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

SHIA	Secure hierarchical in-network aggregation
SIA	Secure Information aggregation
ACK	Positive acknowledgment message
NACK	Negative acknowledgment message

NOMENCLATURE

Alanine	2-Aminopropanoic acid
Valine	2-Amino-3-methylbutanoic acid

GLOSSARY

chick	female, usually young
dude	male, usually young

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. SECURITY/DATA AGGREGATION BACKGROUND

Cite papers read and also summarize

[1] [2]

3. NETWORKING AND CRYPTOGRAPHY TOOLS

Networking - Algorithms of generating tree from a given graph. Optimal tree structure.

Hash

Elliptic curve

4. IN-NETWORK DATA AGGREGATION OVERVIEW

4.1 In-network data aggregation

Sensor networks are used in scientific data collection, emergency fire alarm systems, traffic monitoring, wildfire tracking, wildlife monitoring and many other applications. In sensor networks, thousands of sensor nodes may interact with the physical environment and collectively monitor an area, generating a large amount of data to be transmitted and reasoned about. 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 4.1.1 *[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 4.1.2 ***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.

4.2 Energy consumption

The sensor network's lifetime can be maximized by minimizing the power consumption of the sensor node's radio module. To estimate the power consumption, we have to consider the communication and computation power consumption at each sensor node. The radio module energy dissipation 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 [2]. To transmit a k - bit packet at distance d, the energy dissipated is:

$$E_{tx}(k,d) = E_{elec} \cdot k + \varepsilon_{amp} \cdot k \cdot d^2 \tag{4.1}$$

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

$$E_{rx}(k) = E_{elec} \cdot k \tag{4.2}$$

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.

4.3 Bandwidth analysis

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

$$Congestion = Edge\ congestion \cdot Fanout \tag{4.3}$$

Congestion is very useful factor while analyzing sensor network as it measures how quickly the sensor nodes will exhaust their batteries [6]. 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 **data-item** defined in $X.X$ 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 4.3, 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.

4.4 Security in In-network data aggregation

In-network data aggregation approach saves bandwidth by transmitting less payloads between sensor nodes but it gives more power to the intermediate aggregator 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 [6, 7] assumes that all the sensor nodes in the network are honest. Secure Information Aggregation (SIA) of [8], provides security for the network topology with a single-aggregator model.

Secure hierarchical in-network aggregation (*SHIA*) in sensor networks [9] 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.

5. SECURE HIERARCHICAL IN-NETWORK DATA AGGREGATION

Our work enhances Secure Hierarchical In-network data aggregation *(SHIA)* protocol of [9] 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.

5.1 Network assumptions

We assume a multi hop network with a set $S = \{s1, ..., s_n\}$ of n sensor nodes where all n nodes are alive and reachable. The network is organized in a rooted tree topology. The trusted base station resides outside of the network and has more computation, storage capacity then the sensor nodes in the network. Note 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.

5.2 Attacker model

We consider a model with a polynomially bounded adversary of [8], 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** [8], 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.

5.3 Aggregate definition and security goals

5.3.1 Aggregate definition

[9] 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., $f(a_1, \ldots, a_n)$.

5.3.2 Security goals

Definition 5.3.1 *[9] 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 [10] uses statistical estimation theory to quantify the effects of direct data injection on various aggregates as follows. An **estimator** is an algorithm $f : \mathbb{R}^n \rightarrow \mathbb{R}$ 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):

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

Wagner in [10] 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 $\tau_A : \mathbb{R}^n \rightarrow \mathbb{R}^n$ with the property that the vectors x and $\tau_A(x)$ never differ at more than k positions. We can define the root mean square(r.m.s) error associated with A by

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

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

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

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

Definition 5.3.2 *[10] We say that 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 in the below table.

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

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 5.3.3 *[9] 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**.

5.4 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

5.5 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 5.1 is shown in Figure 5.2.

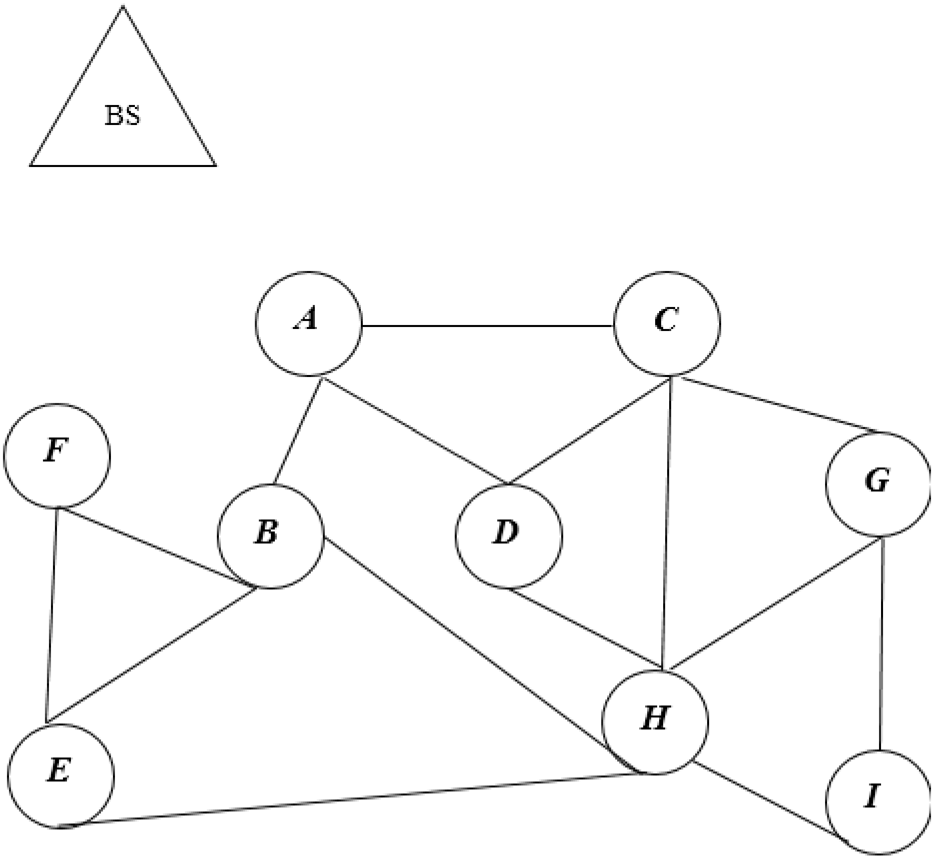

Fig. 5.1.: Network graph

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

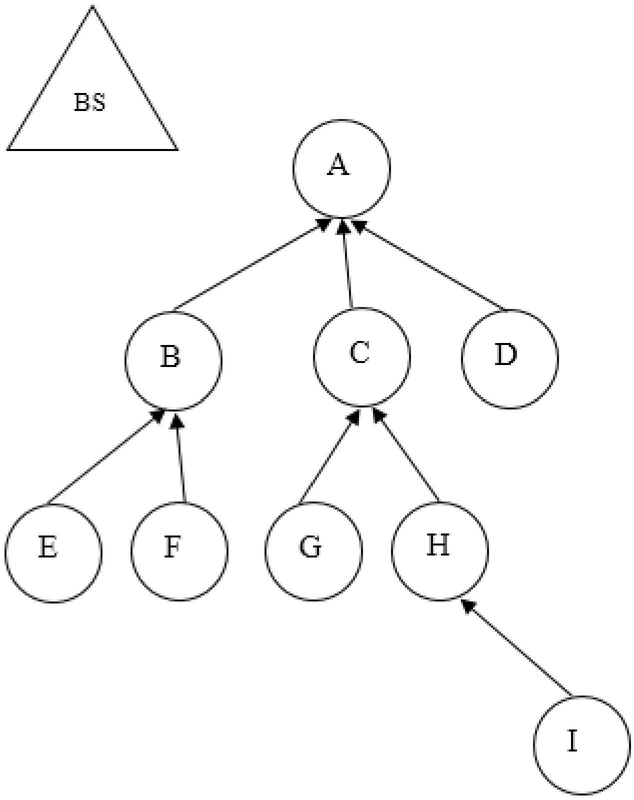

Fig. 5.2.: Aggregation tree for network graph in Figure 5.1

5.6 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.

5.6.1 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 5.6.1. Figure 5.3 shows a commitment tree for the aggregation tree shown in Figure 5.2. Conceptually, a commitment tree is a logical tree built on top of an aggregation tree, with additional aggregate information attached to the nodes to help in the result checking phase.

Definition 5.6.1 *[9] A commitment tree is a tree where each vertex has an associated label representing the data that is passed on to its parent. The labels have the following format:*

<*count, value, complement, commitment*>

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

There is one leaf vertex u_s for each sensor node s, which we call the leaf vertex of s. The label of u_s consists of count = 1, value = a_s where a_s is the data value of s, complement = $r - a_s$ where r is the upper bound on allowable data values, and commitment is the nodes unique ID.
Internal vertices represent aggregation operations, and have labels that are defined based on their children. Suppose an internal vertex has child vertices with the following labels: $u_1, u_2, \ldots, u_q$, where $u_i = < c_i, v_i, \bar{v}_i, h_i >$. Then the vertex has label <c,

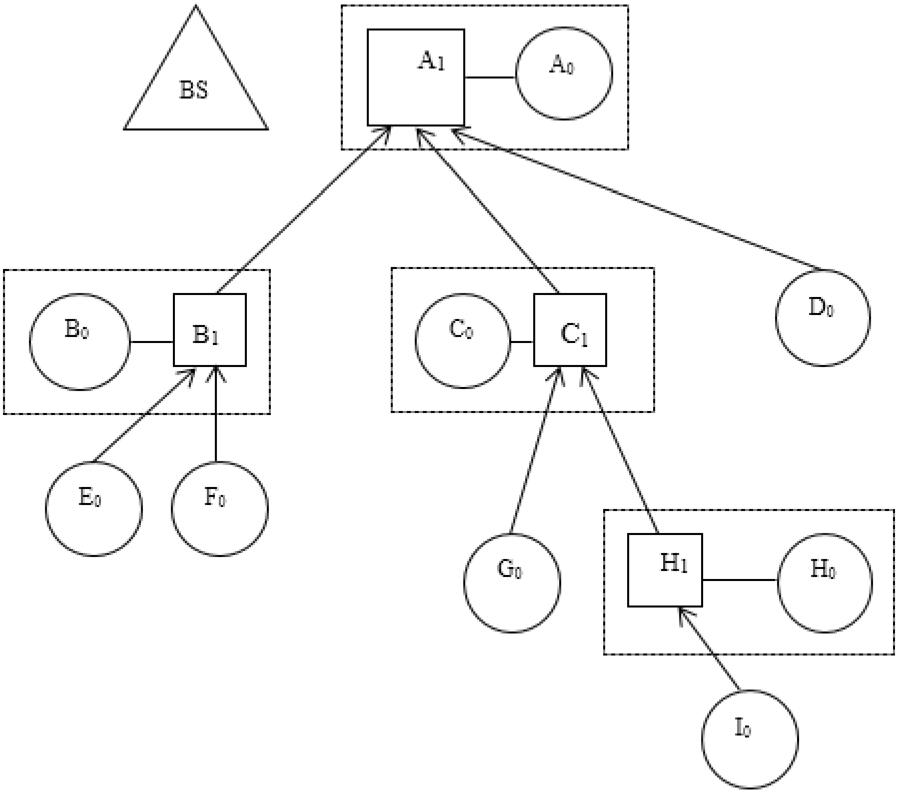

Fig. 5.3.: Naive commitment tree for Figure 5.2. For each sensor node s, s_0 is its leaf vertex, while s_1 is the internal vertex representing the aggregate computation at s (if any). The labels of the vertices on the path of node I to the root are shown from Equation 5.4 to 5.8.

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

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

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

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

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

$$A_1 =< 9, v_{A_1}, r - v_{A_1}, H[N||9||v_{A_1}||\bar{v_{A_1}}||A_0||D_0||B_1||C_1] > \tag{5.8}$$

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

5.6.2 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 5.6.2 *[9] A commitment forest is a set of complete binary commitment trees such that there is at most one commitment tree of any given height.*

The commitment forest is built as follows. Leaf sensor nodes in the aggregation tree originate a single-vertex commitment forest, which they then communicate to their parent sensor nodes. Each internal sensor node s originates a similar single-vertex commitment forest. In addition, s also receives commitment forests from each of its children. Sensor node s keeps track of which root vertices were received from which of its children. It then combines all the forests to form a new forest as follows. Suppose s wishes to combine q commitment forests $F_1, \ldots, F_q$. Note that since all commitment trees are complete binary trees, tree heights can be determined by inspecting the count field of the root vertex. We let the intermediate result be $F = F_1 \cup \ldots \cup F_q$, and repeat the following until no two trees are the same height in F. Let h be the smallest height such that more than one tree in F has height h. Find two commitment trees T_1 and T_2 of height h in F, and merge them into a tree of height $h + 1$ by creating a new vertex that is the parent of both the roots of T_1 and T_2 according to the inductive rule in Definition 5.6.1.

Example 5.6.1 *The commitment-forest generation process for node A of Figure 5.2 is shown here.*

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

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

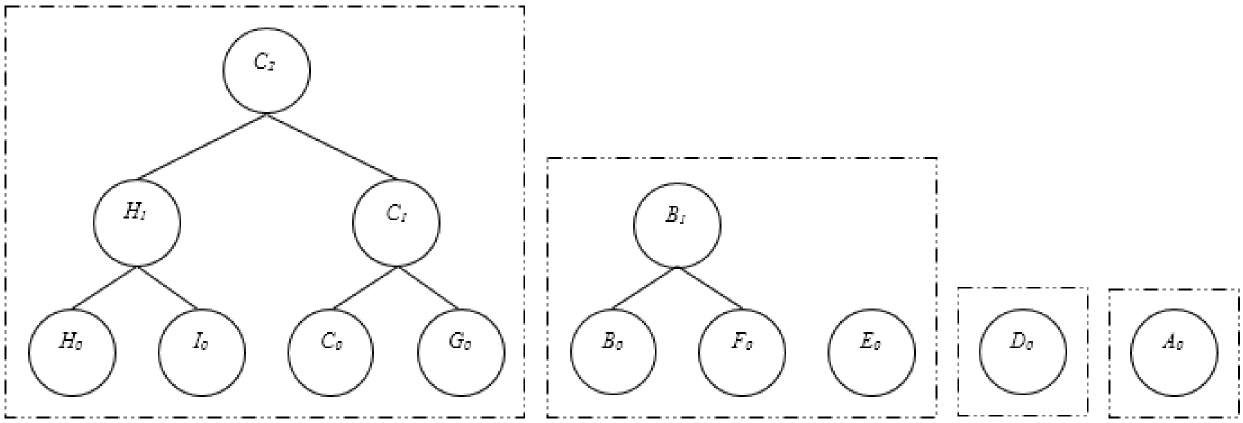

Fig. 5.4.: A receives C_2 from C, (B_1, B_0) from B, D_0 from D and generates A_0. The commitment forest received from a given sensor node is indicated by dashed-line box.

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

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

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

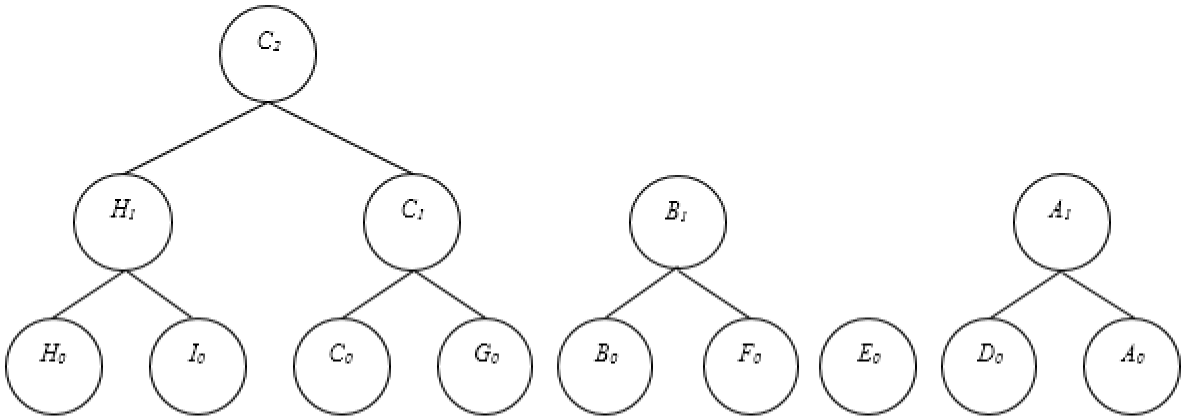

Fig. 5.5.: First Merge: A_1 vertex created by A.

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

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

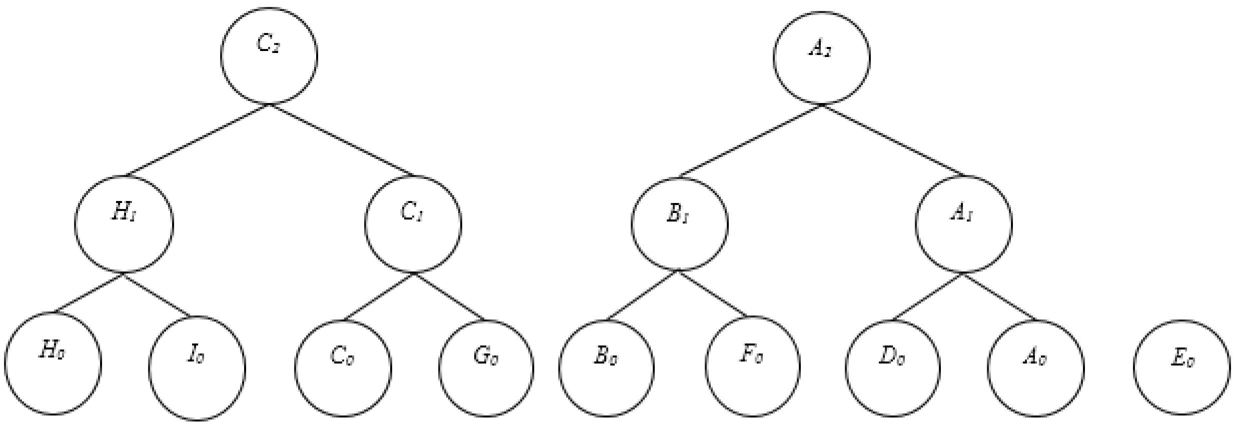

Fig. 5.6.: Second Merge: A_2 vertex created by A.

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

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

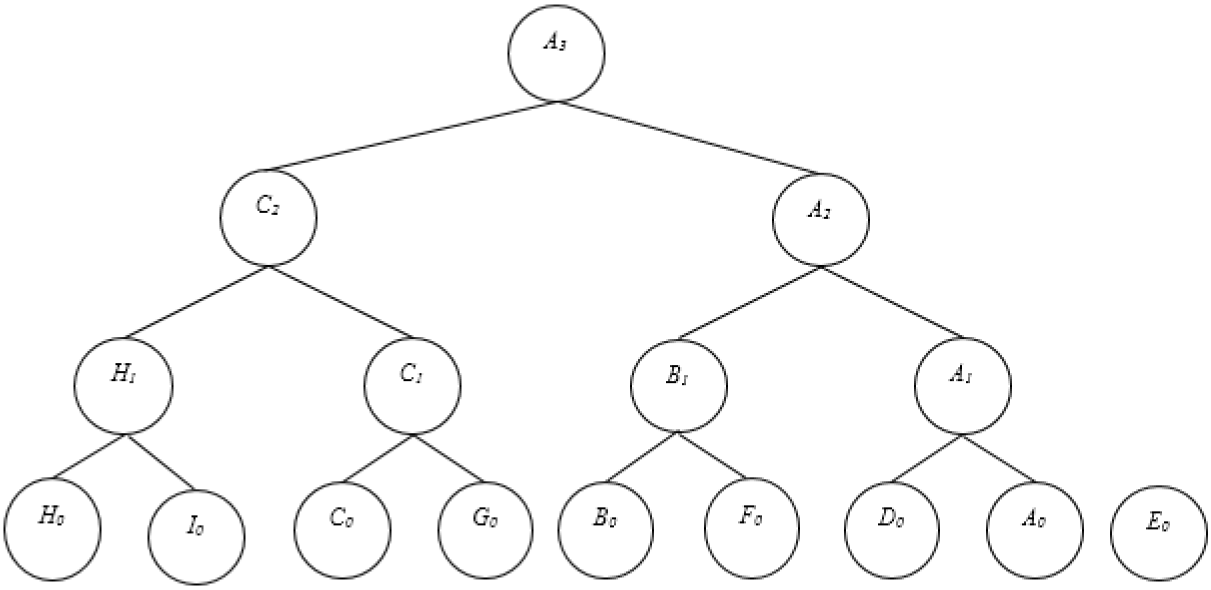

Fig. 5.7.: Third Merge: A_3 vertex created by A.

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

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

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

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

Definition 5.7.1 *[9] 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).*

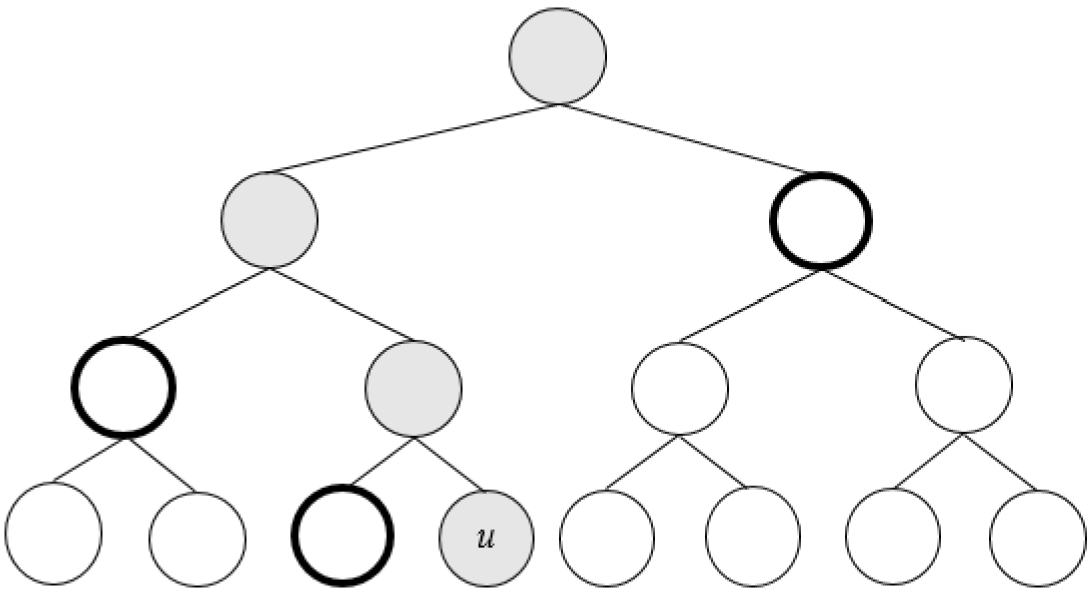

Fig. 5.8.: Off-path vertices from u are highlighted in bold.

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 a verification phase.

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

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

$$MAC_{K_s}(N||ACK) \tag{5.9}$$

$$MAC_{K_s}(N||NACK) \tag{5.10}$$

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 5.7.1 *[9] 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 [10]. Our protocol works on SUM which is non-resilient aggregate and achieves the similar security goals.

6. COMMITMENT TREE GENERATION WITH INTERNAL VERIFICATION

This chapter describes our approach of creating commitment tree which is built on top of commitment tree generation of SHIA mentioned in the previous chapter.

Definition 6.0.2 *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 identity 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.

There is one vertex s_0 for each sensor node s, which we call the leaf vertex of s.

$$s_0 =< s_{id}, 1, s_{value}, H(N||1||s_{value}) > \tag{6.1}$$

$$Sign(s_0) = E_{K_s}(H(s_0)) \tag{6.2}$$

In addition to sending the data-item, each sensor node sends the signature of the data-item to its parent. The parent node verifies the signature using the child node's public key which is included in child node's certificate. After verification of signature it proceeds with the aggregation.

Definition 6.0.3 *A* ***commitment payload*** *is a set of data-items of the root of the commitment trees in the outgoing commitment forest.*

Because of the signature, the sensor node has the proof for the sent data-item. It also prevents the parent or ancestor node from claiming the different received data-item.

For the given aggregation tree the commitment forest is built as follows. Leaf sensor nodes in the aggregation tree creates their leaf vertex according to Equation 6.1 which they send it to their parent in the aggregation tree. Each internal sensor node s in the aggregation tree also creates their leaf vertex. In addition, s also receives the commitment payload from each of its children which creates the commitment forest for s. It then merges all the data-items in its commitment forest with same count value to form a new commitment payload.

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

Example 6.0.1 *The commitment-payload generation process for node A of Figure 5.2 is shown here.*

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

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

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

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

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

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

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

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

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

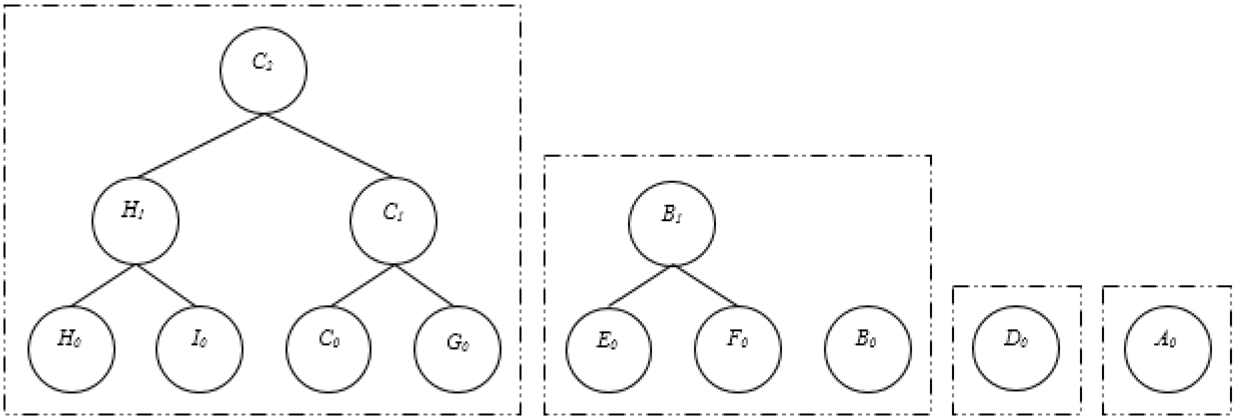

Fig. 6.1.: A receives C_2 from C, (B_1, B_0) from B, D_0 from D and generates A_0. The commitment payload received from a given sensor node is indicated by dashed-line box.

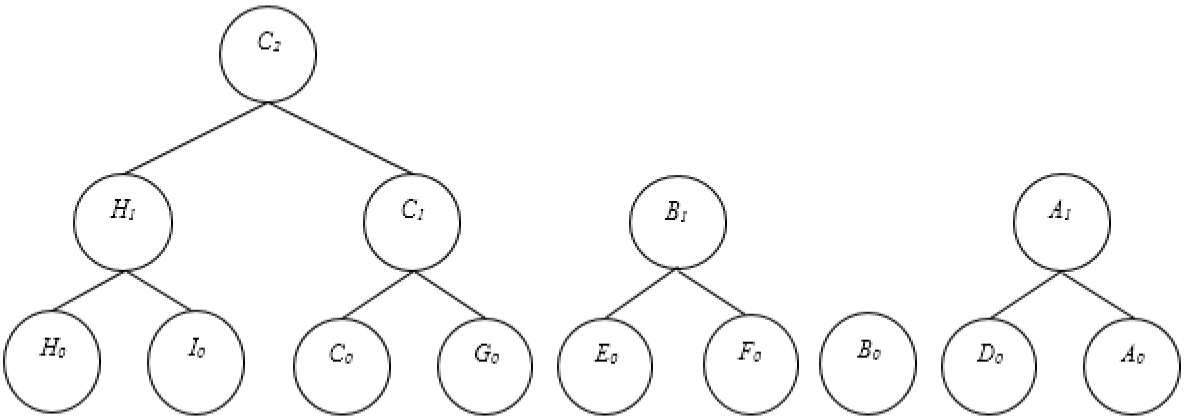

Fig. 6.2.: First Merge: A_1 vertex created by A.

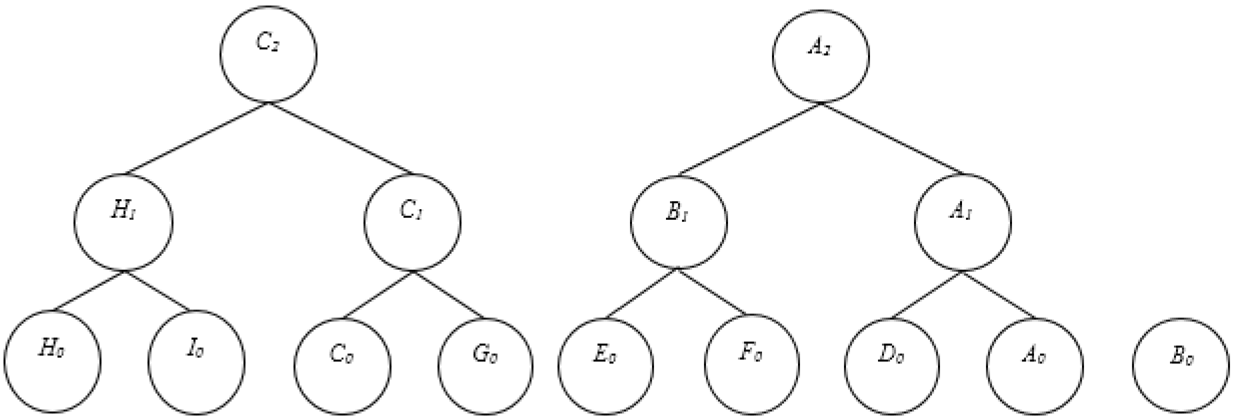

Fig. 6.3.: Second Merge: A_2 vertex created by A.

where $A_{2value} = B_{1value} + A_{1value}$

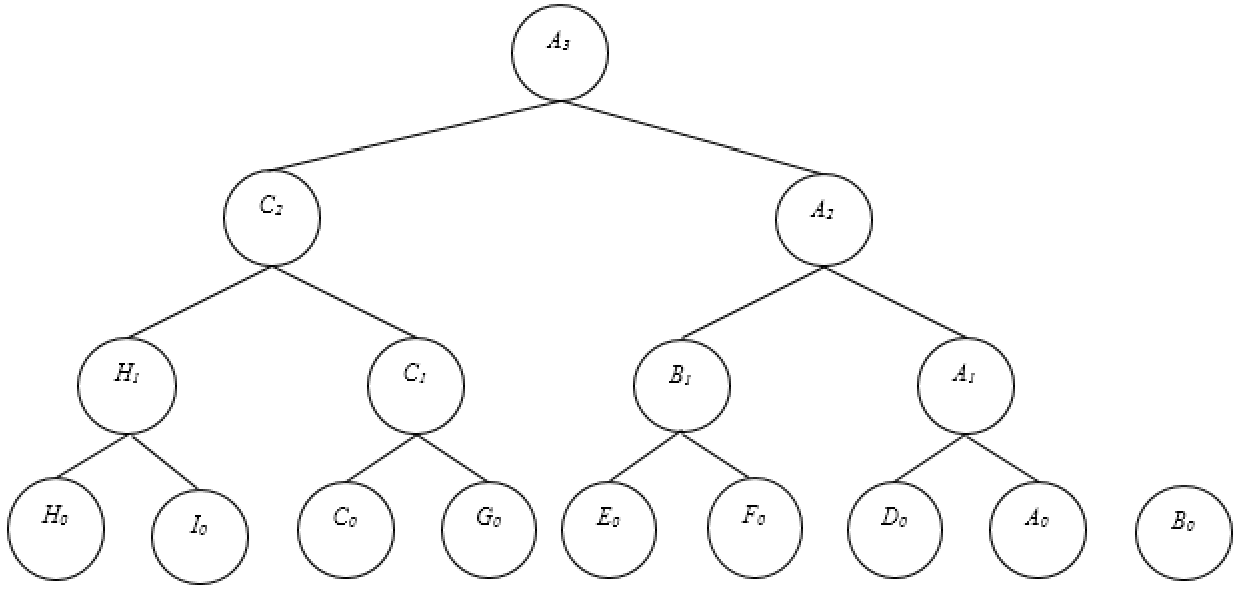

Fig. 6.4.: Third Merge: A_3 vertex created by A.

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

Talk about the certificates:

How many certificates does A need to know in this example ? In the above example, A need to know $D, B, C's$ certificate to verify their signatures. But if we use SHIA'a approach of creating commitment tree then A need to know $E's$ certificate as well. Hence, being root in as many tree as possible is the more efficient.

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.

$$\begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ \hline 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ \hline 1 & 0 & 1 & 1 \end{pmatrix}$$

The base station receives the following:

$$\begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ \hline 0 & 1 & 0 & 0 \end{pmatrix}$$

The base station does the following:

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

$$\begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ \hline 1 & 1 & 0 & 1 \end{pmatrix}$$

And concludes that node 4 is complaining.

7.6 Detect an adversary

Algorithm 1 Pseudo algorithm to detect an adversary

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

8. PROTOCOL

The commitment tree is a tree where each vertex has an associated label representing the data that is passed on to its parent. The messages have the following format: *MESSAGE*

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

SIGNATURE (MESSAGE)

$\text{Encryption}_{secret-key_{node}}$(HASH (MESSAGE))
500 bits

CERTIFICATES

Public key	Signature	ID
1000 bits	500 bits	20 bits

8.1 Aggregation-Commit Phase

In this phase, the network constructs a commitment structure. First, the sensor nodes at the highest depth in the aggregation tree (leaf nodes) send their payloads,defined according to Definition 8.1.1, to their parents in the aggregation tree. Each internal sensor node in the aggregation tree performs an aggregation operation whenever it receives payloads from all of its children. Whenever a sensor node performs an aggregation operation, it creates a commitment to the set of inputs used to compute the aggregate by computing a hash over all the inputs (including the commitments that were computed by its children). Both the aggregation result and the commitment creates a payload for the aggregator. Then the payload, with the signature of the payload signed by the sensor node are passed on to the parent of the sensor node. Once the final payloads and the signatures of those payloads are sent to the ***BaseStation***, if an adversary tries to claim a different commitment structure it gets caught during the verification phase. Our algorithm generates perfectly balanced binary trees to create commitment forests which saves the bandwidth in the verification phase compared to other approaches.

Definition 8.1.1 *[9] A* ***commitment tree*** *is a logical tree build on top of an* ***aggregation tree*** *where each vertex has an associated payload to it, representing data being passed on to its parent. The payload has the following format:*

$\{ \, id, \; count, \; value, \; commitment \, \}$

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

Our payload format is different than the label format in [9]. The payload format adds an ID field and removes the complement field from the label. Our protocol helps detecting an adversary, to achieve this we send the signature of the payload. And to verify the signature, the verifier needs the ID of that node. The complement field was used to verify the upper bound on the aggregation result by the ***BaseStation***.

We claim that the result can be achieved by sending count only, sending complement was redundant and no longer required.

There is one leaf vertex $s.v$ for each sensor node s with the payload,

$$s.p = \{\ s.id,\ 1,\ s.value,\ Hash(N \mid\mid s.id \mid\mid 1 \mid\mid s.value\ )\} \tag{8.1}$$

where N is the query nonce which is disseminated with each query.

Internal vertices represent aggregation operations, and have payloads that are defined based on their children. Suppose an internal vertex has child vertices $s_1.v, s_2.v, \ldots, s_q.v$ with the following payloads: $s_1.p, s_2.p, \ldots, s_q.p$, where

$$s_i.p = \{\ s_i.id,\ s_i.count,\ s_i.value,\ s_i.commitment\ \} \tag{8.2}$$

Then the internal vertex has payload,

$$s.p = \{\ id,\ count,\ value,\ commitment\ \} \tag{8.3}$$

$$id = s.id \tag{8.4}$$

$$count = \sum_{i=1}^{q} s_i.count \tag{8.5}$$

$$value = \sum_{i=1}^{q} s_i.value \tag{8.6}$$

$$commitment = H(N \mid\mid id \mid\mid count \mid\mid value \mid\mid s_1.p \mid\mid s_2.p \mid\mid \cdots \mid\mid s_q.p) \tag{8.7}$$

Every payload has an associated signature to it, which is the encryption of the hash of the payload using node's private key. For example, signature associated with equation 8.1 is the following:

$$s.sign(p) = E_{s.private-key}(H(s.p)) \tag{8.8}$$

We use elliptic curve cryptography to minimize keys and signatures bit size. Also, we use the collision resistant hash function so it's impossible for an adversary to tamper with any of the commitments once they are created.

There is a mapping between the vertices in the commitment tree and the sensor nodes in the aggregation tree, a vertex is a logical element while a node is a physical device. To avoid confusion, we use the term vertex for the members in the commitment tree and node for the members of the aggregation tree.

The ***AggregationTree*** is a rooted tree created from the network graph. To create an optimal ***AggregationTree*** from the given network graph is outside the scope of this thesis. Our algorithm takes any arbitrary ***AggregationTree*** as an input. One possible ***AggregationTree*** for given network graph in Figure 5.1 is shown in the Figure 5.2. We use the term ***BaseStation*** for the trusted third party. In Figure 5.2 BS is the ***BaseStation***.

Each sensor node s has it's own sensor reading $s.value$. The ***BaseStation*** is interested in overall behavior of the network, which is some function f of all n sensor readings.

$$f(s_1.value, s_2.value, s_3.value, \ldots, s_n.value) \tag{8.9}$$

We discuss the case for the SUM function, but the protocols discussed here can be applied to many other functions with little or no modification.

$$SUM = \sum_{i=1}^{n} s_i.value \tag{8.10}$$

8.1.1 No aggregation approach

One way to calculate the SUM is to send all the n payloads to the ***BaseStation***. It means all the internal nodes in the network send all the payloads received from their descendants to their parent. Once the ***BaseStation*** has all n payloads, it computes the summation. For any given node, the *Inforate* is 1. *Inforate* is defined in Definition 8.1.2.

Advantages:

- Perfectly secure.

Disadvantages:

- Requires $O(n)$ bandwidth between the ***BaseStation***and the ***BaseStation***.

- Requires $O(d)$ bandwidth between the internal node and its parent, where d is the number of descendants for a given node.

- Very high *Inforate.*

Definition 8.1.2 *The Inforate for a particular node is the payloads ratio, number of payloads sent / number of payloads received.*

8.1.2 Naive approach

Another way to calculate the *SUM* is to send only one payload to the ***BaseStation***. It means all the internal nodes in the network, after receiving readings from all of their children, does the summation including their own reading and then pass the resulted payload to their parents' as shown in Figure 5.3. For any given node, the *Inforate* is $1/(c+1)$, where c is the number of children for the give node.
Advantages:

- Optimal *Inforate.*

Disadvantages:

- Makes aggregated value more vulnerable to various security attacks.

- Requires more bandwidth in the verification phase.

8.1.3 Aggregate-commit approach

The no aggregation and naive approaches are two extreme approaches. The aggregate-commit approach of [9] is between these two extreme cases. It combines the advantages of both the approaches.

In the naive approach, each sensor node s sends to its parent a single message containing the payload of the root vertex of its commitment subtree T_s. In the aggregate-commit approach, each sensor node s sends the payloads of the root vertices of a set of commitment subtrees $F = \{T_1, T_2, \ldots, T_q\}$, called commitment forest.

Definition 8.1.3 *[9] 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 with n leaf vertices has the following properties:

- The tallest tree in the forest has height at most $\log(n)$.
- There are at most $\log(n)$ trees in the forest.
- The *Inforate* is $\log(n+1)/n$.

We claim that the binary representation of a number x illustrates the forest decomposition of the sensor node s, where $x = 1 +$ number of descendants of s. For example, if sensor node s has 22 descendants then $x = 23$, $(x)_{10} = (10111)_2$. It means s has four complete binary trees in its outgoing forest, with the height of four, two, one and zero. Note that all the trees in the commitment forest are complete binary trees and no two trees have the same height. In the following section we describe how to build commitment forest with an example and also how to reason about commitment forest using binary addition.

8.1.4 Commitment forest generation

The sensor nodes at the highest depth in the aggregation tree (leaf nodes) initiate a single-vertex commitment forest, which they transmit to their parent sensor node. Each internal sensor node s initiates 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 are received from which of its children. It then aggregates all the forests to form a new forest as follows.

Suppose s wishes to combine q commitment forests $F_1, \ldots, F_q$ and create an aggregated forest F. To do so, the sensor node s merges trees with the same height in its forests by creating a new tree with the height incremented by 1. It repeats this process until no two trees in its forest have the same height. Let T_1, T_2 have the height h, where h is the smallest height in F. The sensor node s merges T_1, T_2 into a tree of height $h+1$ by creating a new vertex according to equation 8.2. It repeats this process until all the trees have unique height in the forest.

In aggregation-commit approach there are at most $\log(n+1)$ commitment trees in each forest transmitted by any sensor node to its parent, which is greater than 1. But all the trees have height less than or equal to $\log(n)$, which makes transmission of off-path values communication efficient which will be discussed in the verification phase.

The sensor node A receives the following payloads:

$$A_0 = \{\ A.id,\ 1,\ A.value,\ H(\ N \ || \ A.id \ || \ 1 \ || \ A.value)\ \}\ (internal) \tag{8.11}$$

$$D_0 = \{\ D.id,\ 1,\ D.value,\ H(\ N \ || \ D.id \ || \ 1 \ || \ D.value)\ \}\ (from\ D) \tag{8.12}$$

$$B_0 = \{\ B.id,\ 1,\ B.value,\ H(\ N \ || \ B.id \ || \ 1 \ || \ B.value)\ \}\ (from\ B) \tag{8.13}$$

$$B_1 = \{\ B.id,\ 2,\ B_1.value,\ H(\ N \ || \ B.id \ || \ 2 \ || \ B_1.value)\ \}\ (from\ B) \tag{8.14}$$

$$C_2 = \{\ C.id,\ 4,\ C_2.value,\ H(\ N \ || \ C.id \ || \ 4 \ || \ C_2.value)\ \}\ (from\ C) \tag{8.15}$$

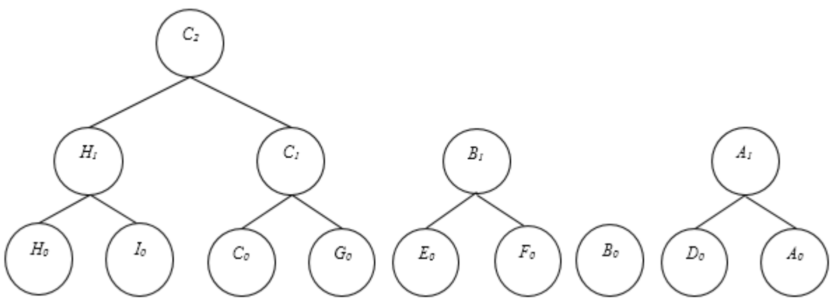

Fig. 8.1.: First Merge: A_1 vertex created by A

$$A_1 = \{\ A.id,\ 2,\ A_1.value,\ H(\ N \ || \ A.id \ || \ 2 \ || \ A1.value\ )\ \} \tag{8.16}$$

$$A_1.value = A.value + D.value \tag{8.17}$$

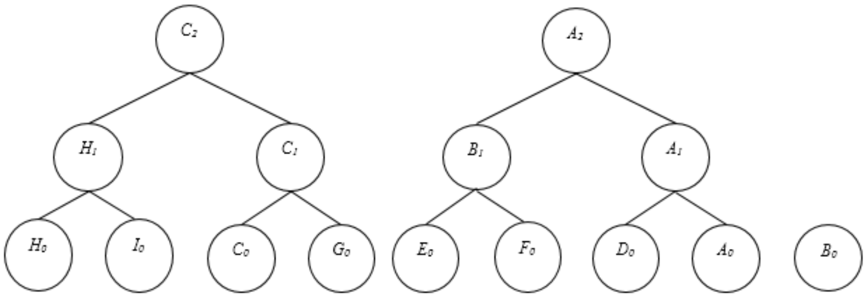

Fig. 8.2.: Second Merge: A_2 vertex created by A

$$A_2 = \{\ A.id,\ 4,\ A_2.value,\ H(\ N\ ||\ A.id\ ||\ 4\ ||\ A_2.value\ )\ \} \tag{8.18}$$

$$A_2.value = A_1.value + B_1.value \tag{8.19}$$

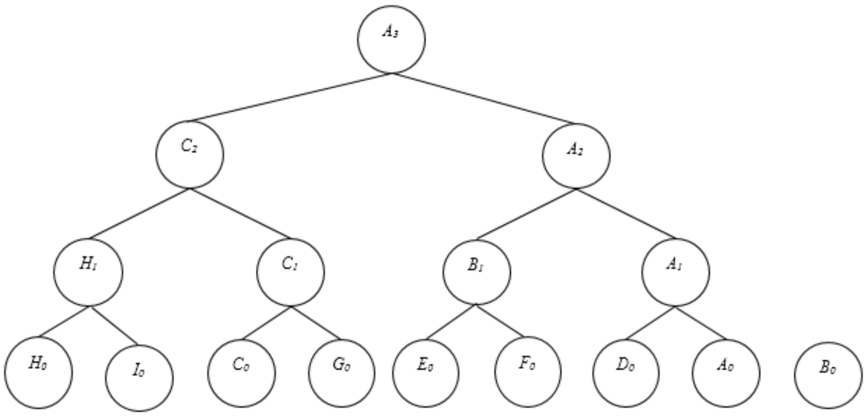

Fig. 8.3.: Third Merge: A_3 vertex created by A

$$A_3 = \{\ A.id,\ 8,\ A_3.value,\ H(\ N\ ||\ A.id\ ||\ 8\ ||\ A_3.value\ )\ \} \tag{8.20}$$

$$A_3.value = A_2.value + C_2.value \tag{8.21}$$

The commitment forest generation process for the sensor node A in Figure 5.2, can be illustrated in the binary format as follows:

Carry	0	1	1	0
B's forest	0	0	0	1
	0	0	1	0
C's forest	0	1	0	0
D's forest	0	0	0	1
A's payload	0	0	0	1
Aggregation	1	0	0	1

The sensor node A receives $B, C, D's$ forests and creates its own single vertex forest. Creating a commitment forest is similar to doing summation on the received forests' binary format. Also, generating a carry in the summation is equivalent to creating a new vertex in the commitment forest by merging two trees. Also, the carry is always generated of the aggregation node. In the analysis chapter, we show that for the aggregation node being root in as many as possible trees is communication efficient.

Observe that A has several ways of doing the summation. For example, while adding lowest significant bits it has $1's$ at three places. The aggregation node A can use any two $1's$ to generate a carry. If A uses $1's$ from $B, D's$ forest than in the final aggregation lowest significant bit represents $A's$ vertex. If vertex related to node is represented by its letter than above summation process can be represented in the following ways.

Carry	0	A	A	0	0	A	A	0	0	A	A	0
B's forest	0	0	0	1	0	0	0	1	0	0	0	1
	0	0	1	0	0	0	1	0	0	0	1	0
C's forest	0	1	0	0	0	1	0	0	0	1	0	0
D's forest	0	0	0	1	0	0	0	1	0	0	0	1
A's payload	0	0	0	1	0	0	0	1	0	0	0	1
Aggregation	A	0	0	A	A	0	0	B	A	0	0	D

Algorithm 2 CommitmentTreeGeneration

d = ***AggregationTree***.MaxDepth
while d $\geq$ 0 **do**
  for all $\mathcal{N} \in$ ***AggregationTree***.depth **do**
    $\mathcal{N}.forest$= NULL
    Create ($\mathcal{N}.msg$, $SIGN_{\mathcal{N}}(\mathcal{N}.msg)$)
    Attach ($\mathcal{N}.msg$, $SIGN_{\mathcal{N}}(\mathcal{N}.msg)$) to $\mathcal{N}.forest$
    if $\mathcal{N}.children \neq 0$ **then**
      for all $\mathcal{C} \in \mathcal{N}.children$ **do**
        for all tree root $\mathcal{R} \in \mathcal{C}.forest$ **do**
          if $\mathcal{N}$ has $\mathcal{R}.cert$ (else get $\mathcal{R}.cert$) **then**
            if $\mathcal{N}$verifies $\mathcal{R}.msg$ (else raise an alarm) **then**
              Add $\mathcal{R}$ into $\mathcal{N}.forest$
        $\mathcal{N}.forest$ = CommitmentTreeCoding ($\mathcal{N}.forest$)
  d = d - 1

Algorithm 3 CommitmentTreeCoding

$temp$ = SortLinkedList($\mathcal{N}$, $\mathcal{N}.forest$) /*returns head of the linked list*/
while $temp.nextTree \neq 0$ **do**
  if $temp.height \neq temp.nextTree.height$ **then**
    $temp = temp.nextTree$
  else
    Create an aggregation node A_N
    $A_N.height = temp.height + 1$
    $A_N.leftChild = temp$
    $A_N.rightChild = temp.nextTree$
    Insert A_N into $\mathcal{N}.forest$
    Remove $temp$
    Remove $temp.nextTree$
    $temp$ = SortLinkedList($\mathcal{N}$, $\mathcal{N}.forest$)
return $temp$

Properties of commitment tree and aggregation tree

If you have $O(n)$ children then you need at least $\Omega(n)$ & at max $O(n \log(n))$ certificates.

If you have $O(n)$ descendants then you need $\Omega(\log(n))$ & at max $O(n \log(n))$ certificates.

8.2 Advantages of this protocol

8.3 Disadvantages of this protocol

9. CHEATING

9.1 Definition

An adversary tampering with the data-items reported by its children to skew the final result is consider as cheating. It can send tampered off-path data-items to its children which are under its control to hide its misbehavior while creating the commitment tree .

9.2 Assumptions

We also assume that the cheater can not send NACK message during verification phase. We assume that an adversary does not tamper, in forwarding the off-path data-items received from its parent. Without those two assumptions, there might be a lot of complainers in the network, creating a lot of traffic in the network. It can cause denial-of-service attack in the network.

Example 9.2.1 *This example shows the significance of the commitment field in the data-item format. Figure 9.1 shows a commitment tree and the vertices have data-items defined as follows.*

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

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

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

- ***No cheating***

  offpath

  A, B receives C_1 from the base station. A receives B_0 from C and vice versa.

  verification

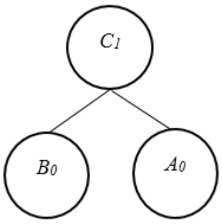

Fig. 9.1.: cheating

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

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

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

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

  offpath

  A, B receives C'_1 *from the base station. A receives* B'_0 *from C and vice versa.*

  verification

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

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

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

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

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

  offpath

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

$A_0 + B_0'' = < 2, 300, H(N||2||300||A_0||B_0'') > \neq C_1' \neq C_1''$ *(by A)*

$A_0'' + B_0 = < 2, 300, H(N||2||300||A_0''||B_0) > \neq C_1' \neq C_1''$ *(by B)*

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

  C tampers A_0*'s value field*

  $A_0' = < A_{id}, 1, 100, H(N||1||10) >$

  $C_1' = < C_{id}, 2, 120, H(N||2||120||A_0||B_0') >$

  C creates $B_0' = < B_{id}, 1, 110, H(N||1||110) >$

  offpath

  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

  $A_0 + B_0' = < 2, 120, H(N||2||120||A_0||B_0') > = C_1'$ *(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 it has to compensate that with relevant off-path value. It means it has to use different data-item in the commitment. So, if an aggregator has two unique children and if it tries to change either one or both childrens' data-item then one of it's children will complain in its verification phase as they will not be able to calculate the same root commitment as they received from the base station.

LIST OF REFERENCES

LIST OF REFERENCES

[1] A. Wang, W. B. Heinzelman, A. Sinha, and A. P. Chandrakasan, "Energy-scalable protocols for battery-operated microsensor networks," *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 29, no. 3, pp. 223–237, 2001.

[2] 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.

[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] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.