SECURE DATA AGGREGATION PROTOCOL 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 Computer Engineering

May 2015

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, May 2015. Secure Data Aggregation Protocol for Sensor Networks. Major Professor: Dr. Brian King.

To increase the lifetime of the sensor network by preserving bandwidth we propose a secure in-network data aggregation protocol with internal verification. For doing secure in-network distributed computations, we show an algorithm for securely computing the sum of sensor readings in the network, which we can generalize to any random tree topology computations for any combination of mathematical functions. We represent an efficient way of doing statistical analysis for the protocol. We propose a novel, distributed, interactive algorithm to trace down the adversary and remove it from the network. In the end, we do bandwidth analysis of the protocol and give the proof for the efficiency of the protocol.

1. INTRODUCTION

Mark Weiser of Xerox PARC [1] coined the term **ubiquitous computing** in 1988. It is the scenario in which computing is omnipresent, and particularly in which devices that do not look like computers are endowed with computing capabilities. For example, light switches, door locks, fridges and shoes have data processing power in them. It is the situation in which all these devices are note only capable of computing but also of communicating. The synergy between these devices make the whole worth more than the sum of the parts. A wireless infrastructure looks more plausible for communication in such networks: as happens with mobile telephones, a base station could cover a cell, and a network of suitable positioned base stations could cover a larger area. But we are interested in a broader picture, in which even this arrangement may not always be possible: think of a photographer taking pictures in the desert and whose camera wants to ask the Global Positioning Systems (GPS) unit what coordinates and time stamp to associate with the picture. The computing and the communications may be ubiquitous, but the network infrastructure might not be. In such cases, the devices will have to communicate as peers and form a local network as needed when they recognize each other's presence. This is what we mean by **ad hoc networking**. The wireless network formed by the camera and the GPS receiver is ad hoc in the sense that it was established just for that specific situation instead of being a permanent infrastructural fixture [2]. Kevin Ashton is a British technology pioneer who co-founded the Auto-ID Center at the Massachusetts Institute of Technology, which created a global standard system for RFID and other sensors. He is known for inventing the term **Internet of Things (IoT)** to describe a system where the Internet is connected to the physical world via ubiquitous sensors [3]. IoT means putting all kinds of devices on the Internet and gaining efficiency from it. For example, Uber is a company built around location awareness [4]. An

Uber driver is a taxi driver with the real-time location awareness. An Uber passenger knows when the taxi will show up. It is about eliminating slack time and worry. It connects passengers, taxi drivers, smart phones and GPS in a way that it provides an efficient and flexible way of transportation. In addition to, the trend in consumer electronics is to embed a microprocessor in everything-cellphones, car stereos, televisions, watches, GPS receivers, digital cameras. In some specific environments such as avionics, electronic devices are already becoming networked; in others, work is underway. Medical device manufacturers want instruments such as thermometers, heart monitors and blood oxygen meters to report to a nursing station; kitchen appliance vendors envisage a future in which the oven will talk to the fridge, which will reorder food over the Internet. It is to be expected that, in the near future, this networking will become much more general. The next step is to embed a short range wireless transceiver into everything; then many gadgets can become more useful and effective by communicating and cooperating with each other. In such scenario, each device by becoming a network node, may take advantage of the services offered by other nearby devices instead of having to duplicate their functionality [2]. At the core of ubiquitous computing and IoT, are the sensors generating data and wireless network providing transmission medium for communications. Collectively, we refer these concepts as **Sensor Networks** which enable economically viable solutions to a variety of applications.

In Sensor networks, the sensors collect raw data, and the data is processed by more powerful machines which converts the raw data into the information. Based on the derived information an important action is taken. The error at any stage in the process can create catastrophic situations. For example, speed sensor failure led to crash of Air France flight - Airbus A330-203 AF 447 on 1st June 2009. France's Bureau of Investigation and Analysis (BEA) released the Airbus final report [30]. According to an official report, the pilots could not reclaim control as the plane dropped out of the sky at a rate of $10,000$ feet per minute. The findings from the flight's black boxes, which were found intact at the bottom of the Atlantic in early May, and their

analysis paints a harrowing picture of Air France flight 447's literal dropping out of the sky. The co-pilots encountered trouble with the speed sensors four hours and 10 minutes into the flight. The flight was on autopilot as the pilot in command took a routine rest out of the cockpit. They were knowingly headed into a turbulent and storm-ridden spot over the Atlantic, and the black boxes show the pilots attempted to maneuver around the storm slightly. For nearly a minute, as the speed sensors jumped, the pilot was not present in the cockpit. By the time the pilot returned, the plane had started to fall at 10,000 feet per minute while violently rolling from side to side. The plane's speed sensors never regained normal functionality as the plane began its three-and-a-half minute freefall. The flight plunged into the Atlantic nose-up, killing all 228 on board. The findings coincide with investigators' earlier theory that the sensors, known as pitot tubes, malfunctioned, possibly because of ice at such a high altitude. In addition to, the small error in the system design could result in significant bad consequences. For example, the recent data breach attack on the company Anthem, exposed Social Security numbers and other sensitive details of 80 million customers. It is considers as one of the biggest thefts of medical related customer data in U.S. history [6].

In sensor networks, sensors are so blended into the physical world that they are hard to distinguish from the physical objects. The sensors may disappear so well that users lose not just the control but even awareness of what is actually going on. While it is a welcome strategy to hide irrelevant complexity, it can easily become a liability. This is a serious problem, if you can't tell what your computer is doing when you are online, how can you be sure it's not transmitting your documents to a malicious destination? And if you already can't tell with your visible computer, how can you hope to tell with the invisible ones inside your appliances? Weiser [5] acknowledges: "If the computational system in invisible as well as extensive, it becomes hard to know what is controlling what, what is connected to what, where information is flowing, how it is being used, what is broken and what are the consequences of any given action?". The networks where computing is invisible, and it is unclear who is responsible for

what, detecting an adversary is a challenging task. We think that the detecting an adversary is necessary in sensor networks for countermeasures (a posteriori remedies). It is essential for the longevity of the sensor network. Hence, we focus on designing the protocol which helps detect an adversary in the sensor networks.

2. SENSOR NETWORKS BACKGROUND

Sensor networks are becoming ubiquitous in our day to day life. It is the vision of a world in which computing power and digital communications are extremely inexpensive commodities. So, cheap that they are embedded in all of the everyday objects that surround us [2]. In this chapter we introduce sensor networks and its applications, we define basic terminologies and then we discuss major barriers to achieve security in sensor networks.

2.1 Applications

In sensor networks, thousands of sensor nodes may interact with the physical world and collectively monitor an area, generating a large amount of data to be transmitted and reasoned about. With the recent advances in hardware technologies of sensors, we can use tiny and cheap sensor nodes to obtain significant amount of useful data about physical world. 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 [7]. 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 [8]. For example, *Palo Alto Research Center*

tries to spot "interesting" vehicles (the vehicles marked specially) using motes equipped with microphones or steerable cameras [9]. 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 Sensor networks can be used to monitor a geographical location. For example, *Meteorology and Hydrology in Yosemite National Park* [10], 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. Sensor networks can be very useful in such situations as they can operate with little or no human intervention.

Health Care Sensors can be used to monitor the patients around the clock. They can report various statistics to the doctors and nurses regarding patients health. Also, it can send reminders to them to take care of the patient periodically. The most important criteria for the such networks are security and reliability. Because based on the sensor readings, doctors decide what treatment or what medicine to prescribe to the patients. If those readings are modified by an adversary then the consequences might be lethal to the patients.

Sustainable Mobility Sensor networks can be used to build digitally connected and coordinated vehicles. With the driver less cars from companies like Google, autonomous vehicle systems seems the future of transportation. Autonomous vehicle systems [11] describes how various various technologies in addition to the sensor networks is used in making the sustainable mobility.

The Active Floor is a system which can detect person walking on the floor using pressure sensors [12]. It also allows to extract information from the raw

sensor data which has reasonable success in identifying people based on their gait. Authors used Hidden Markov Models based analysis techniques originally developed for speech and face recognition to identify people.

Code Blue is a system to enhance emergency medical care with the goal to have a seamless patient transfer between a disaster area and a hospital [13].

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

2.2 Sensor Node Architecture

Sensor networks consists of an individual sensor node and the construction of each node depends on the application. The major components of the sensor nodes are shown in Figure 2.1 [14].

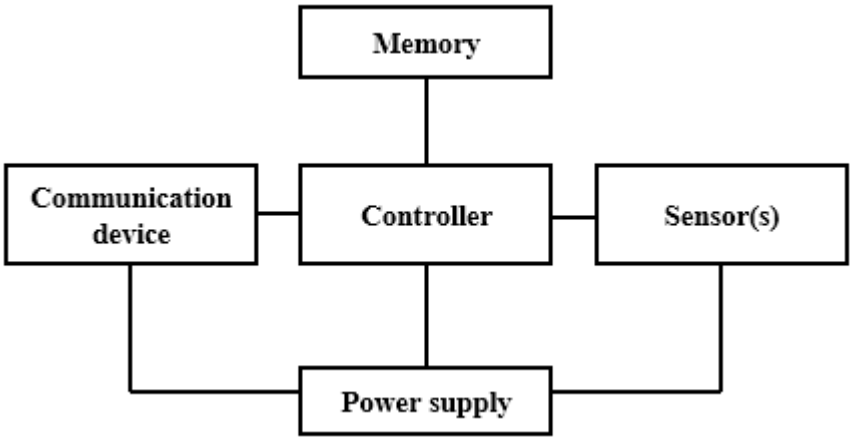

Fig. 2.1.: Sensor Node Architecture

Controller is the Central Processing Unit (CPU) of the node. It collects data from the sensors, processes this data, decides when and where to send it, receives data

from other sensor nodes, and decides on the actuators behavior. It executes various programs, ranging from time-critical signal processing and communication protocols to application programs.

Communication Device is used to exchange data between individual nodes at radio frequency. Radio Frequency (RF)-based communication is widely used in sensor networks as it best fits the requirements of numerous applications. It provides relatively long range and high data rates, acceptable error rates at reasonable energy expenditure, and does not require line of sight between sender and receiver. Both a transmitter and a receiver are required in a sensor node. The essential task for the devices is to convert a bit stream coming from a microcontroller (or a sequence of bytes or frames) and convert them to and from radio waves. It is convenient to use a device that combines these two tasks in a single entity, called transceivers. Usually, half-duplex operation is realized since transmitting and receiving at the same time on a wireless medium is impractical in most cases.

Sensors can be roughly categorized into three categories. *Passive, omnidirectional sensors* can measure a physical quantity at the point of the sensor node without actually manipulating the environment by active probing in this sense, they are passive. Moreover, some of these sensors actually are self-powered in the sense that they obtain the energy they need from the environment energy is only needed to amplify their analog signal. There is no notion of "direction" involved in these measurements. Typical examples for such sensors include thermometer, light sensors, vibration, microphones, humidity, mechanical stress or tension in materials, chemical sensors sensitive for given substances, smoke detectors, air pressure, and so on. *Passive, narrow-beam sensors* are passive as well, but have a well-defined notion of direction of measurement. A typical example is a camera, which can "take measurements" in a given direction, but has to be rotated if need be. *Active sensors* are the last group of sensors actively probes

the environment, for example, a sonar or radar sensor or some types of seismic sensors, which generate shock waves by small explosions. These are quite specific - triggering an explosion is certainly not a lightly undertaken action and require quite special attention.

Power Supply of Sensor Nodes is a crucial system component which has two main aspects. First, storing energy and providing power in the required form; second, attempting to replenish consumed energy by "scavenging" it from some node-external power source over time. Storing power is conventionally done using batteries, the normal AA battery stores about 2.22.5 Ah at 1.5 V. To ensure truly long-lasting nodes and wireless sensor networks, such a limited energy store is unreliable. We can use energy from a nodes environment "energy scavenging". For example, *Vibrations* is almost pervasive form of mechanical energy: walls or windows in buildings are resonating with cars or trucks passing in the streets, machinery often has low frequency vibrations, ventilations also cause it, and so on. The available energy depends on both amplitude and frequency of the vibration. *Pressure variations* is similar to vibrations, a variation of pressure can also be used as a power source. Such piezoelectric generators are in fact used already. One well-known example is the inclusion of a piezoelectric generator in the heel of a shoe, to generate power as a human walks about [15]. This device can produce, on average, 330 $\mu W/cm^2$. *Photovoltaics* effect is used in the solar cells can be used to power sensor nodes. *Temperature gradients* effect utilizing the differences in temperature can be directly converted to electrical energy.

The Memory Component is fairly straightforward. Random Access Memory (RAM) is used to store intermediate sensor readings, packets from other nodes, and so on. While RAM is fast, its main disadvantage is that it loses its content if power supply is interrupted. Program code can be stored in Read-Only Memory

(ROM). Correctly dimensioning memory sizes, especially RAM, can be crucial with respect to manufacturing costs and power consumption.

2.3 Energy Consumption

The sensor network's lifetime can be maximized by minimizing the power consumption of communication devices (or transreceiver) of the sensor nodes. The transreceiver is responsible for all the wireless communications between nodes. To estimate the power consumption, we have to consider the communication and computation power consumption. The transreceiver's energy dissipation depends on two main parameters [16]. The first is $E_{elec}(J/b)$, the energy dissipated to run the transmit or receive electronics. The second is $\varepsilon_{amp}(J/m^2b)$, the energy dissipated by the transmit power amplifier to achieve an acceptable signal to 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 [17]. 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 $\varepsilon_{amp} = 100pJ/m^2b$.

Transreceiver can be put into different states to save energy [14]. It can be in either transmit or receive state and energy consumption of those states are describe above. It can be in Idle state where it is ready to receive packet but is not currently receiving anything. It can be in Sleep state where majority of the parts are switched off, and is not able to receive immediately. To sustain the sensor network for longer times 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 [16] 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.

2.4 Resource Constraints

We introduce the parameters which should be taken into account while designing secure protocol for sensor networks. These parameters can constrain the protocol designer's choices within the protocol.

2.4.1 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, making the sensor readings unreliable and irrelevant . This attack becomes more damaging when multiple adversaries succeeds in injecting false data into the network which may cause catastrophic consequences [18].

2.4.2 Hardware Limitations

As far as we know, one of the first hardware platform for developing sensor network application is MICA [19] developed by University of Berkeley. Another popular platform is Mote from Intel [20]. 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 Texas Instruments are shown in Table 2.1. 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. These hardware limitations can constrain protocol designer's choice of algorithms for applications.

	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 CC2538 from Texas Instruments implementing IEEE 802.15.4 standards

2.4.3 Transmission Medium

In sensor networks, a group of sensor nodes (or processors) communicate over the radio (e.g., Bluetooth, WLAN). Traditionally, wireless mediums have issues due to synchronization, hidden station and expose station terminal problems, distributed arbitration, directional antennas, bandwidth limitations, higher error rate, security, scalability etcetera. For example, wireless networks have approximately 10^6 times

higher bit error rate (BER) than wired networks 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 transmission medium with constrained resources makes it challenging to design the reliable networking protocol as we have to consider all the possible retransmissions.

2.4.4 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 while doing the quality of service in the network.

All these parameters combined contributes to making strong assumptions on the network topology while designing the protocol for sensor networks.

3. DATA AGGREGATION BACKGROUND

Data aggregation is an essential paradigm for wireless routing in sensor networks [21]. The idea is to compress the data coming from different sources enroute eliminating redundancy, minimizing the number of transmissions and thus saving energy. This paradigm shifts the focus from the traditional address centric approaches for networking (finding short routes between pairs of addressable end nodes) to a more data-centric approach (finding routes from multiple sources to a single destination that allows in-network consolidation of redundant data).

One of the fundamental and indispensable functionalities of sensor networks is the ability to answer queries concerning the data acquired by the sensors. For example, in Figure 3.1 [22] the base station who initiates the query to the sensor network might be at the end of the river. The sensor nodes may lie on both sides of the river and they have to response to the queries of the base station. The transreceiver module in the sensor nodes has a limited transreceiving range. So, the sensor nodes cannot communicate to the base station in peer-to-peer fashion. The sensor nodes have to communicate via hop-by-hop to the base station. This resource constraint forces us to design distributed protocol for sensor networks.

3.1 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. The wireless data communications between nodes consume a large portion of the total energy consumption. In-network data aggregation techniques allow the sensor nodes to aggregate the data before sending it to their parents. This technique reduces the wireless communication happening between the nodes and the overall

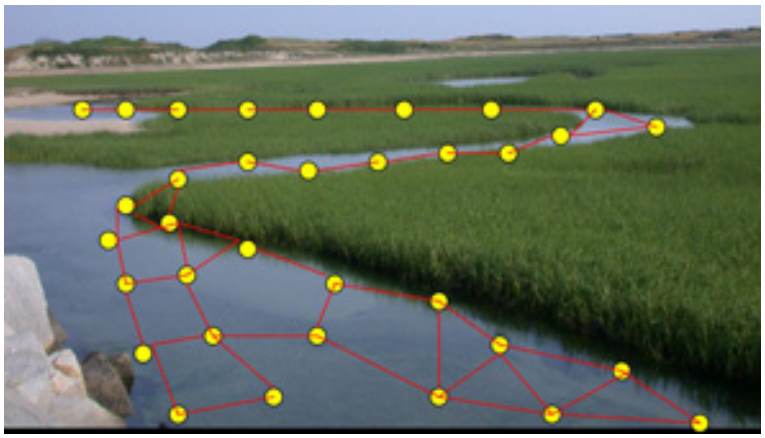

Fig. 3.1.: Routing River

energy consumption in the network. For example, in-network data aggregation of the SUM function can be performed as follows. All the intermediate sensor nodes in the network, receives the sensor readings from all of their children. They aggregate all those readings by applying the summation function to those readings. A network, which has the star topology, shown in Figure 3.2, the root of the tree receives the following sensor readings $S_1(10), S_2(14), S_3(12), S_4(15), S_5(11), S_6(17)$. The root has its own reading of $S_0(15)$. The root node aggregates these seven readings and creates an aggregated result as follows:

$$S = \sum_{i=0}^{6} S_i \tag{3.1}$$

Now, instead of sending all those sensor readings one at a time to its parent, it can send one aggregated sensor reading. In this example the aggregate function is summation, but it can be replaced with other statistical functions such as average, median, count etc., with little or no modification.

According to [23], **data aggregation** can be defined as follows. Consider a sensor network with n sensors collecting data, the data will be propagated in some fashion to the base station. At the base station, the data is aggregated and processed into information. This can be represented as $f(x_1, x_2, ..., x_n)$ where $x_1, x_2, ..., x_n$ represent the sensor readings. Here f is some mapping $f : \mathcal{D}_1 \times \mathcal{D}_2 \times ...\mathcal{D}_n \rightarrow I$, where $\mathcal{D}_i$

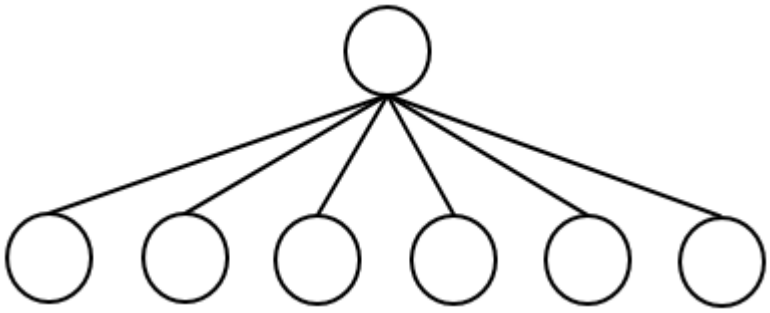

Fig. 3.2.: Star Network

represents sensor i's domain and I represents the set of all possible information. Thus the goal is to compute the information as follows:

$$y = f(x_1, x_2, ..., x_n) \tag{3.2}$$

It is shown that the energy savings achieved by in-network data aggregation are significant [24]. The in-network data aggregation approach requires the sensor nodes to do more computations. But studies have shown that wireless communication of data transmission requires more energy than local computation of the data. In-network data aggregation is an efficient and a widely used approach for saving bandwidth by doing less wireless 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.

Definition 3.1.1 *[25]* ***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 3.1.2 *For a given node* ***Information rate*** *is the ratio of the number of sent payloads over the received payloads.*

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.

3.2 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:

$$Node\ congestion = Edge\ congestion \cdot Fanout \tag{3.3}$$

Congestion is a useful factor while analyzing sensor network as it measures how quickly the sensor nodes will exhaust their batteries [26]. 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 as Definition 3.1.1 to its parent in the aggregation tree. In this approach, the fanout (δ) depends on the given aggregation tree. For example, in the aggregation tree shown in Figure 3.2, with n nodes, organized in the star tree topology, we see the congestion is $O(n)$. For the network organized in the palm tree topology, as shown in Figure 3.3, the congestion is $\theta(1)$. 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.

Even though network topology can be random, in our observation, palm tree, star tree, binary tree are widely used network topologies to do the performance analysis. In the following chapters, we describe how to run our protocol for any random aggre-

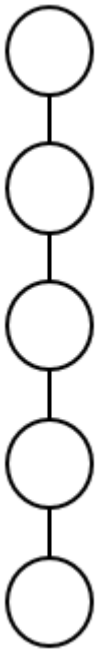

Fig. 3.3.: Palm Tree Network Topology

gation tree. In the end, we do bandwidth analysis of the protocol for star, palm and binary trees.

3.3 Resilient And Non-Resilient Aggregate Functions

Wagner [27] uses statistical estimation theory to quantify the effects of direct data injection on various aggregates functions as follows:

Definition 3.3.1 *[28] 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 $f : \mathbb{R}^n \rightarrow \mathbb{R}$ 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):

$$rms(f) := \mathbb{E}[(\widehat{\Theta} - \theta)^2|\theta]^{1/2} \tag{3.4}$$

Note that r.m.s(f) is a function of θ; we compare functions pointwise. Also, an unbiased estimator is one for which $E[\widehat{\Theta}|\theta] = \theta$ for all θ.

Wagner in [27] 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 r.m.s error associated with A as follows:

$$rms^*(f, A) := \mathbb{E}[(\widehat{\Theta}^* - \theta)^2|\theta]^{1/2} \tag{3.5}$$

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 $rms^*(f, A) \gg rms(f)$, 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

$$rms^*(f, k) := max\{rms^*(f, A) : A \; is \; a \; k - node \; attack\} \tag{3.6}$$

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 3.3.2 *[27] 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. 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. Wagner's work is summarized in the following Table.

Table 3.1.: Summary of Wagner's work

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

3.4 Security Issues

The aggregation schemes can be compared with compression schemes. "Lossless data compression" [29] produces a compressed file from which the original data can be recovered exactly. Facsimile uses lossless data compression. However, lossless data compression schemes are limited in the compression rates they can achieve. "Lossy data compression" [29] schemes produces a compressed file from which only an approximation to the original information can be recovered. Much higher compression ratios are possible. The aggregation scheme will be a *lossy data compression* because once the base station receives the final aggregated value it can not create the original

sensor readings of the sensor nodes. Hence, it is very important that the base station has very high confidence in the received final aggregated value. For example, in Equation 3.2 Sensor II's reading is changed from x_2 (the "true reading") to x_2' by an intermediate aggregator, then an aggregate node computes $y' = f(x_1, x_2', ..., x_n)$. It is very likely that $y' \neq y$ where y is the true information if the true reading was counted. The base station takes an action based on the received information y'. How dangerous would it be to act using y'? Thus, if one "knows there is some likelihood of false data" how do we protect the information generated?

As we know, in-network data aggregation technique saves bandwidth by transmitting less payloads between sensor nodes thus increasing the lifetime of the network. Designing one such protocol for the sensor networks poses a numerous challenges due to resource limitations and inherent characteristics discussed in the previous chapter. Moreover, this technique empowers intermediate aggregate sensor nodes in the network by allowing them to control certain portion of the network. 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. Intermediate sensor nodes adversarial power is proportional to their number of descendants.

While applying the data-aggregation techniques the integrity of the sensor readings becomes more valuable. As one or few faulty sensor readings can deviate the final aggregated result. For example, National Snow and Ice Data Center on June 3, 2008 [31] reported that Arctic sea ice extent had declined through the month of May as summer approaches. Daily ice extents in May continued to be below the long-term average and approached the low levels seen at the same time last year. The spring ice cover was thin, one sign of thin and fairly weak ice was the formation of several polynyas in the ice pack. The Defense Meteorological Satellite Program (DMSP)

F13 [32] satellite, which had been central to their Arctic sea ice analysis for the past several years, is nearing the end of its mission. As the standard data practice, they have transitioned to a newer sensor, in that case the DMSP F15 [33]. The DMSP F15 had the same type of sensor as the DMSP F13. NSIDC had done preliminary inter calibration to assure consistency with the historical record. They said that due to the inter calibration errors the final reported ice extent values might differ on average $\pm 30,000$ square kilometers or $11,600$ square miles per preliminary number reported. The faulty sensor readings can be caused due to malfunctioning sensors, the active attacks on true sensor readings or incorrect interpretations of the data, which can cause catastrophic situations. Hence, it is necessary to take countermeasures while using data-aggregation techniques.

Despite the fact that in-network aggregation makes an intermediate sensor nodes more powerful and aggregated value more vulnerable to various security attacks, some aggregation approaches requires strong network topology assumptions or honest behaviors from the sensor nodes. For example, in-network aggregation schemes in [26, 34] assumes that all the sensor nodes in the network are honest. Secure Information Aggregation (SIA) of [35] enables 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. The SIA provides probabilistic security for the network topology with a single-aggregate model.

An approach to detection and diagnosis of multiple failures in a dynamic system is proposed in [36]. It is based on the Interacting Multiple-Model (IMM) estimation algorithm, which is one of the most cost-effective adaptive estimation techniques for systems involving structural as well as parametric changes. The proposed approach provides an integrated framework for fault detection, diagnosis, and state estimation. It is able to detect and isolate multiple faults substantially more quickly and more reliably than many existing approaches. Its superiority is illustrated in two aircraft

examples for single and double faults of both sensors and actuators, in the forms of total, partial, and simultaneous failures.

Secure hierarchical in-network aggregation (SHIA) in sensor networks [28] 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. We build our work on the foundation of SHIA, which allows to track down the adversary and remove it from the network.

4. NETWORKING AND CRYPTOGRAPHY TOOLS

The word cryptography means "secret writing". It is the art and science of concealing meaning. Cryptanalysis is the science breaking of cryptography schemes. Formally, the basic component of cryptography is a cryptosystem [37].

Definition 4.0.1 *A cryptosystem is a* 5*-tuple* $(\mathcal{E}, \mathcal{D}, \mathcal{M}, \mathcal{K}, \mathcal{C})$*, where* $\mathcal{M}$ *is the set of plaintexts,* $\mathcal{K}$ *is the set of keys,* $\mathcal{C}$ *is the set of ciphertexts,* $\mathcal{E} : \mathcal{M} \times \mathcal{K} \to \mathcal{C}$ *is the set of enciphering functions, and* $\mathcal{D} : \mathcal{C} \times \mathcal{K} \to \mathcal{M}$ *is the set of deciphering functions.*

4.1 Symmetric Key Encryption

Consider an encryption scheme consisting of the sets of encryption and decryption transformations $\{E_e : e \in \mathcal{K}\}$ and $\{D_d : d \in \mathcal{K}\}$, respectively, where $\mathcal{K}$ is the key space. The encryption scheme is said to be *symmetric-key* if for each associated encryption/decryption key pair (e, d), it is computationally "easy" to determine d given e, and to determine e from d [38]. Moreover, most symmetric-key schemes satisfy $e = d$.

A two party communication using symmetric key encryption can be described by the block diagram of Figure 4.1. One major issue with symmetric key system is to find an efficient method to agree upon and exchange keys securely, which is known as *key-distribution problem* [38].

4.2 Asymmetric/Public Key Encryption

Let $\{E_e : e \in \mathcal{K}\}$ be a set of encryption transformations, and let $\{D_d : d \in \mathcal{K}\}$ be the set of corresponding decryption transformations, where $\mathcal{K}$ is the key space. Consider any pair of associated encryption/decryption transformations (E_e, D_d) and

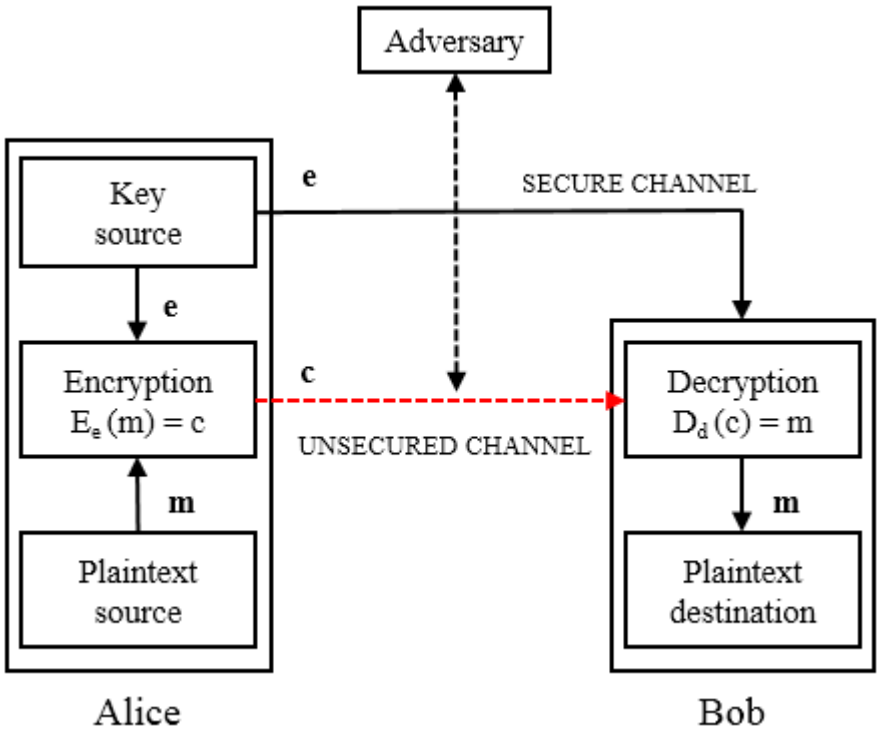

Fig. 4.1.: Two party communication using encryption, with a secure channel for key exchange. The decryption key d can be efficiently computed from the encryption key e.

suppose that each pair has the property that knowing E_e it is computationally infeasible, given a random ciphertext $c \in \mathcal{C}$, to find the message $m \in \mathcal{M}$ such that $E_e(m) = c$. This property implies that given e it is infeasible to determine the corresponding decryption key d. This is unlike symmetric key ciphers where e and d are essentially the same [38].

Consider the two party communication between Alice and Bob illustrated in Figure 4.2. Bob selects the key pair (e, d). Bob sends the encryption key e (called the *public key*) to Alice over any channel but keeps the decryption key d (called the *private key*) secure and secret. Alice may subsequently send a message m to Bob and applying the encryption transformation determined by Bob's public key to get $c = E_e(m)$. Bob decrypts the ciphertext c by applying the inverse transformation D_d uniquely

determined by d. Here the encryption key is transmitted to Alice over an unsecured channel. Since, the encryption key e need not be kept secret, it may be made public.

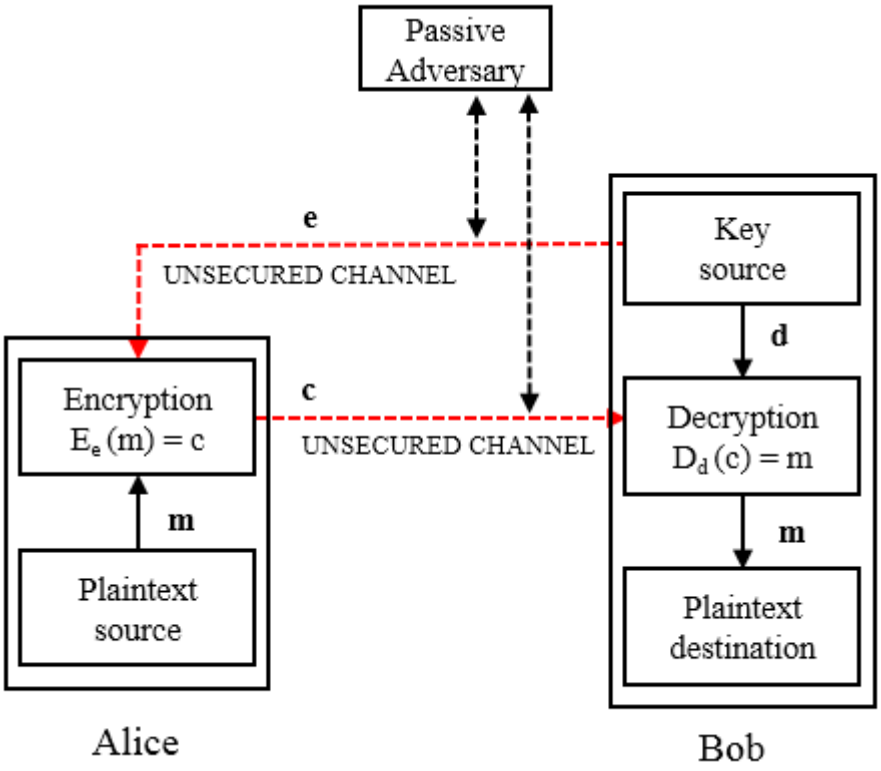

Fig. 4.2.: Encryption using public key techniques

4.3 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 essential mathematical tools to achieve data integrity. A hash function h should have the following properties:

Compression A hash function h maps an input x of arbitrary finite bitlength, to an output $h(x)$ of fixed bitlength n.

Ease of computation For 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 $h(x) = h(x')$.

SHA-256, is a 256-bit hash and provides 128 bits of security against collision attacks [39]. And 128 bits security is adequate for most of the applications.

4.4 Message Authentication Codes

A Message Authentication Code (MAC) is a family of hash functions parameterized by a secret key k, also known as keyed hash function (h_k). It has the following properties:

Ease of computation For a known function h_k, given a value k and an input x, $h_k(x)$ is easy to compute. This result is called MAC.

Compression The function h_k maps an input x of arbitrary finite bitlength to an output $h_k(x)$ of fixed length n.

Computation-resistance Given a description of the function family h, for every fixed allowable value of k (unknown to an adversary), given zero or more text-MAC pairs $(x_i, h_k(x_i))$, it is computationally infeasible to compute any text-MAC pair $(x, h_k(x))$ for any new input $x \neq x^{'}$ (including possibly for $h_k(x) = h_k(x_i)$ for some i). If computation-resistance does not hold, a MAC algorithm is subject to MAC-forgery.

4.5 Digital Signatures

A digital signature is a cryptographic scheme for demonstrating the authenticity of a digital message. A valid digital signature gives a recipient strong 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 $\mathsf{Gen} : N \to \mathcal{K} \times \mathcal{K}'$
5. an efficient signing algorithm $\mathsf{Sign} : \mathcal{M} \times \mathcal{K} \to \mathcal{S}$
6. an efficient verification algorithm $\mathsf{Verify} : \mathcal{S} \times \mathcal{M} \to \{\text{true, false}\}$

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:

$$s = \mathsf{Sign}_{s_k}(m) \tag{4.1}$$

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:

$$\mathsf{Verify}_{p_k}(m, s) = \begin{cases} \textbf{true} \text{ with probability of 1} & \text{if } s = \mathsf{Sign}_{s_k}(m) \\ \textbf{false} \text{ with overwhelming probability} & \text{if } s \neq \mathsf{Sign}_{s_k}(m) \end{cases} \tag{4.2}$$

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. In Figure 4.3, we show the steps for signing and verifying the hashed message [40]. The message is hashed

before its being signed to reduce the message size. If the message is not hashed before signing then the signature can be longer than the message which is problematic for the longer messages.

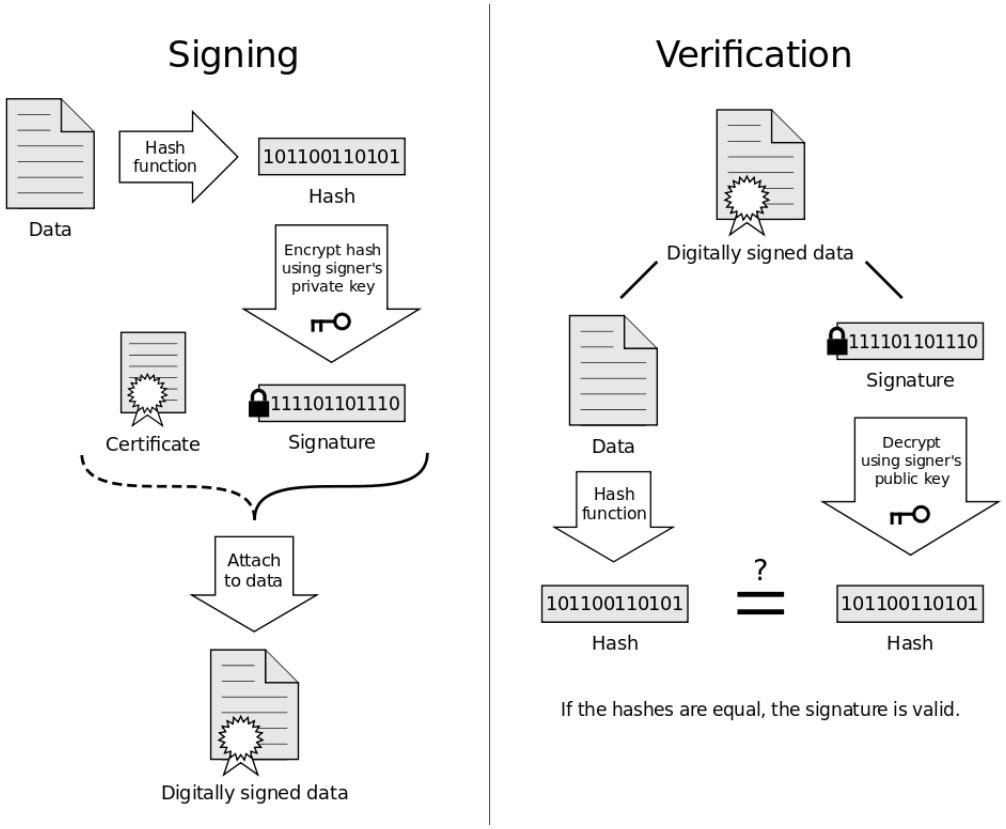

Fig. 4.3.: Signing and verification of digital signatures

In the Digital Signature scheme only the owner of the secret key can generate a valid signature. The digital signature is easily verified by other parties as long as they know the public key. The digital signature is not only tied to the signer but also to the message that is being signed. Digital signatures do not encrypt the message. However, if necessary, a signed message can be encrypted after it is signed.

4.6 Summary

Three different integrity-protection mechanisms HASH, MAC, Signature can be summarized in a matrix like Table 4.1 [2]. The main difference between the various primitives stems from identifying who can generate the code and who can verify it. This determines the suitability of the construction for a given purpose.

	Who can generate it	Who can verify it
Hash	Everyone	Everyone
MAC	Holders of secret	Holders of secret
Signature	Holder of secret	Everyone

Table 4.1.: A comparison of integrity-protecting primitives

5. DATA AGGREGATION WITHOUT INTERNAL VERIFICATION

Secure hierarchical in-network aggregation (SHIA) [28] presents the first and provably secure sensor network data aggregation protocol without internal verification. It is designed for general networks with single or multiple adversaries. Our work enhances SHIA, by making it communication efficient, adds new security services to the protocol, achieves similar security goals with non-resilient aggregation functions and efficient ways of analyzing the protocol. In this chapter, we summarize the important parts of SHIA and relevant terms, to build the foundation to describe our protocol in the following chapters.

5.1 Network Assumptions

We assume a multi hop network with a set $S = \{S_1, ..., S_n\}$ of n sensor nodes where all sensor 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 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 are peer-to-peer and we do not consider the local wireless broadcast. Each sensor node S has a unique ID and shares a unique secret symmetric key K_S with the base station. The keys enable message authentication between sensor node and the base station. And it also allows 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.

5.2 Attacker Model And Security Goal

We consider a model with a polynomially bounded adversary [35], 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. In such a case, the base station receives the inaccurate aggregated data. Since in many applications the information received by the base station provides a basis for critical decisions, false information could have ruinous consequences as described in Section 3.4. 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** [35], 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. 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.

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 5.2.1 *[28] An aggregation algorithm is **optimally secure** if, by tampering with the aggregation process, an adversary is unable to induce the base station to accept any aggregation result which is not already achievable by direct data injection.*

5.3 The SUM Aggregate Algorithm of SHIA

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 reading of the sensor node i. The algorithm has the following 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.

5.4 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) [24]. 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.

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. The fresh nonce is used for each

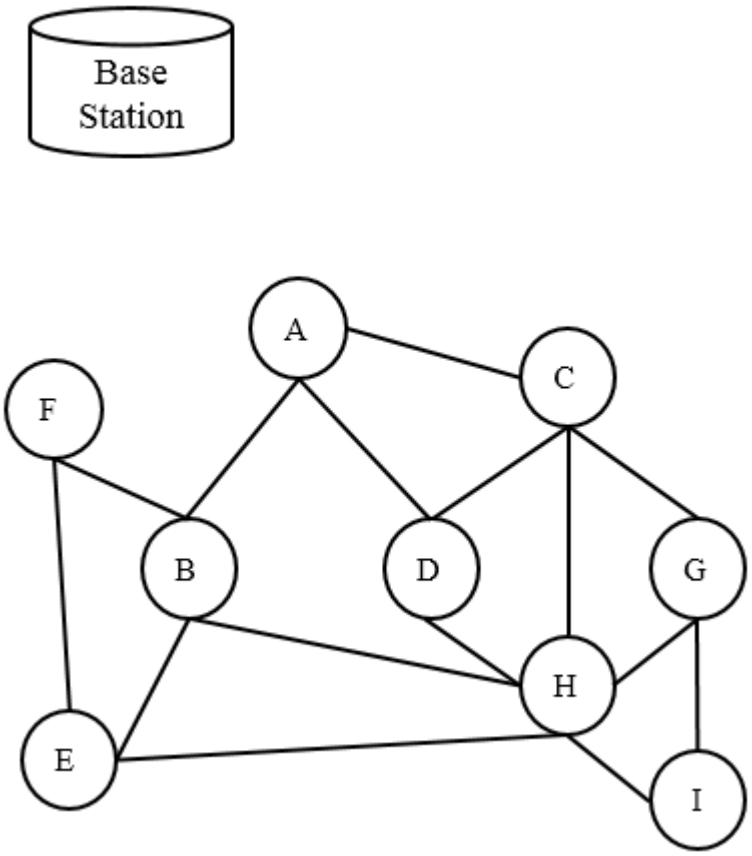

Fig. 5.1.: Network graph

new query to prevent replay attacks in the network. 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.

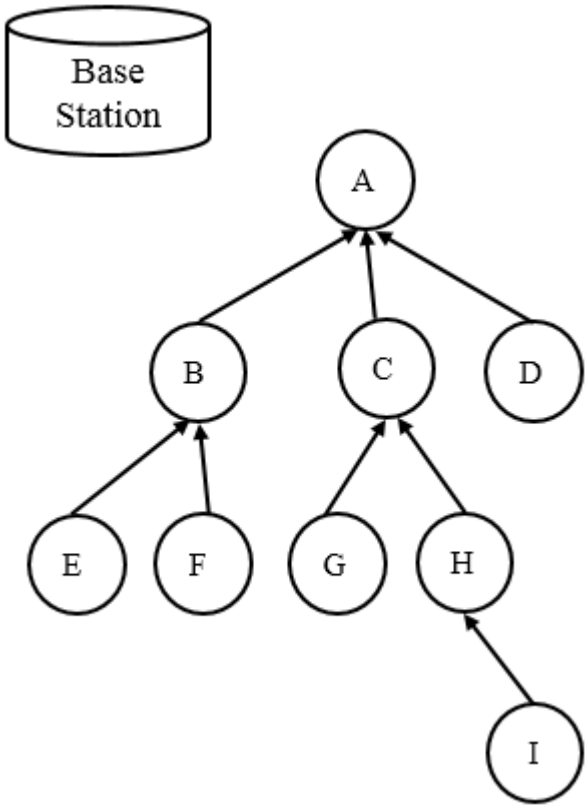

Fig. 5.2.: Simplified Aggregation tree for network graph in Figure 5.1

5.5 Aggregate Commit

The aggregate commit phase constructs cryptographic commitments to the data values and to the intermediate aggregated data-values. 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. The commitment

tree is a logical tree built on top of an aggregation tree. The commitment tree is defined as follows:

Definition 5.5.1 *[28] 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.

The sensor node A creates the label for the leaf vertex A_0 is given as follows :

$$A_0 = < A_{count}, A_{value}, \overline{A_{value}}, A_{commitment} >$$

where A_{value} is the sensor data collected by A, $\overline{A_{value}} = r - A_{value}$ with r being the upper bound on allowable data values. The values of $A_{count} = 1$, and $A_{commitment} = A_{id}$ which is unique ID of A as A_0 is the leaf vertex of A. Internal vertices in the commitment tree represent aggregation operations, and have labels that are defined based on their children. Suppose an internal node I has child vertices with the following labels : $A_1, A_2, \ldots, A_q$, where $A_i = < A_{icount}, A_{ivalue}, \overline{A_{ivalue}}, A_{icommitment}>$. The label for the internal vertex I_j is given as follows:

$$I_j = < I_{jcount}, I_{jvalue}, \overline{I_{jvalue}}, I_{jcommitment} >$$
$$I_{jcount} = \sum A_{icount}$$
$$I_{jvalue} = \sum A_{ivalue}$$
$$\overline{I_{jvalue}} = \sum \overline{A_{ivalue}}$$
$$I_{jcommitment} = H[N||I_{jcount}||I_{jvalue}||\overline{I_{jvalue}}||A_1||A_2||\ldots||A_q]$$

The word vertex is used for the node in the commitment tree and the node is used for the node in the aggregation tree. There is a mapping between the node 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 local computations and wireless communications. The commitment field in the label is calculated using collision resistant hash function which makes impossible for an adversary to change the commitment structure once it is sent to the base station.

5.5.1 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. Conceptually, a commitment tree is a logical tree built on top of an aggregation tree, with additional aggregate information attached to the nodes. The commitment tree for the aggregation tree of Figure 5.2 is shown in Figure 5.3 . In this example, the node B receives E_0 from E, F_0 from F and has its own leaf vertex B_0. The node B aggregates all three vertices by creating a new vertex B_1 and sends B_1 to its parent node A. The label of an internal vertex B_1 is defined as follows:

$$B_1 = < 3, B_{1value}, \overline{B_{1value}}, H[N||3||B_{1value}||\overline{B_{1value}}||B_0||E_0||F_0] > .$$

The labels of the vertices on the path of node I to the root for the commitment tree A are given as follows:

$I_0 = < 1, I_{value}, \overline{I_{value}}, I_{id} >$
$H_1 = < 2, H_{1value}, \overline{H_{1value}}, H[N||2||H_{1value}||\overline{H_{1value}}||H_0||I_0] >; H_{1value} = I_{value} + H_{value}$
$B_1 = < 3, B_{1value}, \overline{B_{1value}}, H[N||3||B_{1value}||\overline{B_{1value}}||B_0||E_0||F_0] >;$
$B_{1value} = B_{value} + E_{value} + F_{value}.$
$C_1 = < 4, C_{1value}, \overline{C_{1value}}, H[N||4||C_{1value}||\overline{C_{1value}}||C_0||G_0||H_1] >$
$C_{1value} = C_{value} + G_{value} + H_{1value}$
$A_1 = < 9, A_{1value}, \overline{A_{1value}}, H[N||9||A_{1value}||\overline{A_{1value}}||A_0||D_0||B_1||C_1] >$
$A_{1value} = A_{value} + D_{value} + B_{1value} + C_{1value}.$

In the naive approach the information rate is $1/n$ where n is the number of children

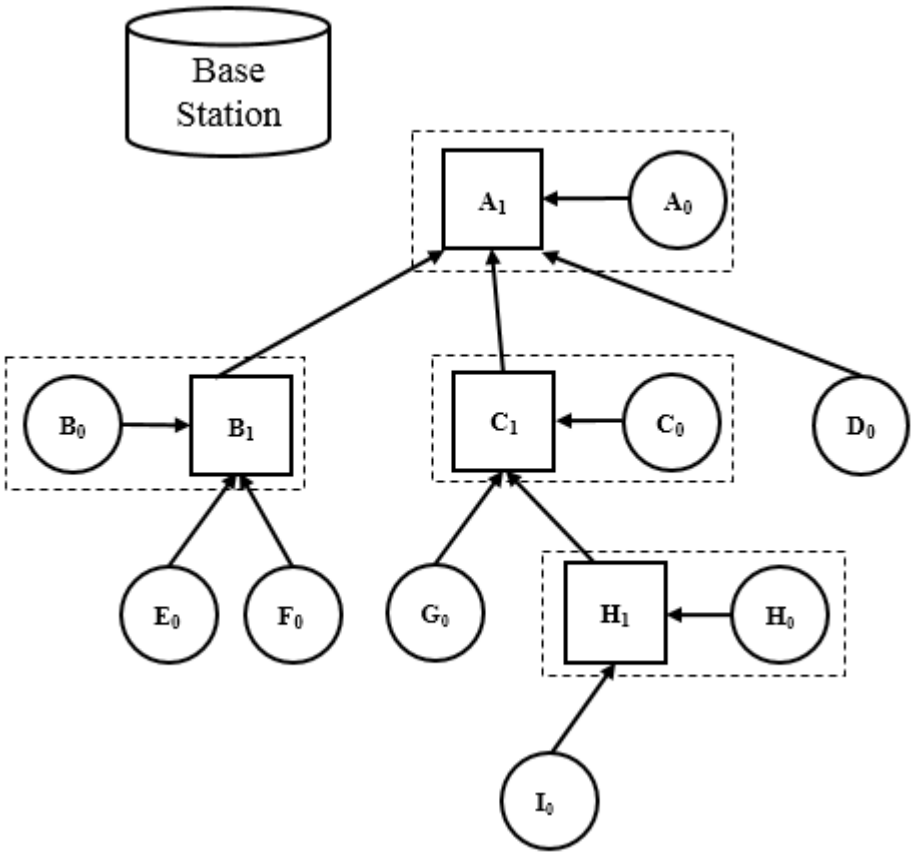

Fig. 5.3.: Naive commitment tree for Figure 5.2. For each sensor node S, S_0 is its leaf vertex, while S_1 is the internal vertex representing the aggregate computation at S (if any). The internal vertices are represented by dashed box.

for the given node. As each node aggregates all the labels into one label before sending it to the parent node.

5.5.2 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 5.5.2 *[28] 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 nodes in the aggregation tree originate a single-vertex commitment forest, which they then communicate to their parent nodes. Each internal node I originates a similar single-vertex commitment forest. In addition, I also receives commitment forests from each of its children. The node I 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 I wishes to combine q commitment forests F_1,...,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 5.5.1. The following example illustrates the commitment tree generation process for the node A of Figure 5.2.

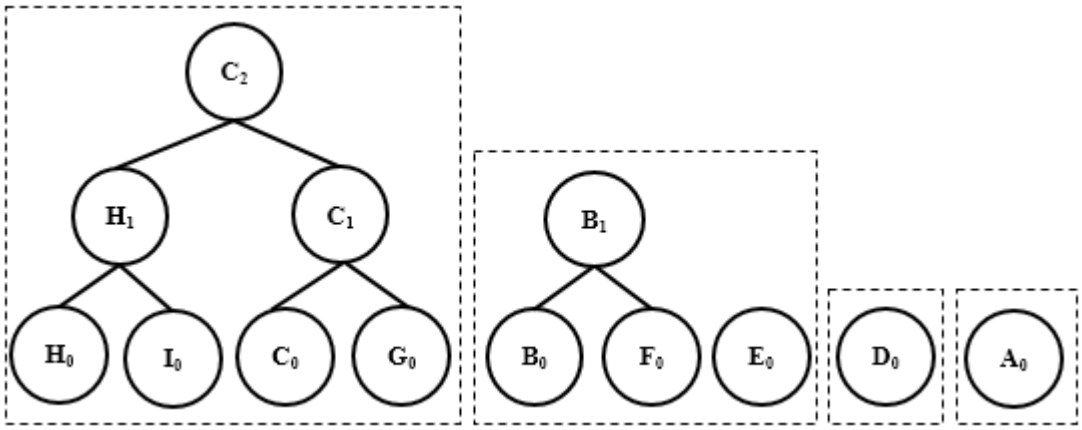

Fig. 5.4.: A receives C_2 from C, (B_1, B_0) from B, D_0 from D and generates A_0. The commitment forest received from a given sensor node is indicated by dashed-line box.

$$A_0 =< 1, A_{value}, \overline{A_{value}}, A_{id} >$$
$$D_0 =< 1, D_{value}, \overline{D_{value}}, D_{id} >$$
$$E_0 =< 1, E_{value}, \overline{E_{value}}, E_{id} >$$
$$B_1 =< 2, B_{1value}, \overline{B_{1value}}, H(N||2||B_{1value}||\overline{B_{1value}}||B_0||F_0) >$$
$$C_2 =< 4, C_{2value}, \overline{C_{2value}}, H(N||4||C_{2value}||\overline{C_{2value}}||H_1||C_1) >$$

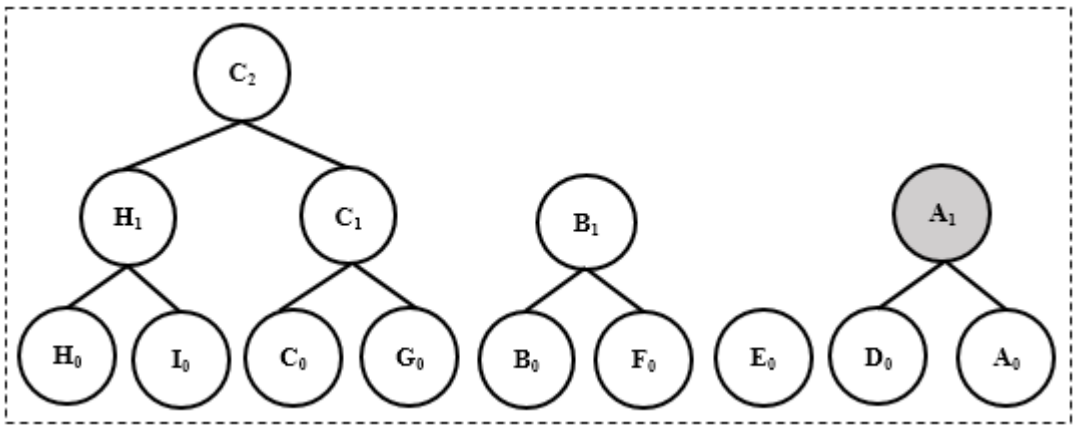

Fig. 5.5.: First Merge: A_1 vertex created by A.

$$A_1 = < 2, A_{1value}, \overline{A_{1value}}, H(N||2||A_{1value}||\overline{A_{1value}}||A_0||D_0) >$$
$$A_{1value} = \ A_{value} + D_{value};\ \overline{A_{1value}} = \overline{A_{value}} + \overline{D_{value}}$$

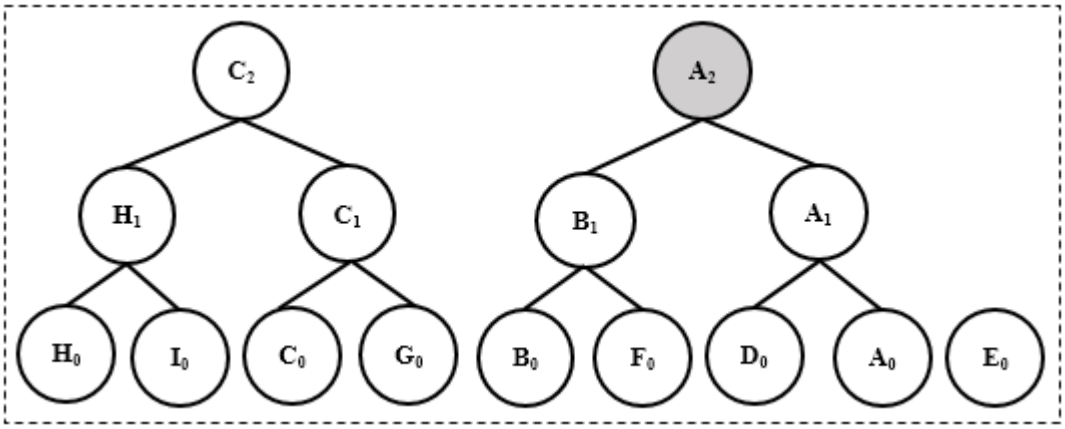

Fig. 5.6.: Second Merge: A_2 vertex created by A.

$$A_2 =< 4, A_{2value}, \overline{A_{2value}}, H(N||4||A_{2value}||\overline{A_{2value}}||B_1||A_1) >$$
$$A_{2value} = A_{1value} + B_{1value};\ \overline{A_{2value}} = \overline{A_{1value}} + \overline{B_{1value}}.$$

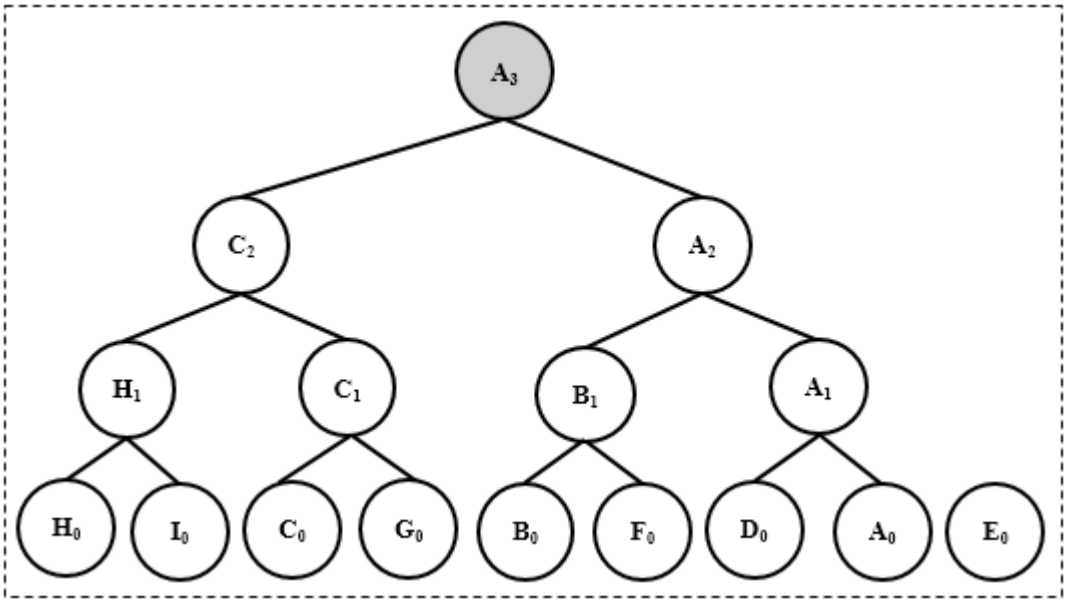

Fig. 5.7.: Third Merge: A_3 vertex created by A.

$$A_3 =< 8, A_{3value}, \overline{A_{3value}}, H(N||8||A_{3value}||\overline{A_{3value}}||C_2||A_2) >$$
$$A_{3value} = A_{2value} + C_{2value};\ \overline{A_{3value}} = \overline{A_{2value}} + \overline{C_{2value}}$$

Once the base station has received the final commitment forest from the root node, it checks that none of the SUM or COMPLEMENT aggregates of the roots of the trees in the forest are negative. If any aggregates are negative, the base station rejects the result and raises an alarm: a negative aggregate is a sure sign of tampering since all the data values (and their complements) are non-negative. Otherwise, the base station then computes the final pair of aggregates SUM and COMPLEMENT. The base station verifies that SUM + COMPLEMENT = nr where r is the upper bound on the range of allowable data values on each node. If this verifies correctly, the base station then initiates the result checking phase.

5.6 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 I to independently verify that its data value (I_{value}) was added into the SUM aggregate, and the complement $\overline{I_{value}}$ of its data value was added into the COMPLEMENT aggregate. First, the base station sends the aggregation results from the aggregation commit phase to every sensor node in the network using authenticated broadcast. 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 contri-

bution to the aggregate has been correctly counted. The result checking process has the following phases.

Distributing Final Commitment Values In this phase, the base station sends each of the received commitment labels to the entire network using authenticated broadcast. Authenticated broadcast means that each sensor node can verify that the message was sent by the base station and no one else.

Distributing 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 5.6.1 *[28] 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).*

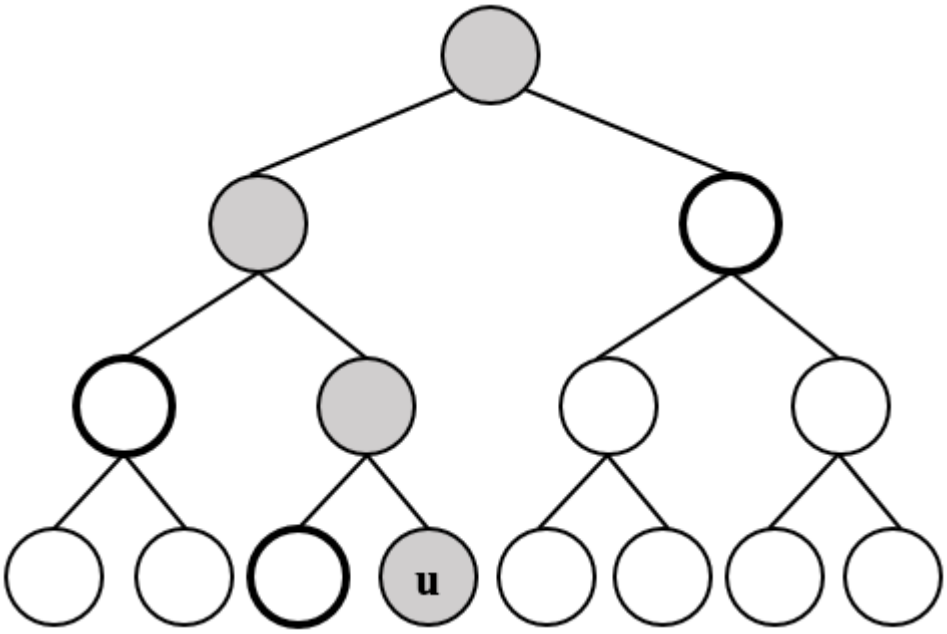

Fig. 5.8.: Off-path vertices of u are highlighted in bold.

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 the verification phase.

Verification of Contribution In this phase, the leaf vertex calculates the root vertex's label using its own label and off-path vertex labels received from its parent. It compares the the calculated root vertex's label with the label received from the base station. If those two labels match then it proceeds to the next step with Acknowledgment (ACK) message or with Negative Acknowledgment (NACK) message. This allows the leaf to verify that its label was not modified on the path to the root during the aggregation commit phase.

Collection of Authentication Codes Once each sensor node verifies its contribution to the root label it sends the relevant authentication code to its parent. The authentication code for sensor node A with ACK and NACK message is given as follows :

$$\mathrm{MAC}_{\mathsf{sk}_\mathsf{A}}(\mathrm{N}||\mathrm{ACK})$$
$$\mathrm{MAC}_{\mathsf{sk}_\mathsf{A}}(\mathrm{N}||\mathrm{NACK})$$

Where ACK, NACK are unique message identifier for positive and negative acknowledgment respectively, MAC is the message authentication algorithm, N is the query nonce and sk_A is secret key that node A 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 hierarchy. Each internal node 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 shares the secret key of all the nodes in the network, it computes the following:

$$\bigoplus_{i=1}^{n} \mathrm{MAC}_{\mathsf{sk_i}}(\mathrm{N}||\mathrm{ACK})$$

Then it compares the computed code with the received code. If those two codes match, then the base station accepts the aggregation result.

Theorem 5.6.1 *[28] Let the final SUM aggregate received by the base station be S. If the base station accepts S, then $S_L \leq S \leq (S_L + \mu \cdot r)$ 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 proof for the above theorem is given in [28]. The theorem gives the proof of correctness of the SHIA protocol. It proves that if the final aggregated value is accepted by the base station then it has to be in the given range. Hence, SHIA achieves security over the Truncated SUM which is a resilient aggregator according to Wagner [27]. Our protocol enhances SHIA to achieve desired security over the SUM which is non-resilient aggregate.

6. SYSTEM DESIGN

This chapter describes the rationale behind the design specifications of our secure aggregation protocol for sensor networks, the mechanism used to achieve it and implementation details.

6.1 Introduction

The most significant design aspect of the sensor network is the *Lifetime of the Network*. A sensor network tends 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 data between sensor nodes, making network bandwidth an expensive resource. The bandwidth is a 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. 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 [41].

Security mechanism is a mechanism that is designed to detect, prevent, or recover from a security attack [41].

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 [41].

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

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.

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. 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. Passive attacks are subtler and outside the scope of our thesis.

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? [42]

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 [37] 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). *Hash functions* provides one way functions to achieve irreversible encryption. *Traffic Padding*, *Routing Control*, *Notarization*, *Authentication Exchange* are possible mechanisms as well but they are outside the scope of our thesis.

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. 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 example, **Protecting banner advertisements on web pages**. The provider of the advertisements do not care if they copy the advertisements and show it to other people. So, there is no confidentiality at all. But they want to prevent people from changing the advertisements to the different types of advertisements. Also, 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, the integrity is important not the confidentiality. In most engineering discipline, it is useful to clarify the requirements carefully before embarking on a project [2]. An important aspect to the computer security, as discussed by Anderson [43], that is often ignored: designed the security before careful thought of the needs. 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.

For example, Wagner's work [27] 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 ag-

gregation. 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) [35] addresses 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) [28], 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. 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. 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. The network has to redo the all the work to create a 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, 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. Since 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. 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*. Further, we want protocol to work on *resilient* and *non-resilient* aggregate functions without compromising any desired security properties. In addition, We want protocol to be secure with *a single* or *multiple adversaries* in the network. Moreover, 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 need 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. Also, 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. Finally, 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 detail in Chapter 4.

7. DATA AGGREGATION WITH INTERNAL VERIFICATION

The concept of an aggregate commit with verification scheme is that all the sensor nodes in the network send the signature of the message along with the message. They send their certificates 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. Once the base station receives the aggregated value it starts the verification process. If there is no malicious activity in the network then it accepts the result and takes an action. If the malicious activity is detected during the verification phase then the base station starts interacting with the nodes in the network and detects an adversary using interactive proof.

7.1 Data-Item

We describe structure of the data-item and its signature, used in creating the commitment tree for the aggregate commit with verification approach. The differences between the data-item and the label structure of SHIA, with rationale behind it are discussed.

Definition 7.1.1 *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.

Each sensor node creates its own data-item during commitment tree generation process which is called the leaf vertex of the node. For example, sensor node A creates its data-item A_0 as follows:

$$A_0 = < A_{id}, 1, A_{value}, H(N||1||A_{value}) > . \tag{7.1}$$

where A_{id}, A_{value} is the unique ID and sensor reading of the node A. The count is 1 as there is only vertex in the subtree rooted at A, H is the collision resistant hash function, and N is the query nonce.

The first difference between SHIA's label structure and our data-item structure is that we remove the *complement* field from the label structure see Definition 5.5.1. The complement field is redundant information in the label. The complement field is used by the base station (the querier in SHIA), before the result checking phase. It is used 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.

The second difference between SHIA's label structure and our data-item structure is that we include unique 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 following sections.

7.1.1 Signing and Verification of the Data-Item

Each sensor node sends the signature of its data-item signed by itself using its own secret key. For example, the signature of A_0 of the Equation 7.1 is given as follows:

$$\mathsf{S} = \mathsf{Sign}_{S_A}(A_0) \tag{7.2}$$

where S_A is the secret key of the sensor node A, Sign is the signing algorithm. The parent node receives the data-item and its signature from its child. It also receives the certificate from its child which is shown in Table 7.1. From the digital certificate

Table 7.1.: Digital Certificate

Unique ID of the sensor node
Public key of the sensor node
Certification Authority's name
Certification Authority's digital signature

the parent node receives the public-key of its child, which is used to decipher the signature. For example, the parent node of B verifies $\mathsf{Sign}_{S_A}(A_0)$ as follows:

$$\mathsf{Verify}_{P_A}(A_0, \mathsf{S}) = \begin{cases} \textbf{true} \text{ with probability of 1} & \text{if } \mathsf{S} = \mathsf{Sign}_{S_A}(A_0) \\ \textbf{false} \text{ with overwhelming probability} & \text{if } \mathsf{S} \neq \mathsf{Sign}_{S_A}(A_0) \end{cases} \tag{7.3}$$

where P_A is the public key of A, Verify is the verification algorithm.

7.1.2 Security Benefits

While creating the commitment tree, the sensor S sends the data-item S_0, and its signature S to its parent in the aggregation tree. The signature allows the parent node

to verify the *authenticity* of the sensor node. As only sensor node S can create the signature using its secret key. The signature S assures the *integrity* of the data-item S_0. Because either the data-item or the signature has been tampered in any way the verification algorithm returns false. It allows the sender to have the proof for the sent data-item and the receiver to have the proof for the received data-item. Hence, providing the security service of *non-repudiation*. The digital signature depends on the message so the parent node can not reuse the signature for other messages in the future. Hence, protecting the network against the *replay attacks*. Hence, it protects against the *active attacks*.

7.2 Commitment Payload

We define commitment payload based on the commitment forest see Definition 5.5.2. We also define transmit payload as follows.

Definition 7.2.1 *A **commitment payload** is a set of data-items of the root vertices of the trees with their respective signatures in the outgoing commitment forest and an additional signature for the transmission.*

Definition 7.2.2 *The **transmit payload** is the concatenation of all the data-items in the commitment payload.*

For brevity, we use the term forest instead of the commitment forest and payload instead of the commitment payload. Consider, the aggregation tree shown in Figure 7.1, where B is the parent of A, C is the parent of B and D is the parent of C. While creating the commitment tree A creates its data-item A_0 according to Equation 7.1. A sends only one data-item to B therefore A's payload (A_{pay}), transmit-payload (A_τ) are given as follows:

$$A_{pay} = < A_0, \mathsf{Sign}_{S_A}(A_0), \mathsf{Sign}_{S_A}(A_\tau) > \ where\ A_\tau = < A_0 > \tag{7.4}$$

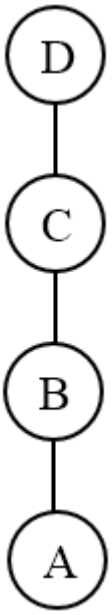

Fig. 7.1.: Palm Shaped Aggregation Tree

The sensor node C's payload is shown in Figure 7.2. The commitment tree generation process is described in the later sections. The sensor node C sends two data-items to D therefore C's payload (C_{pay}), transmit-payload (C_τ) are given as follows:

$$C_{pay} = < C_0, \mathsf{Sign}_{S_C}(C_0), B_1, \mathsf{Sign}_{S_C}(B_1), \mathsf{Sign}_{S_C}(C_\tau) > \ \ where\ C_\tau = < B_1 \ || \ C_0 >$$

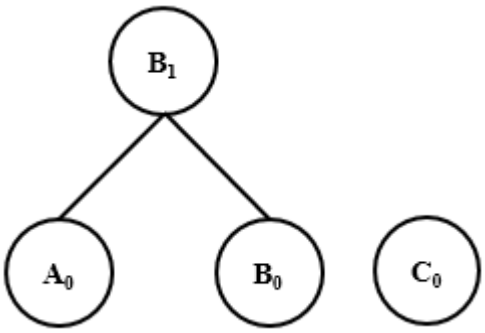

Fig. 7.2.: Commitment Payload of Sensor Node C

The verification of the received signatures in the payload is done by the parent node in the same way described in Section 7.1.1.

7.2.1 Security Benefits

As described in Section 7.1.2, the signature of the transmit-payload $\mathsf{Sign}_{S_S}(S_\tau)$ assures the integrity and authentication of the transmit-payload S_τ. In addition to that, the signature of the transmit-payload is like the signature for the transmission. To the sender, it assures that it is sending only the data-items included in the signature of the transmit-payload. Further, it establish that none of the data-items gets added or remove from the transmit-payload during the transmission. To the receiver, it assures that it receives all the data-items included in the signature of the transmit-payload and none of the data-items were stranded or added additionally to the payload of its child. For example, the signature on the C's transmit-payload $\mathsf{Sign}_{S_C}(C_\tau)$ assures that the sensor node C sent only two data-items C_0, B_1 in its payload. It also establishes that none of the data-items in its payload have been left stranded. As we said, it's like the signature for the transmission.

7.3 Key Differences

In SHIA, all the leaf nodes in the aggregation tree send only their respective data-items to the parent in the aggregation tree. In our approach, all the leaf nodes send the data-item, the signature of the data-item and the signature of the transmit-payload to their parent node in the aggregation tree. The child node sends its certificate as well if the parent node does not have it in its memory already.

In SHIA, the parent node proceeds with the aggregation process without verifying the received data-items. In our protocol, the parent node verifies the received signature using the the verification algorithm. It proceeds with the aggregation only if all the signatures are verified true.

In SHIA, the trusted base station verifies the final received data-items. And upon detecting the malicious activity in the network, the base station raises an alarm. The base station does not do anything to detect malicious node responsible for the malicious activity. In our approach, upon detecting the malicious activity the base station

interacts with several relevant nodes in the network to trace down the malicious node. Also, the base station issues the certificate to the sensor nodes in the network.

7.3.1 Bandwidth

According to Definition 7.1.1, the typical size of the data-item packet is 400 bits as shown below. If one uses Elliptic Curve Digital Signature Algorithm (ECDSA) then

Table 7.2.: Data-Item Size

ID	COUNT	VALUE	COMMITMENT
20 bits	21 bits	20 bits	256 bits

the size of signature is 500 bits [44]. 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.

7.4 Two Ways of Forwarding Payload

As described in the previous sections, we send the data-items with their signature along with the signature of the transmit-payload while creating the commitment tree. Here, we describe two different approaches to send the signatures in the aggregation tree hierarchy based on the two different ways of signing the data-items. To demonstrate two approaches we use the aggregation tree shown in Figure 7.1 and the payload of the sensor node C shown in Figure 7.2. In both the approaches, the sensor node C sends all the data-items in its payload with their respective signatures to its parent sensor node D along with the signature of its transmit-payload C_τ.

In the first approach, C verifies $\mathsf{Sign}_{S_B}(B_1)$ and sends it to D without any modifications as follows:

$$
\begin{aligned}
&C_{pay} = < C_0, \mathsf{Sign}_{S_C}(C_0), B_1, \mathsf{Sign}_{S_B}(B_1), \mathsf{Sign}_{S_C}(C_\tau) > \ \ where\ C_\tau =< C_0 || B_1 > \\
&C_0 = < C_{id}, 1, C_{value}, H(N||1||C_{value}) > \\
&B_1 = < B_{id}, 2, B_{1value}, H(N||2||B_{1value}||A_0||B_0) >;\ B_{1value} = \ B_{value} + A_{value}
\end{aligned}
\tag{7.5}
$$

The sensor node C sends three signatures $\mathsf{Sign}_{S_B}(B_1)$, $\mathsf{Sign}_{S_C}(C_0)$ and $\mathsf{Sign}_{S_C}(C_\tau)$ to its parent D. It requires the parent node D to know the public key of the sensor nodes C and B, hence two certificates.

In the second approach, the sensor node C can verify the $\mathsf{Sign}_{S_B}(B_1)$ then remove the old signature and creates new signature $\mathsf{Sign}_{S_C}(B_1)$ and sends to D as follows:

$$
C_{pay} = < C_0, \mathsf{Sign}_{S_C}(C_0), B_1, \mathsf{Sign}_{S_C}(B_1), \mathsf{Sign}_{S_C}(C_\tau) > \ \ where\ C_\tau =< C_0 || B_1 >
\tag{7.6}
$$

The sensor node C sends three signatures $\mathsf{Sign}_{S_C}(B_1)$, $\mathsf{Sign}_{S_C}(C_0)$ and $\mathsf{Sign}_{S_C}(C_\tau)$ to its parent D. But all the signatures are signed by the sensor node C, it requires the parent node D to know the public key of only the sensor node C, hence D need to know only one certificate.

To give an analogy with a real world application, consider the following example. Suppose, one want to buy a diamond from the local diamond retailer. Some Diamonds are expensive commodity, so the end customer wants to verify its authenticity and integrity before purchasing. Now, 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 and sends the 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 they add their signature and send their 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 they remove the signature of the wholesaler, adds their 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.

We call these two approaches **Forwarding signatures without resigning the data-items (FSwoRD)**, **Forwarding signatures with resigning the data-items (FSwRD)** as shown in Equations 7.5 and 7.6 respectively. Both the approaches 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.

8. OUR PROTOCOL

In this chapter, we describe our protocol with FSwRD approach beginning from query dissemination phase till the detecting an adversary. Our protocol using FSwRD can also be applied using FSwoRD with the obvious modifications. We briefly describe this in Section XX.

8.1 Query Dissemination

The protocol begins with the base station initiating the query request to the sensor network. The base station does the authenticated broadcast of its query to the entire sensor network asking the network to report the aggregated result of the sensor reading values. It includes the query nonce in its query to avoid replay attacks in the future. We use the same hash chain process to generate unique query nonce for the base station as described in Section 5.4. Once the sensor nodes receive the query from the base station they creates their own leaf vertex by creating the data-items of their sensor readings and its signature. The sensor nodes create their payloads and send it to their parent in the aggregation tree. The details of the commitment tree generation is described in the next section.

8.2 Commitment Tree Generation

For the given aggregation tree the commitment forest is built as follows. The commitment tree generation begins from the sensor nodes with the highest depth (leaf nodes) in the aggregation tree. Leaf sensor nodes in the aggregation tree create their leaf vertices by creating data-items, signatures of those data-items and signature of the transmit-payload according to Equations 7.1, 7.2, 7.4, respectively. They then

send the payload to their parents in the aggregation tree. Each internal sensor node in the aggregation tree also creates their leaf vertex. In addition, internal sensor node receives the payload from each of its children, which has created a forest. For each child, internal node first verifies the signature of the transmit-payload and then the signature of the data-item for one child at a time. Once internal node verifies all the received signatures, it merges all the data-items into its forest with same count value. It merges two data-items by creating a new data-item with count value incremented by one and whose value is the addition of value field of the previous two data-items. Note that we can easily determine the height of the commitment tree from the count value. Suppose, after verifying all the signatures from the payloads, an internal sensor node I has to merge i data-items D_1, D_2, ..., D_i in its forest. Let c be the smallest count value in I's forest. The sensor node I finds two data-items D_1 and D_2 from 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 8.1. It repeats the process until no two data-items in its forest have the same count value. The data-item of the A_2 is given as follows:

$$A_2 =< A_{id}, 4, A_{2value}, H(N||4||A_{2value}||B_1||D_1) >;\ A_{2value} = B_{1value} + D_{1value}.$$

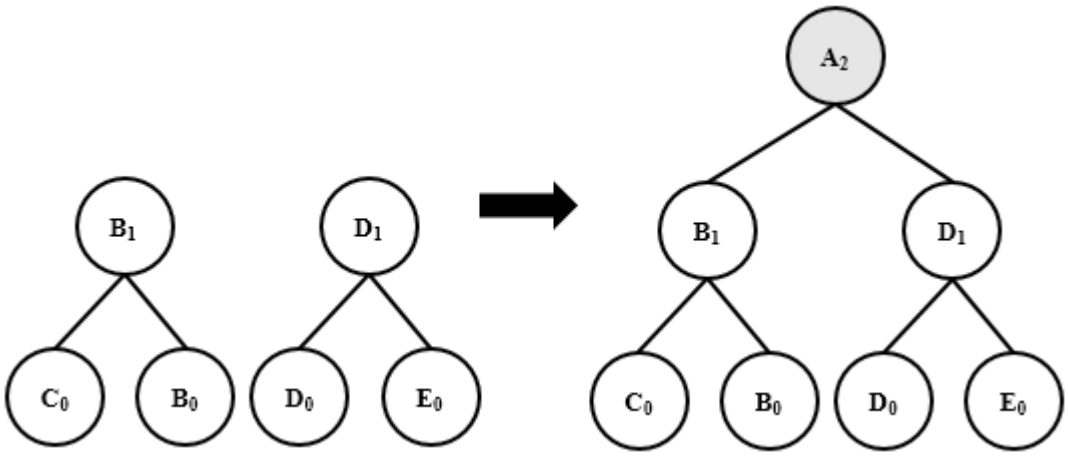

Fig. 8.1.: Input node A has B_1 and C_1 in its forest. It aggregates these two trees and creates A_2.

We demonstrate the commitment tree generation process for the aggregation tree shown in Figure 8.2. In this example, A has three children. A receives one payload

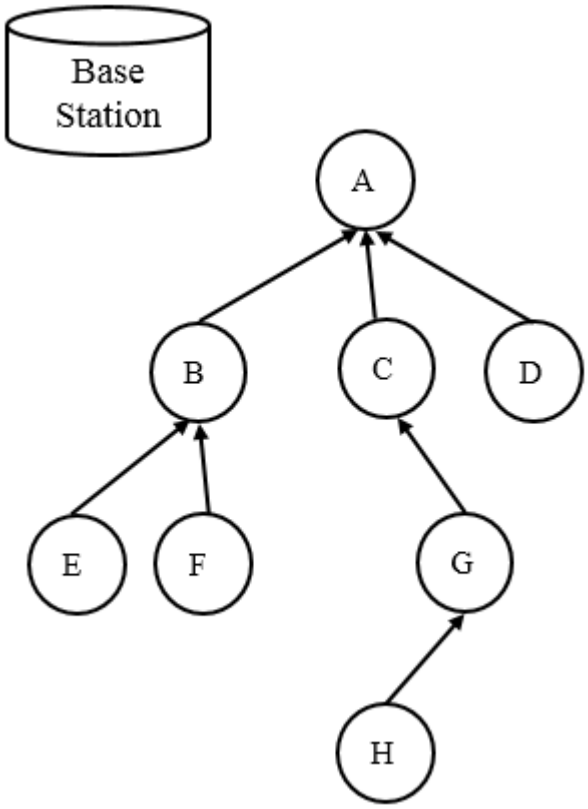

Fig. 8.2.: Aggregation Tree.

from each of its children, which impacts A's forest. After verifying all the signatures from each child's payload, A uses the data-items in its forest to create its payload, which is sent to the base station. We describe the payload generation process for sensor nodes B, C, D and A in order.

The sensor node B creates its payload from its forest. It's forest consists of payloads received from E and F. The leaf sensor nodes E and F creates their payloads according to Equation 7.4 as follows:

$$E_{pay} = < E_0, \mathsf{Sign}_{\mathsf{sk_E}}(E_0), \mathsf{Sign}_{\mathsf{sk_E}}(E_\tau) > \ where\ E_\tau = < E_0 >$$
$$E_0 = < E_{id}, 1, E_{value}, H(N||1||E_{value}) >$$
$$F_{pay} = < F_0, \mathsf{Sign}_{\mathsf{sk_F}}(F_0), \mathsf{Sign}_{\mathsf{sk_F}}(F\tau) > \ where\ F_\tau = < F_0 >$$
$$F_0 = < F_{id}, 1, F_{value}, H(N||1||F_{value}) > .$$

B receives E_{pay}, F_{pay} from E, F respectively. B verifies all the signatures in the received payloads. Then it creates its own data-item B_0. Now, B has B_0, E_0, F_0 in its forest as shown in Figure 8.3. As all the data-items have the same count value, B has an option, when selecting two data-items to merge. B aggregates E_0, F_0 and creates B_1. After creating B_1 none of the data-items have the same count value. So, B creates its payload B_{pay} and sends it to A as follows:

$$B_{pay} = < B_0, \mathsf{Sign}_{\mathsf{sk_B}}(B_0), B_1, \mathsf{Sign}_{\mathsf{sk_B}}(B_1), \mathsf{Sign}_{\mathsf{sk_B}}(B_\tau) > \ where\ B_\tau = < B_0||B_1 >$$
$$B_1 = < B_{id}, 2, B_{1value}, H(N||2||B_{1value}||E_0||F_0) >;\ B_{1value} = E_{value} + F_{value}$$
$$B_0 = < B_{id}, 1, B_{value}, H(N||1||B_{value}) >$$

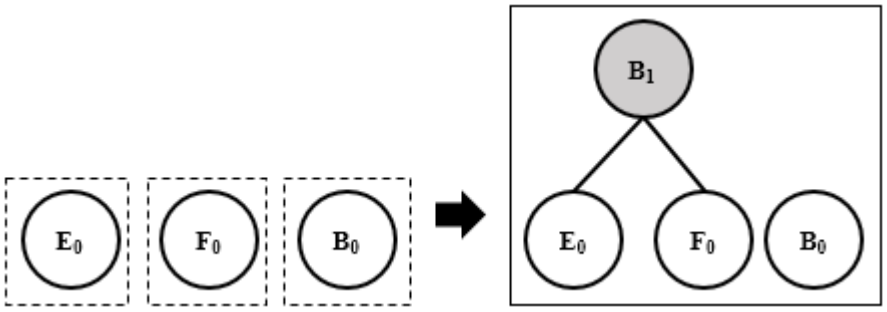

Fig. 8.3.: Transformation from B's forest to its payload. Each dashed-line box shows forest and solid-line box shows payload of the respective sensor node.

In similar way, the sensor node C in the aggregation tree receives G_{pay} from G which is defined as follows:

$$G_{pay} = < G_1, \mathsf{Sign}_{\mathsf{sk_G}}(G_1), \mathsf{Sign}_{\mathsf{sk_G}}(G_\tau) > \ where\ G_\tau = < G_0||H_0 >$$
$$G_1 = < G_{id}, 2, G_{1value}, H(N||2||G_{1value}||G_0||H_0) >;\ G_{1value} = G_{value} + H_{value}$$
$$G_0 = < G_{id}, 1, G_{value}, H(N||1||G_{value}) >$$
$$H_0 = < H_{id}, 1, H_{value}, H(N||1||H_{value}) >$$

C verifies all the signatures in the received payload G_{pay} and creates C_0. Now, C has C_0, G_1 in its forest as shown in Figure 8.4. As none of the data-items have the same count value, C does not merge those two data-items. But C removes the old signature on G_1 and signs G_1 with its secret key. So, C creates its payload C_{pay} and sends it to A as follows:

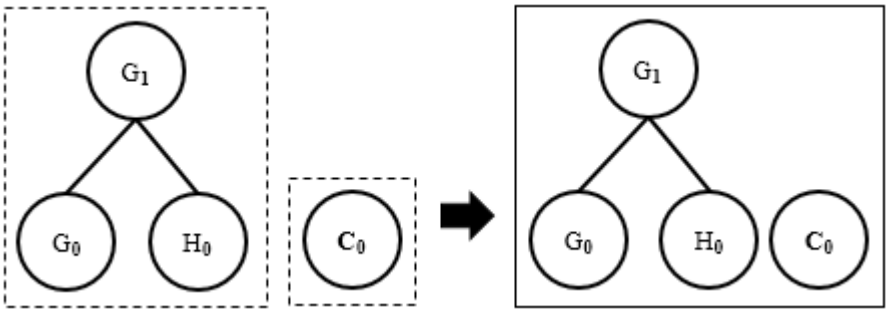

Fig. 8.4.: C's forest aggregation creating its payload.

$$C_{pay} = < C_0, \mathsf{Sign}_{\mathsf{sk_C}}(C_0), G_1, \mathsf{Sign}_{\mathsf{sk_C}}(G_1), \mathsf{Sign}_{\mathsf{sk_C}}(C_\tau) > \ \ where\ C_\tau = < C_0 || G_1 >$$
$$C_0 = < C_{id}, 1, C_{value}, H(N||1||C_{value}) >$$

The sensor node D creates its payload (D_{pay}) and sends it to A as follows:

$$D_{pay} = < D_0, \mathsf{Sign}_{\mathsf{sk_D}}(D_0), \mathsf{Sign}_{\mathsf{sk_D}}(D_\tau) >;\ where\ D_\tau = < D_0 >$$
$$D_0 = < D_{id}, 1, D_{value}, H(N||1||D_{value}) >$$

The root node of the aggregation tree A receives the payloads B_{pay}, C_{pay} and D_{pay} from B, C and D respectively. A verifies all the signatures in the received payloads and creates A_0. Now, A has G_1, C_0, B_1, B_0, D_0 and A_0 in its forest as shown in Figure 8.5. A has four data-items with count value of 1. In the first merge, A aggregates those data-items and creates A_{10} and A_{11} as shown in Figure 8.6. Now, A has four data-items with count value of 2. In the second merge, A aggregates those data-items and creates A_{20} andA_{21} as shown in Figure 8.7. Finally, A has two data-items with count value of 4. In the final merge, A aggregates those data-items and creates A_3

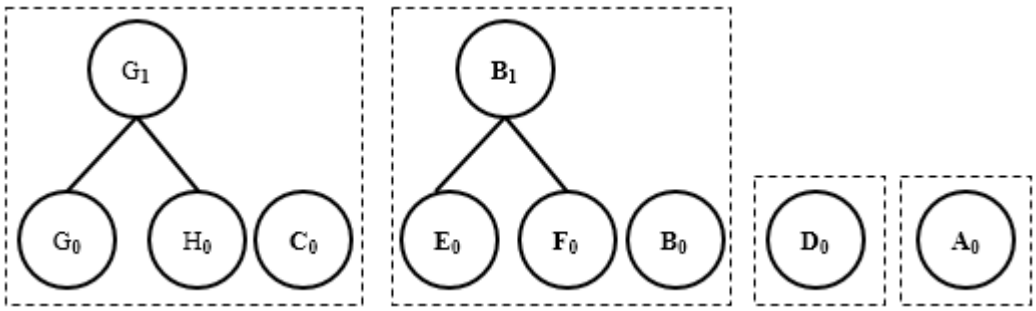

Fig. 8.5.: A's forest: A receives three payloads from C, B, D and creates A_0

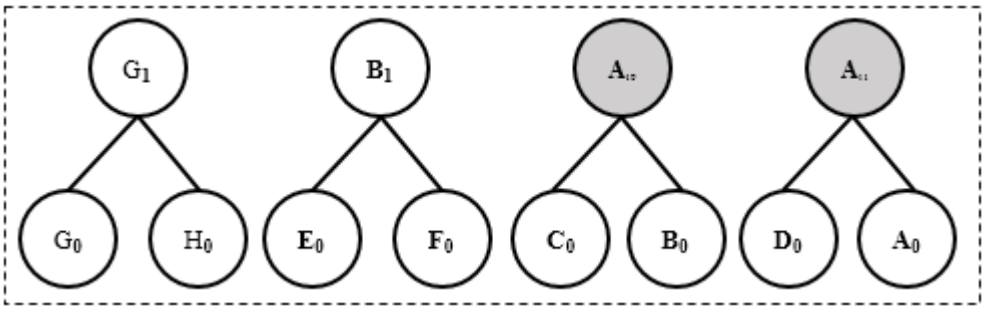

Fig. 8.6.: A's forest: after first merge

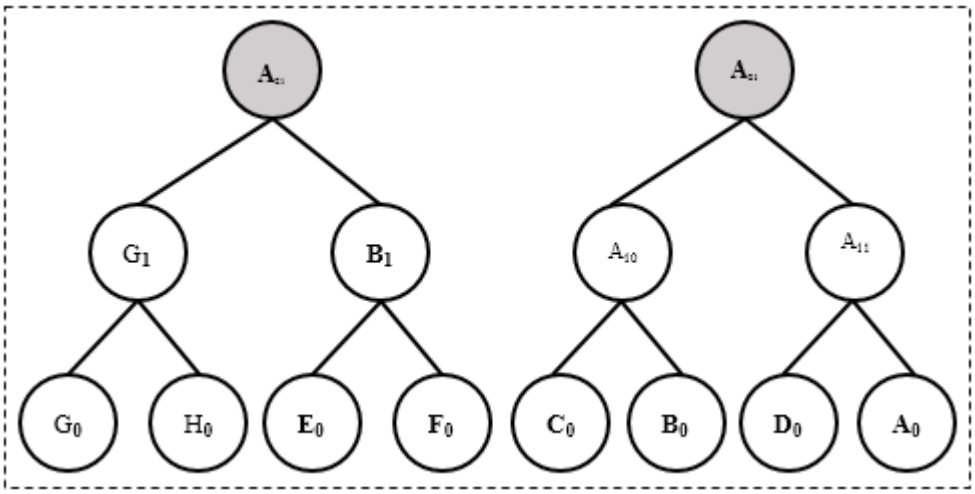

Fig. 8.7.: A's forest: after second merge

as shown in Figure 8.8. Then A creates payload and sends it to the base station as follows:

$A_{pay} = < A_3, \mathsf{Sign}_{\mathsf{sk_A}}(A_3), \mathsf{Sign}_{\mathsf{sk_A}}(A_\tau) > \ where\ A_\tau = < A_3 >$

$A_3 = < A_{id}, 8, A_{3value}, H(N||8||A_{3value}||A_{20}||A_{21}) >;\ A_{3value} = \ A_{20value} + A_{21value}$

$A_{20} = < A_{id}, 4, A_{20value}, H(N||4||A_{20value}||G_1||B_1) >;\ A_{20value} = \ G_{1value} + B_{1value}$

$A_{21} = < A_{id}, 4, A_{21value}, H(N||4||A_{21value}||A_{10}||A_{11}) >;\ A_{21value} = \ A_{10value} + A_{11value}$

$A_{10} = < A_{id}, 2, A_{10value}, H(N||2||A_{10value}||B_0||C_0) >;\ A_{10value} = \ B_{value} + C_{value}$

$A_{11} = < A_{id}, 2, A_{11value}, H(N||2||A_{11value}||D_0||A_0) >;\ A_{11value} = \ D_{value} + A_{value}$

$A_0 = < A_{id}, 1, A_{value}, H(N||1||A_{value}) >$

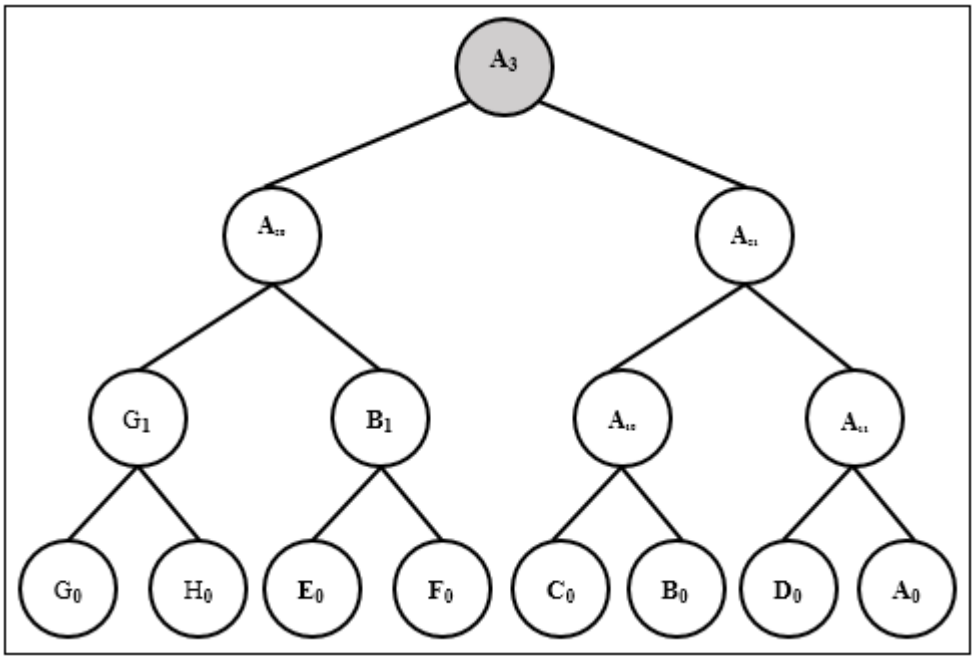

Fig. 8.8.: A's payload : A sends this to the base station.

Once the base station receives the payload from the root of the aggregation tree, it verifies all the signatures in the payload. In the previous example, the base station receives A_{pay} from the sensor node A. It verifies the signatures $\mathsf{Sign}_{\mathsf{sk_A}}(A_3)$, $\mathsf{Sign}_{\mathsf{sk_A}}(A_\tau)$ in the received payload. If the base station verifies all the signatures to true it initiates the result checking phase.

8.3 Result Checking

The purpose of the *result checking* phase is to require that all the sensor nodes verify their individual contributions to the final aggregate value. If there is any inconsistency in the aggregation process then with the help of the base station, trace down the node responsible for causing the inconsistency in the aggregation process.

8.3.1 Dissemination Final Payload by the Base Station

Once the base station receives the payloads of the root node of the aggregation tree, it verifies all the signatures in the payload and then sends each of the data-items in the payload to the entire sensor network using authenticated broadcast. The authenticated broadcast allows the sensor nodes to verify that the data-items are sent by the base station. The authentication ensures no one else is masquerading the base station. In our previous example, the base station receives only one data-item A_3 in the payload sent by A. In that case, the base station's payload B_{pay} which is sent to the entire network is given as follows:

$$\mathsf{B}_{pay} = < A_3, \mathsf{Sign}_{\mathsf{sk_B}}(A_3), \mathsf{Sign}_{\mathsf{sk_B}}(\mathsf{B}_\tau) > \ \ where\ \mathsf{B}_\tau = < A_3 > .$$

8.3.2 Dissemination of Off-Path Values

To enable verification each sensor node must receive all of its off-path values. The off-path values of the sensor nodes can be determined according to the Definition 5.6.1. Each internal vertex I in the commitment tree has two children u_1 and u_2. To disseminate off-path values, I sends the data-item of u_1 to u_2, and vice-versa (I also attaches relevant information tagging u_1 as the right child and u_2 as the left child) along with the signatures of the data-item and the signature of the transmit-payload. In our previous example, internal vertex A_{10}, shown in Figure 8.8, has two children

C_0 and B_0. A_{10} (which is sensor node A in aggregation tree) sends the following off-path values to C and B respectively as follows:

$$< B_0, \mathsf{Sign}_{\mathsf{sk_A}}(B_0), \mathsf{Sign}_{\mathsf{sk_A}}(A_\tau) > \ \ where\ A_\tau = < B_0 >$$
$$< C_0, \mathsf{Sign}_{\mathsf{sk_A}}(C_0), \mathsf{Sign}_{\mathsf{sk_A}}(A_\tau) > \ \ where\ A_\tau = < C_0 > .$$

An internal vertex I receives data-items with their respective signatures from its parent. It verifies the signatures of the received data-items then resigns them and sends those data-items (and left/right tags) with their signatures to both of its children. Continuing the previous example, internal vertex A_{10} receives A_{11} and A_{20} from its parent A_{21}. A_{10} sends the following off-path values to C and B respectively as follows:

$$< B_0, \mathsf{Sign}_{\mathsf{sk_A}}(B_0), A_{11}, \mathsf{Sign}_{\mathsf{sk_A}}(A_{11}), A_{20}, \mathsf{Sign}_{\mathsf{sk_A}}(A_{20}), \mathsf{Sign}_{\mathsf{sk_A}}(A_\tau) >$$
$$where\ A_\tau = < B_0||A_{11}||A_{20} >$$
$$< C_0, \mathsf{Sign}_{\mathsf{sk_A}}(C_0), A_{11}, \mathsf{Sign}_{\mathsf{sk_A}}(A_{11}), A_{20}, \mathsf{Sign}_{\mathsf{sk_A}}(A_{20}), \mathsf{Sign}_{\mathsf{sk_A}}(A_\tau) >$$
$$where\ A_\tau = < C_0||A_{11}||A_{20} > .$$

In FSwRD, all the leaf vertices need to know only one certificate as they receive data-items signed by their parent vertex. In FSwoRD, all the leaf vertices might need to know $\log l$ certificates, where l is the number of leaf-vertices in commitment tree including the leaf vertex. Continuing the previous example, C and B receive three signatures from A_{10}. All these three signatures are signed by A. Hence, C and B need to know the certificate of A. Whereas in FSwoRD, C and B receives three signatures from A. From those three two will be signed by A and one will be signed by either B or C depending on the data-item. Therefore, B need to know the certificates of A and C, and C need to know the certificates of A and B.

We show the significance of the commitment field while distributing the off-path values. The commitment filed helps us detecting any *malicious activity* in the network. Because of the signatures infrastructure eventually we can detect an adversary. For example, if an internal vertex simply forwards incorrect data-item to its children then it the relevant leaf vertex will complain, as the leaf vertex will not be able to

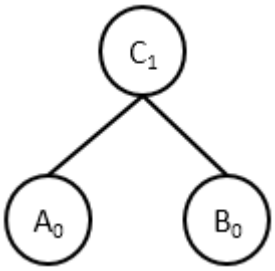

Fig. 8.9.: Smallest Possible Commitment Tree

derive the data-item received using authenticated broadcast from the base station. Because of the signatures internal vertex responsible for forwarding the incorrect data-item will be caught. Another scenario could be that an internal vertex changed the its children's data-item while creating commitment tree and sending the incorrect off-path values to compensate discrepancy. To illustrate the scenario consider the Example 8.3.1.

Example 8.3.1 *Let's say the vertices in the commitment tree of Figure 8.9 have the data-items defined as follows. Note that we did not include the signatures of these data-items in this example for simplification.*

$$A_0 = < A_{id}, 1, 10, H(N||1||10) >$$
$$B_0 = < B_{id}, 1, 20, H(N||1||20) >$$
$$C_1 = < C_{id}, 2, 30, H(N||2||30||A_0||B_0) >$$

C changes A_0 and B_0 to A_0' and B_0' by changing the value fields and then applies aggregate commit algorithm. C sends either C_1' or C_1'' to the base station. It also creates B_0'' and A_0'' off-path values trying to compensate its malicious activity.

$A'_0 = < A_{id}, 1, 100, H(N||1||10) >$
$B'_0 = < B_{id}, 1, 200, H(N||1||20) >$

$C'_1 = < C_{id}, 2, 300, H(N||2||300||A''_0||B_0) >$ *or* $C''_1 = < C_{id}, 2, 300, H(N||2||300||A_0||B''_0) >$

$B''_0 =< B_{id}, 1, 290, H(N||1||20) >$
$A''_0 =< A_{id}, 1, 280, H(N||1||10) >$

During the dissemination of root data-item phase, A and B using authenticated broadcast receives either C'_1 *or* C''_1 *from the base station based on what C send to base station. During the dissemination of off-path values phase, A and B receives* B''_0, A''_0 *from C respectively. During the verification of inclusion phase, A and B derives the root data-item using the received off-path values, and it does not match with the received root data-item as follows:*

$$\begin{aligned} &A \text{ uses } (A_0, B''_0) \text{ and derives} < 2, 300, H(N||2||300||A_0||B''_0) > \neq C'_1 = C''_1 \\ &B \text{ uses } (A''_0, B_0) \text{ and derives} < 2, 300, H(N||2||300||A''_0||B_0) > = C'_1 \neq C''_1. \end{aligned} \tag{8.1}$$

Hence, during the collection of authentication codes phase, A and B send their authentication codes with NACK message.

Above example shows how the commitment field in the data-item 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 malicious activity can be detected by the commitment filed in the data-item. Later, we show that the adversary can be detected with the provided signature infrastructure. Once a vertex has received all the data-items off its off-path vertices, it can proceed to the verification step.

8.3.3 Verification of Inclusion

Once the sensor node has received all the data-items of its off-path vertices from its parent, first it verifies the signature of all the received data-items and then it verifies that no aggregation result tampering has occurred on the path between its leaf vertex and the root of its commitment tree. It also verifies that its senor reading was aggregated correctly by all the intermediate aggregate nodes. For each vertex on the path from the root of its commitment tree, it derives the data-items according to Definition 5.5.1. It is able to do so since the off-path values provide all the necessary information to perform the data-item computation. In the previous example, C receives B_0, A_{11} and A_{20} from its parent node and it has C_0. It aggregates $< C_0, B_0 >$ and derives A_{10}. Then it aggregates $< A_{10}, A_{11} >$ and derives A_{21}. Finally it aggregates $< A_{21}, A_{20} >$ and derives A_3. Then it compares the derived A_3 with the A_3 received from the base station. Based on those data-items are identical or not the node proceeds with the next step accordingly. If those data-items are identical then the node sends the authentication code with ACK message and if those data-items are not identical then the node sends the authentication code with NACK message.

8.3.4 Collection of Authentication Codes

The authentication codes for sensor node I, with their positive and negative acknowledgment message, are defined as follows:

$$\begin{aligned} &\text{Positive}: \text{MAC}_{\mathsf{sk_I}}(\text{N}||\text{ACK}) \\ &\text{Negative}: \text{MAC}_{\mathsf{sk_I}}(\text{N}||\text{NACK}) \end{aligned} \tag{8.2}$$

$\mathsf{sk_I}$ is the secret key of the sensor node I, ACK and 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. Leaf sensor nodes in the aggregation tree first send their authentication codes to their parents in the aggregation tree. Once an internal sensor node has received authen-

tication codes from all its children, it computes the XOR of its own authentication code with all the received codes, and forwards it to its parent. For example, the sensor node B shown in Figure 8.2, receives the authentication codes $\mathrm{MAC}_{\mathsf{sk_E}}(\mathrm{N}||\mathrm{ACK})$, $\mathrm{MAC}_{\mathsf{sk_F}}(\mathrm{N}||\mathrm{ACK})$ from E, F respectively, and it has its own authentication code with ACK message then B sends the following authentication code to A.

$$\mathrm{MAC}_{\mathsf{sk_E}}(\mathrm{N}||\mathrm{ACK}) \oplus \mathrm{MAC}_{\mathsf{sk_F}}(\mathrm{N}||\mathrm{ACK}) \oplus \mathrm{MAC}_{\mathsf{sk_B}}(\mathrm{N}||\mathrm{ACK})$$

Because of the XOR function we do not have to forward all the authentication codes. At the end of the process, the base station receives a single authentication code Δ_{root} from the root of an aggregation tree.

8.3.5 Verification of Authentication Codes

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

$$\Delta_{ack} = \bigoplus_{i=1}^{n} \mathrm{MAC}_{\mathsf{sk_i}}(\mathrm{N}||\mathrm{ACK}) \tag{8.3}$$

Here, the addition represents an XOR operation. The base station can compute Δ_{ack} as it knows the secret key $\mathsf{sk_i}$ for each sensor node i. Then it compares the computed Δ_{ack} with the received root authentication code Δ_{root}. 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 as follows:

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

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

$$c_i := \text{MAC}_{\mathsf{sk_i}}(\text{N}||\text{ACK}) \oplus \text{MAC}_{\mathsf{sk_i}}(\text{N}||\text{NACK}) \tag{8.5}$$

After that, it compares c with all c_i one at a time. The matching code indicates the complainer node. The base station needs to do $C_{n,1}$ [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 8.3.2 *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 as follows:*

$$\begin{aligned}
&MAC_{\mathsf{sk_1}}(N \,||\, ACK) = (1001)_2;\ MAC_{\mathsf{sk_1}}(N \,||\, NACK) = (1101)_2\\
&MAC_{\mathsf{sk_2}}(N \,||\, ACK) = (0110)_2;\ MAC_{\mathsf{sk_2}}(N \,||\, NACK) = (1111)_2\\
&MAC_{\mathsf{sk_3}}(N \,||\, ACK) = (0101)_2;\ MAC_{\mathsf{sk_3}}(N \,||\, NACK) = (0111)_2\\
&MAC_{\mathsf{sk_4}}(N \,||\, ACK) = (0011)_2;\ MAC_{\mathsf{sk_4}}(N||NACK) = (1110)_2
\end{aligned}$$

Let's say the root of an aggregation tree computed the $\Delta_{root} = (0100)_2$ as described in Subsection 8.3.4. The base station calculates $\Delta_{ack} = (1001)_2$ according to the Equation 8.3. Then it calculates the complainer code $c = (1101)_2$ according to the Equation 8.4. The base station also computes the complainer codes for each sensor node according to the Equation 8.5 as follows:

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

The base station compares complainer code c with individual complainer code c_i and finds that $c = c_4$. So, the base station identifies that the s_4 complained, during verification of inclusion phase.

The following binary illustration helps visualize that applying XOR is negating the contribution of the authentication code with NACK. The base station receives the

[1] $C_{n,1} = \binom{n}{1}$

following from the root an aggregation tree, which includes one authentication code with NACK.

$$\underbrace{\begin{array}{cccc} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ \hline 0 & 1 & 0 & 0 \end{array}}_{\Delta_{root}}$$

The base station calculates $\Delta_{ack}, \Delta_{nack}$ and complainer code c as follows:

$$\underbrace{\begin{array}{cccc} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ + \; 0 & 0 & 1 & 1 \\ \hline 1 & 0 & 0 & 1 \end{array}}_{\Delta_{ack}} \quad \underbrace{\begin{array}{cccc} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ \hline 1 & 0 & 1 & 1 \end{array}}_{\Delta_{nack}}$$

$$\underbrace{\begin{array}{cccc} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ \hline 1 & 1 & 0 & 1 \end{array}}_{c}$$

Then the base station calculates the complainer code for each node i as follows:

$$\underbrace{\begin{array}{cccc} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ \hline 0 & 1 & 0 & 0 \end{array}}_{c_1} \quad \underbrace{\begin{array}{cccc} 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ \hline 1 & 0 & 0 & 1 \end{array}}_{c_2} \quad \underbrace{\begin{array}{cccc} 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ \hline 0 & 0 & 1 & 0 \end{array}}_{c_3} \quad \underbrace{\begin{array}{cccc} 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ \hline 1 & 1 & 0 & 1 \end{array}}_{c_4}$$

And concludes that node 4 is complaining. In general, to find k complainers the base station needs to do $C_{n,k}$ [2] calculations and the same number of comparisons to find k complainers.

If there are more that one complainers in the network then the base station uses the following recursive algorithm to find all the complainers in the network.

[2] $C_{n,k} = \binom{n}{k}$

Algorithm 1 Pseudo algorithm to detect more than one complainer

1: BS simulates the commitment trees using ACK messages.

2: BS asks the root of an aggregation tree to send the authentication codes of all the trees in its forest.

3: BS compares the received authentication codes with simulated authentication codes and identifies the complainer tree.

4: BS repeats the process until it finds all the leaves who sent authentication codes with NACK messages.

8.3.6 Detecting an adversary

Once the base station finds the node who send NACK during the verification of inclusion phase, it interacts with the nodes in the network to trace an adversary responsible for it. The bases station utilizes the following algorithm to trace an adversary.

Algorithm 2 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

Talk About Impact.

9. ANALYSIS

The following theorem proves that the complete binary tree structure is optimal for commitment tree generation.

Theorem 9.0.1 *Binary commitment tree is optimal in terms of verification for m-ary tree, as it requires minimum number of off-path values.*

Proof Consider the case of a tertiary tree, other m-ary tree arguments follows the same manner. Let m be the number of leaves in a commitment tree.

For the given binary commitment tree, each leaf vertex needs $\log_2 m$ off-path values in the verification phase. The total off-path values needed in the given commitment tree is $[m \log_2 m]$. For the given tertiary commitment tree, each leaf vertex needs $2 \log_3 m$ off-path values in the verification phase. The total off-path values needed in given commitment tree is $[2m \log_3 m]$.

$$
\begin{aligned}
&\text{Let } y = \log_3 m \\
&y = \tfrac{\log_2 m}{\log_2 3} \text{ where } \log_2 3 > 1 \\
&y \log_2 3 = \log_2 m \\
&\log_3 m \cdot \log_2 3 = \log_2 m \\
&\log_3 m = \tfrac{\log_2 m}{\log_2 3} \\
&2m \log_3 m = 2m \tfrac{\log_2 m}{\log_2 3} = \tfrac{2}{\log_2 3} m \log_2 m \\
&2m \log_3 m = (1.2618) m \log_2 m \\
&2m \log_3 m > m \log_2 m
\end{aligned}
$$

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

9.1 Bandwidth Analysis

Our protocol creates a complete binary tree for any given aggregation tree. Hence, if we have n nodes in an aggregation tree, then at max there will be $2^{\lg n+1}-1 = 2n-1$ vertices in the commitment tree.

In both the approaches FSwRD and FSwoRD, there are at least two signatures associated with all the vertices in the commitment tree. Hence, there are at least $2(2n-1)$ signatures created while creating the commitment tree.

In general, an intermediate node with n descendants receives $\lceil \log_2 n \rceil$ trees from its children. As we know, there are at least two signatures associated with each vertex, the node with n descendants receives at least $2\lceil \log_2 n \rceil$ signatures.

We claim that the binary representation of a non-negative number x illustrates the payload decomposition of the sensor node S, where $x = 1 +$ number of descendants of S. For example, if sensor node S has 22 descendants then $x = 23$, $(x)_{10} = (10111)_2$. This means S has four complete binary trees in its payload, with the height of four, two, one and zero. Note that all the trees in the commitment payload are complete binary trees and no two trees have the same height. The reason 1 is added to the descendants of S is that the sensor node itself is collecting data.

In both the approaches FSwRD and FSwoRD, the number of transmitted signatures within the network remain the same as shown in Table 9.1 (we ignore the signature of the payload). But the number of times a data-item is being signed differ in both the approaches which impacts the number of certificates being transmitted in the network. The certificate is large in size and consumes a lot of bandwidth during the transmission. To do the further analysis, we developed a performance matrix based on number of transmitted certificates for commonly used network topologies. We analyzed different network topologies as shown in Table 9.2.

Table 9.1.: Analysis Table 1

Network Topology	Totality of Signatures Transmitted	
	FSwRD	FSwoRD
Star	$2n$	$2n$
Palm Tree	$2n$	$2n$
Complete Binary	$2n$	$2n$

Table 9.2.: Analysis Table 2

Network Topology	Totality of Certificates	
	FSwRD	FSwoRD
Star	$n-1$	$n-1$
Palm Tree	XX	$n-1$
Complete Binary	$n-1$	$n-1$

10. CONCLUSION AND FUTURE WORK

We give the proof that complete binary tree is an optimal data structure to build the commitment tree. We propose an algorithm which can detect an adversary in the network and remove it for all future queries.

11. 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.

11.1 Introduction

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

Definition 11.1.1 *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 2.4. 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 11.1.1. 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.

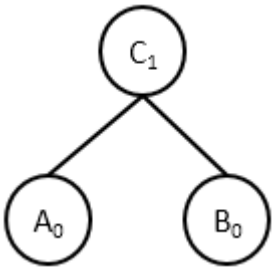

Fig. 11.1.: Smallest possible commitment tree

11.1.1 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 11.2 for the analysis purpose.

Example 11.1.1 *Let's say the vertices in the commitment tree of Figure 11.2 have the data-items defined as follows.*

$$\begin{aligned} A_0 &= < A_{id}, 1, 10, H(N||1||10) > \\ B_0 &= < B_{id}, 1, 20, H(N||1||20) > \\ C_1 &= < C_{id}, 2, 30, H(N||2||30||A_0||B_0) > \end{aligned} \tag{11.1}$$

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

$$\begin{aligned} &A \text{ uses} (A_0, B_0) \text{ to calculate} < 2, 30, H(N||2||30||A_0||B_0) > = C_1 \\ &B \text{ uses} (A_0, B_0) \text{ to calculate} < 2, 30, H(N||2||30||A_0||B_0) > = C_1 \end{aligned} \tag{11.2}$$

  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.

$$\begin{aligned} A'_0 &= < A_{id}, 1, 100, H(N||1||100) > \\ B'_0 &= < B_{id}, 1, 200, H(N||1||200) > \\ C'_1 &= < C_{id}, 2, 300, H(N||2||300||A'_0||B'_0) > \end{aligned} \tag{11.3}$$

  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

$$\begin{aligned} &A \text{ uses } (A_0, B'_0) \text{ to calculate} < 2, 210, H(N||2||210||A_0||B'_0) > \neq C'_1 \\ &B \text{ uses } (A'_0, B_0) \text{ to calculate} < 2, 120, H(N||2||120||A'_0||B_0) > \neq C'_1 \end{aligned} \tag{11.4}$$

  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.

$$\begin{aligned} A'_0 &= < A_{id}, 1, 100, H(N||1||10) > \\ C'_1 &= < C_{id}, 2, 120, H(N||2||120||A_0||B'_0) > \end{aligned} \tag{11.5}$$

  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.

$$\begin{aligned} B'_0 &= < B_{id}, 1, 110, H(N||1||110) > \\ A_0 &= < A_{id}, 1, 10, H(N||1||10) > \end{aligned} \tag{11.6}$$

  verification of inclusion

$$\begin{aligned} &A \text{ uses } (A_0, B'_0) \text{ to calculate } < 2, 120, H(N||2||120||A_0||B'_0) > = C'_1 \\ &B \text{ uses } (A_0, B_0) \text{ to calculate } < 2, 30, H(N||2||30||A_0||B_0) > \neq C'_1 \end{aligned} \tag{11.7}$$

  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.

$$\begin{aligned} A'_0 &= < A_{id}, 1, 100, H(N||1||10) > \\ B'_0 &= < B_{id}, 1, 200, H(N||1||20) > \\ C'_1 &= < C_{id}, 2, 300, H(N||2||300||A''_0||B_0) > \\ C''_1 &= < C_{id}, 2, 300, H(N||2||300||A_0||B''_0) > \end{aligned} \tag{11.8}$$

  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.

$$\begin{aligned} B_0'' &=< B_{id}, 1, 290, H(N||1||20) > \\ A_0'' &=< A_{id}, 1, 280, H(N||1||10) > \end{aligned} \tag{11.9}$$

verification of inclusion

$$\begin{aligned} &A \text{ uses } (A_0, B_0'') \text{ to calculate } < 2, 300, H(N||2||300||A_0||B_0'') > \neq C_1' = C_1'' \\ &B \text{ uses } (A_0'', B_0) \text{ to calculate } < 2, 300, H(N||2||300||A_0''||B_0) > = C_1' \neq C_1'' \end{aligned} \tag{11.10}$$

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.

11.2 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 2 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 11.2.1 *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 8.3.5. And based on the set c it creates a set $a = \{a_1, a_2, \ldots, a_n\}$ of the possible adversaries.*

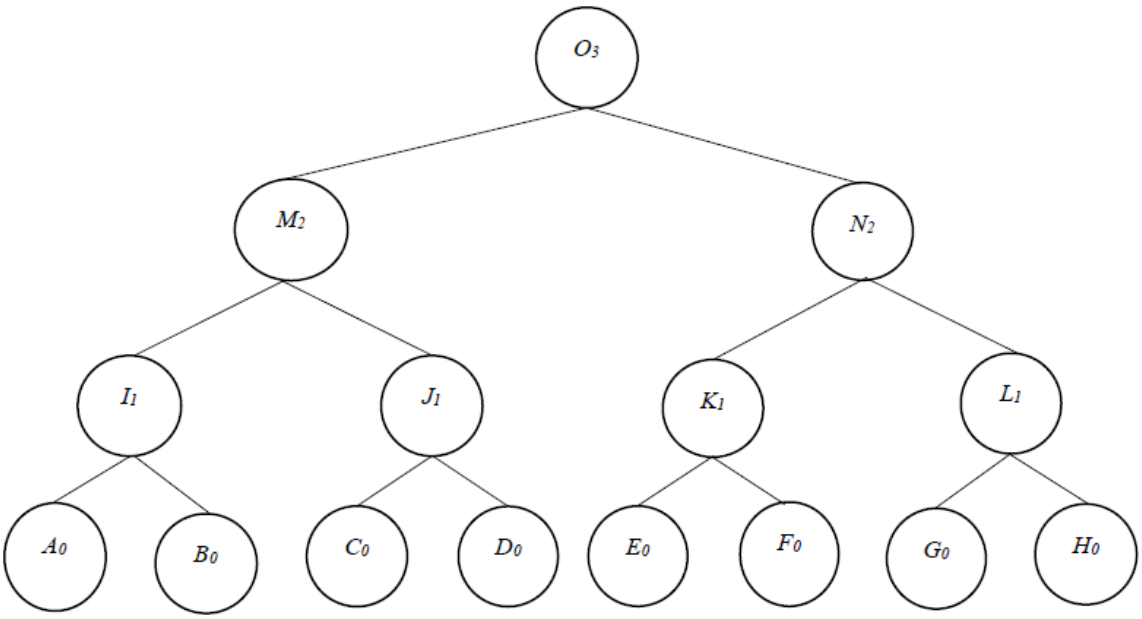

Fig. 11.2.: Smallest possible commitment tree

- *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* $a = \{(J, I)\}$

LIST OF REFERENCES

LIST OF REFERENCES

[1] M. Weiser, "The computer for the 21st century," *Scientific american*, vol. 265, no. 3, pp. 94–104, 1991.

[2] F. Stajano, *Security for Ubiquitous Computing*. John Wiley and Sons, Feb. 2002. [Online]. Available: http://www.cl.cam.ac.uk/~fms27/secubicomp/

[3] K. Ashton, "That internet of things thing," *RFiD Journal*, vol. 22, no. 7, pp. 97–114, 2009.

[4] Uber. https://www.uber.com/. [Online; accessed February 8, 2015].

[5] M. Weiser, R. Gold, and J. S. Brown, "The origins of ubiquitous computing research at parc in the late 1980s," *IBM systems journal*, vol. 38, no. 4, pp. 693–696, 1999.

[6] Anthem inc.'s data breach. http://www.bloomberg.com/news/articles/2015-02-05/signs-of-china-sponsored-hackers-seen-in-anthem-attack. [Online; accessed February 5, 2015].

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] M. D. Addlesee, A. Jones, F. Livesey, and F. Samaria, "The orl active floor," *IEEE Personal Communications*, vol. 4, pp. 35–41, 1997.

[13] 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.

[14] H. Karl and A. Willig, *Protocols and architectures for wireless sensor networks*. John Wiley & Sons, 2007.

[15] N. S. Shenck and J. A. Paradiso, "Energy scavenging with shoe-mounted piezoelectrics," *Ieee Micro*, vol. 21, no. 3, pp. 30–42, 2001.

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

[17] 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.

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

[19] 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.

[20] 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.

[21] B. Krishnamachari, D. Estrin, and S. Wicker, "The impact of data aggregation in wireless sensor networks," in *Distributed Computing Systems Workshops, 2002. Proceedings. 22nd International Conference on*. IEEE, 2002, pp. 575–578.

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

[23] S. ZareAfifi, R. Verma, B. King, P. Salama, and D. Kim, "Secure countermeasures to data aggregation attacks on sensor networks," in *Circuits and Systems (MWSCAS), 2012 IEEE 55th International Midwest Symposium on*. IEEE, 2012, pp. 856–859.

[24] 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.

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

[26] 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.

[27] 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.

[28] 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.

[29] L.-G. Alberto and W. Indra, "Communication networks: fundamental concepts and key architectures," *Mc GrawHill*, pp. 845–857, 2000.

[30] Final report of airbus a330-203. http://www.bea.aero/docspa/2009/f-cp090601.en/pdf/f-cp090601.en.pdf. [Online; accessed January 30, 2015].

[31] Nsidc weather report. http://nsidc.org/arcticseaicenews/2008/06/arctic-sea-ice-still-on-track-for-extreme-melt/. [Online; accessed January 30, 2015].

[32] Dmsp-f13. http://nsidc.org/data/docs/daac/f13_platform.gd.html. [Online; accessed January 30, 2015].

[33] Dmsp-f15. http://nsidc.org/data/docs/daac/f15_platform.gd.html. [Online; accessed January 30, 2015].

[34] 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.

[35] 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.

[36] Y. Zhang and X. Rong Li, "Detection and diagnosis of sensor and actuator failures using imm estimator," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 34, no. 4, pp. 1293–1313, 1998.

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

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

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

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

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

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

[43] R. Anderson, "Why cryptosystems fail," in *Proceedings of the 1st ACM Conference on Computer and Communications Security*. ACM, 1993, pp. 215–227.

[44] F. ECDSA, "186-3," *Digital Signature Standard (DSS)*, 2009.