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 vii

LIST OF FIGURES viii

SYMBOLS ix

ABBREVIATIONS x

NOMENCLATURE xi

GLOSSARY xii

ABSTRACT xiii

1 Cheating 1

1.1 Definition 1

1.2 Aim 1

1.3 Assumptions 1

1.4 What is not cheating ? 1

1.5 Probabilistic bound on a cheater 2

1.6 Why do we need digital signatures ? 4

1.7 Why digital signatures are not sufficient to detect a cheater ? or Why do we need public key infrastructure to detect a cheater ? 5

2 August 6

3 november 8

4 protocol 11

4.1 Aggregation-Commit Phase 12

4.1.1 No aggregation approach 15

4.1.2 Naive approach 16

4.1.3 Aggregate-commit approach 17

4.1.4 Commitment forest generation 17

Page

4.2 Advantages of this protocol 24
4.3 Disadvantages of this protocol 24
5 analysis 25
5.1 Background 25
5.2 Star Tree 25
5.3 Maximum savings 26
5.4 Pseudo Palm Tree 28
5.5 Binary tree 29
6 theorems 30
7 network-flow 34
7.1 Star aggregation tree 34
8 Summary 36
9 Recommendations 37
LIST OF REFERENCES 38

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

1.1 Possible commitment tree 2

1.2 Possible commitment tree 2

1.3 Possible commitment tree 3

1.4 Possible commitment tree 4

4.1 Network graph 14

4.2 Aggregation tree for network graph in figure 4.1 14

4.3 Naive commitment tree 16

4.4 A receives C_2 from C, (B_1, B_0) from B, D_0 from D and generates A_0. The commitment forest received from a given sensor node is indicated by dashed-line box. 18

4.5 First Merge: A_1 vertex created by A 19

4.6 Second Merge: A_2 vertex created by A 20

4.7 Third Merge: A_3 vertex created by A 20

5.1 Star aggregation tree 25

5.2 Symmetric Tree 26

5.3 Pseudo palm tree 28

5.4 Binary tree 29

7.1 Star aggregation tree 35

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. CHEATING

1.1 Definition

If an aggregator changes the sensor readings reported by its children to skew the final aggregated result is consider as cheating.

1.2 Aim

Aim of this section is to detect the cheater with given definition.

1.3 Assumptions

We make an assumption that the cheater can not say NACK during verification phase. If a cheater is allowed to send NACK message then it can send NACK messages all the time and create a lot of traffic in the network which might create Denial of service attack.

1.4 What is not cheating ?

In figure 7.1, A is an aggregator if A is a cheater it can skew the final aggregation result irrespective of B's sensor reading. We do not consider this case as a cheating because A is adjusting its sensor reading, it's not changing the B's sensor reading.

For example, if maximum allowed value = 10

case I: $B_0(2) = 5$, $A_0(2) = 13$, $A_1(2) = 18$. In verification, A will be caught due to out of range off path value.

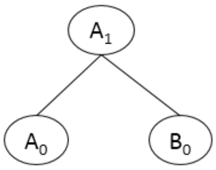

Fig. 1.1.: Possible commitment tree

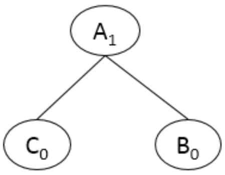

Fig. 1.2.: Possible commitment tree

case II: $B_0(2) = 5$, $A_0(2) = 10$, $A_1(2) = 15$. $B_0'(2) = 6$, $A_0'(2) = 9$. that's not cheating.

Similar arguments can be done for figure 7.2 if A, C both are cheaters. In that case A is adjusting C's sensor reading to skew the final aggregation result and C will not complain as it is a cheater. We do not consider that as cheating either.

1.5 Probabilistic bound on a cheater

To derive Probabilistic bound on a cheater using following example.

In figure 7.3, all vertices in a commitment tree are unique. And, remember cheater can not say NACK during verification phase.

- A_0 says NACK during verification phase it implies that atleast one of the following is {I}, {B, I}, {B, M} is a cheater.
- A_0, B_0 says NACK during verification phase it implies that atleast one of the following is {I}, {M}, {C, D, O} is a cheater.

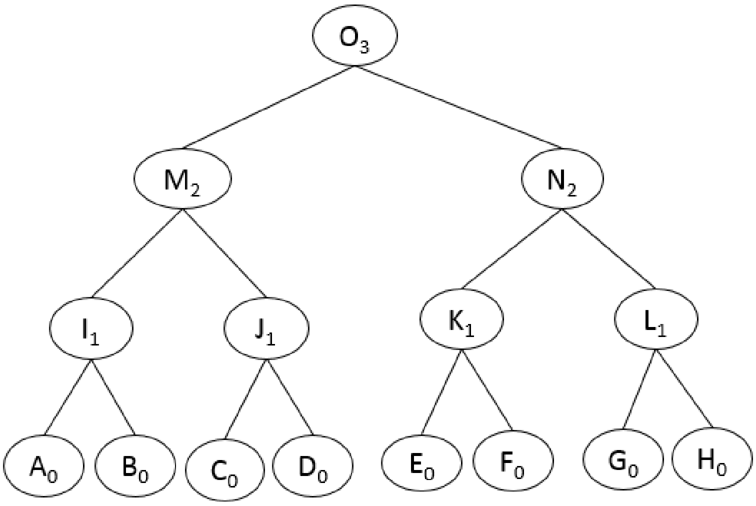

Fig. 1.3.: Possible commitment tree

- A_0, B_0, C_0 says NACK during verification phase it implies that atleast one of the following is {J ,I}, {J, M}, {D, O} is a cheater.

- A_0, B_0, C_0, D_0 says NACK during verification phase it implies that atleast one of the following is {O}, {M}, {I, J}, {E, F, G, H, O} is a cheater.

- A_0, B_0, C_0, D_0, E_0 says NACK during verification phase it implies that atleast one of the following is {O, K}, {M, K}, {I, J, K}, {F, G, H, O} is a cheater.

- $A_0, B_0, C_0, D_0, E_0, F_0$ says NACK during verification phase it implies that atleast one of the following is {I, J, K}, {M, N}, {O, K}, {O, N} is a cheater.

- A_0, C_0 says NACK during verification phase it implies that atleast one of the following is {I}, {J} is a cheater.

Similar, kind of analysis can be done for figure 7.4 in which all the vertices in the commitment tree are different.

From all above examples we can derive the following pattern as well,

If d = depth of a tree,

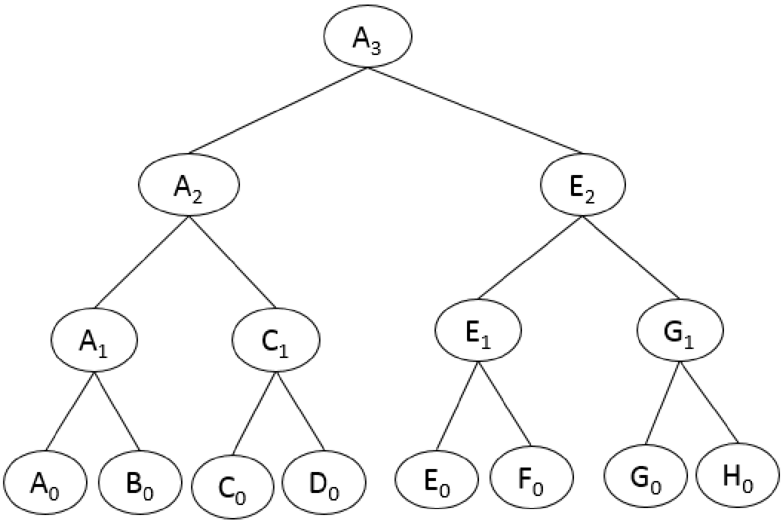

Fig. 1.4.: Possible commitment tree

Depth of a cheater	Minimum number of NACK messages
d - 1	1
d - 2	2
d - 3	4
d - 4	8

1.6 Why do we need digital signatures ?

Digital signatures allow us to achieve authenticity of the message. The labels and signatures have the following format:

id = id

label = $< count, value, commitment >$

signature = $E_{Private_key}(H(N||label))$

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; *id* is the sum of all the leaves id in the subtree; *signature* is a cryptographic scheme for demonstrating the authenticity of a message; N is the query nonce.

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, and signature is the node's unique ID.

Internal vertices represent aggregattion operations, and have labels that are defined based on their children. Write up examples after talking to Dr.King : Do you have to aggregate ID's as well ?

1.7 Why digital signatures are not sufficient to detect a cheater ? or Why do we need public key infrastructure to detect a cheater ?

Digital signatures allow us to achieve authenticity of the message but do not provide any mechanism to achieve integrity of the message. To achieve integrity we need public key infrastructure.

For example, in figure 7.3 one set of possible lables could be the following:

$id_A = 1; A_0 =< 1, 5, H(N||1||5) >; SigA_0 = E_{K_A}(H(N||A_0));$

$id_B = 2; B_0 =< 1, 6, H(N||1||5) >; SigB_0 = E_{K_B}(H(N||B_0));$

$id_I = 3; I_1 =< 2, 11, H(N||2||11||A_0||B_0) >; SigI_1 = E_{K_I}(H(N||I_1));$

$id_J = 4; J_1 =< 2, 15, H(N||2||15||C_0||D_0) >; SigJ_1 = E_{K_J}(H(N||J_1));$

$id_M = 5; M_2 =< 4, 26, H(N||4||26||I_1||J_1) >; SigM_2 = E_{K_M}(H(N||M_2));$

Above labels and signatures are the case where no one is cheating in the network. If A, B say NACK message during the verification phase it means either M or I is a cheater. To preceisely find who is cheater we have following problems:

- M can say it received $(I_1^{'}, SigI_1)$ eventhough it received $(I_1, SigI_1)$ from I.
- M can not verify that it received $(I_1^{'}, SigI_1)$ instead of $(I_1, SigI_1)$ from I.

Because of this we can not not detect cheater between I, M. The fundamental problem is that signatures can be verified only by the base station and not by any of the intermediate nodes. We want the ability in which an intermediate node can verify the signatures from its children. And that is why we need public key infrastructure.

2. AUGUST

Things discussed in meeting:

Analyzed congestion and why is it sub linear ?

In SHIA leaves verify their values with final results not with intermediate results. But in surveillance application data is compared with some base value in such network intermediate values are important.

Analyze the protocol with Digital signatures. How many signatures do we need ?

Analyze properties of commitment tree.

Definitions

A **direct data injection attack** occurs when an attacker modifies the data readings reported by the nodes under its direct control, under the constraint that only legal readings in [0, r] are reported.

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

For example, if A is an aggregator and it receives one reading from B. So, A needs to aggregate two values one of its own and the other is B's value. Suppose, maximum allowed value is 40. A0 = 10, B0 = 20. A1 = 30. A1 ¡= 80. If A reports any value out of that range it will get caught and any cheating within the range falls under direct data injection attack.

Congestion

As a metric for communication overhead, we consider node congestion, which is the worst case communication load on any single sensor node during the algorithm. Congestion is a commonly used metric in ad-hoc networks since it measures how quickly the heaviest-loaded nodes will exhaust their batteries [6, 12]. Since the heaviest-loaded nodes are typically the nodes which are most essential to the connectivity of the network (e.g., the nodes closest to the base station), their failure may cause the network to partition even though other sensor nodes in the network may still have high battery levels. A lower communication load on the heaviest-loaded nodes is thus desirable even if the trade-off is a larger amount of communication in the network as a whole.

For a lower bound on congestion, consider an unsecured aggregation protocol where each node sends just a single message to its parent in the aggregation tree. This is the minimum number of messages that ensures that each sensor node contributes to the aggregation result. There is $\Omega(1)$ congestion on each edge on the aggregation tree, thus resulting in $\Omega(d)$ congestion on the node(s) with highest degree d in the aggregation tree. The parameter d is dependent on the shape of the given aggregation tree and can be as large as $\Theta(n)$ for a single-aggregator topology or as small as $\Theta(1)$ for a balanced aggregation tree. Since we are taking the aggregation tree topology as an input, we have no control over d. Hence, it is often more informative to consider per-edge congestion, which can be independent of the structure of the aggregation tree.

Consider the simplest solution where we omit aggregation altogether and simply send all data values (encrypted and authenticated) directly to the base station, which then forwards it to the querier. This provides perfect data integrity, but induces O(n) congestion at the nodes and edges nearest the base station. For an algorithm to be practical, it must cause only sublinear edge congestion.

Our goal is to design an optimally secure aggregation algorithm with only sublinear edge congestion.

1. remove complement 2. variable range

3. NOVEMBER

Misc. topics to write about:

Why do you want to communicate an entire aggregation tree to the querier ?
If the querier knows the entire aggregation tree and also if it knows the protocol which all the sensor nodes will be running then the querier can simulate the commitment trees on its own. Because of that we do not have to communicate the commitment tree every time we run the protocol which saves a lot of communications in the network. Also, note the fact that aggregation tree does not change often so the communication required to send the aggregation tree is negligible over time.

How to communicate an entire aggregation tree to the querier ?
The base station in the aggregation tree needs to know the entire network topology. It will relay that information to the querier.

How does the base station know the entire aggregation tree topology ?
If every sensor nodes has a small table containing the path to reach to the certain destination then the base station can ask for this information to the individual sensor nodes. While it is receiving this information it can relay the same information to the querier. Note: the base station is also a simple sensor node like all other nodes it can not store all the forwarding tables so it will relay those table information directly to the querier and querier can make big table containing the information related to the aggregation tree.

Mobility
You can talk about the aggregation tree topology is mobile. It's increasingly mobile topology not leap mobility.

Caching of certificates
Certificates are sent only once for the first time. They are cached for subsequent

communications. Every node in the tree needs to know the certificates of all the root nodes in its forest.

Why does the internal vertex in the commitment tree need to send what it received and what it sent to its parent ?

To detect a cheater, if an internal vertex send (to the querier) only the values which it sent to its parent in the commitment tree then it is no value to the querier. Because the querier can not verify that value and the signature. For the querier to verify the aggregated data and its signature it needs both the values over which aggregation has happend.

Why don't you need backward signatures ?

Because according to the protocol, every parent checks its children's message and its signature. If those two do not match then it will not accept the message.

Do you need signature on forest ?

No, we do not need the signature on forest.

Analyses of being root in as many tree as possible:

- *Bandwidth perspective*

  Off path values

  It takes same bandwidth (same hop counts) to distribute off path values in include itself or exclude itself stratergy. You can have inductive argument for it to prove it.

  Certificates Parent node needs to deal with less nodes in the aggregation tree means it needs less certificates, means less memory storage. For example, in pseudo palm tree case if we use include it self streategy then it is possible that one node has to propagate its value from the bottom to the top of the tree. It means all the intermediate nodes need to know its certificate. This can be avoided by using exclude itself(being root in as many possible tree as possible) stratergy.

- *Security perspective*

Exclude itself stratergy is more secure in the sense that aggregator needs to partner with two nodes to achieve cheating. If it includes itself then it has to partner with only one node which is relatively easy.

Why do we need authenticated broadcast from the querier ?

Significance of Nonce

Why do we need public key infrastructure ?

Why don't we use aggregation tree as commitment tree ?

Why is commitment tree binary and not n-ary ? (proof)

How to detect following cheating ?

The querier knows an aggregation tree and a protocol. So the querier can simulate commitment tree. All the nodes in the network are supposed to run the same protocol. Suppose if they don't then the commitment tree will look different. How will you detect such cheating ?

4. 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)

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

CERTIFICATES

Public key	Signature	ID
1000 bits	500 bits	20 bits

4.1 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 4.1.1, 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 4.1.1 *[3] 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 [3]. 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***,

$$s.p = \{\ s.id,\ 1,\ s.value,\ Hash(N \mid\mid s.id \mid\mid 1 \mid\mid s.value\ )\} \tag{4.1}$$

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

$$s_i.p = \{\ s_i.id,\ s_i.count,\ s_i.value,\ s_i.commitment\ \} \tag{4.2}$$

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

$$s.p = \{\ id,\ count,\ value,\ commitment\ \} \tag{4.3}$$

$$id = s.id \tag{4.4}$$

$$count = \sum_{i=1}^{q} s_i.count \tag{4.5}$$

$$value = \sum_{i=1}^{q} s_i.value \tag{4.6}$$

$$commitment = H(N \mid\mid id \mid\mid count \mid\mid value \mid\mid s_1.p \mid\mid s_2.p \mid\mid \cdots \mid\mid s_q.p) \tag{4.7}$$

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 4.1 is the following:

$$s.sign(p) = E_{s.private-key}(H(s.p)) \tag{4.8}$$

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 4.1 is shown in the Figure 4.2. We use the term ***BaseStation*** for the trusted third party. In Figure 4.2 BS is the ***BaseStation***.

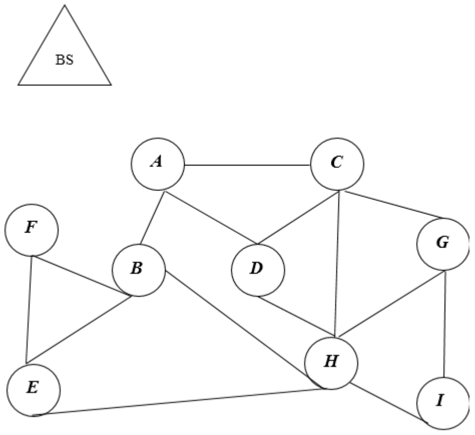

Fig. 4.1.: Network graph

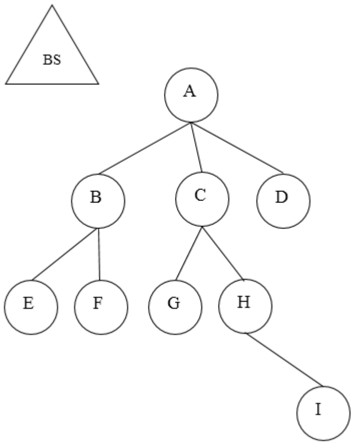

Fig. 4.2.: Aggregation tree for network graph in figure 4.1

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.

$$f(s_1.value, s_2.value, s_3.value, \ldots, s_n.value) \tag{4.9}$$

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

$$SUM = \sum_{i=1}^{n} s_i.value \tag{4.10}$$

4.1.1 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 ***information rate*** is 1. ***information rate*** is defined in Definition 4.1.2.

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 ***information rate***.

Definition 4.1.2 *The **information rate** for a particular node is the **payloads** ratio, number of **payloads** sent / number of **payloads** received.*

4.1.2 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 4.3. For any given node, the ***information rate*** is $1/(c+1)$, where c is the number of children for the give node.

Advantages:

- Optimal ***information rate***.

Disadvantages:

- Makes aggregated value more vulnerable to various security attacks.
- Requires more bandwidth in the verification phase.

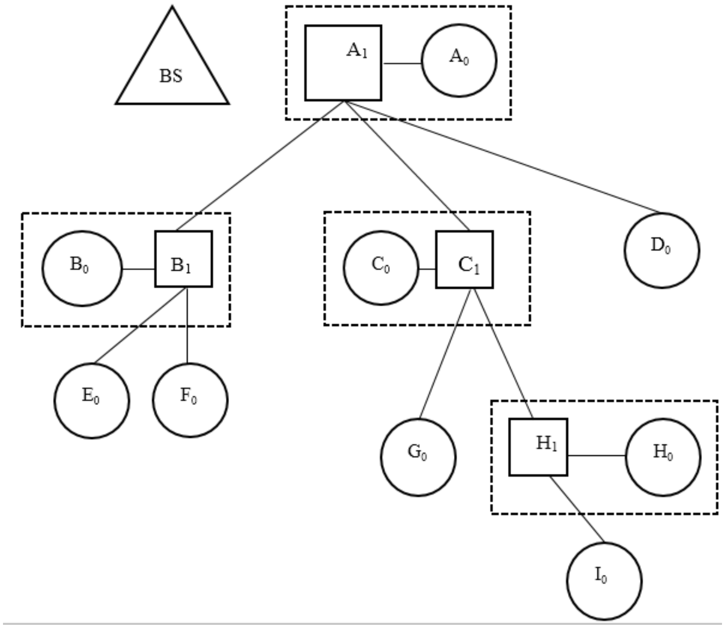

Fig. 4.3.: Naive commitment tree

4.1.3 Aggregate-commit approach

The no aggregation and naive approaches are two extreme approaches. The aggregate-commit approach of [3] 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 4.1.3 *[3] 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 ***information rate*** 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.

4.1.4 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 4.2. 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 4.2 is illustrated in the Figures 4.4, 4.5, 4.6, 4.7.

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.

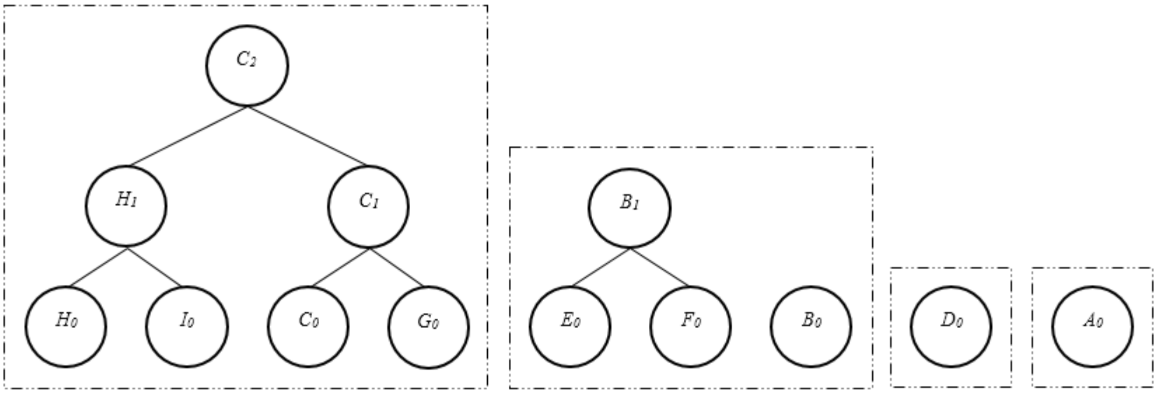

Fig. 4.4.: A receives C_2 from C, (B_1, B_0) from B, D_0 from D and generates A_0. The commitment forest received from a given sensor node is indicated by dashed-line box.

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

$$A_0 = \{\ A.id,\ 1,\ A.value,\ H(\ N \mid\mid A.id \mid\mid 1 \mid\mid A.value)\ \}\ (internal) \tag{4.11}$$

$$D_0 = \{\ D.id,\ 1,\ D.value,\ H(\ N \mid\mid D.id \mid\mid 1 \mid\mid D.value)\ \}\ (from\ D) \tag{4.12}$$

$$B_0 = \{\ B.id,\ 1,\ B.value,\ H(\ N \mid\mid B.id \mid\mid 1 \mid\mid B.value)\ \}\ (from\ B) \tag{4.13}$$

$$B_1 = \{\ B.id,\ 2,\ B_1.value,\ H(\ N \mid\mid B.id \mid\mid 2 \mid\mid B_1.value)\ \}\ (from\ B) \tag{4.14}$$

$$C_2 = \{\ C.id,\ 4,\ C_2.value,\ H(\ N \mid\mid C.id \mid\mid 4 \mid\mid C_2.value)\ \}\ (from\ C) \tag{4.15}$$

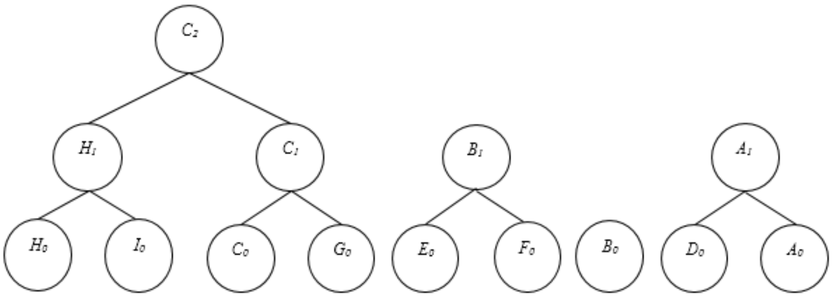

Fig. 4.5.: First Merge: A_1 vertex created by A

$$A_1 = \{\ A.id,\ 2,\ A_1.value,\ H(\ N \mid\mid A.id \mid\mid 2 \mid\mid A1.value\ )\ \} \tag{4.16}$$

$$A_1.value = A.value + D.value \tag{4.17}$$

The commitment forest generation process for the sensor node A in Figure 4.2, 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

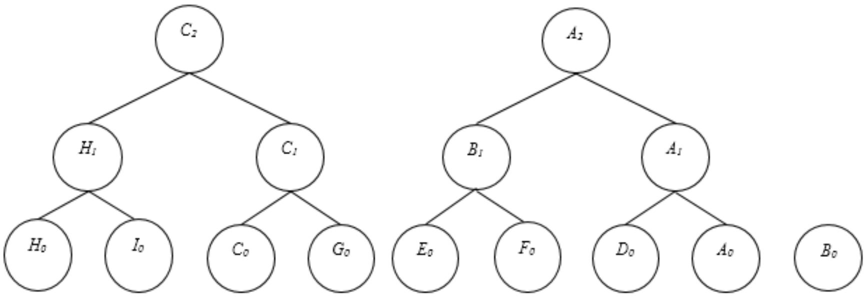

Fig. 4.6.: Second Merge: A_2 vertex created by A

$$A_2 = \{ \ A.id, \ 4, \ A_2.value, \ H(\ N \ || \ A.id \ || \ 4 \ || \ A_2.value \ ) \ \} \tag{4.18}$$

$$A_2.value = A_1.value + B_1.value \tag{4.19}$$

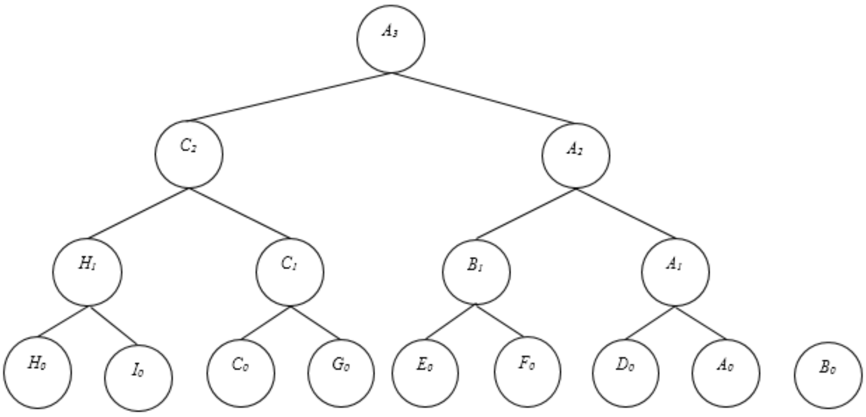

Fig. 4.7.: Third Merge: A_3 vertex created by A

$$A_3 = \{ \ A.id, \ 8, \ A_3.value, \ H(\ N \ || \ A.id \ || \ 8 \ || \ A_3.value \ ) \ \} \tag{4.20}$$

$$A_3.value = A_2.value + C_2.value \tag{4.21}$$

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

```
depth = $\boldsymbol{AggregationTree}$.MaxDepth
**while** depth $\geq 0$ **do**
    **for all** $\mathcal{N} \in$ $\boldsymbol{AggregationTree}$.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 $\mathcal{R}$ to $\mathcal{N}.forest$
                $\mathcal{N}.forest$ = CommitmentTreeCoding ( $\mathcal{N}.forest$ )
    depth = depth - 1
```

Algorithm 2 CommitmentTreeCoding

```
$temp$ = SortLinkedList( $\mathcal{N}.forest$ )
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 $A_N$ to $\mathcal{N}.forest$
        Remove $temp$
        Remove $temp.nextTree$
        $temp$ = SortLinkedList( $\mathcal{N}.forest$ )
return $temp$
```

Algorithm 3 Pseudo algorithm to detect a cheater

```
$\boldsymbol{BaseStation}$ finds out all the $\mathcal{C}_{\mathcal{N}} \in \boldsymbol{AggregationTree}$ using a complainer detecting algorithm
for all $\mathcal{C}_{\mathcal{N}}$ do
    $\mathcal{Q}$ gets $\mathcal{N}_0$, $SIGN_{\mathcal{N}}$ ( $\mathcal{N}_0$ )
$\mathcal{Q}$ finds possible $CHEATER$ based on $\mathcal{C}_{\mathcal{N}}$
for all $CHEATER$ do
    $\mathcal{Q}$ gets $\mathcal{N}_{\mathcal{I}}$, $SIGN_{\mathcal{N}}$ ( $\mathcal{N}_I$ ) $CHEATER$ receives and sends.
    If needed $\mathcal{Q}$ gets $\mathcal{N}_{\mathcal{I}}$, $SIGN_{\mathcal{N}}$ ( $\mathcal{N}_I$ ) of the $\mathcal{P}$ $CHEATER$
$\mathcal{Q}$ determines the $CHEATER$ based on recived information
```

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.

4.2 Advantages of this protocol

4.3 Disadvantages of this protocol

5. ANALYSIS

5.1 Background

Aggregation tree Commitment tree Analogy with binary representation

5.2 Star Tree

To do : Material on star tree, star tree analysis gives you 1 cert savings.

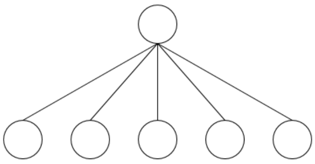

Fig. 5.1.: Star aggregation tree

5.3 Maximum savings

Analysis is true for any n bit forest size. Give names to the following tolopogies. **Maximum savings**, with n(=4) bit forest, fanout(=2), savings of n(=4) certificates:

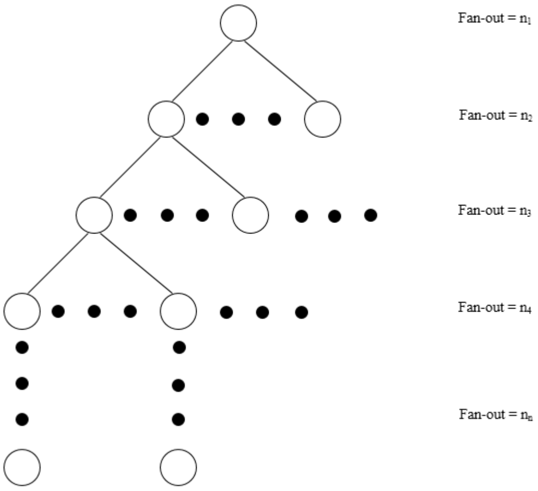

Fig. 5.2.: Symmetric Tree

$$\begin{array}{c|cccc} 1 & 1 & 1 & 1 & 0 \\ \hline 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ \hline \mathrm{A} & \mathrm{C} & \mathrm{C} & \mathrm{C} & \mathrm{C} \end{array} \quad — \quad \begin{array}{c|cccc} 1 & 1 & 1 & 1 & 0 \\ \hline 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ \hline \mathrm{A} & \mathrm{A} & \mathrm{A} & \mathrm{A} & \mathrm{A} \end{array}$$

No savings, with n(=4) bit forest with alternate bit positions, fanout(=2, 3, 5) :

$$\begin{array}{c|cccc} 1 & 0 & 1 & 0 & 0 \\ \hline 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ \hline \mathrm{A} & 0 & \mathrm{A} & 0 & \mathrm{A} \end{array} \quad — \quad \begin{array}{c|cccc} 1 & 0 & 1 & 0 & 0 \\ \hline 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ \hline \mathrm{A} & \mathrm{C} & \mathrm{A} & \mathrm{C} & \mathrm{A} \end{array} \quad — \quad \begin{array}{cc|cccc} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ \hline \mathrm{A} & \mathrm{A} & 0 & 0 & \mathrm{C} & \mathrm{A} \end{array}$$

Savings of n - 1 certificates, with n(=4) bit forest, fanout(=3) :

$$\begin{array}{cc|cccc} 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ \hline 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ \hline \mathrm{A} & 0 & \mathrm{C} & \mathrm{C} & \mathrm{C} & 0 \end{array} \text{ — } \begin{array}{cc|cccc} 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ \hline 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ \hline \mathrm{A} & 0 & \mathrm{A} & \mathrm{A} & \mathrm{A} & 0 \end{array}$$

Savings of n - 1 certificates, with n(=4) bit forest, fanout(=4) :

$$\begin{array}{cc|cccc} 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ \hline 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ \hline \mathrm{A} & \mathrm{A} & \mathrm{C} & \mathrm{C} & 0 & \mathrm{C} \end{array} \text{ — } \begin{array}{cc|cccc} 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ \hline 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ \hline \mathrm{A} & \mathrm{A} & \mathrm{A} & \mathrm{A} & 0 & \mathrm{A} \end{array}$$

5.4 Pseudo Palm Tree

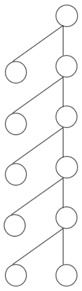

Fig. 5.3.: Pseudo palm tree

Theorem 5.4.1 *At any given level, if an aggregator priorities aggregating its childrens' messages over its own message, it can save bandwidth by not sending its childrens' certificates to its parent.*

Proof We can see from Figure 5.3 that every aggregator has odd number of messages to aggregate, including itself. It means an aggregator at each level has odd number of $1's$ in therir least significant bits. If an aggregator aggregates messages of its children and creates a carry then it needs to send its own certificate to its parent or else it has to send one of its children's certificate as well. Following example illustrates the idea, where C means an aggregator has to send its child's certificate to its parent, A means an aggregator has to send its own certificate to its parent and X is don't care.

0	0	0	1	0		0	0	0	1	0
0	0	0	0	1		0	0	0	0	1
0	0	0	0	1	—	0	0	0	0	1
X	X	X	X	1		X	X	X	X	1
X	X	X	X	C		X	X	X	X	A

Hence, this approach saves bandwidth by sending one less certificate at each level.

■

5.5 Binary tree

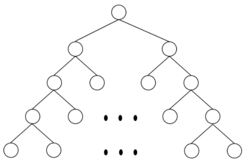

Fig. 5.4.: Binary tree

Theorem 5.5.1 *At any given level, if an aggregator priorities aggregating its childrens' messages over its own message, it can save bandwidth by not sending* $\lceil lg(n/2) \rceil$ *certificates,* n *is number of descendants for given node, to its parent.*

6. THEOREMS

Tree properties & notations

Every node is identical, following the same procedure

Add picture of the topology

d_i is the depth at i

n_i is the fanout at d_{i-1}

c_i is the number of certificates forwarded by each node at d_i to its parent

$= \lceil log((n_{i+1} * c_{i+1}) + 1) \rceil$

N_i is the number of nodes at d_i

$= \Pi_{k=1}^{i} n_k$

T_i is the totality at d_i where totality is the number of certificates received/needed

$= N_i * (n_{i+1} * c_{i+1})$

Total number of nodes in a tree

$= N + 1$

$= N_0 + N_1 + N_2 + +N_3 + ... + N_{n-2} + N_{n-1} + N_n$

$= n_0! + n_1! + n_2! + n_3! + ... + n_{n-2}! + n_{n-1}! + n_n!$

$= 1 + n_1! + n_2! + n_3! + ... + n_{n-2}! + n_{n-1}! + n_n!$

Theorem 6.0.2 *Given an aggregation tree with N + 1 nodes, having $N_i = n_i!$ nodes at depth d_i, equally distributed among thier $n_{i-1}!$ parents then in totality network needs $O(N * log(C))$ certificates where C is a constant.*

Proof Case I: For d_n ; $N_n = n_n!$; $T_n = 0$; $c_n = 1$

Case II: For d_{n-1}

$N_{n-1} = n_{n-1}!$

$T_{n-1} = n_{n-1}! * (n_n * c_n) = n_{n-1}! * (n_n)$

$c_{n-1} = 2$

Case III: For d_{n-2}

$N_{n-2} = n_{n-2}!$

$T_{n-2} = n_{n-2}! * (n_{n-1} * c_{n-1}) = n_{n-2}! * (n_{n-1} * 2)$

$c_{n-2} = \lceil log((n_{n-1} * c_{n-1}) + 1) \rceil = \lceil log((n_{n-1} * 2) + 1) \rceil$

Case IV: For d_{n-3}

$N_{n-3} = n_{n-3}!$

$T_{n-3} = n_{n-3}! * (n_{n-2} * c_{n-2}) = n_{n-3}! * (n_{n-2} * \lceil log((n_{n-1} * 2) + 1) \rceil)$

$c_{n-3} = \lceil log((n_{n-2} * c_{n-2}) + 1) \rceil = \lceil log(n_{n-2} * \lceil log((n_{n-1} * 2) + 1) \rceil + 1) \rceil$

■

Theorem 6.0.3 *Given an aggregation tree with N + 1 nodes, having $N_i = n_i!$ nodes at depth d_i, equally distributed among thier $n_{i-1}!$ parents then in totality network needs $\Omega(N)$ certificates.*

Proof Case I: For d_n ; $N_n = n_n!$; $T_n = 0$; $c_n = 1$

Case II: For d_{n-1}

$N_{n-1} = n_{n-1}!$

$T_{n-1} = n_{n-1}! * (n_n * c_n) = n_{n-1}! * n_n$

$c_{n-1} = 1$

Case III: For d_{n-2}

$N_{n-2} = n_{n-2}!$

$T_{n-2} = n_{n-2}! * (n_{n-1} * c_{n-1}) = n_{n-2}! * n_{n-1}$

$c_{n-2} = 1$

Case IV: For d_{n-3}

$N_{n-3} = n_{n-3}!$

$T_{n-3} = n_{n-3}! * (n_{n-2} * c_{n-2}) = n_{n-3}! * n_{n-2}$

$c_{n-3} = 1$

Case n-2: For d_2

$N_2 = n_2!$

$T_2 = n_2! * (n_3 * c_3) = n_2! * n_3$

$c_2 = 1$

Case n-1: For d_1

$N_1 = n_1!$

$T_1 = n_1! * (n_2 * c_2) = n_1! * n_2$

$c_{n-1} = 1$

■

Things to include in above analysis:
Every node does the same thing
Individual throughput for each node

Theorem 6.0.4 *Given an aggregation tree with N nodes, in totality network needs $\Omega(N)$ certificates.*

Proof Let T represent a node in an aggregation tree whose number of children are $T.CHILDREN$, $C$is one of its children and P is its parent.
We can say that T needs certificates of all of its children because while creating a commitment tree T receives at least one $C.msg$ and $SIGN_C$ $(C.msg)$ from all $T.CHILDREN$

If your aggregation tree is such that $\forall T$ needs to send only one message to P then every P receives only $P.CHILDREN$ number of messages. Hence, P needs $P.CHILDREN$ number of certificates.

So, if every P needs certificates only of its children and we have N nodes in the network then since every node has a unique parent as aggregation tree is a rooted tree, in totality we need only N certificates in the network.

■

Theorem 6.0.5 *Given an aggregation tree with N + 1 nodes, having $N_i = n_i!$ nodes at depth d_i, equally distributed among thier $n_{i-1}!$ parents then in totality network needs $\Omega(N)$ certificates.*

Proof Let say we have N + 1 nodes in an aggregation tree, also d_i represents depth at level i. Tree is constructed such that root has n_1 children, all n_1 nodes at d_1 have n_2 children, all $(n_1 * n_2)$ nodes at d_2 have n_3 children and all $(n_1 * n_2 * n_3 ... n_{n-1})$ nodes at d_{n-1} have n_n children.
$N + 1 = (1 + (n_1) + (n_2 * (n_1)) + (n_3 * (n_2 * n_1)) + ... + (n_n * (n_{n-1} * n_{n-2} * n_{n-3} ... n_1)))$

All $(n_{n-1} * n_{n-2} * n_{n-3} ... n_1)$ nodes at d_{n-1} need to know certificates of all of their n_n children. If there is only one carry after aggregation then all nodes at d_{n-1} need to send only one certificate to their parent.
All $(n_{n-2} * n_{n-3} * n_{n-4} ... n_1)$ nodes at d_{n-2} need to know certificates of all of their $n - 1$ children.
All n_1 nodes at d_1 need to know certificates of all of their $n - 2$ children.
The root need to know the certificates of all of its n_1 children.
Also, the querier needs to know the certificate of the root.
So, in totality we need $(1 + (n_1) + (n_2 * (n_1)) + (n_3 * (n_2 * n_1)) + ... + (n_n * (n_{n-1} * n_{n-2} * n_{n-3} ... n_1)))$ which is $\Omega(N)$. Hence, prooved.

■

7. NETWORK-FLOW

MESSAGE

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

SIGNATURE (MESSAGE)

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

CERTIFICATES

Public key	Signature	ID
1000 bits	500 bits	20 bits

NETWORK FLOW

For the given aggregation tree, when a child communicates for the first time with its parent, it sends three things: Message, Signature of message & Certificate. For any subsequent communications a child sends Message & Signature of messsage.

7.1 Star aggregation tree

In star aggregation topology root has to create (n - 1) intermediate vertices, where n is the number of children root has. *Note*: number of certificates = signatures = messages = public keys

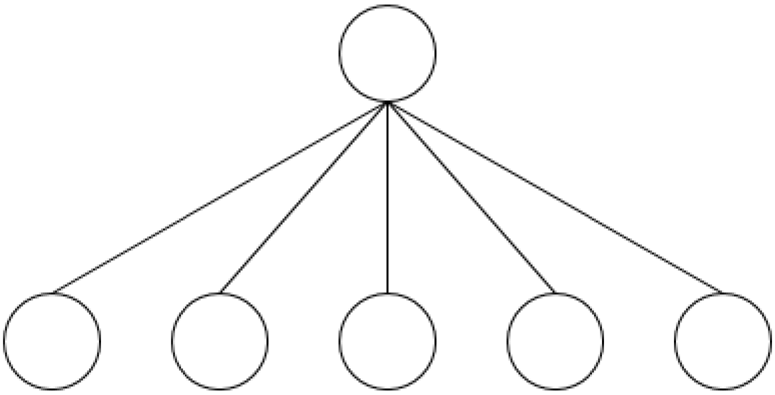

Fig. 7.1.: Star aggregation tree

	#Messages	#Certificates
To root	O (n)	O (n)
From root	1	1

The following property might be true for all possible topologies:

If you have N children, each of your children has n descendent then following equality holds true:

$N < numberofcertificatesneeded < Nlog(n)$

8. SUMMARY

This is the summary chapter.

9. RECOMMENDATIONS

Buy low. Sell high.

LIST OF REFERENCES

LIST OF REFERENCES

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

[2] 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.

[3] 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.