SECURE DATA AGGREGATION SCHEME

FOR SENSOR NETWORKS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Kavit Shah

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Electrical and Electronics Engineering

December 2014

Purdue University

Indianapolis, Indiana

This is the dedication.

ACKNOWLEDGMENTS

This is the acknowledgments.

PREFACE

This is the preface.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

LIST OF FIGURES

SYMBOLS

s	Sensor node
N	Query nonce
H	Hash function
d	Distance
D	Data-item
X	Random variable
δ	Fanout of a sensor node
f	Function
v	Vertex
A	An attack
α	Resilient factor

ABBREVIATIONS

SHA	Secure hash algorithm
SHIA	Secure hierarchical in-network aggregation
SIA	Secure information aggregation
ACK	Positive acknowledgment message
NACK	Negative acknowledgment message
BER	Bit error rate

ABSTRACT

Shah, Kavit Master, Purdue University, December 2014. Secure data aggregation scheme for sensor networks. Major Professor: Dr. Brian King.

This is the abstract.

1. INTRODUCTION

2. SENSOR NETWORKS BACKGROUND

Sensor networks are becoming ubiquitous in our day to day life. They are known also known as Internet of Things. In the following sections we show the applications and the rise of sensor networks. Then we describe why doing security in the sensor network presents the number of unique challenges and solutions to tackle those challenges.

2.1 Sensor Networks and Applications

In sensor networks, thousands of sensor nodes may interact with the physical environment and collectively monitor an area, generating a large amount of data to be transmitted and reasoned about. With the recent advances in sensor network research, we can use tiny and cheap sensor nodes to obtain a lot of useful information about physical environment. For example, we can use them to discover temperature, humidity, water quality, lightning condition, pressure, noise level, carbon dioxide level, oxygen level, soil moisture, magnetic field, characteristics of objects such as speed, direction, and size, the presence or absence of certain kinds of objects, and all kinds of values about machinery like mechanical stress level or movement [1]. These versatile types of sensors, allow us to use sensor network in a wide variety of scenarios. For example, sensor networks are used in habitat and environment monitoring, health care, military surveillance, industrial machinery surveillance, home automation, scientific data collection, emergency fire alarm systems, traffic monitoring, wildfire tracking, wildlife monitoring and many other applications.

Military Application Sensor networks can be used for enemy tracking, battlefield surveillance or target classification [2]. For example, *Palo Alto Research Center* tries to spot "interesting" vehicles (the vehicles marked specially) using motes equipped with microphones or steerable cameras [3]. The objective is to coordinate a number

of this kind of sensors in order to keep sensing the track of a chosen moving object minimizing any information gaps about the track that may occur.

Environmental Monitoring For example, *Meteorology and Hydrology in Yosemite National Park* [4], a sensor network was deployed to monitor the water system across and within the Sierra Nevada, especially regarding natural climate fluctuation, global warming, and the growing needs of water consumers. Research of the water system in the Sierra Nevada is difficult, because of its geographical structure. And sensor networks can be real blessing in such situation as they can operate with little or no human intervention.

Health Care Sensors can be used to monitor the patients round the clock. The most important criteria for the such networks are security and reliability. *CodeBlue* [5] is a system to enhance emergency medical care with the goal to have a seamless patient transfer between a disaster area and a hospital.

Sustainable Mobility With the driver less cars from companies like Google, connected and coordinated vehicles seems the future of transportation. Autonomous vehicle systems [6] describes how various various technologies in addition to the sensor networks is used in making sustainable mobility.

The application of a sensor network usually determines the design of the sensor nodes and the design of the network itself. As far as we know, there is no general architecture for sensor networks. Therefore, developing a protocol for sensor networks can certainly be challenging.

2.2 Energy Consumption

The sensor network's lifetime can be maximized by minimizing the power consumption of the sensor node's radio module. To estimate the power consumption, we have to consider the communication and computation power consumption at each sensor node. The radio module energy dissipation depends on two main parameters [7]. The first is $E_{elec}(J/b)$, the energy dissipated to run the transmit or receive electronics.

The second is $\varepsilon_{amp}(J/b \cdot m^2)$, the energy dissipated by the transmit power amplifier to achieve an acceptable noise ratio E_b/N_o at the receiver. We assume the d^2 energy loss for transmission between sensor nodes since the distances between sensors are relatively short [8]. To transmit a k - bit packet at distance d, the energy dissipated is:

$$E_{tx}(k, d) = E_{elec} \cdot k + \varepsilon_{amp} \cdot k \cdot d^2 \tag{2.1}$$

and to receive the k - bit packet, the radio expends

$$E_{rx}(k) = E_{elec} \cdot k \tag{2.2}$$

For μAmp wireless sensor, $E_{elec} = 50nJ/b$ and $\varepsilon_{amp} = 100pJ/b \cdot m^2$.

To sustain the sensor network for longer time all aspects of the sensor network should be efficient. For example, the networking algorithm for routing should be such that it minimizes the distance d between the nodes. The signal processing algorithm should be such that it process the networking packets with less computations. It is shown in [7] that by distributing the Fast Fourier Transform (FFT) algorithm in the network requires less communication between the sensor nodes. Also, for minimal energy dissipation, a processor should operate at the lowest voltage for a given clock frequency.

2.3 Data Aggregation

The sensor nodes in the network often have limited resources, such as computation power, memory, storage, communication capacity and most significantly, battery power. Furthermore, data communications between nodes consume a large portion of the total energy consumption. The in-network data aggregation reduces the energy consumption by aggregating the data before sending it to the parent node in the network which reduces the communications between nodes. For example, in-network data aggregation of the SUM function can be performed as follows. Each intermediate sensor node in the network forwards a single sensor reading containing the sum of all

the sensor readings from all of its descendants, rather than forwarding each descendants' sensor reading one at a time to the base station. It is shown that the energy savings achieved by in-network data aggregation are significant [9]. The in-network data aggregation approach requires the sensor nodes to do more computations. But studies have shown that transmitting the data requires more energy than computing the data. Hence, in-network data aggregation is an efficient and a widely used approach for saving bandwidth by doing less communications between sensor nodes and ultimately giving longer battery life to sensor nodes in the network.

We define the following terms to help us define the goals of in-network data aggregation approach.

Definition 2.3.1 *[10]* ***Payload*** *is the part of the transmitted data which is the fundamental purpose of the transmission, to the exclusion of information sent with it such as meta data solely to facilitate the delivery.*

Definition 2.3.2 ***Information rate*** *for a given node is the ratio of the payloads, number of payloads sent divided by the number of payloads received.*

The goal of the aggregation process is to achieve the lowest possible information-rate. In the following sections, we show that lowering information-rate makes the intermediate sensor nodes (aggregator) more powerful. Also, it makes aggregated payload more fragile and vulnerable to various security attacks.

2.4 Bandwidth Analysis

Congestion is a widely used parameter while doing bandwidth analysis of networking applications. The congestion for any given node is defined as follows:

$$Node\ congestion = Edge\ congestion \cdot Fanout \tag{2.3}$$

Congestion is a useful factor while analyzing sensor network as it measures how quickly the sensor nodes will exhaust their batteries [11]. Some nodes in the sensor

network have more congestion than the others, the highly congested nodes are the most important to the the network connectivity. For example, the nodes closer to the base station are essential for the network connectivity. The failure of the highly congested nodes may cause the sensor network to fail even though most of the nodes in the network are alive. Hence, it is desirable to have a lower congestion on the highly congested nodes even though it costs more congestion within the overall sensor network.

To distribute the congestion uniformly across the network, we can construct an aggregation protocol where each node transmits a single payload Defined in 2.3.1 to its parent in the aggregation tree. In this approach, δ is dependent on the given aggregation tree. For example, an aggregation tree shown in Figure 2.1, with n nodes,

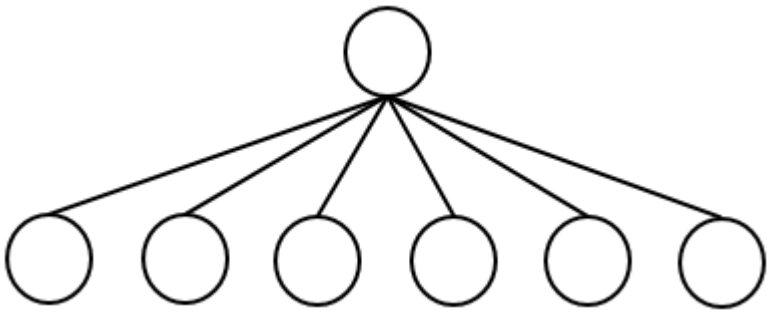

Fig. 2.1.: Star Network

organized in the star tree topology congestion is $O(n)$. And the network organized in the palm tree topology as shown in Figure 2.2 the congestion is $O(1)$. This approach can create some highly congested nodes in the aggregation tree which is undesirable. In most of the real world applications we cannot control δ as the aggregation tree is random. Hence, it is desirable to have uniform distribution of congestion across the aggregation tree.

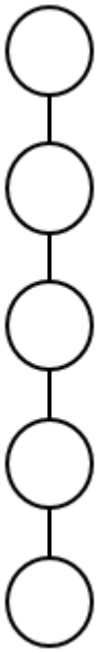

Fig. 2.2.: Palm Tree Network

2.5 Security in Sensor Networks

Here, we depict, what parameters should be taken into account while designing secure protocol for sensor networks. And how those parameters constrains protocol designer's choices of the protocol. Then we depict how data-aggregation can help overcome some of those limitation and some examples where security is ignored by making some strong network assumptions.

2.5.1 Physical Limitations

Sensor nodes are often deployed in open, hostile and unattended environments, so they are vulnerable to physical tampering due to the lower physical security. An adversary can obtain the confidential information from a compromised sensor and reprogram it with malicious software. The compromised node may then report an arbitrary false sensor readings to its parent node in the tree hierarchy, causing the

final aggregation result to far deviate from the true aggregate result. This attack becomes more damaging when multiple adversaries succeeds in injecting false data into the network which may cause catastrophic consequences [12].

2.5.2 Hardware Limitations

As far as we know, one of the first hardware platform for developing sensor network application is MICA [13] developed by University of Berkeley. Another popular platform is Mote from Intel [14]. Due to lower manufacturing cost of sensor nodes, they have low speed processor, limited storage, a short range trans receiver. For example, the major specifications for the latest ZigBee chip supporting IEEE 802.15.4 standard, CC2538 from TexasInstruments are shown in Table 6.1. This chip can do most of the security algorithms but has really little amount of memory storage. It has limited output power which constraints its transmission range which forces us to use multi-hop routing in the network as one node can not communicate with the node outside of its transmission range. This hardware limitations constrains protocol designer's choice of algorithms for applications.

2.5.3 Transmission Medium

In sensor networks, a group of sensor nodes (or processors) communicate over the radio (e.g., Bluetooth, WLAN). Traditionally, wireless medium has the issues of in synchronization, hidden station and expose station terminal problems, distributed arbitration, directional antennas, bandwidth limitations, higher error rate, security, scalability etcetera. For example, wireless network has approximately 10^6 times higher bit error rate (BER) than wired network which causes frequent link loss and then path loss. Hence, making unstable routing in the network. Higher BER creates higher collision rate in the network, generating higher overhead of retransmission and lowering the channel utilization and the throughput of the network. This kind of

	CC2538
Device Type	Wireless Micro controller
Frequency	2.4GHz
Processor Integration	ARM Cortex-M3
Flash	Up to 512 KB
RAM	Up to 32 KB
Security	AES-128/256;ECC-128/256; SHA2; RSA
RX Current	20 (mA)
Output Power	7 (dBm)
Data Rate(Max)	250 kbps
Type of Battery	AAA; AA; Rechargable(Li-ION)

Table 2.1.: System-on-Chip specifications for IEEE 802.15.4 from TexasInstruments

transmission medium with constrained resources makes it challenging to design the secure data-aggregation protocol for sensor networks.

2.5.4 Mobility

As we know, sensor nodes communicate via radio and the availability of the transmission medium changes over time due to link failure, bandwidth limitations or change in network topology. Nodes may be able to move, which further contributes to the instability of the communication link. The mobility issue makes difficult to do the routing in the network with the directional antennas in place. It also requires network to be agile enough to do the reconfiguration for the newer network topology. It impedes whiled doing the quality of service in the network. All these parameters combined contributes to making strong assumptions on the network topology while designing the secure data aggregation protocol for sensor networks.

2.5.5 In-network Data Aggregation

Designing secure data aggregation protocol for the wireless sensor network poses a numerous challenges due to resource limitations and inherent characteristics discussed in the previous subsections. One widely used approach to overcome the bandwidth limitation is to use in-network data aggregation. In-network data aggregation approach saves bandwidth by transmitting less payloads between sensor nodes. But that empowers an intermediate aggregate sensor nodes in the network by allowing it to control certain part of the network. For example, a malicious intermediate sensor node who does aggregation over all of its descendants payloads, needs to tamper with only one aggregated payload instead of tampering with all the payloads received from its descendants. Thus, a malicious intermediate sensor node needs to do less work to skew the final aggregated payload. An adversary controlling few sensor nodes in the network can cause the network to return unpredictable payloads, making an entire sensor network unreliable. Notice that the more descendants an intermediate sensor node has the more powerful it becomes.

Despite the fact that in-network aggregation makes an intermediate sensor nodes more powerful, some aggregation approaches requires strong network topology assumptions or honest behaviors from the sensor nodes. For example, in-network aggregation schemes in [11,15] assumes that all the sensor nodes in the network are honest. Secure Information Aggregation (SIA) of [16], provides security for the network topology with a single-aggregator model. Secure hierarchical in-network aggregation (SHIA) in sensor networks [17] presents the first and provably secure sensor network data aggregation protocol for general networks and multiple adversaries. We discuss the details of the protocol in the next chapter. SHIA limits the adversary's ability to tamper with the aggregation result with the tightest bound possible but it does not help detecting an adversary in the network. Also, we claim that same upper bound can be achieved with compact label format defined in the next chapter.

3. NETWORKING AND CRYPTOGRAPHY TOOLS

Here we describe the networking and cryptographic tools used in this protocol referred from [18, 19]. For example, we describe the Hash function, its properties and how it helps us achieve data integrity.

3.1 Encryption

3.2 Hash Function

A hash function takes a message as its input and outputs a fixed length message called hash code. The hash code represents a compact image of the message like a digital fingerprint. Hash functions are used to achieve data integrity.

A hash function h should have the following properties:

- Compression - h maps an input x of arbitrary finite bitlength, to an output $h(x)$ of fixed bitlength n.
- Ease of computation - given h, x it is easy to compute $h(x)$.
- Preimage resistance - for all pre-specified outputs, it is computationally infeasible to find any input which hashes to that output, i.e., to find any preimage x' such that $h(x') = y$ when given y for which a corresponding input is not known.
- 2nd-preimage resistance - it is computationally infeasible to find any second input which has the same output as any specified input, i.e, given x, to find a 2nd-preimage $x' \neq x$ such that $h(x') = h(x)$.
- Collision resistance - it is computationally to find any two distinct inputs x, x' which hash to the same output, i.e., such that $h(x) = h(x')$.

We use SHA-256 [20] hash algorithm as a collision resistant hash function.

3.3 Digital Signatures

A digital signature is a mathematical scheme for demonstrating the authenticity of a digital message. A valid digital signature gives a recipient reason to believe that the message was created by a known sender, such that the sender cannot deny having sent the message (authentication and non-repudiation) and that the message was not altered in transit (integrity). A Digital Signature scheme consists of the following:

1. a plain text message space $\mathcal{M}$ (set of strings over alphabets)
2. a signature space $\mathcal{S}$ (set of possible signatures)
3. a signing key space $\mathcal{K}$ (set of possible keys for signature generation) and a verification space $\mathcal{K}'$ (a set of possible verification keys)
4. an efficient key generation algorithm $\mathsf{Gen} : N \rightarrow \mathcal{K} \times \mathcal{K}'$
5. an efficient signing algorithm $\mathsf{Sign} : \mathcal{M} \times \mathcal{K} \rightarrow \mathcal{S}$
6. an efficient verification algorithm $\mathsf{Verify} : \mathcal{S} \times \mathcal{M} \rightarrow \{\text{true, false}\}$

For any secret key $s_k \in \mathcal{K}$ and any $m \in \mathcal{M}$, the message m is signed using key s_k as follows:

$$s = \mathsf{Sign}_{s_k}(m) \tag{3.1}$$

For any s_k let p_k denote public key and for all $m \in \mathcal{M}$ and $s \in \mathcal{S}$, s as follows:

$$\mathsf{Verify}_{p_k}(m, s) = \begin{cases} \textbf{true} \text{ with probability of 1} & if\ s = \mathsf{Sign}_{s_k}(m) \\ \textbf{false} \text{ with overwhelming probability} & if\ s \neq \mathsf{Sign}_{s_k}(m) \end{cases} \tag{3.2}$$

where the probability space is determine by the $\mathcal{M}, \mathcal{S}, \mathcal{K}, \mathcal{K}'$ and perhaps the signing and verification algorithms. The "overwhelming probability" for the signature scheme determines the probability that the scheme allows for a forgery. Note that the Digital Signature scheme satisfies the following requirements:

- Only the owner of the secret key can generate a valid signature.
- The digital signature is easily verified by other parties.
- The digital signature is not only tied to the signer but also to the message that is being signed.
- Digital signatures cannot be separated from the message and attached to another message.
- Digital signatures do not encrypt the message. However, if necessary, a signed message can be encrypted after it is signed.

The Figure 3.1 from [21], shows the steps for signing and verifying the hashed message. The message is hashed before its being signed to reduce the message size. If the

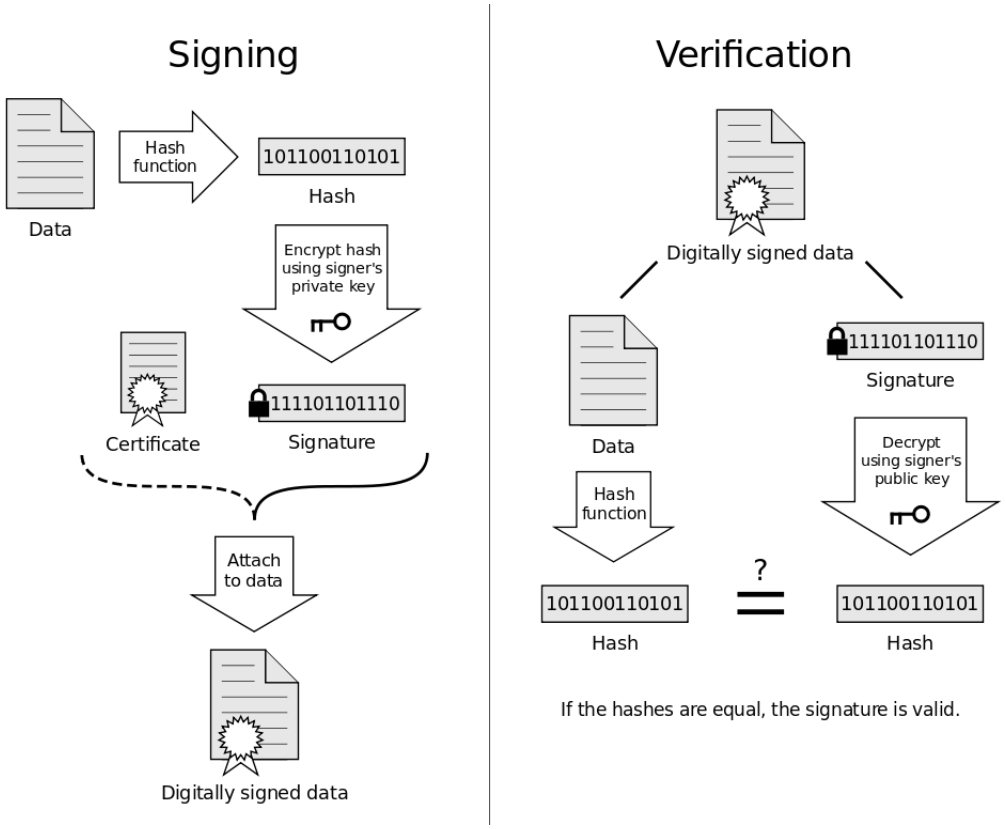

Fig. 3.1.: Signing and verification of digital signatures

message is not hashed before signing then the signature can be longer than the message which is problematic for the longer messages. **Talk about non-repudiation, authenticity, integrity. Talk about similarities and differences between physical world signature non-repudiation, authenticity, integrity.** In physical world, your signatures are the same for all the messages. But this can not be possibly true for digital signatures as the attacker can obtain one signature and then use the same signature to pretend some one else.

3.4 Tree generation algorithms

3.5 XOR function

3.6 Elliptic curve

4. DATA AGGREGATION BACKGROUND

One of the fundamental and indispensable functionalities of sensor networks is the ability to answer queries over the data acquired by the sensors. For example, in Figure 4.1 the base station might be at the end of the river who initiates the query to the sensor network. And all the sensor nodes on both sides of the river has to response to the queries of the base station.

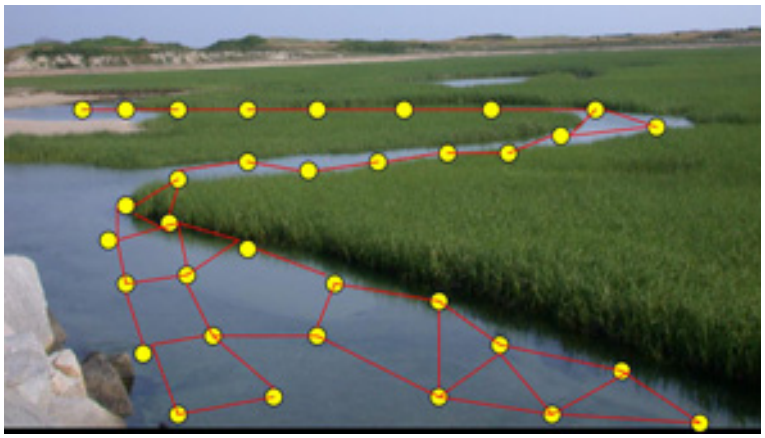

Fig. 4.1.: Routing River [22]

The resource constraints and security issues make designing mechanisms for information aggregation in large sensor networks particularly challenging.

Literature survey on data aggregation

Read SIA; you can take inspiration from it for SIA

talk about compression

lossy compression

lossless compression

data aggregation is lossy compression as there is no way the base station can know the original sensor readings

Hence, confidence in data is so important.

then talk about SHIA

5. DATA AGGREGATION WITHOUT INTERNAL VERIFICATION

Our work enhances Secure Hierarchical In-Network Data Aggregation (SHIA) protocol of [17] by making it communication efficient, adding new capabilities to the protocol, achieving similar security goals with non-resilient aggregation functions and efficient way of analyzing the protocol. In this chapter, we summarize the important parts of SHIA protocol and relevant terms, to build the foundation to describe our protocol in the following chapters.

SHIA computes the aggregate functions (such as *truncated SUM*, *AVERAGE*, *COUNT*, $\Phi - QUANTILE$) of the sensor readings. For example, a network with 5 nodes having the following sensor readings $10, 11, 8, 15, 12$ has the overall all sum of 56, the average of 11.2 and the median ($\Phi = 0.5$) of 11. The goal of the SHIA is to compute the aggregate functions while assuming that partially a network is controlled by an adversary which is attempting to skew the final aggregated result.

5.1 Network Assumptions

We assume a multi hop network with a set $S = \{s1, ..., s_n\}$ of n sensor nodes where all n nodes are alive and reachable. The network is organized in a rooted tree topology. The trusted base station resides outside of the network and has more computation, storage capacity then the sensor nodes in the network. **Note**: SHIA names the base station as the querier and the root of the tree as the base station. The base station knows total number of sensor nodes n in the network and the network topology. It also has the capacity to directly communicate with every sensor node in the network using authenticated broadcast. All the wireless communications between the nodes is peer-to-peer and we do not consider the local wireless broadcast.

Each sensor node has a unique identifier s and shares a unique secret symmetric key K_s with the base station. The keys enable message authentication, and encryption if the data confidentiality is required. All the sensor nodes are capable of doing symmetric key encryption and symmetric key decryption. They are also capable of computing collision resistant cryptographic hash function H (SHA-256) [20].

5.2 Attacker Model

We consider a model with a polynomially bounded adversary of [16], which has a complete control over some of the sensor nodes in the network. Most significantly, the adversary can change the measured values reported by sensor nodes under its control. An adversary can perform a wide variety of attacks. For example, an adversary could report fictitious values (probably completely independent of the measured reported values), instead of the real aggregate values and the base station receives the fictitious aggregated information. Since in many applications the information received by the base station provides a basis for critical decisions, false information could have ruinous consequences. However, we do not want to limit ourselves to just a few specific selected adversarial models. Instead, we assume that the adversary can misbehave in any arbitrary way, and the only limitations we put on the adversary are its computational resources (polynomial in terms of the security parameter) and the fraction of nodes that it can have control over. We focus on **stealthy attacks** [16], where the goal of an adversary is to make the base station accept false aggregation results, which are significantly different from the true results determined by the measured values, while not being detected by the base station. In this setting, denial-of-service (DoS) attacks such as not responding to the queries or always responding with negative acknowledgment at the end of verification phase clearly indicates to the base station that something is wrong in the network and therefore is not a stealthy attack. One of the security goals of SHIA is to detect the stealthy attacks. And one of the

security goals of our work is to detect an adversary who caused the stealthy attacks and remove it from the network to prevent these attacks in the future.

5.3 Resilient and Non-Resilient Aggregate Functions

Wagner [23] uses statistical estimation theory to quantify the effects of direct data injection on various aggregates functions. To understand Wagner's work we define following terms.

Definition 5.3.1 *According to [17], each sensor node s_i has a data value a_i assuming that all the data values are non-negative bounded by real value $a_i \in [0, r]$, where r is the maximum allowed data value. The objective of the* ***aggregation process*** *is to compute some function f over all the data values, i.e., $f(a_1, \ldots, a_n)$.*

Definition 5.3.2 *According to [17], a* ***direct data injection*** *attack occurs when an adversary modifies the data readings reported by the nodes under its direct control, under the constraint that only legal readings in $[0, r]$ are reported.*

An **estimator** is an algorithm $f : \mathbb{R}^n \to \mathbb{R}$ where $f(x_1, \ldots, x_n)$ is intended as an estimate of some real valued function of θ. We assume that θ is real valued and that we wish to estimate θ itself. Next, we define $\widehat{\Theta} := f(X_1, \ldots, X_n)$, where $X_1, \ldots, X_n$ are n random variables. We can define the root-mean-square(r.m.s) error (at θ):

$$rms(f) := \mathbb{E}[(\widehat{\Theta} - \theta)^2 | \theta]^{1/2} \tag{5.1}$$

Wagner in [23] defines **resilient estimators and resilient aggregation** as follows. A k-node attack A is an algorithm that is allowed to change up to k of the values $X_1, \ldots, X_n$ before the estimator is applied. In particular, the attack A is specified by a function $\tau_A : \mathbb{R}^n \to \mathbb{R}^n$ with the property that the vectors x and $\tau_A(x)$ never differ at more than k positions. We can define the root mean square(r.m.s) error associated with A by

$$rms^*(f, A) := \mathbb{E}[(\widehat{\Theta}^* - \theta)^2 | \theta]^{1/2} \tag{5.2}$$

where $\widehat{\Theta}^* := f(\tau_A(X_1, \ldots, X_n))$. To explain, $\widehat{\Theta}^*$ is a random variable that represents the aggregate calculated at the base station in the presence of the k-node attack A, and $rms^*(f, A)$ is a measure of inaccuracy of the aggregate after A's intrusion. If $rms^*(f, A) >> rms(f)$, then the attack has succeeded in noticeably affecting the operation of the sensor network. If $rms^*(f, A) \approx rms(f)$, the attack had little or no effect. We define

$$rms^*(f, k) := max\{rms^*(f, A) : A \; is \; a \; k - node \; attack\} \tag{5.3}$$

so that $rms^*(f, k)$ denotes the r.m.s. error of the most powerful k-node attack possible. Note that $rms^*(f, 0) = rms(f)$. We think of an aggregation function f as an instance of the resilient aggregation paradigm if $rms^*(f, k)$ grows slowly as a function of k.

Definition 5.3.3 *According to [23], an aggregation function f is (k, α)-**resilient** (with respect to a parameterized distribution $p(X_i|\theta)$) if $rms^*(f, k) \leq \alpha \cdot rms(f)$ for the estimator f.*

The intuition is that the (k, α)-resilient functions, for small values of α, are the ones that can be computed meaningfully and securely in the presence of up to k compromised or malicious nodes. His work is summarized in the Table 5.1. According to this quantitative study measuring the effects of direct data injection on various aggregates, concludes that the **aggregates (truncated SUM and AVERAGE) can be resilient** under such attacks.

5.4 Security Goal

Without precise knowledge of application, the direct data injection attacks are indistinguishable from the malicious sensor readings. The goal of SHIA is to design an **optimally secure** aggregation algorithm with only **sublinear edge congestion**.

Definition 5.4.1 *According to [17], an aggregation algorithm is **optimally secure** if, by tampering with the aggregation process, an adversary is unable to induce the*

Aggregate(f)	Security Level
minimum	insecure
maximum	insecure
sum	insecure
average	insecure
count	acceptable
$[l, u]$-truncated average	problematic
5% -trimmed average	better
median	much better

Table 5.1.: Summary of Wagner's work

base station to accept any aggregation result which is not already achievable by direct data injection.

5.5 The SUM Aggregate Algorithm

In this algorithm, the aggregate function f is summation meaning that we want to compute $a_1 + a_2 + \ldots + a_n$, where a_i is the sensed data value of the node i. This algorithm has three main phases:

- Query dissemination - initiates the aggregation process
- Aggregate commit - initiates the commitment tree generation process
- Result checking - initiates the distributed, interactive verification process

5.6 Query dissemination

Prior to this phase, an aggregation trees is created using a tree generation algorithm. We can use any tree generation algorithm as this protocol works on any

aggregation tree structure. For completeness of this protocol, one can use Tiny Aggregation Service (TaG) [9]. TaG uses broadcast message from the base station to initiate a tree generation. Each node selects its parent from whichever node it first receives the tree formation message. One possible aggregation tree for given network graph in Figure 5.1 is shown in Figure 5.2.

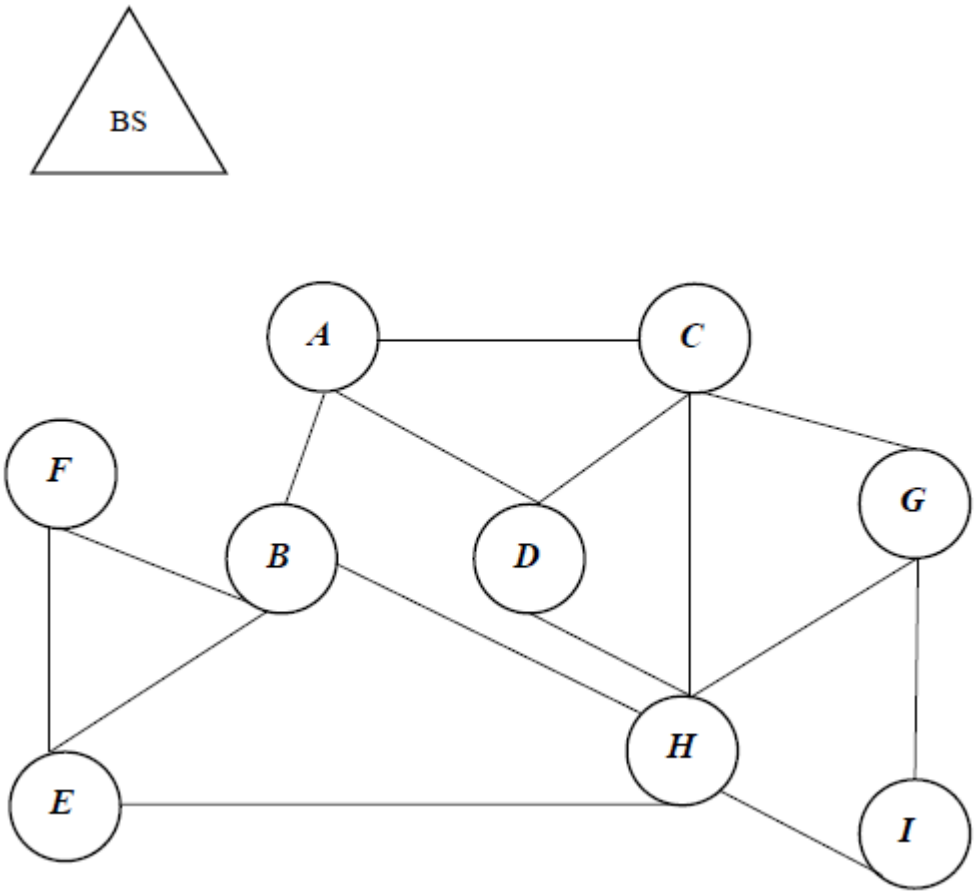

Fig. 5.1.: Network graph

To initiate the query dissemination phase, the base station broadcasts the query request message with the query nonce N in the aggregation tree. The query request message contains new query nonce N for each query to prevent replay attacks in the network. It is very important that the same nonce is never re-used by the base station. $SHIA$ uses **hash chain** to generate new nonce for each query. A hash

chain is constructed by repeatedly evaluating a pre-image resistant hash function h on some initial random value, the final value (or "anchor value") is preloaded on the nodes in the network. The base station uses the pre-image of the last used value as the nonce for the next broadcast. For example, if the last known value of the hash chain is $h^i(X)$, then the next broadcast uses $h^{i-1}(X)$ as the nonce; X is the initial random value. When a node receives a new nonce $N^{'}$, it verifies that $N^{'}$ is a precursor to the most recently received (and authenticated) nonce N on the hash chain, i.e., $h^i(N^{'}) = N$ for some i bounded by a fixed k of number of hash applications. A hash chain prevents an adversary from predicting the query nonce for future queries as it has to reverse the hash chain computation to get an acceptable pre-image.

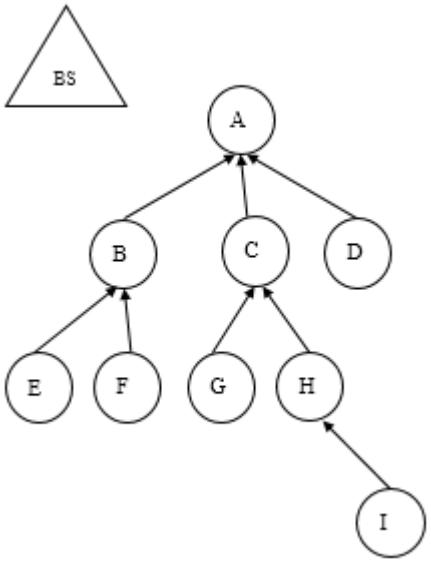

Fig. 5.2.: Aggregation tree for network graph in Figure 5.1

5.7 Aggregate commit

The aggregate commit phase constructs cryptographic commitments to the data values and to the intermediate in-network aggregation operations. These commitments are then passed on to the base station by the root of an aggregation tree.

The base station then rebroadcasts the commitments to the sensor network using an authenticated broadcast so that the rest of the sensor nodes in the network can verify that their respective data values have been incorporated into the final aggregate value.

5.7.1 Aggregate commit: Naive Approach

In the naive approach, during aggregation process each sensor node computes a cryptographic hash of all its inputs (including its own data value). The aggregation result along with the hash value called a label, is then passed on to the parent in the aggregation tree. The label is format is described in Definition 5.7.1. Figure 5.3 shows a commitment tree for the aggregation tree shown in Figure 5.2. Conceptually, a commitment tree is a logical tree built on top of an aggregation tree, with additional aggregate information attached to the nodes to help in the result checking phase.

Definition 5.7.1 *[17] A commitment tree is a tree where each vertex has an associated label representing the data that is passed on to its parent. The labels have the following format:*

<count, value, complement, commitment>

Where count is the number of leaf vertices in the subtree rooted at this vertex; value is the SUM aggregate computed over all the leaves in the subtree; complement is the aggregate over the COMPLEMENT of the data values; and commitment is a cryptographic commitment.

There is one leaf vertex u_s for each sensor node s, which we call the leaf vertex of s. The label of u_s consists of count = 1, value = a_s where a_s is the data value of s, complement = $r - a_s$ where r is the upper bound on allowable data values, and commitment is the nodes unique ID.

Internal vertices represent aggregation operations, and have labels that are defined based on their children. Suppose an internal vertex has child vertices with the following labels: u_1, u_2, ... , u_q, where $u_i = < c_i, v_i, \bar{v_i}, h_i >$. Then the vertex has label $<c,$

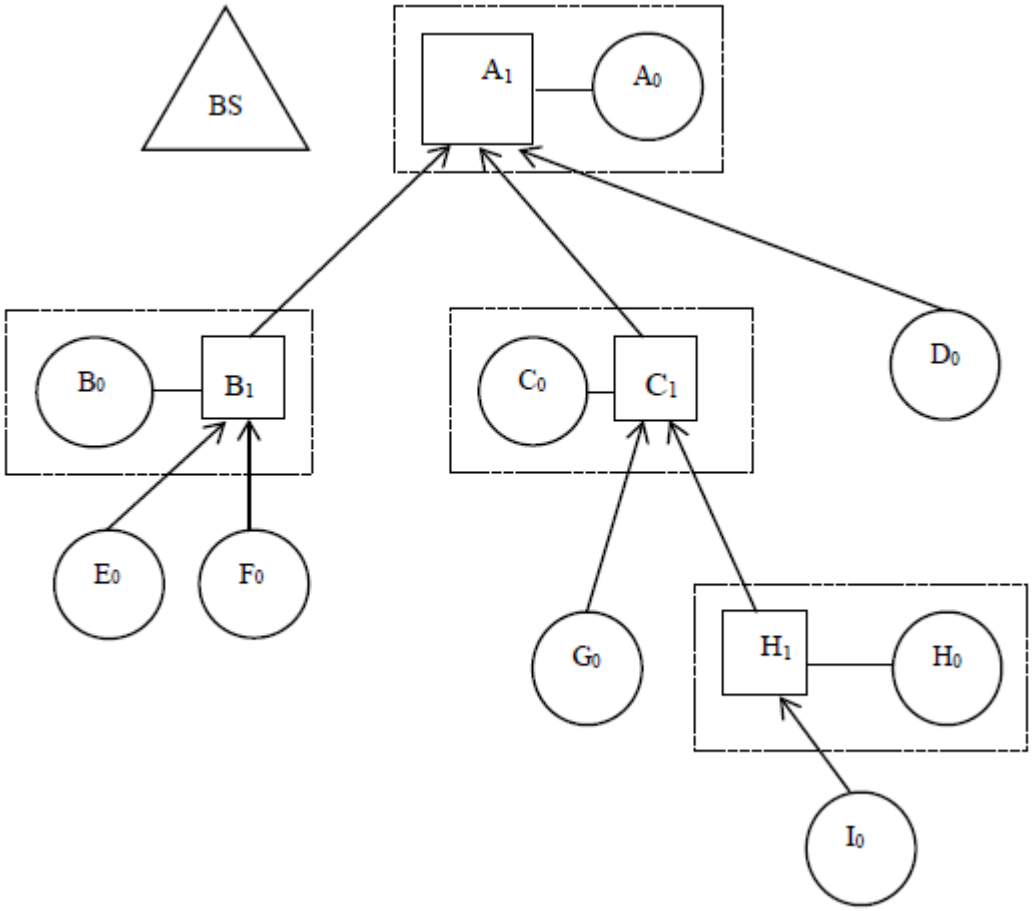

Fig. 5.3.: Naive commitment tree for Figure 5.2. For each sensor node s, s_0 is its leaf vertex, while s_1 is the internal vertex representing the aggregate computation at s (if any). The labels of the vertices on the path of node I to the root are shown from Equation 5.4 to 5.8.

v, $\bar{v}$, $h>$, with $c = \sum c_i$, $v = \sum v_i$, $\bar{v} = \sum \bar{v_i}$ and $h = H[N||c||v||\bar{v}||u_1||u_2||\ldots||u_q]$. The labels of the vertices of the commitment tree of Figure 5.3 are shown below.

$$I_0 =< 1, a_I, r - a_I, I > \tag{5.4}$$

$$H_1 =< 2, v_{H_1}, r - v_{H_1}, H[N||2||v_{H_1}||\bar{v_{H_1}}||H_0||I_0] > \tag{5.5}$$

$$B_1 =< 3, v_{B_1}, r - v_{B_1}, H[N||3||v_{B_1}||\bar{v_{B_1}}||B_0||E_0||F_0] > \tag{5.6}$$

$$C_1 =< 4, v_{C_1}, r - v_{C_1}, H[N||4||v_{C_1}||\bar{v_{C_1}}||C_0||G_0||H_1] > \tag{5.7}$$

$$A_1 =< 9, v_{A_1}, r - v_{A_1}, H[N||9||v_{A_1}||v_{A_1}^{-}||A_0||D_0||B_1||C_1] > \quad (5.8)$$

The word vertices is used for the nodes in the commitment tree and the word node is used for the nodes in the aggregation tree. There is a mapping between the nodes in the aggregation tree and the vertices in the commitment tree, a vertex is a logical element in the commitment tree where as the node is a physical sensor node which does all the communications. The collision resistant hash function makes it impossible for an adversary to change the commitment structure once it is sent to the base station. Our payload format is compact than the label format which is discussed in the next chapter.

5.7.2 Aggregate commit: Improved approach

The aggregation tree is a subgraph of the network graph so it may be randomly unbalanced. This approach tries to separate the structure of the commitment tree from the structure of the aggregation tree. So, the commitment tree can be perfectly balanced.

In the naive approach, each sensor node always computes the aggregate sum of all its inputs which is a greedy approach. SHIA uses delayed aggregation approach, which performs an aggregation operation if and only if it results in a complete, binary commitment tree.

We now describe SHIA's delayed aggregation algorithm for producing balanced commitment trees. In the naive commitment tree, each sensor node passes to its parent a single message containing the label of the root vertex of its commitment subtree T_s. In the delayed aggregation algorithm, each sensor node passes on the labels of the root vertices of a set of commitment subtrees $F = \{T_1, \ldots, T_q\}$. We call this set a commitment forest, and we enforce the condition that the trees in the forest must be complete binary trees, and no two trees have the same height. These constraints are enforced by continually combining equal-height trees into complete binary trees of greater height.

Definition 5.7.2 *[17] A commitment forest is a set of complete binary commitment trees such that there is at most one commitment tree of any given height.*

The commitment forest is built as follows. Leaf sensor nodes in the aggregation tree originate a single-vertex commitment forest, which they then communicate to their parent sensor nodes. Each internal sensor node s originates a similar single-vertex commitment forest. In addition, s also receives commitment forests from each of its children. Sensor node s keeps track of which root vertices were received from which of its children. It then combines all the forests to form a new forest as follows. Suppose s wishes to combine q commitment forests $F_1,\ldots,F_q$. Note that since all commitment trees are complete binary trees, tree heights can be determined by inspecting the count field of the root vertex. We let the intermediate result be $F = F_1 \cup \ldots \cup F_q$, and repeat the following until no two trees are the same height in F. Let h be the smallest height such that more than one tree in F has height h. Find two commitment trees T_1 and T_2 of height h in F, and merge them into a tree of height $h+1$ by creating a new vertex that is the parent of both the roots of T_1 and T_2 according to the inductive rule in Definition 5.7.1.

Example 5.7.1 *The commitment-forest generation process for node A of Figure 5.2 is shown here.*

$$\begin{aligned}
A_0 &=< 1, a_A, r - a_A, A > \\
D_0 &=< 1, a_D, r - a_D, D > \\
E_0 &=< 1, a_E, r - a_E, E > \\
B_1 &=< 2, v_{B_1}, v_{\bar{B_1}}, H(N||2||v_{B_1}||v_{\bar{B_1}}||B_0||F_0) > \\
C_2 &=< 4, v_{C_2}, v_{\bar{C_2}}, H(N||4||v_{C_2}||v_{\bar{C_2}}||H_1||C_1) >
\end{aligned} \tag{5.9}$$

$$\begin{aligned}
A_1 &=< 2, v_{A_1}, v_{\bar{A_1}}, H(N||2||v_{A_1}||v_{\bar{A_1}}||A_0||D_0) > \\
v_{A_1} &= a_A + a_D; v_{\bar{A_1}} = r - a_A + r - a_D
\end{aligned} \tag{5.10}$$

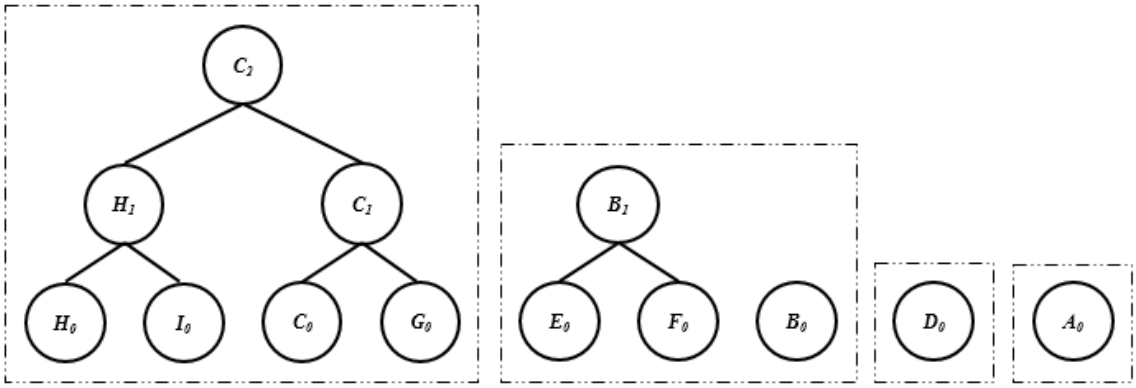

Fig. 5.4.: A receives C_2 from C, (B_1, B_0) from B, D_0 from D and generates A_0. The commitment forest received from a given sensor node is indicated by dashed-line box.

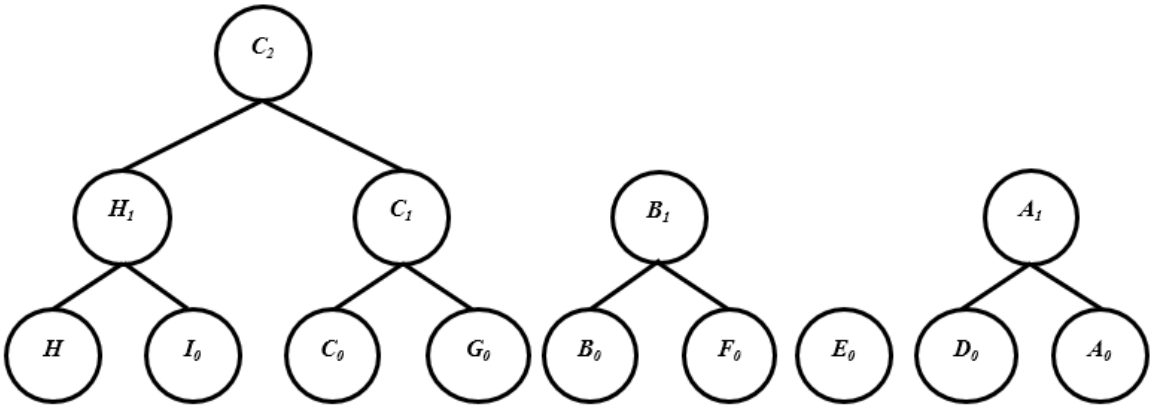

Fig. 5.5.: First Merge: A_1 vertex created by A.

$$A_2 =< 4, v_{A_2}, v_{\bar{A}_2}, H(N||4||v_{A_2}||v_{\bar{A}_2}||B_1||A_1) > \\ v_{A_2} = v_{A_1} + v_{B_1}; v_{\bar{A}_2} = r - v_{A_1} + r - v_{B_1} \tag{5.11}$$

$$A_3 =< 8, v_{A_3}, v_{\bar{A}_3}, H(N||8||v_{A_3}||v_{\bar{A}_3}||C_2||A_2) > \\ v_{A_3} = v_{A_2} + v_{C_2}; v_{\bar{A}_3} = r - v_{A_2} + r - v_{C_2} \tag{5.12}$$

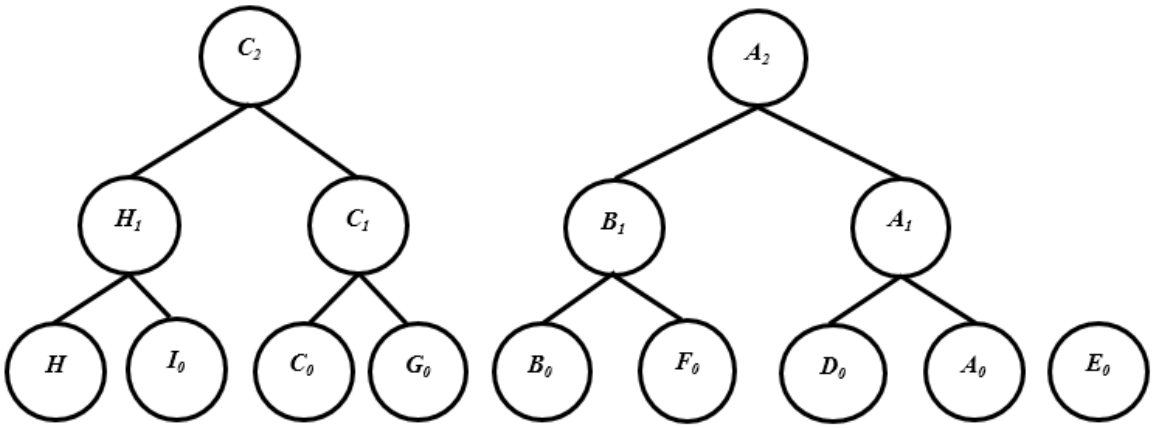

Fig. 5.6.: Second Merge: A_2 vertex created by A.

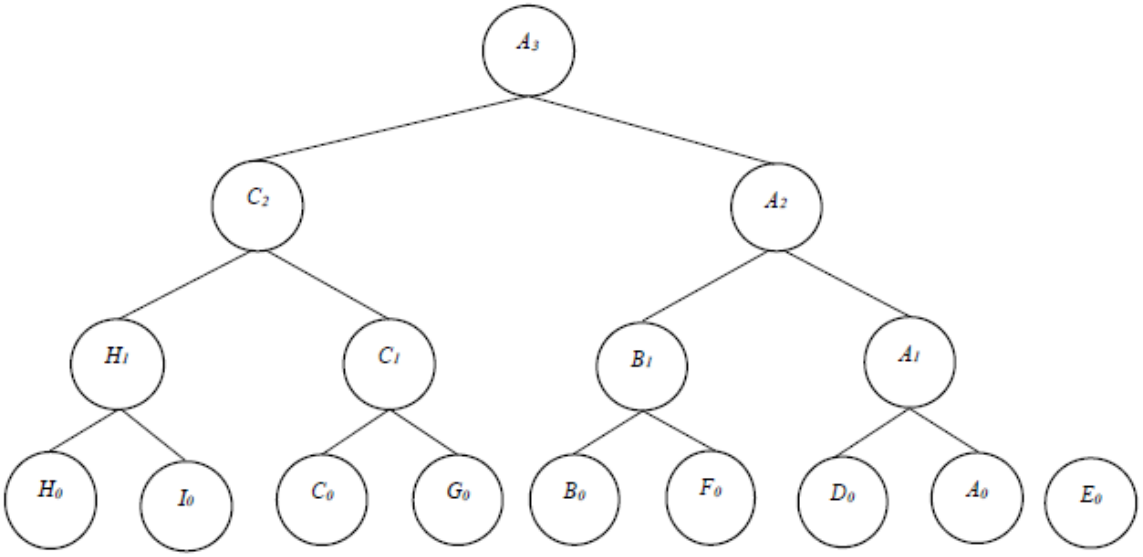

Fig. 5.7.: Third Merge: A_3 vertex created by A.

5.8 Result checking

SHIA presents novel distributed verification algorithm achieving provably optimal security while maintaining sublinear edge congestion. In our work, we take similar approach and add new capabilities to help find an adversary. Here, we describe the SHIA's result checking phase to build the basis for our work. The purpose of the result checking phase is to enable each sensor node s to independently verify that its data

value as was added into the SUM aggregate, and the complement $(r - a_s)$ of its data value was added into the COMPLEMENT aggregate. First, the aggregation results from the aggregation-commit phase are sent by the base station using authenticated broadcast to every sensor node in the network. Each sensor node then individually verifies that its contributions to the respective SUM and COMPLEMENT aggregates were indeed counted. If so, it sends an authentication code to the base station. The authentication code is also aggregated for communication efficiency. When the base station has received all the authentication codes, it is then able to verify that all sensor nodes have checked that their contribution to the aggregate has been correctly counted. The result checking process has the following phases.

Dissemination of final commitment values. Once the base station receives final commitment labels from the root of the commitment forest, it sends each of those commitment labels to the entire network using authenticated broadcast. Authenticated broadcast means that the each sensor node can verify that the message was sent by the base station and no one else.

Dissemination of off-path values. Each vertex must receive all of its off-path values to do the verification. The off-path values are defined as follows.

Definition 5.8.1 *[17] The set of off-path vertices for a vertex u in a tree is the set of all the siblings of each of the vertices on the path from u to the root of the tree that u is in (the path is inclusive of u).*

Vertex receives its off-path values from its parent. Each internal vertex has two children. For example, an internal vertex k has two children k_1, k_2. k sends the label of k_1 to k_2 and vice versa. k tags the relevant information of its left and right child. Once a vertex receives all of its off-path values it begins a verification phase.

Verification of contribution. The leaf vertex calculates the root vertex's label using its own label and off-path vertex labels. This allows the leaf to verify that its label was not modified on the path to the root during the aggregation-commit process. Then it compares the the calculated root vertex's label with the label received from the base station via authenticated broadcast. If those two labels match then it proceeds

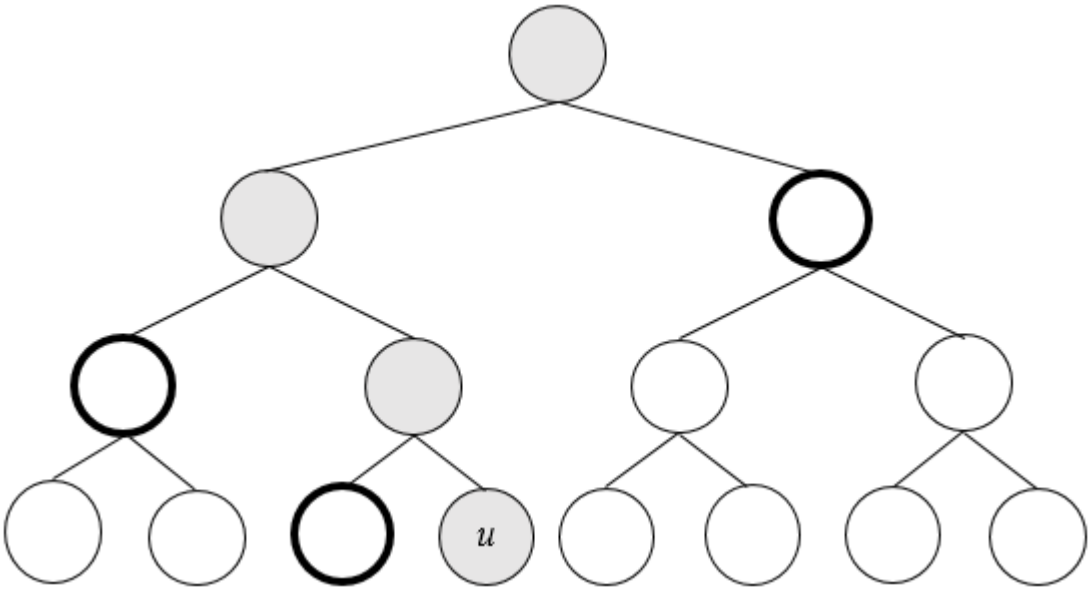

Fig. 5.8.: Off-path vertices from u are highlighted in bold.

to the next step with ACK message or with NACK message.

Collection of authentication codes. Once each sensor node s does verification of contribution for its leaf vertex v_s it sends the relevant authentication code to the base station. The authentication code for sensor node s with ACK and NACK message has the following format.

$$MAC_{K_s}(N||ACK) \tag{5.13}$$

$$MAC_{K_s}(N||NACK) \tag{5.14}$$

Where ACK, NACK are unique message identifier for positive acknowledgment and negative acknowledgment respectively, N is the query nonce and K_s is secret key that s shares with the base station. Collection of authentication code starts with the leaf nodes in the aggregation tree. Leaf nodes in the aggregation tree send their authentication codes to their parent. Once the parent node has received the authentication from all of its children it does XOR operation on all the authentication codes including its own authentication code and sends it to its parent in the aggregation tree. Each internal sensor node s in the aggregation tree repeats the process. Finally,

the root of an aggregation tree sends a single authentication code to the base station which is an XOR of all the authentication codes of the aggregation tree.

Verification of confirmations. Since the base station knows the key K_s for each sensor node s, it verifies that every sensor node has released its authentication code by computing the XOR of the authentication codes for all the sensor nodes in the network, i.e., $\bigoplus_{i=1}^{n} MAC_{K_i}(N||ACK)$. The base station then compares the computed code with the received code. If the two codes match, then the base station accepts the aggregation result.

Theorem 5.8.1 *[17] Let the final SUM aggregate received by the base station be S. If the base station accepts S, then $S_L \leq S \leq (SL + \mu \cdot r)$ where S_L is the sum of the data values of all the legitimate nodes, μ is the total number of malicious nodes, and r is the upper bound on the range of allowable values on each node.*

The above theorem is proven by SHIA. SHIA achieves security over the truncated SUM which is a resilient aggregator according to Wagner [23]. Our protocol works on SUM which is non-resilient aggregate and achieves the similar security goals.

6. DATA AGGREGATIIT WITH INTERNAL VERIFICATION

This chapter describes the rational behind the design specifications of secure aggregation protocol for sensor networks, mechanism used to achieve it and implementation details. Then we elaborate the nitty-gritty of the commitment tree generation process with internal verification.

6.1 System Design

The most significant design aspect of the sensor network is the *Lifetime of the Network*. The sensor network tend to have limited life span as they are powered by the battery. The lifetime of the sensor network is inversely proportional to the sensor nodes' power consumption. One of the most dominating factor for the power consumption is transmitting and receiving the data between sensor nodes, making network bandwidth an expensive resource. The bandwidth is more expensive resource than the local data computation, as trans-receiving activity consumes more power than computation. The obvious solution to increase the lifespan of the sensor network is to decrease the bandwidth usage in the network. As we know, data aggregation techniques can greatly reduce the bandwidth usage in the network, increasing the lifespan of the network. Hence, data-aggregation techniques are one of the key tool in our tool box while designing protocol for the sensor networks.

The second most significant factor in the design of the data aggregation protocol is *Security Architecture* . The International Telecommunications Union (ITU) Telecommunication Standardization Sector (ITU-T) Recommendation X.800, *Security Architect for Open Systems Interconnection* (OSI) provides a systematic approach for it [24]. The OSI security architecture focuses on security attacks, mechanism and

services defined as follows:

Security attack is any action that compromises the security of information owned by an organization or entity.

Security mechanism is a mechanism that is designed to detect, prevent, or recover from a security attack.

Security service is a service that enhances the security of the data processing systems and the information transfers of an organization. The services are intended to counter security attacks, by using one or more security mechanisms.

Security attacks can be classified as *passive attacks* and *active attacks*.

Passive attacks attempts to learn or make use of information from the system but does not affect the system resources. They are in the nature of eavesdropping on, or monitoring of, transmissions. Two types of passive attacks are *release of message contents* and *traffic analysis*.

The *release of message contents* is well understood. A telephone conversation, an electronic mail, and a transferred file may contain sensitive or confidential information. We would like to prevent an opponent from learning the contents of these transmissions.

The *traffic analysis* is subtler. Suppose that we had a way of masking the message so that even if the opponents captured the message, could not extract it. The common mechanism for masking the message is encryption. If we had encryption protection in place, an opponent might still be able to observe the pattern of these message communication. The opponent could determine the location and identity of communicating hosts and could observe the frequency and length of message being exchanged. These patterns might be useful in guessing the nature of the communication that was taking place.

Passive attacks are difficult to detect as they do not involve any alteration of the message. Typically, the message traffic is sent and received in an apparently normal way and neither the sender nor receiver is aware that a third party has sniffed the message or observed the traffic pattern. However, it is feasible to prevent the success

of these attacks, usually by means of encryption. Thus, the emphasis on dealing with passive attacks is on prevention rather that detection.

Active attacks involve some modification of the data stream or the creating of a false stream and can be subdivided into four categories: *replay, masquerade, modification of messages, and denial of service.*

Replay involves the passive capture of a data unit and its subsequent retransmission to produce an unauthorized effect.

A *masquerade* takes place when one entity pretends to be a different entity. For example, authentication sequence can be captured and replayed after a valid authentication sequence has taken place, thus enabling an authorized entity with few privileges to obtain extra privileges by impersonating an entity that has those privileges.

Modification of message means that some portion of a legitimate message is altered or that the messages are delayed or reordered, to produce an unauthorized effect. For example, a message stating "Allow Barack Obama to read confidential file accounts." is modified to say "Allow Michelle Obama to read confidential file accounts."

The *denial of service* inhibits the normal use of communication facilities. For example, an entity may suppress all messages directed to a particular destination. Another form of service denial is the disruption of an entire network, either by disabling the network or by overloading it with messages so as to degrade performances.

Active attacks present the opposite characteristics of passive attacks. Whereas passive attacks are difficult to detect, measures are available to prevent their success. On the other hand, it is quite difficult to prevent active attacks absolutely, because to do so would require physical protection of all communication facilities and paths at all times. Instead, the goal is to detect them and to recover from any disruption or delays caused by them. Because the detection has a deterrent effect, it may also contribute to prevention.

Security services are divided into six categories: *Authentication, Access Control, Data Confidentiality, Data Integrity, Non-repudiation, Availability.*

Authentication service assures that a communication is authenticate. Two types of authentication services are described in the standard: *peer entity authentication* and *data origin authentication*.

The *peer entity authentication* provides for the corroboration of the identity of a peer entity in an association. Two entities are considered as peer if they implement the same protocol in different systems (e.g., two TCP users in two communicating systems). Peer entity authentication is used at the establishment of, or at times during the data transfer phase of, a connection. It attempts to provide confidence that an entity is not performing either a masquerade or an unauthorized replay of a previous connection.

Data origin authentication provides the corroboration of the source of a data unit. It does not provide protection against the duplication or modification of data units. This type of service supports applications like electronic mail where there are no prior interactions between the communicating entities.

Data Confidentiality ensures the protection of transmitted data from passive attacks. It is also known as secrecy or privacy. For example, student grade information is an asset whose confidentiality is considered to be highly important by students. In the United States, the release of such information is regulated by the Family Educational Rights and Privacy Act (FERPA). Grade information should only be available to students, their parents, and employees that require the information to do their job. The other aspect of confidentiality is the protection of traffic flow from analysis. This requires that an attacker not be able to observe the source and destination, frequency, length, or other characteristics of the traffic on a communications facility. Ensuring confidentiality can be difficult. For example, who determines which parties are authorized to access the information? By "accessing" information, do we mean that an authorized party can access a single bit? the whole collection? pieces of information out of context? Can someone who is authorized disclose that information to other parties? [25]

Data Integrity assures that the only authorized entities can modify the information; where modification means writing, creating, deleting and changing the information. A hospital patient's allergy information stored in a database illustrates the several aspects of the data integrity. The doctor should be able to trust that the information is correct and current. Now, suppose a nurse who is authorized to view and update the information deliberately falsifies the data to cause harm to the hospital. The database needs to be restored to a trusted basis quickly, and it should be possible to trace the error back to the person responsible. Patient allergy information is an example of an asset with a high requirement for integrity. Inaccurate information could result in serious harm or death to a patient and expose the hospital to massive liability. Because the integrity service relates to active attacks, we are concerned with detection rather than prevention. If a violation of integrity is detected, then the service may simply report this violation, and some other portion of software or human intervention is required to recover from the violation. Alternatively, there are mechanisms available to recover from the loss of integrity of data, as we will review subsequently. The incorporation of automated recovery mechanisms is, in general, the more attractive alternative.

Availability means that the information of a system being accessible and usable by an authorized parties according to performance specification at appropriate times. In other words, if some person or system has legitimate access to a particular set of objects, that access should not be prevented. Availability applies to the information and services. For example, information or service is available can mean the following. It is present in the usable form, having the enough capacity to meet the service need. If it is in wait mode, it is making enough progress, and it has a bounded waiting time, meaning there is timely response to our requests. Resources are allocated fairly so that some requests are not favored over the others. The service can be used easily and in its intended way. Availability addresses the concern raised by the denial of service attacks. It depends on proper management and control of system resources and thus depends on access control service and other security services.

Non repudiation service requires neither the sender nor the receiver can deny the transmission. Thus, when a message is sent, the receiver can prove that the alleged sender in fact sent the message. Similarly, when a message is received, the sender can prove that the alleged receiver in fact received the message. This is very important in electronic commerce applications, where it is important that a consumer cannot deny the authorization of a purchase.

Access Control is the ability to restrict and control the access to host systems and applications via communications links. To achieve this, each entity trying to gain access must first be identified, or authenticated, so that access rights can be tailored to the individual. Login credentials and Locks are two analogues mechanism of access control.

Security mechanisms follow one of the most fundamental principle of Open Design [26] defined as follows:

Definition 6.1.1 *The principle of open design states that the security of a mechanism should not depend on the secrecy of its design or implementation.*

Designers and implementers of a program must not depend on secrecy of the details of their design and implementation to ensure security. Others can ferret out such details either through technical means, such as disassembly and analysis, or through nontechnical means, such as searching through garbage receptacles for source code listings (called "dumpster-diving"). If the strength of the program's security depends on the ignorance of the user, a knowledgeable user can defeat that security mechanism. The term "security through obscurity" captures this concept exactly. This is true of cryptographic software and systems. Because cryptography is a highly mathematical subject, and companies who sell these softwares want to keep their algorithms secret. Issues of proprietary software and trade secrets complicate the application of this principle. In some cases, companies may not want their designs made public, lest their competitors use them. The principle then requires that the design and implementation be available to people barred from disclosing tit outside the company. Note that keeping cryptographic keys and passwords does not violate this principle, because a

key is not an algorithm. However, keeping the enciphering and deciphering algorithms secret would violate it. The following security mechanisms may be incorporated into the appropriate protocol layer in order to provide some of the OSI services.

Encryption is the use of mathematical algorithms to transform data into a form that is not readily intelligible. The transformation and subsequent recovery of the data depend on an algorithm and zero or more encryption.

Digital Signature appends the data to a cryptographic transformation of data unit that allows a recipient of the data unit to prove the source and integrity of the data unit and protects against the forgery (e.g., by the recipient).

Traffic Padding is the insertion of bits into gaps in a data stream to frustrate traffic analysis attempts.

Routing Control enables selection of particular physically secure routes for certain data and allows routing changes, especially when a breach of security is suspected.

Notarization is the use of a trusted third party to assure certain properties of a data exchange.

Authentication Exchange is a mechanism intended to ensure the identity of an entity by means of information exchange.

Hash functions provides one way functions to achieve irreversible encryption.

As you can see, expectations from secure networking protocol are far-reaching. It is very ambitious for any system to have all the above mentioned security services at the same time. If we have to achieve all the security services for sensor networks we can make each sensor node signs its reading data and then send data with its signature to its parent. And the network forwards all the data with their respective signatures to the base station. The base station verifies all the signatures and then calculates the aggregate function. Clearly, this approach is not practical as it requires n signatures to traverse through the link between the base station and the root node of the network; where n is the number of nodes in the network. Which makes that link the bottleneck link of the network. And if that link breaks we loose the entire network connectivity.

In reality all the security services are not always required. For an instance, when a client downloads a file from the File server using Internet, he can verify the integrity of the File using the checksum. But it is okay if somebody on the network sniffed the downloading activity as far as it did not change the content of the File. In this application, a client requires the integrity of the File but the privacy of the client, the File server and the Internet service provider are not required. To give an analogy with a physical world application, when a person writes a postcard, he puts his own signature on the it. When that postcard delivers successfully by the postal service, the receiving party can verify the integrity of the message from the handwriting and the signature of the person. The postal service knows the sender and the receiver of the postcard. It can even read the postcard. Again, here integrity is important not the confidentiality. It is crucial to find out which security services are desired for the particular application, so we can use the relevant security mechanisms to provide those services. In practice, protocol designer finds out the most important security services for the applications before designing the protocol.

For example, Wagner's work [23] describes the attacks on standard schemes for data aggregation and introduces the problem of securing aggregation in the presence of malicious or spoofed data. He proposes a mathematical theory of security for aggregation. The theory quantifies, in a principled way, the robustness of an aggregation operator against malicious data. It draws novel connections to statistical estimation theory and to the field of robust statistics. He identifies techniques for aggregation that provide robustness against attack. It provides helpful guidance to sensor network implementors or designers in selecting appropriate aggregate functions.

Secure Information Aggregation (SIA) [16] address the problem of how to enable secure information aggregation, such that the user accepts the data with high probability if the aggregated result is within a desired bound, but that the user detects cheating with high probability and rejects the result if it is outside of the bound. SIA provides statistical security under the assumption of a single-aggregator model. In the single-aggregator model, sensor nodes send their data to a single aggregator

node, which computes the aggregate and sends it to the base station. This form of aggregation reduces communications only on the link between the aggregator and the base station, and is not scalable to large multihop sensor deployments. SIA provides the probabilistic data-integrity service under strong network assumptions.

Secure Hierarchical In-network Aggregation (SHIA) [17], in many ways enhances Secure Information Aggregation (SIA). SHIA presents the provably secure sensor network data aggregation protocol for general networks and multiple adversarial nodes, in compare to SIA which provides probabilistic security for a single aggregate network topology. SHIA limits the adversarys ability to manipulate the aggregation result with the tightest bound possible, with no knowledge of the distribution of sensor data values. SHIA provides data-integrity service for any hierarchical sensor networks because of that it can detect malicious activity in the network.

The third most significant aspect in the design of secure data aggregation protocol is *Data Integrity*. The protocol collects the data from the sensors and aggregates the reading on the way to the base station. The base station takes an important decision based on the received value. For example, if the sensor network is deployed to measure the certain harmful chemical levels in the lab atmosphere and raise an alarm if the it increases above certain level. And if the base station fails to raise an alarm because of the false aggregated data, can create catastrophic and lethal situation. Hence, the data integrity is so important.

As we know, data integrity can be achieved with the error detection and error correction techniques. The first step towards achieving the data integrity in the sensor network is to *detect any malicious activity* in the network. And the second step is the ability to *locate the malicious node or an adversary* in the network. We think detecting a malicious activity without tracing down the malicious node responsible for it, is of no value. To give an analogy with the physical world, it is like you know there is a crime in the city and you do not do anything about it. Locating the criminals responsible for the crime is mandatory to abolish the crime in the city. In similar way, locating the malicious node and removing it from the network is an important

security service. Failing to provide such a security service, allows an adversary to continue doing the malicious activity in the future, which makes aggregated sensor data garbage. And network has to redo the all the work to create response for the query from the base station, which consumes lots of bandwidth. An adversary can repeat this process until the network dies due to low battery power, creating a denial of service attack in the network. Hence, we think detecting a malicious node who is responsible for the malicious activity is equally important. If we can track down the malicious node in the network then we can remove that node from the network for all future queries. And make sure that all the future queries are not manipulated by any malicious node in the network. Hence, the fourth and fifth most significant design aspects of secure aggregation protocol are *detect the malicious activity* (in terms of data aggregation) and can *detect the adversary* in the network, respectively.

To detect an adversary (or prove someone guilty), we need proof that the adversary is responsible for the malicious activity. Consider the following example showing the analogy with the signature scheme in the physical world, used by the postal services. When a postman delivers the package, the receiving party has to sign the document informing that he verified and received the package. As only the receiving party can create his signature, in the future he can not claim of not receiving the package or receiving the damaged or incorrect package. And if he claims such, the postal company has the signed document as the proof mentioning that the package was received successfully, by the receiving party. The signed document also ensures to the postal company that the postman did not misplace or steal the package. The signature scheme used by the postal service promises non repudiation security service. Hence, we require *non-repudiation* security service in the sensor network.

6.2 System Design Specifications

We want to design a secure aggregation protocol which maximizes the *network lifetime*. The protocol can be applied to *any hierarchical sensor network*. We want

protocol to work on *resilient* and *non-resilient* aggregate functions without compromising any desired security properties. We want protocol to be secure with *a single* or *multiple adversaries* in the network. We want the protocol to protect against any *active attacks*. If the aggregation result is accepted by the base station it should have very high confidence in the result, meaning that we want the highest level of *data integrity* security service in the protocol. We want the capabilities to *detect any malicious activity*. If there is any malicious activity in the network, we should be able to *locate an adversary* responsible for it, in the network. We want the *non-repudiation* security service so neither sender nor receiver can deny the transmission or receiving of the message, which is mandatory to locate an adversary. We want to achieve this with the least amount of *bandwidth and computation* overhead in the network. So, the protocol can be easily implemented in the real world sensor networks. It is not that difficult to provide mentioned security properties to protect against active attacks in the sensor network. We will show that it requires sub-linear edge congestion to provide these services.

6.3 Security Mechanisms

To detect any malicious activity in the network which is an important part of achieving data-integrity in the network we use *Hash Functions* as security mechanism. To protect against any active security attacks, provide authentication and non-repudiation security services we use *Digital Signatures* as the security mechanisms. Both of these mechanisms are described in details in Chapter 3.

6.4 System Design Implementations

The high level idea of the aggregate commit with verification scheme is that all the nodes in the network send the signature of the message along with the message itself. It sends its certificate if the parent node does not have it already. The parent node verifies all the received signatures from its children. And proceeds with the

aggregation process. After aggregation, the parent node can throw away all the signatures from its children and signs the message of its children or it can pass its children's signatures to its parent. The pros and cons of each approach are discussed in the following sections.

6.5 Data-Item

We describe structure of the data-item, used in creating the commitment tree for the aggregate commit with verification approach. And differences between the data-item and the label structure of SHIA, with rational behind it.

Definition 6.5.1 *A commitment tree is a binary tree where each vertex has an associated data-item representing the data that is passed on to its parent. The data-items have the following format:*

$$< \textbf{\textit{id, count, value, commitment}} >$$

Where id is the unique ID of the node; count is the number of leaf vertices in the subtree rooted at this vertex; value is the SUM aggregate computed over all the leaves in the subtree and commitment is a cryptographic commitment.

We remove the *complement* field from the label structure Defined 5.7.1. We think the complement filed is redundant information in the label. The complement field is used by the base station (the querier, according to SHIA), before the result checking phase, to verify $\mathbf{SUM + COMPLEMENT = n \cdot r}$; where $\mathbf{n}$ is the number of nodes in the network, $\mathbf{r}$ is the upper bound on the allowed sensor readings. We can achieve the same upper bound without the complement field. As the querier knows n, r and it gets SUM from the root of the aggregation tree. If $\mathbf{SUM > n \cdot r}$, then the base station knows some node or nodes in the network reported out of range readings.

We include *id* of the node in its data-item. SHIA does not have the ID field in their label structure as they do not do internal verification while creating a commitment tree and while distributing off-path values. Also, in the label format ID of the node is hashed in the commitment field after the first aggregation and virtually gets lost.

Hence, SHIA can not provide security services such as authenticity, non-repudiation and is vulnerable to all sorts of active attacks. We do internal verification while creating the commitment tree and distributing off-path values. So, it is necessary for any aggregate node to know the ID of all the received data-items in its forest, for the verification of the received signatures as shown in the next section.

6.6 Signing the Data-Item

During commitment tree generation phase, there is one vertex S_0 for each sensor node S, which we call the leaf vertex of S. The data-item for leaf vertex S_0 and associated signature to it is defined as follows:

$$S_0 = < S_{id}, 1, S_{value}, H(N||1||S_{value}) >; \ \mathsf{Sign}_{S_S}(S_0) \tag{6.1}$$

where H is a collision resistant hash function, S_S is the secret key of sensor node S, N is the query nonce.

The key difference between the SHIA's approach and our approach is that, in addition to sending the data-item, each sensor node sends the signature of the data-item to its parent. It sends its certificate as well if the parent node does not have it in its memory already. The parent node gets the public key of the child node from its certificate, which is used in verifying the signature. The parent node verifies all the received signature using its children node's public key. Details of the signing and verification processes are shown in Figure 3.1. After doing the verification the parent node proceeds to the aggregation and creating the commitment tree.

6.6.1 Bandwidth Analysis

Cite resources for this.

Typical size of the data-item packet is 400 bits. If one uses Elliptic curve cryptography then the size of signature is 500 bits. And the certificate size is 1500 bits. So, at max we have to send additional 2000 bits with the data-item. We think it is

worthwhile to send these additional bits. Because of all the security benefits we gain from it. **Note:** The packets size are close approximate to the actual packet size. The actual packet size may differ based on the implementation.

6.6.2 Security Benefits

The signature allows the parent node to verify the *authenticity* of the sensor node. It also allows to verify the *integrity* of the received data-item. It allows the sender to have the proof for the sent data-item. And the receiver for the proof for the received data-item, providing the *non-repudiation*. Hence, it protects against the *active attacks*.

6.7 Signing the Commitment Payload

We define commitment payload based on the commitment forest Defined in 5.7.2.

Definition 6.7.1 *A **commitment payload** is a set of data-items of the root vertices of the trees in the outgoing commitment forest.*

For brevity, we use the term forest, payload instead of commitment forest, commitment payload respectively.

An aggregate node sends an additional signature to its parent, which is a signature of all the data-items in its payload. The signature on the payload assures that an aggregate node sent only the data-items included in the payload signature. It assures the authenticity, non-repudiation, integrity of all the data-items included in the payload. For example, the aggregation tree shown in Figure 6.1, the payload of sensor node C is show in Figure 6.2. The sensor node C sends all the data-items in its payload with their signatures to its parent sensor node D. Furthermore, C sends the signature of its payload $\mathsf{Sign}_{S_C}(C_0||B_1)$ to D.

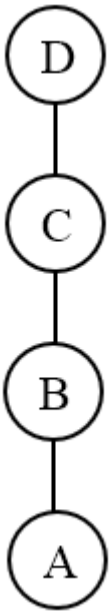

Fig. 6.1.: Palm Shaped Aggregation Tree

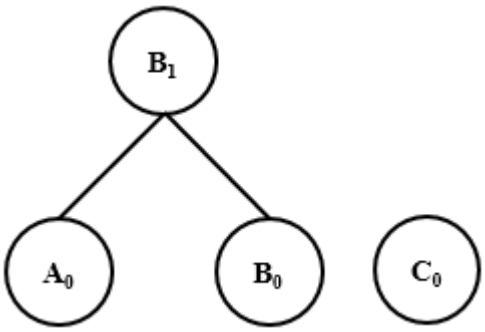

Fig. 6.2.: Commitment Payload of Sensor Node C

$$
\begin{aligned}
&B_1 = < B_{id}, 2, B_{1value}, H(N||2||B_{1value}||A_0||B_0) >;\ \mathsf{Sign}_{S_B}(B_1) \\
&C_0 = < C_{id}, 1, C_{value}, H(N||1||C_{value}) >;\ \mathsf{Sign}_{S_C}(C_0) \\
&\mathsf{Sign}_{S_C}(C_0||B_1)
\end{aligned}
\tag{6.2}
$$

6.7.1 Security Benefits

In addition to all the security benefits mentioned in Subsection 6.6.2, signature on the C's payload $\mathsf{Sign}_{S_C}(C_0||B_1)$ assures that the sensor node C sent only two data-items C_0, B_1 in its payload. And none of the data-items in its payload have been left stranded.

6.8 Forwarding the Commitment Payload

The sensor node C has two data-items C_0, B_1 in its payload as shown in Equation 6.2. It sends the $\mathsf{Sign}_{S_B}(B_1)$, $\mathsf{Sign}_{S_C}(C_0)$ and $\mathsf{Sign}_{S_C}(C_0||B_1)$ to its parent node D. It requires the parent node D to know the public key of the sensor nodes C and D, hence their certificates. Instead of that the sensor node C can verify the $\mathsf{Sign}_{S_B}(B_1)$ then remove the old signature and create new signature $\mathsf{Sign}_{S_C}(B_1)$ signed by itself on the data-item B_1 as shown below:

$$\begin{aligned} B_1 &= < B_{id}, 2, B_{1value}, H(N||2||B_{1value}||A_0||B_0) >;\ \mathsf{Sign}_{S_C}(B_1) \\ C_0 &= < C_{id}, 1, C_{value}, H(N||1||C_{value}) >;\ \mathsf{Sign}_{S_C}(C_0) \\ &\mathsf{Sign}_{S_C}(C_0||B_1) \end{aligned} \tag{6.3}$$

We call these two approaches **Forwarding signatures without resigning the data-items**, **Forwarding signatures with resigning the data-items** as shown in Equations 6.2 and 6.3 respectively .

To give an analogy with the real world application consider the following example. One want to buy a diamond from the local diamond retailer. Diamond is an expensive commodity so the end customer wants to verify its authenticity and integrity before purchasing. Suppose, the diamond was created by the manufacturer in Africa, it was sold to a national wholesaler in the United States. The national wholesaler sells it to the state level reseller and he sells it to the city or county level retailer from whom the customer purchases the diamond.

One approach to verify the authenticity of the commodity is to make each entity in the supply chain to verify all the signatures on the received entity and sign on top of

it. And then forward the commodity with all the signatures to the next entity in the supply chain. The next entity repeats the same procedure. Hence, any entity in the supply chain need to verify the signatures of all its descendants in the supply chain. In our example, it means to make the manufacturer from Africa signs the diamond and sells the signed diamond with his certificate to the national level wholesaler in United States. The national level wholesaler in United States verifies the signature from the manufacturer using manufacturer's certificate. Then he adds his signature and certificate, and sells the diamond signed with two signatures and two certificates to the state level reseller. The state level reseller verifies both the signatures on the diamond using the respective certificates. Then he adds his signature and certificate, and sells the diamond singed with three signatures and three certificates to the city level retailer. The city level retailer does the same thing before selling the diamond to the end customer. In the end, the customer needs to verify all four signatures, using the respective certificates.

An alternative approach to verify the authenticity of the commodity is to make each entity in the supply chain verify the signature, throw away the old signature, and then add its own signature on it. It means the next entity in the supply chain need to verify only a single signature. The next entity repeats the same procedure. Hence, any entity in the supply chain need to verify the signature of only its direct peer in the supply chain. In our example, it means to make the manufacturer from Africa signs the diamond and sells the signed diamond with his certificate to the national level wholesaler in United States. The national level wholesaler in United States verifies the signature from the manufacturer using the manufacturer's certificate. Then he removes the signature of the manufacturer, adds his own signature and certificate, and sells the diamond signed with one signature and one certificate to the state level reseller. The state level reseller verifies only the signature from the wholesaler using the wholesaler's certificate. Then he removes the signature of the wholesaler, adds his own signature and certificate, and sells the diamond signed with one signature and one certificate to the city level retailer. The city level retailer does the same

thing before selling the diamond to the end customer. In the end, the customer needs to verify only one signature of the city level retailer using retailer's certificate. This approach requires very few number of certificates overall in the supply chain.

Both theapproaches have their pros and cons and the perfect approach depends heavily on the application. The various aspects of both the approaches for sensor nodes are discussed in the following sections.

6.9 Forwarding signatures with resigning the data-items

If we throw away the old signatures and resign the data-item with current aggregate node then following is true:

- Each parent needs the certificates of only its direct children.
- Each child needs to know the certificate of its parent only.
- Number of signatures remain the same as previous approach.
- Number of certificates needed in the network is $O(n)$; n is the number of nodes in the network.
- We do not need the signature of the payload.

As the commitment tree is binary we need $\Omega(2^h - 1)$ signatures; where h is the height of the commitment tree in both approaches . Note that we send signatures while distributing off-path values.

6.10 Forwarding signatures without resigning the data-items

6.11 Commitment Tree Generation

For the given aggregation tree the commitment forest is built as follows. Leaf sensor nodes in the aggregation tree create their leaf vertex by creating data-items and their respective signatures according to Equation 6.1, **??** which they send it to

	With resigning	Without resigning
Number of signatures	Test	T
Number of signing activity	Test	T
Number of verifying activity	Test	T
Number of certificates	Test	T

Table 6.1.: System-on-Chip specifications for IEEE 802.15.4 from TexasInstruments

their parent as a payload in the aggregation tree. Each internal sensor node I in the aggregation tree also creates their leaf vertex and its signature. In addition, I receives the payload from each of its children which creates the forest for I. Once I verifies all the received signatures, it merges all the data-items in its forest with same count value to create its payload. Note that we can determine the height of the commitment tree from the count value.

Suppose I have to create its payload by merging i data-items D_1, D_2, ..., D_i in its forest. First, I verifies the received signatures $Sign(D_1)$, $Sign(D_2)$, ..., $Sign(D_i)$. Once verified, I starts merging the data-items as follows. Let c be the smallest count value in I's forest. The sensor node I finds two data-items D_1, D_2 in its forest with the same count value c and merges them into a new data-item with the count of $c+1$ as shown in Figure 6.3.
It repeats the process until no two data-items in its forest have the same count value. An example of generating the payload by merging the data-items in the forest for the sensor node A in Figure 5.2 is illustrated in the following example.

Example 6.11.1 *The commitment-payload generation process for node A of Figure 5.2 is shown here.*

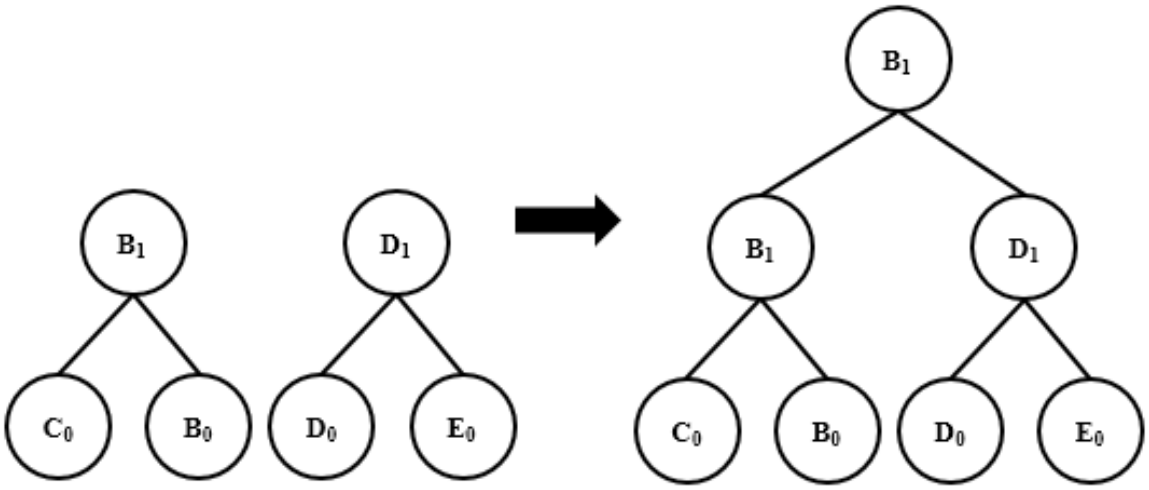

Fig. 6.3.: A has B_1, C_1 in his forest and aggregates those two trees and creates A_2.

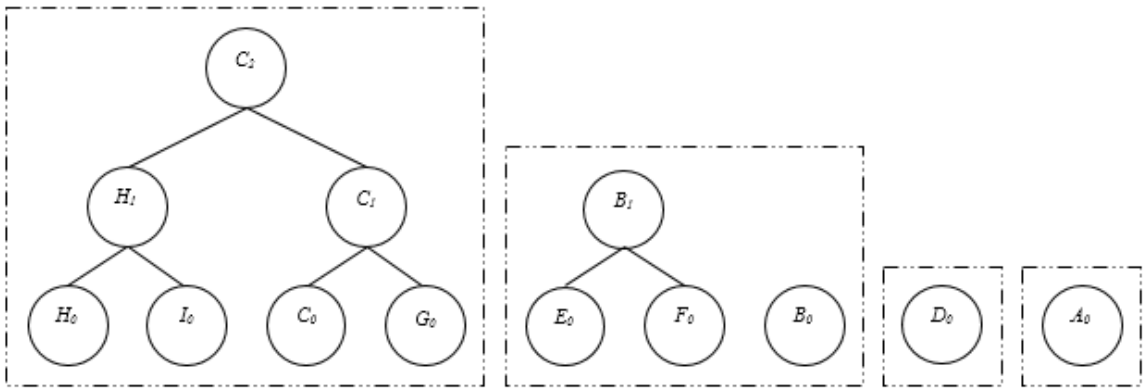

Fig. 6.4.: A receives C_2 from C, (B_1, B_0) from B, D_0 from D and generates A_0. The commitment payload received from a given sensor node is indicated by dashed-line box.

$$
\begin{aligned}
A_0 &=< A_{id}, 1, A_{value}, H(N||1||A_{value}) >; \mathit{Sign}_{S_A}(A_0) \\
D_0 &=< D_{id}, 1, D_{value}, H(N||1||D_{value}) >; \mathit{Sign}_{S_D}(D_0) \\
B_0 &=< B_{id}, 1, B_{value}, H(N||1||B_{value}) >; \mathit{Sign}_{S_B}(B_0) \\
B_1 &=< B_{id}, 2, B_{value}, H(N||2||B_{value}||E_0||F_0) >; \mathit{Sign}_{S_B}(B_1) \\
&\mathit{Sign}_{S_B}(B_0||B_1), benefits \\
C_2 &=< C_{id}, 4, C_{value}, H(N||4||C_{value})||H_1||C_1) >; \mathit{Sign}_{S_C}(C_2)
\end{aligned}
\tag{6.4}
$$

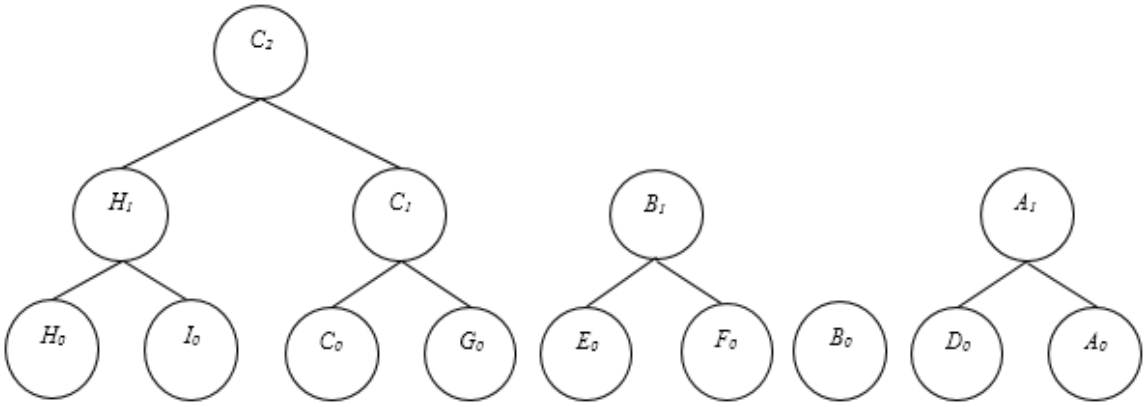

Fig. 6.5.: First Merge: A_1 vertex created by A.

$$A_1 =< A_{id}, 2, A_{1value}, H(N||2||A_{1value}||A_0||D_0) >; Sign_{S_A}(A_1)$$
$$where\ A_{1value} = A_{value} + D_{value} \tag{6.5}$$

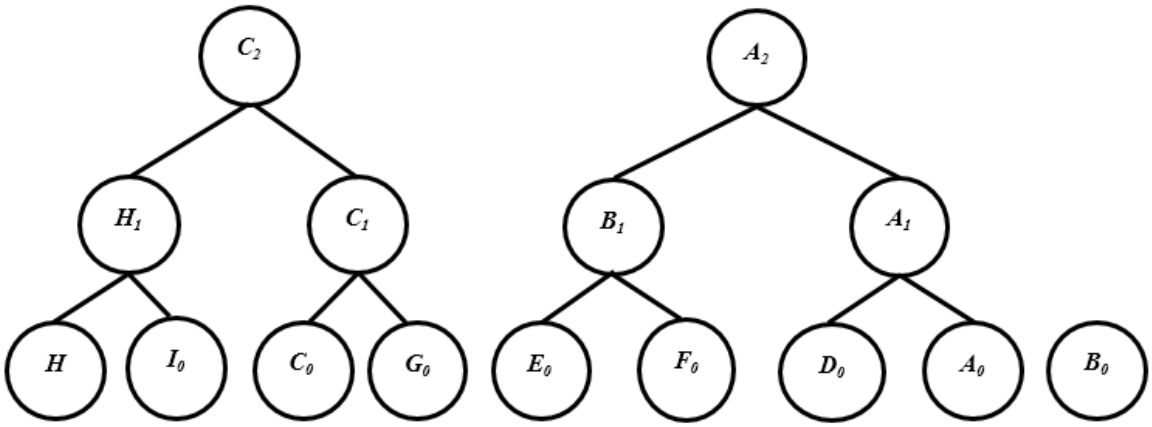

Fig. 6.6.: Second Merge: A_2 vertex created by A.

$$A_2 =< A_{id}, 4, A_{2value}, H(N||4||A_{2value}||B_1||A_1) >; Sign_{S_A}(A_2)$$
$$where\ A_{2value} = B_{1value} + A_{1value} \tag{6.6}$$

$$A_3 =< A_{id}, 8, A_{3value}, H(N||8||A_{3value}||C_2||A_2) >; Sign_{S_A}(A_3)$$
$$where\ A_{3value} = A_{2value} + C_{2value} \tag{6.7}$$

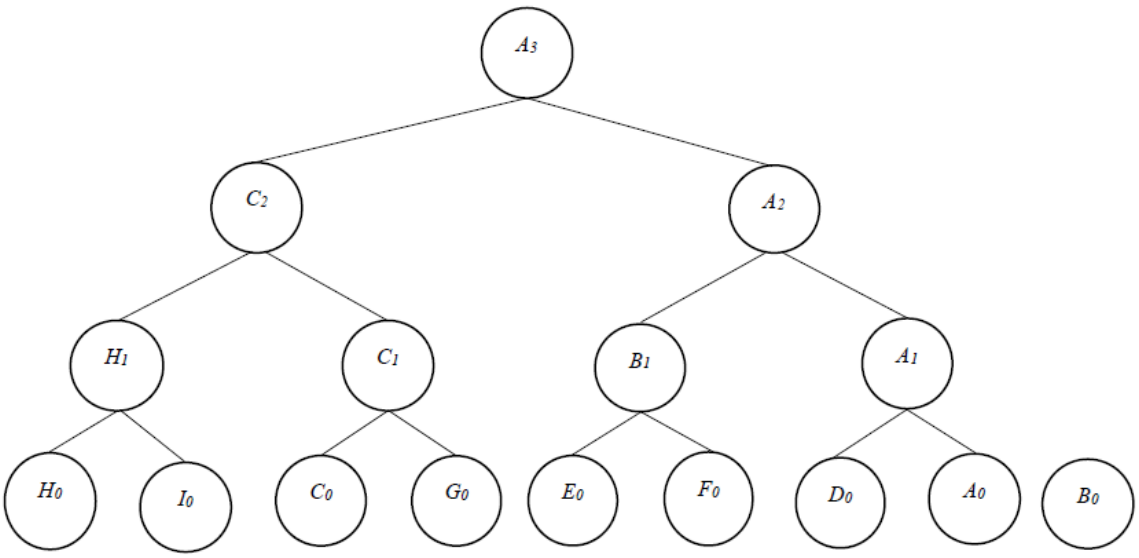

Fig. 6.7.: Third Merge: A_3 vertex created by A.

6.12 Bandwidth Analysis

For any given sensor node's forest with n leaf vertices, has at most $\log n$ data-items in its payload. It has at most $(\log n) + 1$ signatures in its payload. The highest possible count value is $\log n$, as all the trees are binary.

An intermediate sensor node S with β descendants in the aggregation tree, has at most $\log(\beta+1)$ data-items with their respective $\log(\beta+1)$ signatures in its payload. S might need to send its payload signature $Sign(S_p)$. At max, S has to send a payload with $\log(\beta+1)$ data-items and $\log(\beta+1)+1$ signatures to its parent in the aggregation tree.

Hence, sending signatures of the data-items causes $O(\log \beta)$ bandwidth overhead for each node in the network, where β is the number of descendants of the sensor node.

6.13 Result checking

6.14 Performance Analysis

In addition to calculating its own data-items, all intermediate sensor nodes with β descendants and ζ direct children need to do the following:

- To calculate and verify $O(\log \beta)$ signatures, creating $O(\log \beta)$ calculation overhead.
- Needs sufficient memory to cache $O(\log \beta)$ certificates.
- Needs enough memory to cache $\Omega(\zeta)$ certificates.

6.15 Applications

The signature based aggregation scheme can be applied to do the **voting** in the network. And voting scheme can be used to solve many sensor network problems. For example, voting can be used to design the distributed algorithm for selecting a cluster head or node revocation system. In the voting scheme, following are the major security concerns:

- The aggregate node needs to know that the vote is coming from the legit voter, no other voter is impersonating the vote of the legit voter.
- Only the intended aggregate node should be able to verify the vote.
- The aggregate node should not be able to tamper with the votes.
- The aggregate node needs the proof that it aggregated the verified votes.
- The voter need the proof for which vote it sent to its aggregator.

For example, the base station wants to know the overall vote-count in the network. To do so, all the leaf nodes send their votes and the signature of their votes to their respective aggregate nodes in the network. The aggregate nodes receive votes with

their signatures from all of their children voters. The aggregate nodes verify all the votes and count those votes. Then they forward the count and the signature of that count signed by the aggregate node to their respective parent in the aggregation tree. This process is repeated until the final count and its signature, is sent to the base station by the root of the aggregation tree.

Node power level, **Surveillance Application**

7. SECURITY ANALYSIS

This chapter describes what does it mean to cheat a secure aggregation algorithm. It depicts detailed examples in which an adversary tries different ways of cheating but fails to do so because of the commitment and signatures.

7.1 Introduction

To discuss different ways of cheating first we define the cheating as follows:

Definition 7.1.1 *A sensor node tampering with the data-item to skew the final aggregate data-item without being detected is consider as* ***cheating****.*

Because of the way an aggregate commit algorithm works, an aggregate node has the highest power to do the cheating as described in Section 2.5. The aggregate node gains more power to cheat as it climbs up in the aggregation tree hierarchy. An aggregate node can cheat at the following phases:

- During commitment tree generation phase
- During dissemination of off-path phase
- At both phases together (the most powerful attack possible)

We can detect a cheating activity with the help of the commitment field in the data-item as shown in Example 7.1.1. But the commitment field is not enough to detect an adversary. And to identify an adversary we need the help from the base station and the signature infrastructure created in the network.

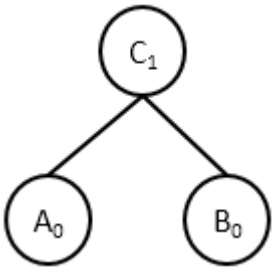

Fig. 7.1.: Smallest possible commitment tree

7.1.1 Assumptions

We make following assumptions for the adversary.

- It can not send an authentication code with NACK message during verification of inclusion phase.
- It does not have the capability to masquerade by reproduce the signatures of any other sensor node.

Without these assumptions, there will be a lot of complainers in the network, creating a lot of traffic in the network. Ultimately, draining the battery levels of the sensor nodes until they die, making some sensor nodes in the network unreachable and potentially causing the denial-of-service attack in the network. Following example shows the different ways an adversary can cheat in the smallest possible commitment tree and how the commitment field in the data-item can help us detect the cheating. And theoretically, all possible commitment trees can be presented as Figure 7.2 for the analysis purpose.

Example 7.1.1 *Let's say the vertices in the commitment tree of Figure 7.2 have the data-items defined as follows.*

$$\begin{aligned} A_0 &= < A_{id}, 1, 10, H(N||1||10) > \\ B_0 &= < B_{id}, 1, 20, H(N||1||20) > \\ C_1 &= < C_{id}, 2, 30, H(N||2||30||A_0||B_0) > \end{aligned} \tag{7.1}$$

Note that we did not include the signatures of these data-items in this example for simplification.

- ***No cheating***

  C aggregates B_0, C_0 according to the aggregate commit algorithm.

  dissemination of root data-item

  A, B receives C_1 from the base station using authenticated broadcast.

  dissemination of offpath values

  A receives B_0 from C and vice versa.

  verification of inclusion

  $$\begin{aligned} &A\ uses (A_0, B_0)\ to\ calculate < 2, 30, H(N||2||30||A_0||B_0) > = C_1 \\ &B\ uses (A_0, B_0)\ to\ calculate < 2, 30, H(N||2||30||A_0||B_0) > = C_1 \end{aligned} \tag{7.2}$$

  collection of authentication codes

  A, B sends their authentication codes with ACK messages.

- ***Cheating by replacing data-items***

  C replaces A_0, B_0 with A'_0, B'_0 and then applies aggregate commit algorithm.

  $$\begin{aligned} A'_0 &= < A_{id}, 1, 100, H(N||1||100) > \\ B'_0 &= < B_{id}, 1, 200, H(N||1||200) > \\ C'_1 &= < C_{id}, 2, 300, H(N||2||300||A'_0||B'_0) > \end{aligned} \tag{7.3}$$

  dissemination of root data-item

  A, B receives C'_1 from the base station using authenticated broadcast.

  dissemination of offpath values

  A receives B'_0 from C and vice versa.

  verification of inclusion

  $$\begin{aligned} &A\ uses\ (A_0, B'_0)\ to\ calculate < 2, 210, H(N||2||210||A_0||B'_0) > \neq C'_1 \\ &B\ uses\ (A'_0, B_0)\ to\ calculate < 2, 120, H(N||2||120||A'_0||B_0) > \neq C'_1 \end{aligned} \tag{7.4}$$

  collection of authentication codes

  A, B send their authentication codes with $NACK$ messages.

- ***Cheating by tampering with a single data-item***

  C replaces A_0 with A'_0 and then applies aggregate commit algorithm.

$$\begin{aligned} A'_0 &= < A_{id}, 1, 100, H(N||1||10) > \\ C'_1 &= < C_{id}, 2, 120, H(N||2||120||A_0||B'_0) > \end{aligned} \tag{7.5}$$

  dissemination of root data-item

  A, B receives C'_1 from C

  dissemination of offpath values

  A, B receives B'_0, A_0 from C respectively.

$$\begin{aligned} B'_0 &= < B_{id}, 1, 110, H(N||1||110) > \\ A_0 &= < A_{id}, 1, 10, H(N||1||10) > \end{aligned} \tag{7.6}$$

  verification of inclusion

$$\begin{aligned} &A \text{ uses } (A_0, B'_0) \text{ to calculate } < 2, 120, H(N||2||120||A_0||B'_0) > = C'_1 \\ &B \text{ uses } (A_0, B_0) \text{ to calculate } < 2, 30, H(N||2||30||A_0||B_0) > \neq C'_1 \end{aligned} \tag{7.7}$$

  collection of authentication codes

  A, B send their authentication codes with ACK,NACK message respectively.

- ***Cheating by tampering with data-items***

  C changes A_0, B_0 to A'_0, B'_0 by changing the value fields and then applies aggregate commit algorithm.

$$\begin{aligned} A'_0 &= < A_{id}, 1, 100, H(N||1||10) > \\ B'_0 &= < B_{id}, 1, 200, H(N||1||20) > \\ C'_1 &= < C_{id}, 2, 300, H(N||2||300||A''_0||B_0) > \\ C''_1 &= < C_{id}, 2, 300, H(N||2||300||A_0||B''_0) > \end{aligned} \tag{7.8}$$

  dissemination of root data-item

  A, B receives either C'_1 or C''_1 from the base station using authenticated broadcast based on what C send to base station.

dissemination of offpath values

A, B receives B_0'', A_0'' from C respectively.

$$\begin{aligned} B_0'' &=< B_{id}, 1, 290, H(N||1||20) > \\ A_0'' &=< A_{id}, 1, 280, H(N||1||10) > \end{aligned} \tag{7.9}$$

verification of inclusion

$$\begin{aligned} &A \text{ uses } (A_0, B_0'') \text{ to calculate } < 2, 300, H(N||2||300||A_0||B_0'') > \neq C_1' = C_1''' \\ &B \text{ uses } (A_0'', B_0) \text{ to calculate } < 2, 300, H(N||2||300||A_0''||B_0) > = C_1' \neq C_1''' \end{aligned} \tag{7.10}$$

collection of authentication codes

A, B send their authentication codes with NACK message.

This example shows how the commitment provides **data-integrity** to the data-items. It makes it nearly impossible for an adversary to tamper with the data-items while creating commitment tree and/or while distributing off-path values. This cheating activity can be detected by the commitment filed in the data-item. Later we will show that the adversary can be detected with the provided signature infrastructure.

7.2 Possible Cheater Analysis

The following example demonstrates that if the base station knows the commitment tree topology and the complainers in the network then it can guess possible adversaries. Then applies interactive protocol described in Algorithm 1 in the next chapter to identify the adversary. The details on how the base station finds complainers in the network are described in the next chapter. For simplicity, we selected the commitment tree where all the vertices are unique. But this analysis holds true for any commitment tree topology.

Example 7.2.1 *The base station creates a set $c = \{c_1, c_2, \ldots, c_n\}$ for the complainers based on the protocol described in the Section 8.5. And based on the set c it creates a set $a = \{a_1, a_2, \ldots, a_n\}$ of the possible adversaries.*

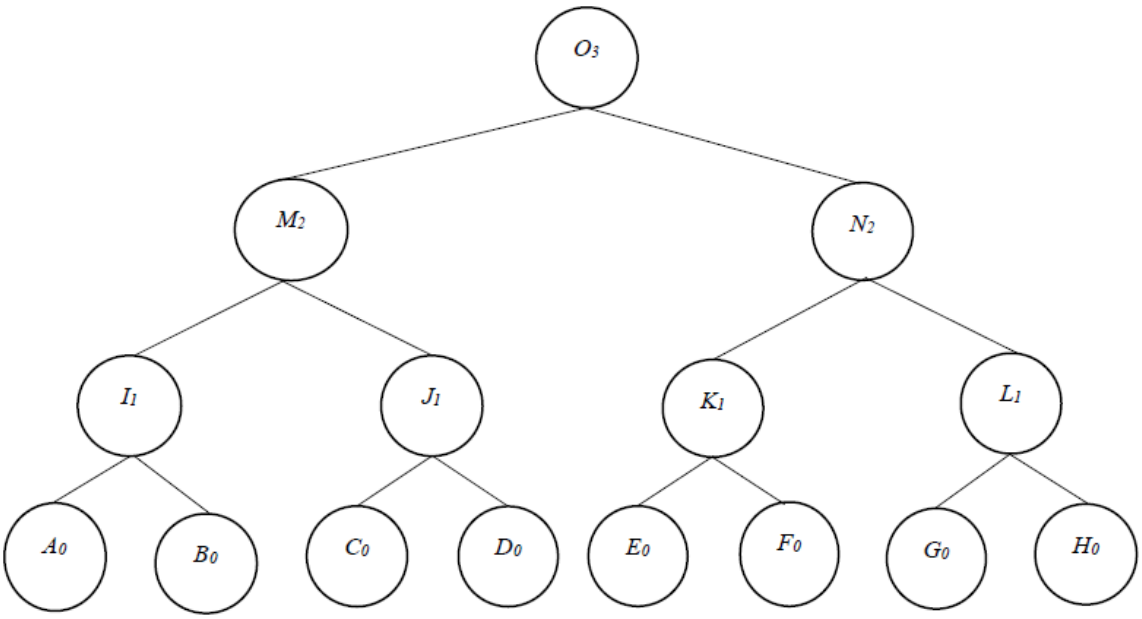

Fig. 7.2.: Smallest possible commitment tree

- *If* $c = \{A_0\}$ *then* $a = \{I, (B, I), (B, M), (B, I, M)\}$
- *If* $c = \{A_0, B_0\}$ *then* $a = \{I, M, (I, M), (C, D, O)\}$
- *If* $c = \{A_0, B_0, C_0\}$ *then* $a = \{(D, O), (I, J), (M, J), (M, J, I)\}$
- *If* $c = \{A_0, B_0, C_0, D_0\}$ *then* $a = \{M, O, (I, J), (E, F, G, H, O)\}$
- *If* $c = \{A_0, B_0, C_0, D_0, E_0\}$ *then* $a = \{(O, K), (M, K), (J, I, K), (F, G, H, O)\}$
- *If* $c = \{A_0, B_0, C_0, D_0, E_0, F_0\}$ *then* $a = \{(O, K), (M, N), (N, O), (J, I, K)\}$
- *If* $c = \{A_0, C_0\}$ *then* $a = \{(J, I)\}$

8. RESULT CHECKING

8.1 dissemination final commitment

8.2 dissemination of off-path values

Two cases:

- With signatures
- Without signatures

8.3 verification of inclusion

8.4 collection of authentication codes

8.5 verification of authentication codes

The authentication codes for sensor node s, with either positive or negative acknowledgment message, are defined as follows:

$$MAC_{K_s}(N \mid\mid ACK) \tag{8.1}$$

$$MAC_{K_s}(N \mid\mid NACK) \tag{8.2}$$

K_s is the key that s shares with the base station; ACK, $NACK$ are special messages for positive and negative acknowledgment respectively. The authentication code with ACK message is sent by the sensor node if it verifies its contribution correctly to the root commitment value during the *verification of inclusion* phase and vice versa.

To verify that every sensor node has sent its authentication code with ACK, the base station computes the Δ_{ack} as follows:

$$\Delta_{ack} = \bigoplus_{i=1}^{n} MAC_{K_i}(N||ACK) \tag{8.3}$$

The base station can compute Δ_{ack} as it knows K_s for each sensor node s. Then it compares the computed Δ_{ack} with the received root authentication code Δ_{root} from the root of the aggregation tree. If those two codes match then it accepts the aggregated value or else it proceeds further to find an adversary.

To detect an adversary, the base station needs to identify which nodes in the aggregation tree sent its authentication codes with *NACK* during the verification of inclusion phase. The node who sent authentication code with *NACK* during the verification of inclusion phase is called a *complainer*. We claim that if there is a single complainer in the aggregation tree during the verification of inclusion phase then the base station can find the complainer in linear time. To find a complainer, the base station computes the complainer code c.

$$c := \Delta_{root} \oplus \Delta_{ack} \tag{8.4}$$

Then it computes the complainer code c_i for all node $i = 1, 2, \ldots, n$.

$$c_i := MAC_{K_i}(N \mid\mid ACK) \oplus MAC_{K_i}(N \mid\mid NACK) \tag{8.5}$$

Then it compares c with all c_i one at a time. The matching code indicates the complainer node. The base station needs to do $\binom{n}{1}$ calculations according to Equation 8.5 and same number of comparisons to find a complainer in the aggregation tree. Hence, the base station can find a single complainer in linear time.

Example 8.5.1 *If there are four nodes s_1, s_2, s_3, s_4 in an aggregation tree and their authentication codes with ACK, NACK message in the binary format are defined below.*

$MAC_{K_1}(N \mid\mid ACK) = (1001)_2$; $MAC_{K_1}(N \mid\mid NACK) = (1101)_2$

$MAC_{K_2}(N \mid\mid ACK) = (0110)_2$; $MAC_{K_2}(N \mid\mid NACK) = (1111)_2$

$MAC_{K_3}(N \mid\mid ACK) = (0101)_2$; $MAC_{K_3}(N \mid\mid NACK) = (0111)_2$

$MAC_{K_4}(N \mid\mid ACK) = (0011)_2$; $MAC_{K_4}(N \mid\mid NACK) = (1110)_2$

$\Delta_{root} = (0100)_2$

$\Delta_{ack} = (1101)_2$

$c_1 = (0100)_2$, $c_2 = (1001)_2$, $c_3 = (0010)_2$, $c_4 = (1101)_2$

$c = (1101)_2$ *c is equal to* c_4.

So, the base station identifies that the s_4 *complained, during verification of inclusion phase.*

In general, to find k complainers the base station needs to do $\binom{n}{k}$ calculations and the same number of comparisons to find k complainers.

How XOR is negating the contribution of NACK.

$$\left(\begin{array}{cccc} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ \hline 1 & 0 & 0 & 1 \end{array}\right) \left(\begin{array}{cccc} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ \hline 1 & 0 & 1 & 1 \end{array}\right)$$

The base station receives the following:

$$\left(\begin{array}{cccc} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ \hline 0 & 1 & 0 & 0 \end{array}\right)$$

The base station does the following:

$$\left(\begin{array}{cccc|cccc|cccc|cccc} 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ \hline 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \end{array}\right)$$

$$\left(\begin{array}{cccc} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ \hline 1 & 1 & 0 & 1 \end{array}\right)$$

And concludes that node 4 is complaining.

8.6 Detect an adversary

Algorithm 1 Pseudo algorithm to detect an adversary

1: BS identifies all the complainer and creates $c = \{c_1, c_2, \ldots, c_n\}$
2: **for all** $N \in c$ **do**
3: BS asks N to send data-items with its signature, sent during commitment tree generation phase
4: BS identifies possible adversary based on c and creates $a = \{a_1, a_2, \ldots, a_n\}$
5: **for all** $A \in a$ **do**
6: BS asks A to send data-items with its signature, received and sent by A during commitment tree generation phase
7: If needed BS asks the parent of A to send data-items with its signature
8: BS determines the adversary based on the verification of signatures

Theorem 8.6.1 *Binary commitment tree is optimal in terms of verification as it requires minimum number of off-path values.*

Proof Let us say n is the number of leaves in the given commitment tree.

$\log_3(n) = y$

$3^y = n$

$\log_2(3^y) = \log_2(n)$

$y * \log_2(3) = \log_2(n)$

$\log_3(n) * \log_2(3) = \log_2(n)$

$\log_3(n) = \frac{\log_2(n)}{\log_2(3)}$

$2 * \log_3(n) = [2/\log_2(3)] * log_2(n) = (1.2618) * log_2(n)$

$2 * log_3(n) > log_2(n)$

For the given binary commitment tree, each leaf vertex needs $\log_2(n)$ off-path values in the verification phase. The total off-path values needed in the given commitment tree is $n \cdot \log_2(n)$.

For the given tertiary commitment tree, each leaf vertex needs $2 \cdot \log_3(n)$ off-path values in the verification phase. The total off-path values needed in given commitment tree is $2 \cdot n \cdot \log_3(n)$.

Hence, in totality the binary commitment tree requires the minimum number of off-path values. ■

Theorem 8.6.2 *[17] The Aggregate Commit with verification induces* $O(\log^3 n)$ *edge congestion. And* $O(\delta \log^3 n)$ *node congestion in the aggregation tree.*

Proof While creating the commitment tree every sent message is at most $O(\log n)$ size. And while the off-path value dissemination step is the ■

LIST OF REFERENCES

LIST OF REFERENCES

[1] H.-J. Hof, "Applications of sensor networks," in *Algorithms for Sensor and Ad Hoc Networks*. Springer, 2007, pp. 1–20.

[2] D. Li, K. D. Wong, Y. H. Hu, and A. M. Sayeed, "Detection, classification, and tracking of targets," *Signal Processing Magazine, IEEE*, vol. 19, no. 2, pp. 17–29, 2002.

[3] M. Chu, J. Reich, and F. Zhao, "Distributed attention in large scale video sensor networks," in *Intelligent Distributed Surveilliance Systems, IEE*. IET, 2004, pp. 61–65.

[4] J. D. Lundquist, D. R. Cayan, and M. D. Dettinger, "Meteorology and hydrology in yosemite national park: A sensor network application," in *Information Processing in Sensor Networks*. Springer, 2003, pp. 518–528.

[5] K. Lorincz, D. J. Malan, T. R. Fulford-Jones, A. Nawoj, A. Clavel, V. Shnayder, G. Mainland, M. Welsh, and S. Moulton, "Sensor networks for emergency response: challenges and opportunities," *Pervasive Computing, IEEE*, vol. 3, no. 4, pp. 16–23, 2004.

[6] R. Benenson, S. Petti, T. Fraichard, and M. Parent, "Towards urban driverless vehicles," *International Journal of Vehicle Autonomous Systems*, vol. 6, no. 1, pp. 4–23, 2008.

[7] A. Wang and A. Chandrakasan, "Energy-efficient dsps for wireless sensor networks," *Signal Processing Magazine, IEEE*, vol. 19, no. 4, pp. 68–78, 2002.

[8] M. Ettus, "System capacity, latency, and power consumption in multihop-routed ss-cdma wireless networks," in *Radio and Wireless Conference, 1998. RAWCON 98. 1998 IEEE*. IEEE, 1998, pp. 55–58.

[9] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "Tag: A tiny aggregation service for ad-hoc sensor networks," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 131–146, 2002.

[10] Payload computing. http://en.wikipedia.org/wiki/Payload_(computing). [Online; accessed January 9, 2015].

[11] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "The design of an acquisitional query processor for sensor networks," in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. ACM, 2003, pp. 491–502.

[12] D. Wagner and R. Wattenhofer, *Algorithms for sensor and ad hoc networks: advanced lectures*. Springer-Verlag, 2007.

[13] J. L. Hill and D. E. Culler, "Mica: A wireless platform for deeply embedded networks," *Micro, IEEE*, vol. 22, no. 6, pp. 12–24, 2002.

[14] O. Arazi, I. Elhanany, D. Rose, H. Qi, and B. Arazi, "Self-certified public key generation on the intel mote 2 sensor network platform," in *Wireless Mesh Networks, 2006. WiMesh 2006. 2nd IEEE Workshop on.* IEEE, 2006, pp. 118–120.

[15] Y. Yao and J. Gehrke, "The cougar approach to in-network query processing in sensor networks," *ACM Sigmod Record*, vol. 31, no. 3, pp. 9–18, 2002.

[16] B. Przydatek, D. Song, and A. Perrig, "Sia: Secure information aggregation in sensor networks," in *Proceedings of the 1st international conference on Embedded networked sensor systems.* ACM, 2003, pp. 255–265.

[17] H. Chan, A. Perrig, and D. Song, "Secure hierarchical in-network aggregation in sensor networks," in *Proceedings of the 13th ACM conference on Computer and communications security.* ACM, 2006, pp. 278–287.

[18] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography.* CRC press, 2010.

[19] D. R. Stinson, *Cryptography: theory and practice.* CRC press, 2005.

[20] sha256 standards from nist. http://csrc.nist.gov/groups/STM/cavp/documents/shs/sha256-384-512.pdf. [Online; accessed January 9, 2015].

[21] Digital signature. http://en.wikipedia.org/wiki/Digital_signature#mediaviewer/File:Digital_Signature_diagram.svg. [Online; accessed January 9, 2015].

[22] Routing river. http://www.cse.msu.edu/rgroups/elans/project_files/wsn_project2.html. [Online; accessed January 16, 2015].

[23] D. Wagner, "Resilient aggregation in sensor networks," in *Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks.* ACM, 2004, pp. 78–87.

[24] W. Stallings and L. Brown, *Computer Security.* Pearson Education, 2008, no. s 304.

[25] C. P. Pfleeger and S. L. Pfleeger, *Security in computing.* Prentice Hall Professional Technical Reference, 2002.

[26] M. Bishop, *Introduction to computer security.* Addison-Wesley Professional, 2004.