SECURE DATA AGGREGATION SCHEME FOR SENSOR NETWORKS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Kavit Shah

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Electrical and Electronics Engineering

December 2014

Purdue University

Indianapolis, Indiana

This is the dedication.

ACKNOWLEDGMENTS

This is the acknowledgments.

PREFACE

This is the preface.

TABLE OF CONTENTS

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

ABSTRACT

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

This is the abstract.

1. INTRODUCTION

2. SENSOR NETWORKS/DATA AGGREGATION/SECURITY BACKGROUND

2.1 Sensor Networks

In sensor networks, thousands of sensor nodes may interact with the physical environment and collectively monitor an area, generating a large amount of data to be transmitted and reasoned about. With the recent advances in sensor network research, we can use tiny and cheap sensor nodes to obtain a lot of useful information about physical environment. For example, we can use them to discover temperature, humidity, water quality, lightning condition, pressure, noise level, carbon dioxide level, oxygen level, soil moisture, magnetic field, characteristics of objects such as speed, direction, and size, the presence or absence of certain kinds of objects, and all kinds of values about machinery like mechanical stress level or movement [1]. These versatile types of sensors, allow us to use sensor network in a wide variety of scenarios. For example, sensor networks are used in habitat and environment monitoring, health care, military surveillance, industrial machinery surveillance, home automation, scientific data collection, emergency fire alarm systems, traffic monitoring, wildfire tracking, wildlife monitoring and many other applications. **Meteorology and Hydrology in Yosemite National Park** [2], 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. Therefore, the sensor network was designed to operate with little or no human interaction. The application of a sensor network usually determines the design of the sensor nodes and the design of the network itself. As far as we know, there is no general architecture for sensor networks. Therefore, developing a protocol for sensor networks can certainly be challenging.

2.2 Data Aggregation

The sensor nodes in the network often have limited resources, such as computation power, memory, storage, communication capacity and most significantly, battery power. Furthermore, data communications between nodes consume a large portion of the total energy consumption. The in-network data aggregation reduces the energy consumption by aggregating the data before sending it to the parent node in the network which reduces the communications between nodes. For example, in-network data aggregation of the *SUM* function can be performed as follows. Each intermediate sensor node in the network forwards a single sensor reading containing the sum of all the sensor readings from all of its descendants, rather than forwarding each descendants' sensor reading one at a time to the base station. It is shown that the energy savings achieved by in-network data aggregation are significant [3]. The in-network data aggregation approach requires the sensor nodes to do more computations. But studies have shown that transmitting the data requires more energy than computing the data. Hence, in-network data aggregation is an efficient and a widely used approach for saving bandwidth by doing less communications between sensor nodes and ultimately giving longer battery life to sensor nodes in the network.

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

Definition 2.2.1 *[4]* ***Payload*** *is the part of the transmitted data which is the fundamental purpose of the transmission, to the exclusion of information sent with it such as meta data solely to facilitate the delivery.*

Definition 2.2.2 ***Information rate*** *for a given node is the ratio of the payloads, number of payloads sent divided by the number of payloads received.*

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

2.3 Energy Consumption

The sensor network's lifetime can be maximized by minimizing the power consumption of the sensor node's radio module. To estimate the power consumption, we have to consider the communication and computation power consumption at each sensor node. The radio module energy dissipation can be measured in two ways [5]. The first is measured in $E_{elec}(J/b)$, the energy dissipated to run the transmit or receive electronics. The second is measured in $\varepsilon_{amp}(J/b/m^2)$, the energy dissipated by the transmit power amplifier to achieve an acceptable 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 [6]. To transmit a k - bit packet at distance d, the energy dissipated is:

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

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

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

For μAmp wireless sensor, $E_{elec} = 50nJ/b$ and $\varepsilon_{amp} = 100pJ/b/m^2$ [5]. To sustain the sensor network for longer time all aspects of the sensor network should be efficient. For example, the networking algorithm for routing should be such that it minimizes the distance d between nodes. The signal processing algorithm should be such that it process the networking packets with less computations.

2.4 Bandwidth Analysis

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

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

Congestion is very useful factor while analyzing sensor network as it measures how quickly the sensor nodes will exhaust their batteries [7]. Some nodes in the sensor network have more congestion than the others, the highly congested nodes are the most important to the the network connectivity. For example, the nodes closer to the base station are essential for the network connectivity. The failure of the highly congested nodes may cause the sensor network to fail even though most of the nodes in the network are alive. Hence, it is desirable to have a lower congestion on the highly congested nodes even though it costs more congestion within the overall sensor network.

To distribute the congestion uniformly across the network, we can construct an aggregation protocol where each node transmits a single payload Defined in 2.2.1 to its parent in the aggregation tree. It implies there is $\Omega(1)$ congestion on each edge in the aggregation tree, thus resulting in $\Omega(\delta)$ congestion on the node according to Definition 2.3, where δ is the fanout of the node. In this approach, δ is dependent on the given aggregation tree. For an aggregation tree with n nodes, organized in the star tree topology congestion is $O(n)$ and the network organized in the palm tree topology the congestion is $O(1)$. This approach can create some highly congested nodes in the aggregation tree which is undesirable. In most of the real world applications we cannot control δ as the aggregation tree is random. Hence, it is desirable to have uniform distribution of congestion across the aggregation tree.

2.5 Security in In-network Data Aggregation

Sensor nodes are often deployed in open, hostile and unattended environments, so they are vulnerable to physical tampering due to the lower physical security and manufacturing cost. An adversary can obtain the confidential information from a compromised sensor and reprogram it with malicious software. The compromised node may then report an arbitrary false sensor readings to its parent node in the tree hierarchy, causing the final aggregation result to far deviate from the true aggregate result. This attack becomes more damaging when multiple adversaries succeeds in injecting false data into the network which may cause catastrophic consequences.

In-network data aggregation approach saves bandwidth by transmitting less payloads between sensor nodes but it gives more power to the intermediate aggregate sensor nodes. For example, a malicious intermediate sensor node who does aggregation over all of its descendants payloads, needs to tamper with only one aggregated payload instead of tampering with all the payloads received from all of its descendants. Thus, a malicious intermediate sensor node needs to do less work to skew the final aggregated payload. An adversary controlling few sensor nodes in the network can cause the network to return unpredictable payloads, making an entire sensor network unreliable. Notice that the more descendants an intermediate sensor node has the more powerful it becomes. Despite the fact that in-network aggregation makes an intermediate sensor nodes more powerful, some aggregation approaches requires strong network topology assumptions or honest behaviors from the sensor nodes. For example, in-network aggregation schemes in [7, 8] assumes that all the sensor nodes in the network are honest. Secure Information Aggregation (SIA) of [9], provides security for the network topology with a single-aggregator model.

Secure hierarchical in-network aggregation (*SHIA*) in sensor networks [10] presents the first and provably secure sensor network data aggregation protocol for general networks and multiple adversaries. We discuss the details of the protocol in the next chapter. *SHIA* limits the adversary's ability to tamper with the aggregation result with the tightest bound possible but it does not help detecting an adversary in the network. Also, we claim that same upper bound can be achieved with compact label format defined in the next chapter.

3. NETWORKING AND CRYPTOGRAPHY TOOLS

3.1 Hash Function

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

A hash function h should have the following properties:

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

We use SHA-256 hash algorithm as a hash algorithm.

3.2 Tree generation algorithms

3.3 Elliptic curve

4. SECURE HIERARCHICAL IN-NETWORK DATA AGGREGATION

Our work enhances Secure Hierarchical In-network data aggregation *(SHIA)* protocol of [10] by making it communication efficient, adding new capabilities to the protocol, achieving similar security goals with non-resilient aggregation functions and efficient way of analyzing the protocol. In this chapter, we summarize the important parts of *SHIA* protocol and relevant terms, to build the foundation to describe our protocol in the following chapters.

The goal of *SHIA* is to compute aggregate functions (such as *truncated* SUM, $AVERAGE$, $COUNT$, $\Phi - QUANTILE$) of the sensed values by the sensor nodes while assuming that partially a network is controlled by an adversary which is attempting to skew the final result.

4.1 Network Assumptions

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

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

4.2 Attacker Model

We consider a model with a polynomially bounded adversary of [9], which has a complete control over some of the sensor nodes in the network. Most significantly, the adversary can change the measured values reported by sensor nodes under its control. An adversary can perform a wide variety of attacks. For example, an adversary could report fictitious values (probably completely independent of the measured reported values), instead of the real aggregate values and the base station receives the fictitious aggregated information. Since in many applications the information received by the base station provides a basis for critical decisions, false information could have ruinous consequences. However, we do not want to limit ourselves to just a few specific selected adversarial models. Instead, we assume that the adversary can misbehave in any arbitrary way, and the only limitations we put on the adversary are its computational resources (polynomial in terms of the security parameter) and the fraction of nodes that it can have control over. We focus on **stealthy attacks** [9], where the adversary's goal 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 the SHIA is to prevent stealthy attacks.

4.3 Aggregate Definition and Security Goals

Definition 4.3.1 *According to [10], each sensor node s_i has a data value a_i assuming that all the data values are non-negative bounded by real value $a_i \in [0, r]$, where r is the maximum allowed data value. The objective of the* ***aggregation process*** *is to compute some function f over all the data values, i.e., $f(a_1, \ldots, a_n)$.*

4.3.1 Security goals

Definition 4.3.2 *According to [10], 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.*

Wagner [11] uses statistical estimation theory to quantify the effects of direct data injection on various aggregates as follows. An **estimator** is an algorithm $f : \mathbb{R}^n \to \mathbb{R}$ where $f(x_1, \ldots, x_n)$ is intended as an estimate of some real valued function of θ. We assume that θ is real valued and that we wish to estimate θ itself. Next, we define $\widehat{\Theta} := f(X_1, \ldots, X_n)$, where $X_1, \ldots, X_n$ are n random variables. We can define the root-mean-square(r.m.s) error (at θ):

$$rms(f) := \mathbb{E}[(\widehat{\Theta} - \theta)^2 | \theta]^{1/2} \tag{4.1}$$

Wagner in [11] defines **resilient estimators and resilient aggregation** as follows. A k-node attack A is an algorithm that is allowed to change up to k of the values $X_1, \ldots, X_n$ before the estimator is applied. In particular, the attack A is specified by a function $\tau_A : \mathbb{R}^n \to \mathbb{R}^n$ with the property that the vectors x and $\tau_A(x)$ never differ at more than k positions. We can define the root mean square(r.m.s) error associated with A by

$$rms^*(f, A) := \mathbb{E}[(\widehat{\Theta}^* - \theta)^2 | \theta]^{1/2} \tag{4.2}$$

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

$$rms^*(f, k) := max\{rms^*(f, A) : A \ is \ a \ k - node \ attack\} \tag{4.3}$$

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

Definition 4.3.3 *According to [11], 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. The summary of the Wagner's work is summarized

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

Table 4.1.: Summary of Wagner's work

in the Table 4.1. According to this quantitative study measuring the effects of direct data injection on various aggregates, and concludes that the aggregates (truncated SUM and AVERAGE) can be resilient under such attacks.

Without precise knowledge of application, the direct data injection attacks are indistinguishable from the malicious sensor readings. Hence, an optimal level of aggregation security is defined as follows.

Definition 4.3.4 *According to [10], 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.*

The goal of SHIA is to design an **optimally secure** aggregation algorithm with only **sublinear edge congestion**.

4.4 The SUM Aggregate Algorithm

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

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

4.5 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) [3]. 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 4.1 is shown in Figure 4.2.

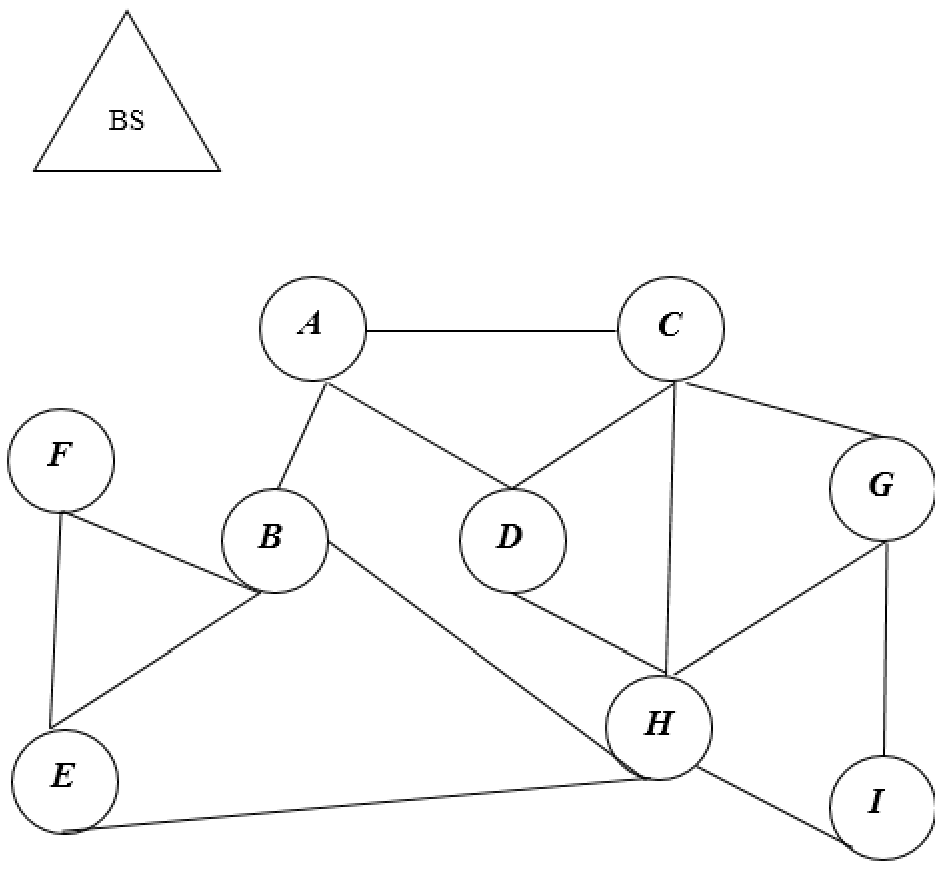

Fig. 4.1.: Network graph

To initiate the query dissemination phase, the base station broadcasts the query request message with the query nonce N in the aggregation tree. The query request message contains new query nonce N for each query to prevent replay attacks in the network. It is very important that the same nonce is never re-used by the base station. $SHIA$ uses **hash chain** to generate new nonce for each query. A hash chain is constructed by repeatedly evaluating a pre-image resistant hash function h on some initial random value, the final value (or “anchor value”) is preloaded on the nodes in the network. The base station uses the pre-image of the last used value as the nonce for the next broadcast. For example, if the last known value of the hash chain is $h^i(X)$, then the next broadcast uses $h^{i-1}(X)$ as the nonce; X is the initial random value. When a node receives a new nonce N, it verifies that N is a precursor to the most recently received (and authenticated) nonce N on the hash chain, i.e., $h_i(N) = N$ for some i bounded by a fixed k of number of hash applications. A hash chain prevents an adversary from predicting the query nonce for future queries as it has to reverse the hash chain computation to get an acceptable pre-image.

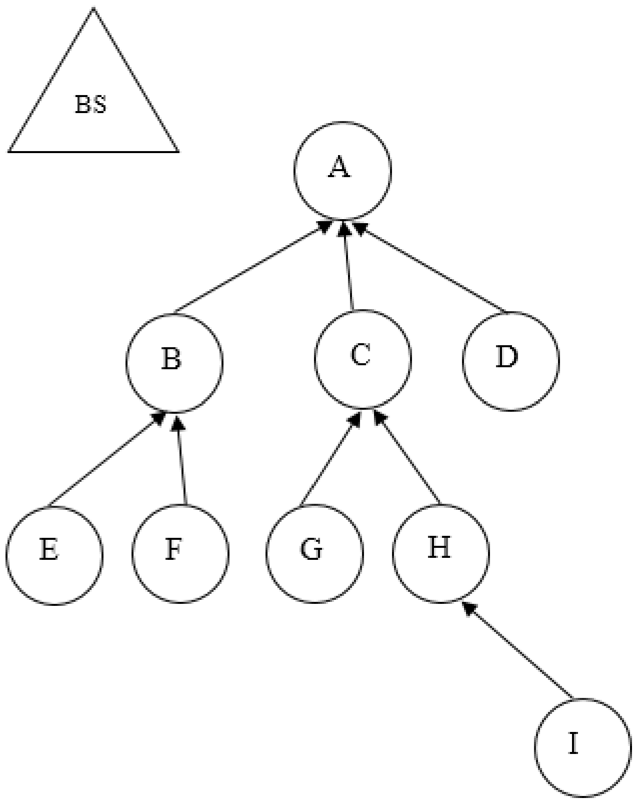

Fig. 4.2.: Aggregation tree for network graph in Figure 4.1

4.6 Aggregate commit

The aggregate commit phase constructs cryptographic commitments to the data values and to the intermediate in-network aggregation operations. These commitments are then passed on to the base station by the root of an aggregation tree. The base station then rebroadcasts the commitments to the sensor network using an authenticated broadcast so that the rest of the sensor nodes in the network can verify that their respective data values have been incorporated into the final aggregate value.

4.6.1 Aggregate commit: Naive Approach

In the naive approach, during aggregation process each sensor node computes a cryptographic hash of all its inputs (including its own data value). The aggregation result along with the hash value called a label, is then passed on to the parent in the aggregation tree. The label is format is described in Definition 4.6.1. Figure 4.3 shows a commitment tree for the aggregation tree shown in Figure 4.2. Conceptually, a commitment tree is a logical tree built on top of an aggregation tree, with additional aggregate information attached to the nodes to help in the result checking phase.

Definition 4.6.1 *[10] A commitment tree is a tree where each vertex has an associated label representing the data that is passed on to its parent. The labels have the following format:*

<count, value, complement, commitment>

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

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

Internal vertices represent aggregation operations, and have labels that are defined

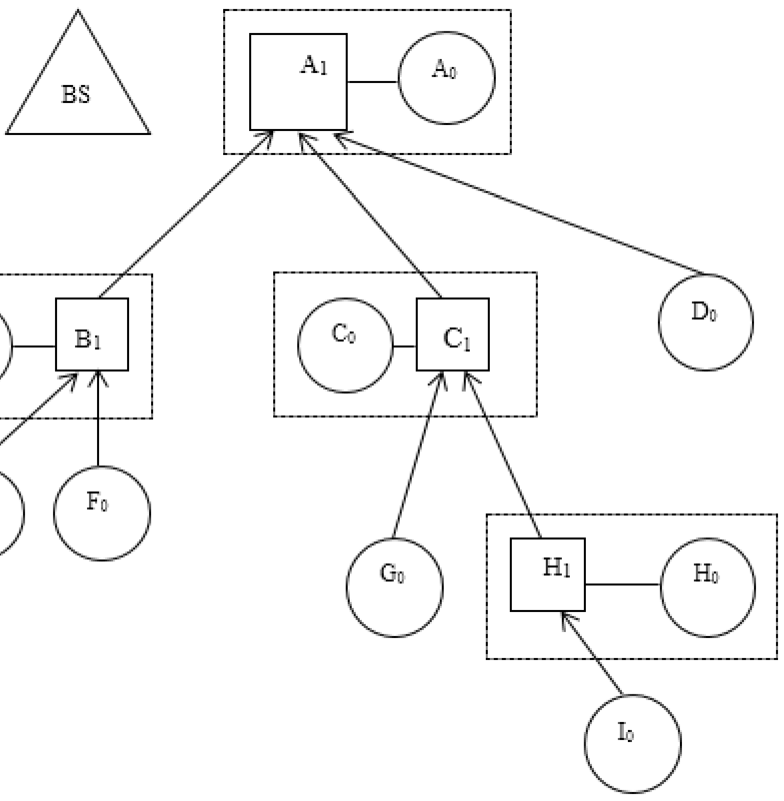

Fig. 4.3.: Naive commitment tree for Figure 4.2. For each sensor node s, s_0 is its leaf vertex, while s_1 is the internal vertex representing the aggregate computation at s (if any). The labels of the vertices on the path of node I to the root are shown from Equation 4.4 to 4.8.

based on their children. Suppose an internal vertex has child vertices with the following labels: u_1, u_2, ... , u_q, where $u_i = < c_i$, v_i, $\bar{v_i}$,h_i>. Then the vertex has label <c, v, $\bar{v}$, h>, with $c = \sum c_i$, $v = \sum v_i$, $\bar{v} = \sum \bar{v_i}$ and $h = H[N||c||v||\bar{v}||u_1||u_2||\ldots||u_q]$. The labels of the vertices of the commitment tree of Figure 4.3 are shown below.

$$I_0 =< 1, a_I, r - a_I, I > \tag{4.4}$$

$$H_1 =< 2, v_{H_1}, r - v_{H_1}, H[N||2||v_{H_1}||\bar{v_{H_1}}||H_0||I_0] > \tag{4.5}$$

$$B_1 =< 3, v_{B_1}, r - v_{B_1}, H[N||3||v_{B_1}||\bar{v_{B_1}}||B_0||E_0||F_0] > \tag{4.6}$$

$$C_1 =< 4, v_{C_1}, r - v_{C_1}, H[N||4||v_{C_1}||\bar{v_{C_1}}||C_0||G_0||H_1] > \tag{4.7}$$

$$A_1 =< 9, v_{A_1}, r - v_{A_1}, H[N||9||v_{A_1}||\bar{v_{A_1}}||A_0||D_0||B_1||C_1] > \quad (4.8)$$

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

4.6.2 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 4.6.2 *[10] A commitment forest is a set of complete binary commitment trees such that there is at most one commitment tree of any given height.*

The commitment forest is built as follows. Leaf sensor nodes in the aggregation tree originate a single-vertex commitment forest, which they then communicate to their parent sensor nodes. Each internal sensor node s originates a similar single-vertex commitment forest. In addition, s also receives commitment forests from each of its children. Sensor node s keeps track of which root vertices were received from which of its children. It then combines all the forests to form a new forest as follows. Suppose s wishes to combine q commitment forests $F_1,\ldots,F_q$. Note that since all commitment trees are complete binary trees, tree heights can be determined by inspecting the count field of the root vertex. We let the intermediate result be $F = F_1 \cup \ldots \cup F_q$, and repeat the following until no two trees are the same height in F. Let h be the smallest height such that more than one tree in F has height h. Find two commitment trees T_1 and T_2 of height h in F, and merge them into a tree of height $h+1$ by creating a new vertex that is the parent of both the roots of T_1 and T_2 according to the inductive rule in Definition 4.6.1.

Example 4.6.1 *The commitment-forest generation process for node A of Figure 4.2 is shown here.*

$A_0 =< 1, a_A, r - a_A, A >$

$D_0 =< 1, a_D, r - a_D, D >$

$E_0 =< 1, a_E, r - a_E, E >$

$B_1 =< 2, v_{B_1}, \bar{v_{B_1}}, H(N||2||v_{B_1}||\bar{v_{B_1}}||B_0||F_0) >$

$C_2 =< 4, v_{C_2}, \bar{v_{C_2}}, H(N||4||v_{C_2}||\bar{v_{C_2}}||H_1||C_1) >$

$A_1 =< 2, v_{A_1}, \bar{v_{A_1}}, H(N||2||v_{A_1}||\bar{v_{A_1}}||A_0||D_0) >$

$v_{A_1} = a_A + a_D;\ \bar{v_{A_1}} = r - a_A + r - a_D$

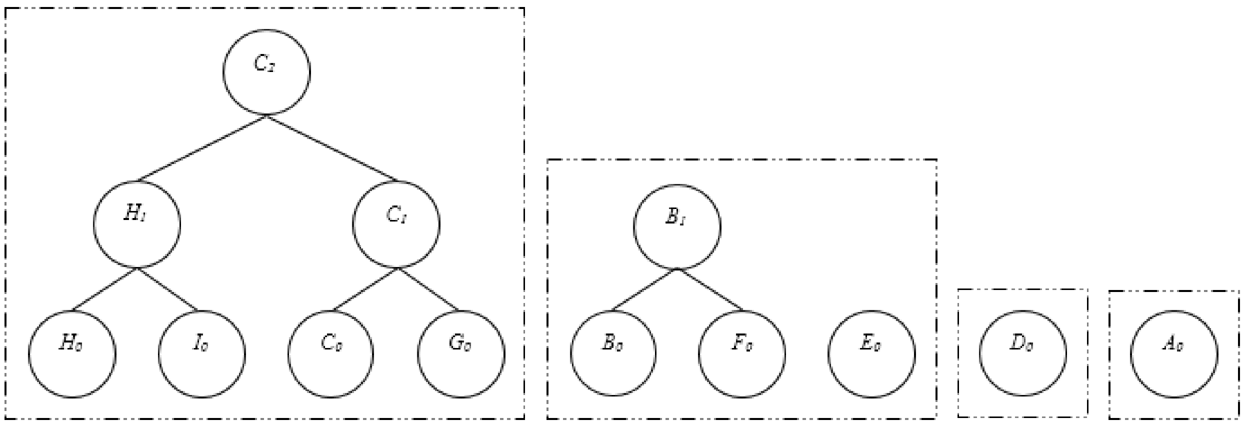

Fig. 4.4.: A receives C_2 from C, (B_1, B_0) from B, D_0 from D and generates A_0. The commitment forest received from a given sensor node is indicated by dashed-line box.

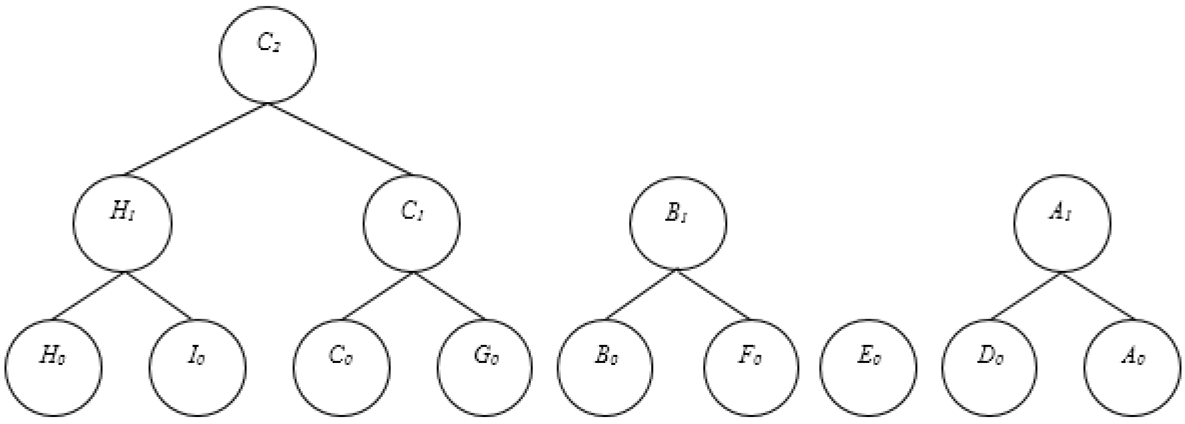

Fig. 4.5.: First Merge: A_1 vertex created by A.

$A_2 =< 4, v_{A_2}, \bar{v_{A_2}}, H(N||4||v_{A_2}||\bar{v_{A_2}}||B_1||A_1) >$

$v_{A_2} = v_{A_1} + v_{B_1}$; $\bar{v_{A_2}} = r - v_{A_1} + r - v_{B_1}$

$A_3 =< 8, v_{A_3}, \bar{v_{A_3}}, H(N||8||v_{A_3}||\bar{v_{A_3}}||C_2||A_2) >$

$v_{A_3} = v_{A_2} + v_{C_2}$; $\bar{v_{A_3}} = r - v_{A_2} + r - v_{C_2}$

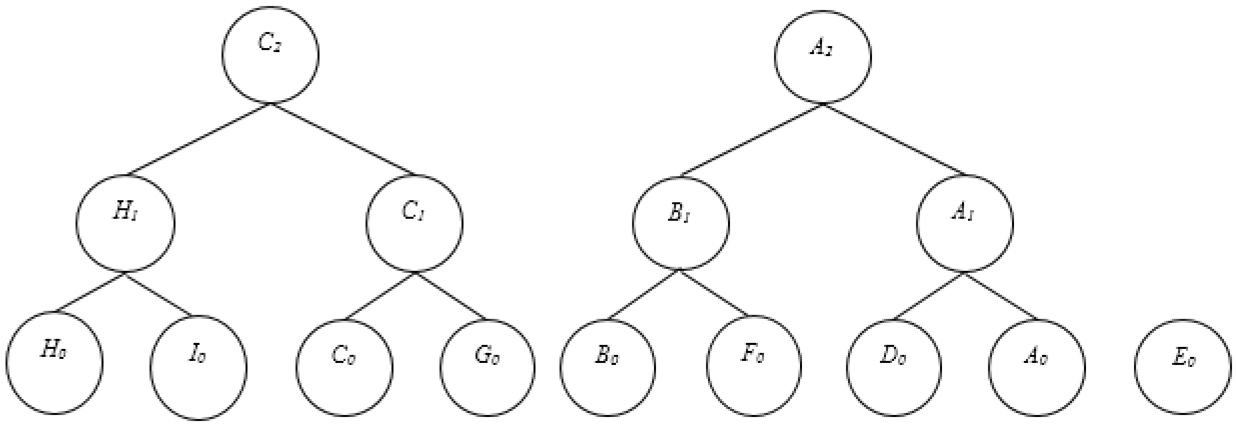

Fig. 4.6.: Second Merge: A_2 vertex created by A.

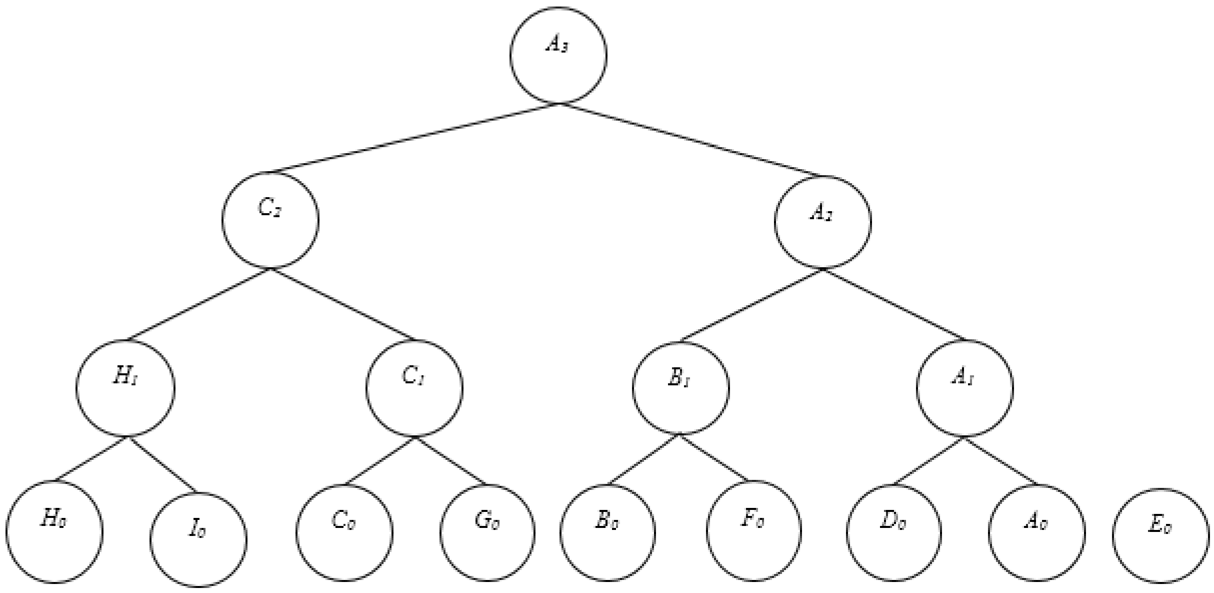

Fig. 4.7.: Third Merge: A_3 vertex created by A.

4.7 Result checking

SHIA presents novel distributed verification algorithm achieving provably optimal security while maintaining sublinear edge congestion. In our work, we take similar approach and add new capabilities to help find an adversary. Here, we describe the SHIA's result checking phase to build the basis for our work. The purpose of the result checking phase is to enable each sensor node s to independently verify that its data value as was added into the SUM aggregate, and the complement $(r - a_s)$ of its data value was added into the COMPLEMENT aggregate. First, the aggregation results from the aggregation-commit phase are sent by the base station using authenticated broadcast to every sensor node in the network. Each sensor node then individually verifies that its contributions to the respective SUM and COMPLEMENT aggregates were indeed counted. If so, it sends an authentication code to the base station. The authentication code is also aggregated for communication efficiency. When the base station has received all the authentication codes, it is then able to verify that all sensor nodes have checked that their contribution to the aggregate has been correctly counted. The result checking process has the following phases.

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

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

Definition 4.7.1 *[10] The set of off-path vertices for a vertex u in a tree is the set of all the siblings of each of the vertices on the path from u to the root of the tree that u is in (the path is inclusive of u).*

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

Verification of contribution. The leaf vertex calculates the root vertex's label using its own label and off-path vertex labels. This allows the leaf to verify that its label was not modified on the path to the root during the aggregation-commit process. Then it compares the the calculated root vertex's label with the label received from the

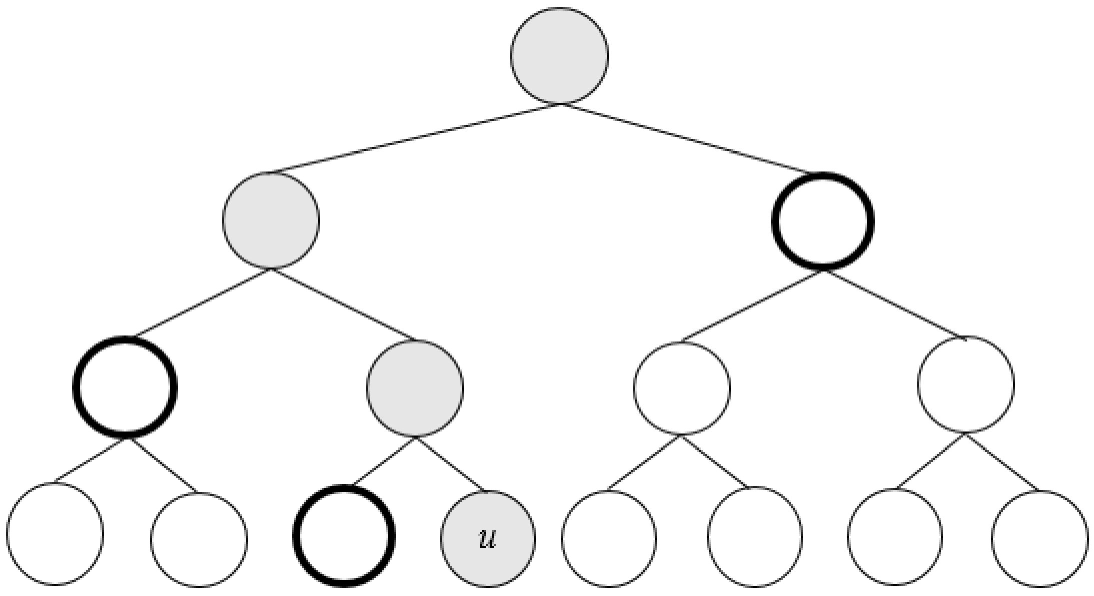

Fig. 4.8.: Off-path vertices from u are highlighted in bold.

base station via authenticated broadcast. If those two labels match then it proceeds to the next step with ACK message or with NACK message.

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

$$MAC_{K_s}(N||ACK) \tag{4.9}$$

$$MAC_{K_s}(N||NACK) \tag{4.10}$$

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

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

Theorem 4.7.1 *[10] Let the final SUM aggregate received by the base station be S. If the base station accepts S, then $S_L \leq S \leq (SL + \mu \cdot r)$ where S_L is the sum of the data values of all the legitimate nodes, μ is the total number of malicious nodes, and r is the upper bound on the range of allowable values on each node.*

The above theorem is proven by SHIA. SHIA achieves security over the truncated SUM which is a resilient aggregator according to Wagner [11]. Our protocol works on SUM which is non-resilient aggregate and achieves the similar security goals.

5. AGGREGATE COMMIT WITH VERIFICATION

In the previous chapter, we saw that SHIA limits the adversary's ability to manipulate the aggregation result with the tightest bound possible. SHIA does not require prior knowledge of network topology and works on hierarchical sensor network which might include multiple malicious sensor nodes, with only suboptimal congestion overhead. SHIA helps the sensor node verify, its reported sensor reading was aggregated correctly or not, by an an aggregate node. If an an aggregate node has tampered with the reported sensor reading then the relevant sensor node can detect the tampering and raise an alarm. But SHIA does not help detecting and revoking the malicious aggregate node from the network. We develop the protocol which identifies the malicious aggregate node in the network.

The high level idea of the aggregate commit with verification scheme is that all the leaves in the aggregation tree sends the signature of its data-item signed by itself, in addition to the data-item. The aggregate node proceeds to the aggregation only after verifying all the received the data-items. The aggregate node also signs its payload in addition to its data-item which is discussed in detail in the following sections.

5.1 Signing the Data-Item

Here, we describe structure of the data-item, how it is different from the label structure of SHIA and rational behind it.

Definition 5.1.1 *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:*

<id, count, value, commitment>

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

We remove the complement field from the label structure Defined 4.6.1. We think sending the complement filed is redundant. The complement field is used by the base station (the querier according to SHIA), before the result checking phase, to verify SUM + COMPLEMENT = $n \cdot r$; where n is the number of nodes in the network, r is the upper bound on the allowed sensor readings. We can achieve the similar upper bound without sending the complement field. As the querier knows n, r and it gets SUM from the root of the aggregation tree. If SUM $> n \cdot r$, then the base station knows some node or nodes in the network reported out of range readings.

There is one vertex s_0 for each sensor node s, which we call the leaf vertex of s. The data-item for leaf vertex s_0 is defined according to Equation 5.1 and associated signature to it is defined according to Equation 5.2.

$$s_0 \ = < s_{id}, 1, s_{value}, H(N||1||s_{value}) > \tag{5.1}$$

$$Sign(s_0) = E_{K_s}(H(s_0)) \tag{5.2}$$

where H is a collision resistant hash function, K_s is the private key of s, N is the query nonce. In aggregate commit with verification scheme, in addition to sending the data-item, each sensor node sends the signature of the data-item to its parent. The parent node verifies all the received signature and then proceeds to the aggregation.

5.1.1 Security Analysis

- A sensor node has the proof for the sent data-item and can not claim sending the different data-item to its parent in the future.
- Each data-item has an associated signature to it, which helps an aggregate node verify the authenticity of the data-item.

- The parent node has the proof for the received data-item, and can not claim receiving different data-items form its children in the future.

5.2 Signing the Commitment Payload

We define commitment payload based on the commitment forest Defined in 4.6.2.

Definition 5.2.1 *A **commitment payload** is a set of data-items of the root vertices of the trees in the outgoing commitment forest.*

For brevity, we use the term forest, payload instead of commitment forest, commitment payload respectively. In addition to sending all the data-items with their respective signatures, an aggregate node sends an additional signature to its parent, which is a signature of all the data-items in its payload.

Example 5.2.1 *The aggregation tree shown in Figure 5.1, the payload of sensor node C is show in Figure 5.4. The sensor node C sends all the data-items in its payload*

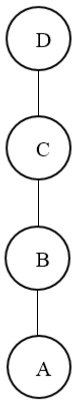

Fig. 5.1.: Palm aggregation tree

with their signatures to its parent sensor node D. Furthermore, C sends the signature of its payload $Sign(C_p)$ to D.

$$B_1 = < B_{id}, 2, B_{1value}, H(N||2||B_{1value}||A_0||B_0) >; Sign(B_1) = E_{K_B}(H(B_1)) \quad (5.3)$$

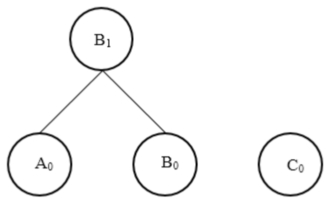

Fig. 5.2.: C's commitment commitment-payload

$$C_0 = < C_{id}, 1, C_{value}, H(N||1||C_{value}) >; Sign(C_0) = E_{K_C}(H(C_0)) \tag{5.4}$$

$$Sign(C_p) = \; E_{K_C}(H(C_0||B_1)) \tag{5.5}$$

5.2.1 Security Analysis

This additional signature signifies the following:

- An intermediate sensor node has verified all the data-items in its forest before creating its payload.
- From the parent node perspective, it has received all the verified data-items from the signer node and not from anywhere else.

5.3 Commitment Tree Generation

For the given aggregation tree the commitment forest is built as follows. Leaf sensor nodes in the aggregation tree create their leaf vertex by creating data-items and their respective signatures according to Equation 5.1, 5.2 which they send it to their parent as a payload in the aggregation tree. Each internal sensor node I in the aggregation tree also creates their leaf vertex and its signature. In addition, I receives the payload from each of its children which creates the forest for I. Once I verifies all the received signatures, it merges all the data-items in its forest with same count value to create its payload. Note that we can determine the height of the commitment tree from the count value.

Suppose I have to create its payload by merging i data-items D_1, D_2, …, D_i in its forest. First, I verifies the received signatures $Sign(D_1)$, $Sign(D_2)$, …, $Sign(D_i)$. Once verified, I starts merging the data-items as follows. Let c be the smallest count value in I's forest. The sensor node I finds two data-items D_1, D_2 in its forest with the same count value c and merges them into a new data-item with the count of $c+1$ as shown in Figure 5.3.

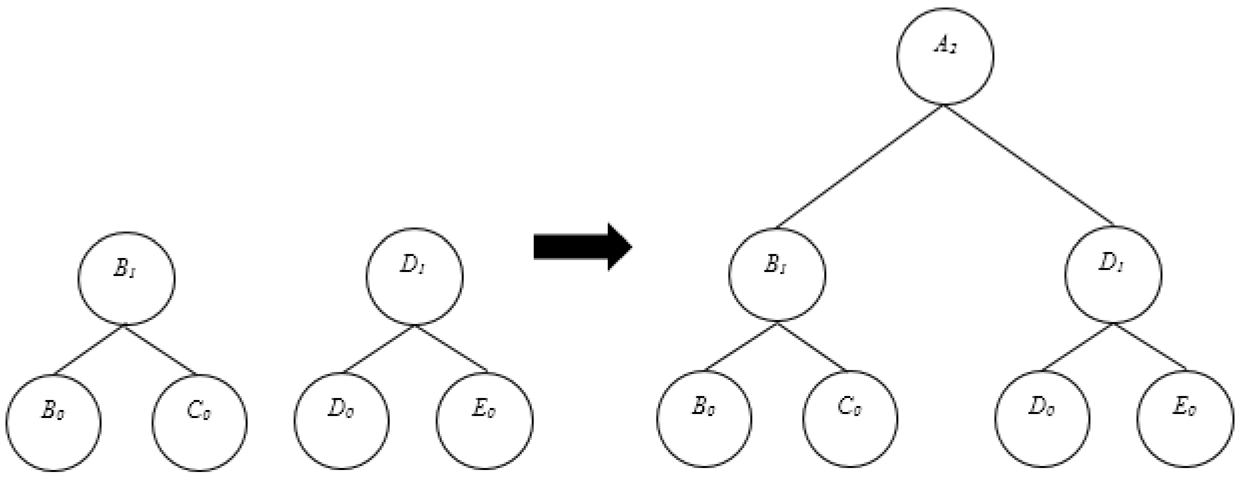

Fig. 5.3.: A has B_1, C_1 in his forest and aggregates those two trees and creates A_2.

It repeats the process until no two data-items in its forest have the same count value. An example of generating the payload by merging the data-items in the forest for the sensor node A in Figure 4.2 is illustrated in the following example.

Example 5.3.1 *The commitment-payload generation process for node A of Figure 4.2 is shown here.*

$A_0 =< A_{id}, 1, A_{value}, H(N||1||A_{value}) >;\ Sign(A_0) = E_{K_A}(H(A_0))$

$D_0 =< D_{id}, 1, D_{value}, H(N||1||D_{value}) >;\ Sign(D_0) = E_{K_D}(H(D_0))$

$B_0 =< B_{id}, 1, B_{value}, H(N||1||B_{value}) >;\ Sign(B_0) = E_{K_B}(H(B_0))$

$B_1 =< B_{id}, 2, B_{value}, H(N||2||B_{value}||E_0||F_0) >;\ Sign(B_1) = E_{K_B}(H(B_1))$

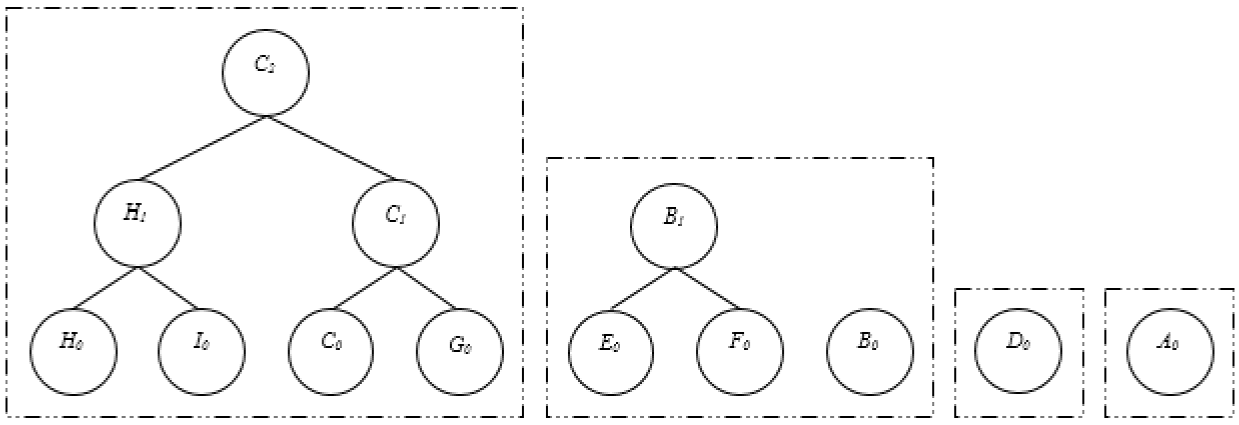

Fig. 5.4.: A receives C_2 from C, (B_1, B_0) from B, D_0 from D and generates A_0. The commitment payload received from a given sensor node is indicated by dashed-line box.

$Sign(B_P) = E_{K_B}(H(B_0||B_1))$

$C_2 =< C_{id}, 4, C_{value}, H(N||4||C_{value})||H_1||C_1) >; Sign(C_2) = E_{K_C}(H(C_2))$

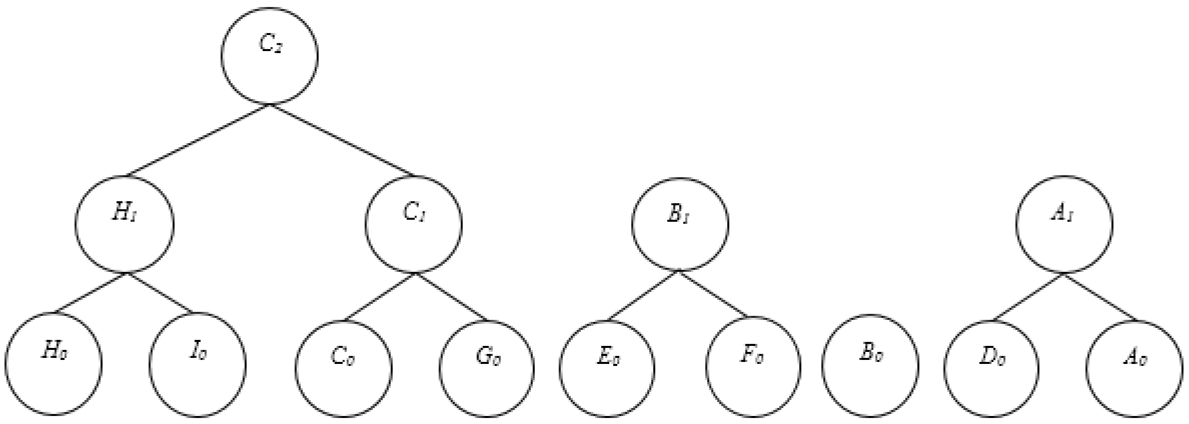

Fig. 5.5.: First Merge: A_1 vertex created by A.

$A_1 =< A_{id}, 2, A_{1value}, H(N||2||A_{1value}||A_0||D_0) >; Sign(A_1) = E_{K_A}(H(A_1))$

$where\ A_{1value} = A_{value} + D_{value}$

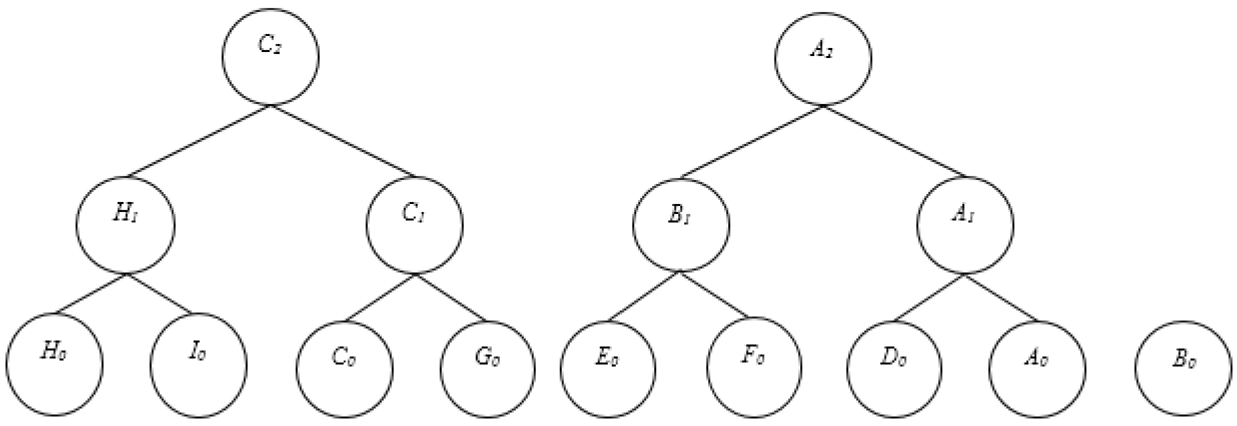

Fig. 5.6.: Second Merge: A_2 vertex created by A.

$A_2 =< A_{id}, 4, A_{2value}, H(N||4||A_{2value}||B_1||A_1) >; Sign(A_2) = E_{K_A}(H(A_2))$
$where\ A_{2value} = B_{1value} + A_{1value}$

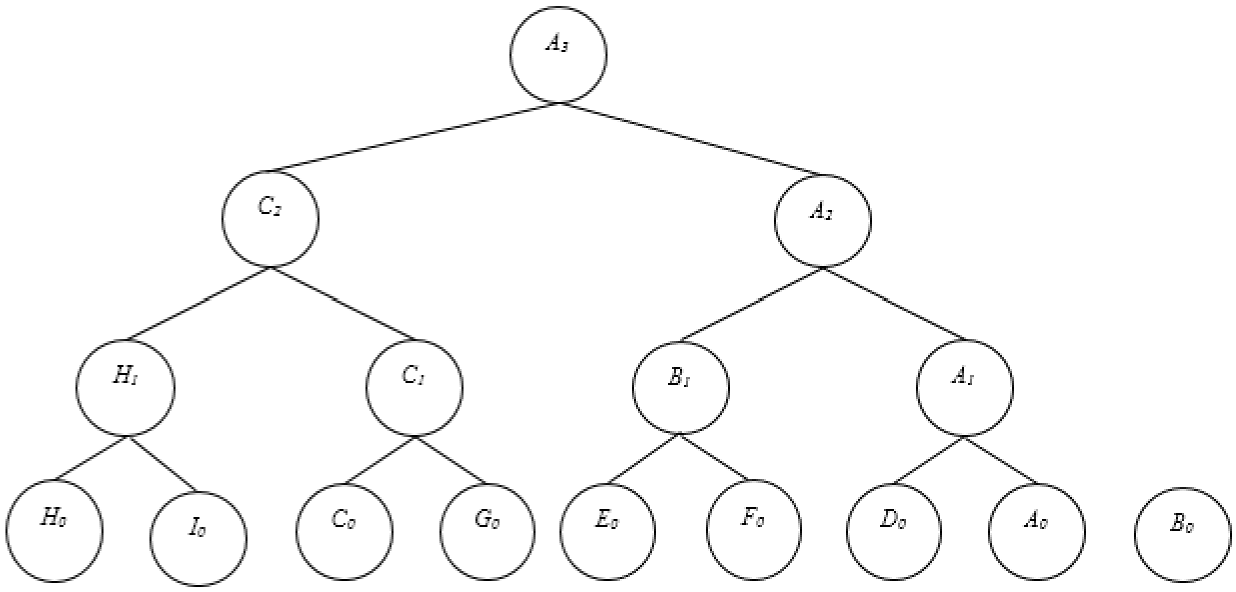

Fig. 5.7.: Third Merge: A_3 vertex created by A.

$A_3 =< A_{id}, 8, A_{3value}, H(N||8||A_{3value}||C_2||A_2) >; Sign(A_3) = E_{K_A}(H(A_3))$
$A_{3value} = A_{2value} + C_{2value}$

5.4 Bandwidth Analysis

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

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

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

5.5 Performance Analysis

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

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

5.6 Applications

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

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

For example, the base station wants to know the overall vote-count in the network. To do so, all the leaf nodes send their votes and the signature of their votes to their respective aggregate nodes in the network. The aggregate nodes receive votes with their signatures from all of their children voters. The aggregate nodes verify all the votes and count those votes. Then they forward the count and the signature of that count signed by the aggregate node to their respective parent in the aggregation tree. This process is repeated until the final count and its signature, is sent to the base station by the root of the aggregation tree.

Node power level, **Surveillance Application**

6. CHEATING

Definition 6.0.1 *A sensor node tampering with, the data-item to skew the final aggregate data-item or off-path values to conceal its tampering activity is consider as a* ***cheating****.*

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

If an aggregate node cheats, it has to cheat at two phases in the protocol to conceal its cheating. First, it has to cheat while creating a commitment tree by changing one or more fields in the final aggregate data-item at the end of its aggregation. Secondly, it has to cheat while distributing the off-path values to conceal its cheating activity from its children. We show that because of the commitment field in the data-item it is impossible for an aggregate node to cheat without being detected.

6.1 Assumptions

We make following assumptions for the adversary.

- It does not tamper with the off-path data-items received from its parent.
- 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

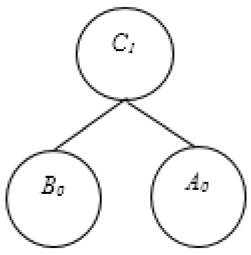

Fig. 6.1.: Smallest possible commitment tree

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.

Example 6.1.1 *Let's say the vertices in the commitment tree of Figure 6.2 have the data-items defined as follows. We did not include the signatures of these data-items as we assume that an aggregate node does not have the capability to reproduce the signatures of its childrens' data-items.*

$A_0 \; = < A_{id}, 1, 10, H(N||1||10) >$

$B_0 \; = < B_{id}, 1, 20, H(N||1||20) >$

$C_1 \; = < C_{id}, 2, 30, H(N||2||30||A_0||B_0) >$

- ***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

  $A_0 + B_0 \; = < 2, 30, H(N||2||30||A_0||B_0) > \; = C_1$ *(by A)*

  $A_0 + B_0 \; = < 2, 30, H(N||2||30||A_0||B_0) > \; = C_1$ *(by B)*

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

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

  $A'_0 = < A_{id}, 1, 100, H(N||1||100) >$

  $B'_0 = < B_{id}, 1, 200, H(N||1||200) >$

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

  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

  $A_0 + B'_0 = < 2, 210, H(N||2||210||A_0||B'_0) > \neq C'_1$*(by A)*

  $A'_0 + B_0 = < 2, 120, H(N||2||120||A'_0||B_0) > \neq C'_1$*(by B)*

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

  C tampers only with the value field in A_0, B_0's data-item and then applies aggregate commit algorithm.

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

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

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

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

  dissemination of root data-item

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

  dissemination of offpath values

  A receives $B''_0 = < B_{id}, 1, 290, H(N||1||20) >$ from C

  B receives $A''_0 = < A_{id}, 1, 280, H(N||1||10) >$ from C

  verification of inclusion

  $A_0 + B''_0 = < 2, 300, H(N||2||300||A_0||B''_0) > \neq C'_1 = C''_1$*(by A)*

  $A''_0 + B_0 = < 2, 300, H(N||2||300||A''_0||B_0) > = C'_1 \neq C''_1$*(by B)*

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

  C tampers A_0's value field and then applies aggregate commit algorithm.

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

  $C'_1 = < C_{id}, 2, 120, H(N||2||120||A_0||B'_0) >$

  C creates $B'_0 = < B_{id}, 1, 110, H(N||1||110) >$

  dissemination of root data-item

  A, B *receives* C'_1 *from C*

  dissemination of offpath values

  A receives $B'_0 = < B_{id}, 1, 110, H(N||1||110) >$ *from C*

  B receives $A_0 = < A_{id}, 1, 10, H(N||1||10) >$ *from C*

  verification of inclusion

  $A_0 + B'_0 = < 2, 120, H(N||2||120||A_0||B'_0) > = C'_1$ *(by A)*

  $A_0 + B_0 = < 2, 30, H(N||2||30||A_0||B_0) > \neq C'_1$ *(by B)*

Above example shows the significance of the commitment field in the data-item. If an aggregator changes the value field in one of its children's data-item then to hide its misbehavior from its children, it has to compensate the difference with the relevant off-path data-item. If an aggregator has two unique children (not including itself) and if it tries to change either one or both childrens' data-item then it can not create a data-item which will be accepted by both of its children. One of it's children will complain in the verification phase as they will not be able to calculate the same root data-item received from the base station.

6.2 Possible Cheater Analysis

Example 6.2.1 • *A_0 sends an authenticated code with NACK during verification of inclusion. Then possible adversaries are the following:*

  - I
  - (B, I)
  - (B, M)

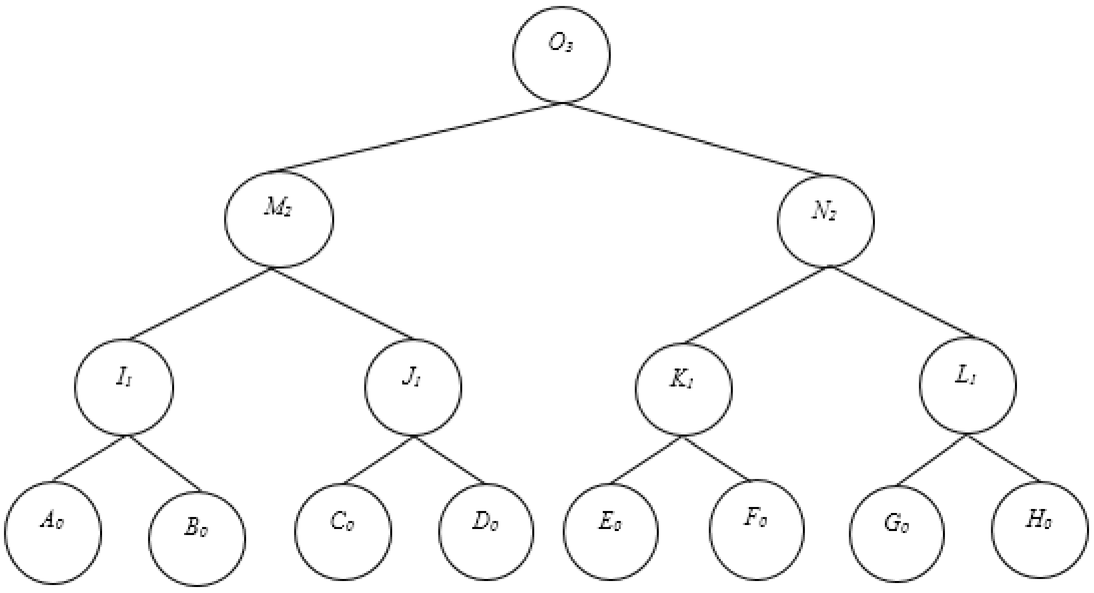

Fig. 6.2.: Smallest possible commitment tree

– (B, I, M)

7. VERIFICATION

7.1 dissemination final commitment

7.2 dissemination of off-path values

Two cases:

- With signatures
- Without signatures

7.3 verification of inclusion

7.4 collection of authentication codes

7.5 verification of authentication codes

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

$$MAC_{K_s}(N \mid\mid ACK) \tag{7.1}$$

$$MAC_{K_s}(N \mid\mid NACK) \tag{7.2}$$

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

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

$$\Delta_{ack} = \bigoplus_{i=1}^{n} MAC_{K_i}(N||ACK) \tag{7.3}$$

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

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

$$c := \Delta_{root} \oplus \Delta_{ack} \tag{7.4}$$

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

$$c_i := MAC_{K_i}(N \mid\mid ACK) \oplus MAC_{K_i}(N \mid\mid NACK) \tag{7.5}$$

Then it compares c with all c_i one at a time. The matching code indicates the complainer node. The base station needs to do $\binom{n}{1}$ calculations according to Equation 7.5 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 7.5.1 *If there are four nodes s_1, s_2, s_3, s_4 in an aggregation tree and their authentication codes with ACK, NACK message in the binary format are defined below.*

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

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

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

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

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

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

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

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

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

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

How XOR is negating the contribution of NACK.

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

The base station receives the following:

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

The base station does the following:

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

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

And concludes that node 4 is complaining.

7.6 Detect an adversary

Algorithm 1 Pseudo algorithm to detect an adversary

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

Theorem 7.6.1 *Binary commitment tree is optimal in terms of verification as it requires minimum number of off-path values.*

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

$\log_3(n) = y$

$3^y = n$

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

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

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

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

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

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

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

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

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

8. NOTES

- Do we hash node's id in the commitment field of its data-item?
- Do we send the payload signature in case of Figure 5.4?

LIST OF REFERENCES

LIST OF REFERENCES

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

[2] 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.

[3] 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.

[4] Payload computing. [Online]. Available: http://en.wikipedia.org/wiki/Payload_(computing)

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.