorphism," in *Proceedings of the 5th International Conference on Information Security (ISC '02)*. Springer-Verlag, 2002, pp. 471–483.

[44] F. Ye, H. Luo, S. Lu, and L. Zhang, "Statistical en-route filtering of injected false data in sensor networks," in *Proc. of IEEE Infocom*, 2004.

[45] W. Zhang and G. Cao, "Group rekeying for filtering false data in sensor networks: A predistribution and local collaboration-based approach," in *Proc. of IEEE Infocom*, 2005.

[46] S. Zhu, S. Setia, S. Jajodia, and P. Ning, "An interleaved hop-by-hop authentication scheme for filtering injected false data in sensor networks," in *Proc. of IEEE Symposium on Security and Privacy*, 2004.

[47] W. Du, J. Deng, Y. S. Han, and P. Varshney, "A pairwise key pre-distribution scheme for wireless sensor networks," in *Proc. of the 10th ACM Conference on Computer and Communications Security (CCS '03)*, 2003.

[48] Mica Motes, http://www.xbow.com.

[49] 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*, 1988, pp. 369–378.

[50] 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.

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

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

[53] 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*, vol. 59, no. 307, pp. 935–956, 1964.

# Curriculum Vitae

Sankardas Roy was born in India in 1974. He completed his Bachelor of Engineering in Electrical Engineering from the Bengal Engineering College, West Bengal, India, in 1997 and his Master of Technology in Computer Science from the Indian Statistical Institute, Kolkata, India, in 2001. During a brief period, he worked as an assistant professor at the Institute of Engineering and Management, Kolkata, India. In the fall of 2002, he began studies in the George Mason University, Fairfax, Virginia to pursue a PhD. His research interests include sensor network security, ad hoc network security, and network security in general.