SECURE DATA AGGREGATION SCHEME

FOR SENSOR NETWORKS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Kavit Shah

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Electrical and Electronics Engineering

December 2014

Purdue University

Indianapolis, Indiana

This is the dedication.

ACKNOWLEDGMENTS

This is the acknowledgments.

PREFACE

This is the preface.

TABLE OF CONTENTS

Page

LIST OF TABLES

Table Page

LIST OF FIGURES

SYMBOLS

m	mass
v	velocity

ABBREVIATIONS

abbr	abbreviation
bcf	billion cubic feet
BMOC	big man on campus

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 monitors 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 communication between nodes consumes a large portion of the total energy consumption. The in-network data aggregation reduces the energy consumption by eliminating redundant data being transmitted to the base station. 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 data transmission requires more energy than data computation. Hence, in-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 metadata 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 reducing ***information rate*** makes the intermediate (aggregator) sensor nodes more powerful. Also, it makes aggregated ***payload*** more fragile and vulnerable to various security attacks. Now, we describe bandwidth analysis of different in-network aggregation approaches.

4.2 Bandwidth analysis

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

$$Congestion = edgeCongestion * fanout \tag{4.1}$$

Congestion is very useful factor while analyzing sensor network as it measures how quickly the sensor nodes will exhaust their batteries [5]. To transmit a k - bit packet at distance d, the energy dissipated is:

$$E_{tx}(k, d) = E_{elec} * k + \varepsilon_{amp} * k * d^2 \tag{4.2}$$

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

$$E_{rx}(k) = E_{elec} * k \tag{4.3}$$

For μAmp wireless sensor, $E_{elec} = 50nJ/b$ and $\varepsilon_{amp} = 100pJ/b/m^2$. cite papers and add more details on the equations

Some nodes in the sensor network have more congestion than the other. 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 working. 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 achieve the lowest possible ***information rate***, 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 $\Omega(f)$ congestion on the node, where f is the fanout of that node according to Definition 4.1. In this approach, f is dependent on the given aggregation tree, which can be $O(n)$ for the star tree topology and $O(1)$ for the palm tree topology. This can create some highly congested nodes in the aggregation tree which is highly undesirable. In most of the real world applications we cannot control f as the aggregation tree is random. Hence, it is desirable to have almost uniform ***information rate***across the aggregation tree.

Talk about No aggregation approach.

SHIA tries to achieve uniform congestion in the network.

4.3 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 is doing 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 [5, 6] assumes that all the sensor nodes in the network are honest. Secure Information Aggregation (SIA) of [7], provides security for the network topology with a single-aggregator model.

Secure hierarchical in-network aggregation (*SHIA*) in sensor networks [8] 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

We describe the Secure Hierarchical In-network data aggregation *(SHIA)* protocol of [8] as our work enhances this protocol by making it more efficient and adding new capabilities to the protocol. The goal of *SHIA* is to compute aggregate functions (such as *SUM*, *AVERAGE*, *COUNT*) of the sensed values by the sensor nodes while assuming that a portion of the sensor nodes are controlled by an adversary which is attempting to skew the final result.

5.1 Network Assumptions

5.2 Security Infrastructure

5.3 Attacker Model

5.4 Problem Definition

5.5 The SUM Aggregate Algorithm

In this algorithm, the aggregate function f is addition 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
- Aggregate commit
- Result checking

5.5.1 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.

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 $h^m(R)$, then the next broadcast uses $h^{m-1}(R)$ as the nonce; R is the initial random value. 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.

5.5.2 Aggregate commit

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

Then elaborate your approach. Two differences: data-item format CT generation being root in as many trees as possible

6. A PROTOCOL FOR COMMITMENT TREE GENERATION

7. 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

7.1 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 7.1.1, 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 7.1.1 *[8] 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 [8]. 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***,

$$s.p = \{\ s.id,\ 1,\ s.value,\ Hash(N \mid\mid s.id \mid\mid 1 \mid\mid s.value\ )\} \tag{7.1}$$

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

$$s_i.p = \{\ s_i.id,\ s_i.count,\ s_i.value,\ s_i.commitment\ \} \tag{7.2}$$

Then the internal vertex has ***payload***,

$$s.p = \{\ id,\ count,\ value,\ commitment\ \} \tag{7.3}$$

$$id = s.id \tag{7.4}$$

$$count = \sum_{i=1}^{q} s_i.count \tag{7.5}$$

$$value = \sum_{i=1}^{q} s_i.value \tag{7.6}$$

$$commitment = H(N \mid\mid id \mid\mid count \mid\mid value \mid\mid s_1.p \mid\mid s_2.p \mid\mid \cdots \mid\mid s_q.p) \tag{7.7}$$

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 7.1 is the following:

$$s.sign(p) = E_{s.private-key}(H(s.p)) \tag{7.8}$$

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 7.1 is shown in the Figure 7.2. We use the term ***BaseStation*** for the trusted third party. In Figure 7.2 BS is the ***BaseStation***.

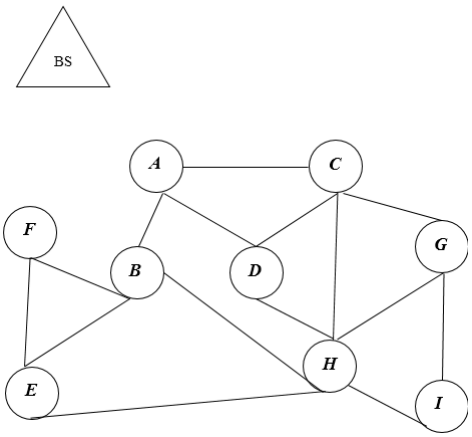

Fig. 7.1.: Network graph

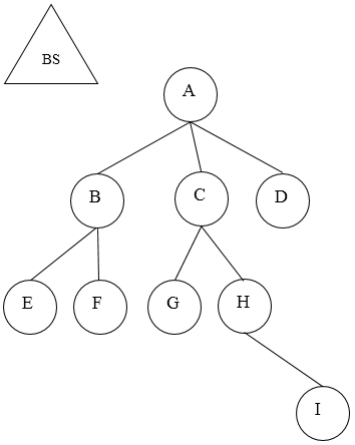

Fig. 7.2.: Aggregation tree for network graph in figure 7.1

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.

$$f(s_1.value, s_2.value, s_3.value, \ldots, s_n.value) \tag{7.9}$$

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.

$$SUM = \sum_{i=1}^{n} s_i.value \tag{7.10}$$

7.1.1 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 7.1.2.

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

7.1.2 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 7.3. 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.

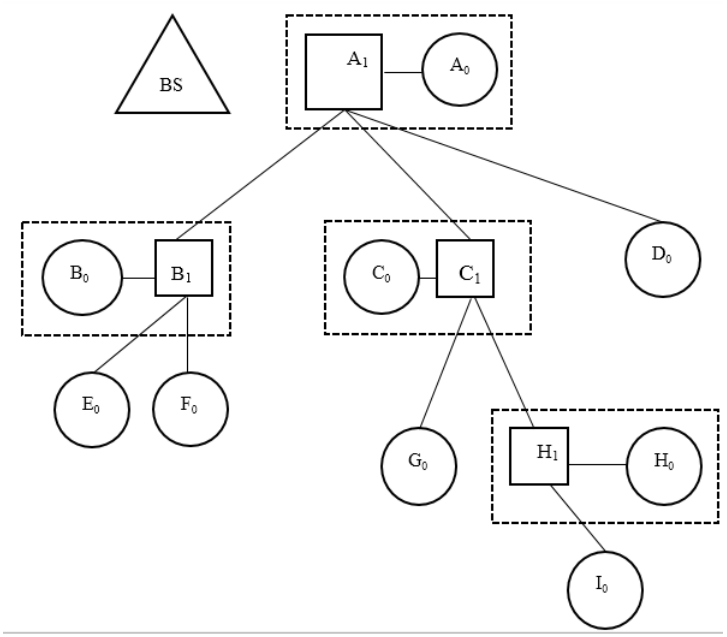

Fig. 7.3.: Naive commitment tree

7.1.3 Aggregate-commit approach

The no aggregation and naive approaches are two extreme approaches. The aggregate-commit approach of [8] 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 7.1.3 *[8] 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.

7.1.4 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 7.2. It repeats this process until all the trees have unique height in the forest. An example of the commitment forest generation process for the sensor node A in Figure 7.2 is illustrated in the Figures 7.4, 7.5, 7.6, 7.7.

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.

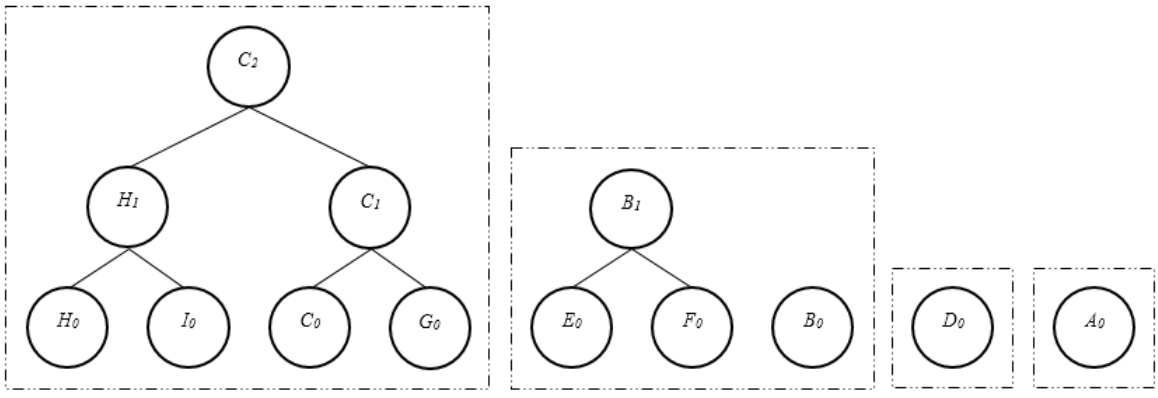

Fig. 7.4.: A receives C_2 from C, (B_1, B_0) from B, D_0 from D and generates A_0. The commitment forest received from a given sensor node is indicated by dashed-line box.

The sensor node A receives the following ***payloads***:

$$A_0 = \{\ A.id,\ 1,\ A.value,\ H(\ N\ ||\ A.id\ ||\ 1\ ||\ A.value)\ \}\ (internal) \tag{7.11}$$

$$D_0 = \{\ D.id,\ 1,\ D.value,\ H(\ N\ ||\ D.id\ ||\ 1\ ||\ D.value)\ \}\ (from\ D) \tag{7.12}$$

$$B_0 = \{\ B.id,\ 1,\ B.value,\ H(\ N\ ||\ B.id\ ||\ 1\ ||\ B.value)\ \}\ (from\ B) \tag{7.13}$$

$$B_1 = \{\ B.id,\ 2,\ B_1.value,\ H(\ N\ ||\ B.id\ ||\ 2\ ||\ B_1.value)\ \}\ (from\ B) \tag{7.14}$$

$$C_2 = \{\ C.id,\ 4,\ C_2.value,\ H(\ N\ ||\ C.id\ ||\ 4\ ||\ C_2.value)\ \}\ (from\ C) \tag{7.15}$$

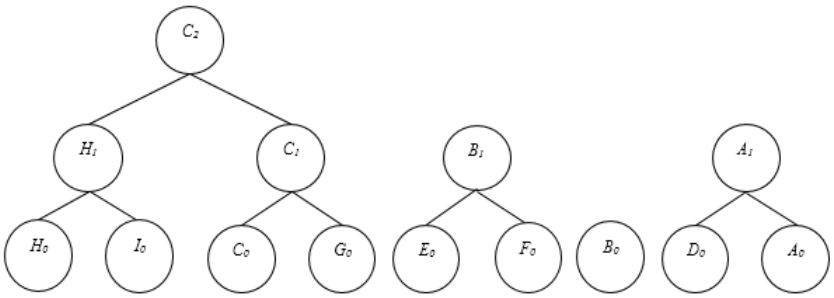

Fig. 7.5.: First Merge: A_1 vertex created by A

$$A_1 = \{\ A.id,\ 2,\ A_1.value,\ H(\ N\ ||\ A.id\ ||\ 2\ ||\ A1.value\ )\ \} \tag{7.16}$$

$$A_1.value = A.value + D.value \tag{7.17}$$

The commitment forest generation process for the sensor node A in Figure 7.2, 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

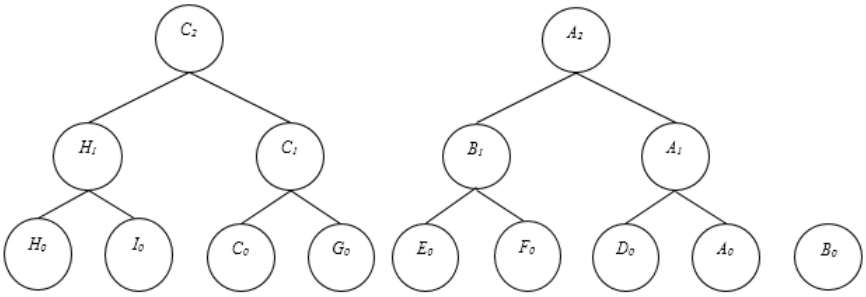

Fig. 7.6.: Second Merge: A_2 vertex created by A

$$A_2 = \{\ A.id,\ 4,\ A_2.value,\ H(\ N\ ||\ A.id\ ||\ 4\ ||\ A_2.value\ )\ \} \tag{7.18}$$

$$A_2.value = A_1.value + B_1.value \tag{7.19}$$

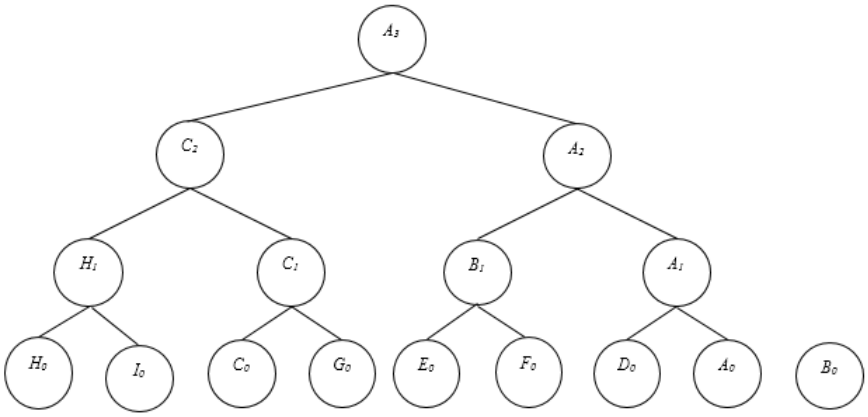

Fig. 7.7.: Third Merge: A_3 vertex created by A

$$A_3 = \{\ A.id,\ 8,\ A_3.value,\ H(\ N\ ||\ A.id\ ||\ 8\ ||\ A_3.value\ )\ \} \tag{7.20}$$

$$A_3.value = A_2.value + C_2.value \tag{7.21}$$

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 1 CommitmentTreeGeneration

1: d = ***AggregationTree***.MaxDepth
2: **while** d $\geq$ 0 **do**
3: **for all** $\mathcal{N} \in$ ***AggregationTree***.depth **do**
4: $\mathcal{N}.forest$= NULL
5: Create ($\mathcal{N}.msg$, $SIGN_{\mathcal{N}}(\mathcal{N}.msg)$)
6: Attach ($\mathcal{N}.msg$, $SIGN_{\mathcal{N}}(\mathcal{N}.msg)$) to $\mathcal{N}.forest$
7: **if** $\mathcal{N}.children \neq$ 0 **then**
8: **for all** $\mathcal{C} \in \mathcal{N}.children$ **do**
9: **for all** tree root $\mathcal{R} \in \mathcal{C}.forest$ **do**
10: **if** $\mathcal{N}$ has $\mathcal{R}.cert$ (else get $\mathcal{R}.cert$) **then**
11: **if** $\mathcal{N}$verifies $\mathcal{R}.msg$ (else raise an alarm) **then**
12: Add $\mathcal{R}$ into $\mathcal{N}.forest$
13: $\mathcal{N}.forest$ = CommitmentTreeCoding ($\mathcal{N}.forest$)
14: d = d - 1

Algorithm 2 CommitmentTreeCoding

1: $temp$ = SortLinkedList($\mathcal{N}$, $\mathcal{N}.forest$) /*returns head of the linked list*/
2: **while** $temp.nextTree \neq 0$ **do**
3: **if** $temp.height \neq temp.nextTree.height$ **then**
4: $temp = temp.nextTree$
5: **else**
6: Create an aggregation node A_N
7: $A_N.height = temp.height + 1$
8: $A_N.leftChild = temp$
9: $A_N.rightChild = temp.nextTree$
10: Insert A_N into $\mathcal{N}.forest$
11: Remove $temp$
12: Remove $temp.nextTree$
13: $temp$ = SortLinkedList($\mathcal{N}$, $\mathcal{N}.forest$)
14: 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.

7.2 Advantages of this protocol

7.3 Disadvantages of this protocol

8. VERIFICATION

8.1 dissemination final commitment

8.2 dissemination of off-path values

8.3 verification of inclusion

8.4 collection of authentication codes

8.5 verification of authentication codes

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

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

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

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

To verify that every sensor node has sent its authentication code with ACK, the base station computes the $MAC_{root}(ACK)$ as follows:

$$MAC_{root}(ACK) = MAC_{K_1}(N \mid\mid ACK) \oplus MAC_{K_2}(N \mid\mid ACK) \oplus \ldots \oplus MAC_{K_n}(N \mid\mid ACK) \tag{8.3}$$

The base station can compute $MAC_{root}(ACK)$ as it knows K_s for each sensor node s. Then it compares the computed $MAC_{root}(ACK)$ with the received root authentication code MAC_{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 according to Equation 8.4.

$$c = MAC_{root} \oplus MAC_{root}(ACK) \tag{8.4}$$

Then it computes the complainer code c_i for node i according to Equation 8.5.
$\forall i \in [1, n]$

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

Then it compares c with all c_i one at a time. The matching code indicates the complainer node. The base station needs to do n comparison to find a complainer in the aggregation tree. Hence, the base station can find a single complainer in linear time. For example, if there are four nodes s_1, s_2, s_3, s_4 in the aggregation tree and their authentication codes with ACK, $NACK$ messages in the binary format are defined below. The base station receives $MAC_{root} = (0100)_2$.

$$MAC_{K_1}(N \mid\mid ACK) = (1001)_2 \; ; \; MAC_{K_1}(N \mid\mid NACK) = (1101)_2 \tag{8.6}$$

$$MAC_{K_2}(N \mid\mid ACK) = (0110)_2 \; ; \; MAC_{K_2}(N \mid\mid NACK) = (1111)_2 \tag{8.7}$$

$$MAC_{K_3}(N \mid\mid ACK) = (0101)_2 \; ; \; MAC_{K_3}(N \mid\mid NACK) = (0111)_2 \tag{8.8}$$

$$MAC_{K_4}(N \mid\mid ACK) = (0011)_2 \; ; \; MAC_{K_4}(N \mid\mid NACK) = (1110)_2 \tag{8.9}$$

Then according to Equation 8.3, $MAC_{root}(ACK) = (1101)_2$
And according to Equation 8.5, $c_1 = (0100)_2$, $c_2 = (1001)_2$, $c_3 = (0010)_2$, $c_4 = (1101)_2$
And according to Equation 8.4, $c = (1101)_2$ So, the base station identifies that the s_4 complained, during verification of inclusion phase.

for more than one complainer it becomes exponential.

How XOR is negating the contribution of NACK.

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

The base station receives the following:

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

The base station does the following:

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

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

And concludes that node 4 is complaining.

8.6 Detect an adversary

Algorithm 3 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

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] 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.

[6] 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.

[7] 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.

[8] 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.