Congestion Lower Bounds for Secure In-network Aggregation

Raghav Bhaskar
Microsoft Research India
Bangalore, India
rbhaskar@microsoft.com

Ragesh Jaiswal
Indian Institute of Technology
New Delhi, India
rjaiswal@cse.iitd.ac.in

Sidharth Telang
Cornell University
Ithaca, USA
sidtelang@cs.cornell.edu

ABSTRACT

In-network aggregation is a technique employed in Wireless Sensor Networks (WSNs) to aggregate information flowing from the sensor nodes towards the base station. It helps in reducing the communication overhead on the nodes in the network and thereby increasing the longevity of the network. We study the problem of maintaing integrity of the aggregate value, when the aggregate function is SUM, in the presence of compromised sensor nodes. We focus on one-round, end-to end, secure aggregation protocols and give a strong, formal security defintion. We show that a worst-case lower bound of $\Omega(n)$ applies on the congestion (maximum size of message between any two nodes) in such protocols, where n is the number of nodes in the network. This is the first such result showing that the most basic protocols are the best one-round in-network aggregation protocols with respect to congestion. We also show that against a weaker adversary (which does not compromise nodes), we can achieve secure in-network aggregation protocols with a congestion of $O(\log_2 n)$.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*

General Terms

Security

Keywords

in-network aggregation, security, wireless security

1. INTRODUCTION

In-network aggregation refers to on the fly computations performed on data by the nodes in a network as the data is being sent towards a fixed node called the base station. It is a popular technique, often employed in wireless sensor networks (WSNs), to prolong the longevity of the network. For example, consider a sensor network where sensors sense the temperature at their location and send it towards the base station. The base station is often interested in knowing the average of the observed readings rather than the individual readings. Thus, while these readings are being transmitted towards the base station, sensor nodes add their reading to the received sum of the readings before forwarding it. Thus, in a single sweep of the network, the base station gets the sum of all the readings and can calculate the average from it. Some of the common aggregate functions are sum, average, histogram, max and min. Not only is the total number of messages that need to be sent are reduced using in-network aggregation, but also the the maximum number of bits communicated over any link (congestion) are reduced. To get an idea of the communication saving made by employing in-network aggregation, consider a network of n nodes in a balanced binary tree topology with the base station at the root and sensors occupying the other nodes. Also, assume the aggregation function is SUM and that each node contributes an integer reading between 0 and $(R-1)$. Then, Table 1 shows the communication costs in the case of no aggregation (when all the messages are sent) and in-network aggregation (when the messages are summed up before sending). Thus, in-network aggregation can reduce both the number of messages sent and congestion significantly for a binary tree topology.

	No aggregation	With aggregation
No. of messages	$O(n \log_2 n)$	n
Congestion	$O(n \log_2 R)$	$O(\log_2 nR)$

Table 1: Communication with and without aggregation for a SUM query in a balanced binary tree topology.n denotes the number of nodes in the network, $[0, R-1]$ - message range

As WSNs are deployed in hostile environments, they are prone to attacks. Attacks can be both from external sources (like other wireless devices, jamming stations) or sensor nodes themselves (which have been compromised by an adversary). While standard cryptographic authentication primitives (like Message Authentication Codes and Digital Signatures [11]) can be used to prevent several kinds of attacks from external sources on the in-network aggregation protocol, more innovative techniques are required to counter attacks from compromised nodes. A compromised node (using its knowledge of keying material of the aggregation protocol) can launch various kinds of attacks without getting detected. For in-

WiSecŠ12, April 16–18, 2012, Tucson, Arizona, USA.

stance, it could drop some received readings, add a huge reading in the current aggregate, not relay any aggregate at all, thereby resulting in incorrect aggregate results. While denial of service attacks (dropping messages etc.) launched by the compromised nodes are beyond the scope of this paper, we define a strong security notion for the SUM aggregate function in the setting where each reading comes from a fixed range $[0, R-1]$. Informally, our security defintion ensures that a compromised node cannot influence the aggregate value by more than $(R-1)$. This insures that even if some nodes report a reading much smaller than $(R-1)$, the compromised node cannot inflate the reading of these nodes to $(R-1)$. Though, a compromised node can always report its reading as $(R-1)$ (or 0), but that is not considered as an attack in our security defintion. Ideally, one would like secure in-network aggregation protocols that impose minimum overhead on existing aggregation protocols. For instance, secure aggregation protocols which continue to work in a single round and do not require the sensor nodes to do much more computation than simply aggregating messages are highly desirable (verification only happens at the end). In this work, we focus on **single round**, **end-to-end secure** in-network aggregation protocols which meet our strict security definition.

Related Works: A number of prior works have addressed the problem of authentication of aggregate value in sensor networks. Data aggregation in the presence of compromised nodes was first studied by [8]. Hu et al. [8] use delayed aggregation to propose protocols which are secure when at most one node is compromised. [3, 6, 9] consider a single aggregator model in which all nodes report to a single party called the aggregator which aggregates the value and forwards it to the base station. [6] works in this model using 'witness' nodes which serve as additional aggregators. [9] uses threshold signatures and provides security guarantees when a upper bounded fraction of the nodes are compromised. [12] works in a similar setting by dividing the aggregation tree into subtrees each of which aggregate messages and commit to them. Suspect aggregates are later attested using these commitments. Both [4] and [7] achieve strong security against compromised nodes and have a congestion of $O(\log n)$ and $O(\log^2 n)$ respectively, but they require multiple rounds of commmunication. All of the above protocols involve more than one round of interaction and are basically 'commit and re-check' schemes. In this work, we focus on single round in-network aggregation protocols. [2] talk abouts the impossibility of achieving a straightforward extension of the security goals for Message Authentication Codes to an aggregation setting with compromised nodes. It proposes a weaker notion of security in which nodes that are compromised do not contribute to the aggregate. In our model, each sensor node is both an aggregator and a contributor. [10] proposes a much weaker security notion in which the attacker can inflate the readings of other nodes to the maximum value without violating their security notion. Also, many other works explore the problem of maintaing secrecy of the aggregated value. In this work, we are not worried about the secrecy of the aggregate, in fact, the aggregate could be sent out in the clear in the network and observable by even a *passive* adversary. Our goal is to authenticate the aggregate value, that is, to make sure that illegitimate modifications are not made to the aggregate before it reaches the base station.

Our main result is that any single round, end-to-end secure in-network aggregation protocol (in the sense of our definition) must have a worst-case congestion of $\Omega(n)$, thereby defeating the purpose of aggregation. Note our lower bound is independent of the topology of the network. We then show that for a weaker adversary that does not compromise any sensor node, we can achieve our security definition with a congestion of $O(\log_2 n)$, which is the optimal for a tree topology. The paper is organized as follows: in Section 2 we formally define in-network aggregation. In Section 3, we provide a security definition via a security game to study the security of aggregation protocols against compromised nodes as well as any external adversary. We, then, present two protocols secure against compromised nodes with $O(n)$ congestion. We follow this with a proof which shows $\Omega(n)$ as a congestion lower bound for any protocol secure against compromised nodes. This is a first formal proof to show that strict security definitions may not be achievable with less than $\Omega(n)$ congestion in the internal adversary case. Finally, we propose a protocol secure against an external adversary with a congestion of $O(\log_2 n)$.

2. PROBLEM DEFINITION

We will now introduce our notation and then define an in-network aggregation protocol formally. The security parameter is denoted by k and n denotes the number of nodes in the network. We will assume that n is ploynomial in the security parameter, *i.e.* $n = poly(k)$. Let $S_1, ..., S_n$ denote the sensor nodes and let V denote the verifier. We assume that each node S_i shares a secret key K_i with the verifier V[1]. The aggregation protocol may be run several times, and each run is referred to as a session and identified by an unique session identifier *sid* (this may be just the number of the aggregation session). We assume that the start of a new session along with its *sid* is known to all the sensor nodes. We do not consider that as part of the in-network aggregation protocol. We now define an in-network aggregation protocol for the SUM aggregation function.

2.1 In-network Aggregation

An in-network aggregation protocol consists of the following four functions. All the functions defined below can be computed in time polynomial in the security parameter k and take the session identifier *sid* inherently as input.

- $\mathsf{Setup}(1^k, K_i)$: This function is executed by the sensor nodes at the beginning of each aggregation session. For node S_i, the input to the function is the security parameter, the global secret key K_i, and the session identifier *sid*. The output of the function for node S_i is the secret key k_i for the session. We call this the session key of the node S_i.
- $\mathsf{GenerateTag}(m_i, k_i)$: This algorithm is run by each node S_i to generate an authentication tag for its contributed message m_i. The input to this function is the message m_i and the session key k_i and the output is a tag σ_i that belongs to some tag space $\mathcal{T}$.
- $\mathsf{AggregateTag}(\sigma_1, \sigma_2)$: This algorithm is run by internal nodes to aggregate authentication tags that it

[1]In the public key setting, the key K_i could be derived using a Diffie-Hellman Key exchange protocol between the verifier and the node.

receives from the other sensor nodes. The input to the function are two authentication tags, say $\sigma_1 \in \mathcal{T}$ and $\sigma_2 \in \mathcal{T}$ and the output of the function is the aggregate tag $\sigma_1 \oplus \sigma_2$ obtained by applying some combination operation $\oplus$. Moreover, $(\mathcal{T}, \oplus)$ forms an abelian group such that finding the inverse in the group is computationally efficient.

Node S_i uses this function in the following manner: Once it receives messages and tags from all incoming nodes, it aggregates all these tags along with its own tag and sends the aggregate tag (and sum of all the corresponding messages) along its outgoing edge.

- Verify(M, Σ): This algorithm is run by the verifier V at the end of each aggregation session. The input to this function is the aggregate message M and an aggregate tag Σ. The output of the function is 1 denoting successful authentication and 0 denoting failure.

Given the above functions, the protocol description is very simple. At the beginning of each aggregation session (denoted by session identifier *sid*), each node executes the Setup to obtain the secret key of the session. The verifier also runs the setup to compute the session keys of all the nodes. Now in a given session, suppose node S_i receives message/tag pairs from nodes S_j, S_k, and S_l and is supposed to send message/tag to node S_p. Suppose S_i intends to contribute m_i and it receives the following message tag pairs (m_j, σ_j), (m_k, σ_k), and (m_l, σ_l). The node S_i does the following: It first uses the function GenerateTag to generate the tag σ_i corresponding to its message m_i. Then it aggregates the tags using the AggregateTag function and send the pair ($m_i + m_j + m_k + m_l$, $\sigma_i \oplus \sigma_j \oplus \sigma_k \oplus \sigma_l$) to the node S_p. Finally the verifier on receiving the pair (M, Σ) executes the Verify function and outputs 1 or 0. In the next subsection, we discuss the security definitions for an in-network aggregation protocol.

3. SECURITY DEFINTIONS

We study the security of an in-network aggregation protocol in two different scenarios. In the first scenario, we consider a strong adversary who can not only read and modify messages exchanged between any two nodes in the network, it can also compromise any subset of the sensor nodes or in other words can get access to any of the secret keys of a subset of the nodes. In the second scenario, we consider a weaker adversary who cannot compromise any sensor node. We call the first class of adversary an *insider adversary* and the second one an *outsider adversary*[2]. The message space $\mathcal{M}$ of messages m_i is assumed to be $[0, R-1]$ for some integer R (again polynomial in the security parameter) and we assume that all the n node participate in the protocol.

For completeness, we provide below the standard definition of a pseudorandom family of functions, which we use later.

DEFINITION 1. *A family of functions $H = \{H_k\}_{k \in \mathcal{K}}$ where $H_k : \mathbb{Z}_p \to \mathbb{Z}_p$ is a pseudorandom family of functions if for every probabilistic polynomial time algorithm A, $\mathbf{Adv}_H^{prf}(A)$ is negligible in the security parameter, where $\mathbf{Adv}_H^{prf}(A)$ is defined as*

$$|Pr[k \leftarrow \mathcal{K} : A^{H_k} = 1] - Pr[RF \leftarrow \mathcal{F} : A^{RF} = 1]|$$

where $\mathcal{F}$ is the set of all functions $f : \mathbb{Z}_p \to \mathbb{Z}_p$

[2]For the sake of simplicity, we do not model concurrent executions of the in-network aggregation protocol, in the following security definitions. Against an adversary, which can choose to attack a session after observing many other sessions, our lower bound will continue to hold.

3.1 Security against insider adversaries

Here the adversary gets complete control over a subset of sensor nodes. Since it is not possible for the verifier to distinguish between a compromised node and a non-compromised one, a compromised node may add as large a value that a node is allowed to add. What the verifier can ensure is that a compromised node should not be able to do much more than that. That is, it should not be able to change (increase or decrease) the readings of the non-compromised nodes without being detected by the verifier. This motivates the following security game.

Game $G_A^{insider}$

procedure Initialize
Let $\mathcal{S} = \{S_1, ..., S_n\}$
$U \leftarrow \{\}$; $M_h \leftarrow 0$
Run Setup to fix the session keys $k_1, ..., k_n$

procedure RevealKeys($\mathcal{U}$)
$U \leftarrow \mathcal{U}$
return all the session keys of nodes in the subset $\mathcal{U}$.

procedure RevealTags$((m_1, S_{i_1}), ..., (m_k, S_{i_k}))$
Generate and return tags $\sigma_1, ..., \sigma_k$ for messages $m_1, ..., m_k$ corresponding to nodes $S_{i_1}, ..., S_{i_k}$.
$M_h \leftarrow \sum_{j, S_{i_j} \in \mathcal{S} \setminus U} m_j$

procedure Finalize(M, Δ)
If $(U = \mathcal{S})$ then return 0
If (Verify$(M, \Delta) = 1$) and
$((M - M_h > |U| \cdot (R-1))$ or $(M < M_h))$
then return 1 else return 0

With respect to the above game, we define the advantage of an adversary in breaking a protocol P in the following manner:

$$\mathbf{Adv}_P^{insider}(A) = \mathbf{Pr}[G_P^{insider} \Rightarrow 1]$$

In simpler words, an adversary is allowed to obtain the secret keys of a subset $\mathcal{U}$ of nodes of its choice. The nodes for which the adversary does not have the secret key are called the honest nodes. It may obtain one correctly generated tag for any message of its choice corresponding to the honest nodes. Finally it attacks with a message, tag pair (M, Σ) and it is said to succeed if Verify succeeds for this pair and the message is larger than the sum of messages queried in the **RevealTags** queries for the honest nodes (M_h) by an additive factor of $(R-1)$ times the number of dishonest nodes or less than it. This captures the case we talked about earlier where we said the the adversary cannot be stopped from tampering the reading of the compromised node but it should not be able to tamper readings of the non-compromised nodes.

DEFINITION 2 (INSIDER SECURITY). *We say a protocol P is insider secure if for every probabilistic polynomial time adversary A, $\mathbf{Adv}_P^{insider}(A)$ is negligible in the security parameter k.*

Now we will first see two protocol constructions that are insider secure but both have linear congestion. Finally, we will argue why any protocol that is insider secure cannot have less than linear amount of congestion.

3.1.1 Aggregate messages, do not aggregate tags

Here we argue that the protocol defined below is insider secure. In order to define the protocol, we just need to define the four functions of Section 2.1. Here is the description of the protocol. We call this protocol P_1:

- Let $MAC_1 : \mathcal{K}_1 \times \mathcal{S} \to \mathcal{K}_2$ and $MAC_2 : \mathcal{K}_2 \times [0,\ldots,R-1] \to \mathcal{U}$ be two message authentication code (MAC) algorithms.
- Setup: The node S_i (and the verifier) applies MAC_1 at K_i and *sid* to compute the session key k_i.
- GenerateTag: The node S_i uses the message authentication code MAC_2 to generate the tag

$$\sigma_i = (0, ..., \underbrace{MAC_2(k_i, m_i)}_{i^{th} \text{ term}}, 0, ..., 0).$$

  The tag space is $\mathcal{T} = (\mathcal{U} \cup \{0\}) \times ... \times (\mathcal{U} \cup \{0\})$ where $\mathcal{U}$ denotes the range of MAC_2.
- AggregateTag: Here we define the combination operator $\oplus$ that operates on elements of $\mathcal{T}$. Given $\sigma_1 = (t_1, ..., t_n)$ and $\sigma_2 = (t'_1, ..., t'_n)$, then $\sigma = (t''_1, ..., t''_n) = \sigma_1 \oplus \sigma_2$ is defined as follows: for all i,

$$t''_i = \begin{cases} t_i + t'_i & \text{if either } t_i = 0 \text{ or } t'_i = 0; \\ 0 & \text{otherwise.} \end{cases}$$

  One can easily verify that $(\oplus, \mathcal{T})$ is a group and finding inverse of any element is simple.
- Verify: Given an input (M, Δ), where $\Delta = (t_1, ..., t_n)$ the verifier does the following: for each i it checks if there exists a message $m_i \in \{0, ..., R-1\}$ such that $MAC(k_i, m_i) = t_i$. If such messages $m_1, ..., m_n$ exist and $\sum_i m_i = M$ then it outputs 1 else it outputs 0. It also returns 0 if any of t_i is 0.

Security.

The security of the above aggregate protocol hinges on the security of the underlying MAC scheme. Given an adversary A with non negligible advantage in $G_P^{insider}$, we can constuct an adversary B that can forge the underlying MAC scheme. The following lemma formalizes this.

THEOREM 3. *Let A be an adversary attacking the authenticated aggregation protocol P_1 defined above and having a running time of t. Then there is an adversary B that attacks the message authentication code MAC_2 such that:*

$$\mathbf{Adv}_{MAC_2}^{uf-cma}(B) \geq (1/n) \cdot \mathbf{Adv}_{P_1}^{insider}(A)$$

Furthermore, B makes at most 1 mac generation query and 1 verification query and has a running time of $O(t + n \cdot R)$. [3]

PROOF. We construct the adversary B that attacks MAC_2 using the adversary A that attacks the aggregation protocol P_1. Here is the description of this adversary.

Algorithm B
1. $H \leftarrow [n]$
2. Pick session keys $k_1, ..., k_n$ randomly from the keyspace.
3. Run A
4. When A asks for session keys for indices in the set $\mathcal{U}$:
5. return keys $k_{i_1}, ..., k_{i_{|\mathcal{U}|}}$ to A
6. Let $H = [n] \backslash \mathcal{U}$.
7. When A asks to reveal tags, i.e., it sends a query $(m_1, S_{i_1}), ..., (m_k, S_{i_k})$:
8. Pick an index j randomly from the set H.
9. For each i_r, if $i_r \neq j$, then return tag $MAC_2(k_{i_r}, M_r)$ for message m_r. else make a call to the MAC_2 generation oracle for B and use the response of the oracle as a tag for message m_r.
10. Suppose A halts and returns $(M, (t_1, ..., t_n))$:
11. For all $i \neq j$, find the message $m'_i \in [0, R-1]$ such that $MAC_2(k_i, m'_i) = t_i$.
12. If the above messgages are found: B sends $((M - \sum_{i \neq j} m'_i), t_j)$ to the verification oracle. else B sends $(\perp, \perp)$ to the verification oracle.

The analysis of the adversary B that attacks MAC_2 is simple. B executes A. We will assume that A queries tags of chosen messages for *all* the nodes. We can do this since if there is an adversary that makes tag queries for only a subset of nodes, then there is another adversary with higher chance of success that makes tag queries for all nodes. Consider a run of the game G_{P_1} with adversary A in which A wins. Let the set of honest nodes be H and A's final output be $(M, (t_1, \ldots, t_n))$. Since $\mathsf{Verify}(M, (t_1, \ldots, t_n)) = 1$, there exist $(m'_1, \ldots, m'_n)$ such that $\sum_{i=1}^n m'_i = M$ and for each i, $m'_i \in \{0, \ldots, R-1\}$ and $t_i = MAC_2(k_i, m'_i)$. Let $M'_h = \sum_{i \in H} m'_i$. Since $M'_h \neq M_h$, there exists $p \in H$ such that $m_p \neq m'_p$ where m_p is the message in the tag query for the node p. Conditioned on the success of A's attack, we note that B's attack succeeds if the randomly chosen index j in line (8) is equal to p. The probability that this occurs is at least $(1/n)$. This gives the statement of the theorem. □

3.1.2 Aggregate (XOR) tags, do not aggregate messages

Here we give another protocol that is insider secure but the congestion is linear in the number of nodes in the network. It differs from our protocol in the previous section in that here the length of the aggregate tag is small, whereas the messages are sent without aggregation. Again, we define the the four functions of Section 2.1. Let us call this protocol P_2.

[3] The security definitions for message authentication codes above is standard. The reader is requested to refer to notes by Mihir Bellare and Philip Rogaway [1] for this.

- Let $MAC_1 : \mathcal{K}_1 \times \mathcal{S} \rightarrow \mathcal{K}_2$ and $H : \mathcal{K}_2 \times [0, \ldots, R-1] \rightarrow \{0,1\}^m$ be two message authentication code (MAC) algorithms.

- Setup: The node S_i (and the verifier) applies MAC_1 at K_i and *sid* to compute the session key k_i.

- GenerateTag: The node S_i uses the message authentication code H to generate the tag $\sigma_i = H_{k_i}(m_i)$. The tag space is $\mathcal{T} = \{0,1\}^m$.

- AggregateTag: Here we define the combination operator $\oplus$ that operates on elements of $\mathcal{T}$. Given $\sigma_1 \in \{0,1\}^m$ and $\sigma_2 \in \{0,1\}^m$, then $\sigma = \sigma_1 \oplus \sigma_2$ is just the bitwise XOR of σ_1 and σ_2.

  One can easily verify that $(\oplus, \mathcal{T})$ is a group and finding inverse of any element is simple.

- Verify: The verifier receives the pair (M, Δ), where M in this case is a tuple of messages $M = (m_1, ..., m_n)$ and $\Delta \in \{0,1\}^m$. The Verify function is defined in the following manner: Check if $H_{k_i}(m_i) \oplus ... \oplus H_{k_n}(m_n) = \Delta$ and $\forall i, 0 \leq m_i \leq R-1$. If both conditions are satisfied, then output 1 else output 0.

Security.

As with the protocol discussed earlier, the security of this protocol hinges on the security of the underlying MAC scheme. Given an adversary A with non negligible advantage in $G_{P_2}^{insider}$, we can constuct an adversary B that can forge the underlying MAC scheme, H. The following lemma formalizes this.

THEOREM 4. *Let A be an adversary attacking the authenticated aggregation protocol P_2 defined above and having a running time of t. Then there is an adversary B that attacks the message authentication code H such that:*

$$\mathbf{Adv}_H^{uf-cma}(B) \geq (1/n) \cdot \mathbf{Adv}_{P_2}^{insider}(A)$$

Furthermore, B makes at most 1 mac generation query and 1 verification query and has a running time of $O(n+t)$.

PROOF. As for protocol P_1, we will construct an adversary B that attacks the underlying MAC, H using an adversary A that attacks the protocol P_2. The description of this adversary is very similar to the adversary in the previous subsection.

Algorithm B
1. $H \leftarrow [n]$
2. Pick session keys $k_1, ..., k_n$ randomly from the keyspace.
3. Run A
4. When A asks for session keys for indices in the set $\mathcal{U}$:
5. return keys $k_{i_1}, ..., k_{i_{|\mathcal{U}|}}$ to A
6. Let $H = [n] \backslash \mathcal{U}$.
7. When A asks to reveal tags, i.e., it sends a query $(m_1, S_{i_1}), ..., (m_k, S_{i_k})$:
8. Pick an index j randomly from the set H.
9. For each i_r, if $i_r \neq j$, then return tag $H_{k_{i_r}}(m_r)$ for message m_r. else make a call to the MAC generation oracle for B and use the response of the oracle as a tag for message m_r.
10. Suppose A halts and returns $((m'_1, ..., m'_n), \Delta)$:
11. Compute $T = \oplus_{i \neq j} H_{k_i}(m'_i) \oplus \Delta$
12. B sends (m'_j, T) to the verification oracle.

B executes A. We will assume that A queries tags of chosen messages for all the nodes. We can do this since if there is an adversary that makes tag queries for only a subset of nodes, then there is another adversary with higher chance of success that makes tag queries for all nodes. Consider a run of the game $G_{P_2}^{insider}$ with adversary A in which A wins. Let the set of honest nodes be H and A's final output be $((m_1, \ldots, m_n), \Delta)$. Since $\mathsf{Verify}((m_1, \ldots, m_n), \Delta) = 1$, for each i, $m_i \in \{0, \ldots, R-1\}$. Let $M'_h = \sum_{i \in H} m_i$. We have that $\sum_{i=1}^{n} m_i - M_h$ is either < 0 or $> (n - |H|) \cdot (R-1)$. This implies $M'_h \neq M_h$. Hence there exists $p \in H$ such that $m_p \neq m'_p$ where m_p is the message used in the tag query for node p. Conditioned on the success of A's attack, we note that B's attack succeeds if the randomly chosen index j in line (8) is equal to p. The probability that this occurs is at least $(1/n)$. This gives the statement of the theorem. □

3.2 Congestion lower bound for insider secure protocols

In the previous subsections, we talked about two protocols that are insider secure but there is linear amount of congestion. Here we show that any protocol that is insider secure will necessarily have $\Omega(n)$ congestion.

We will show that if the congestion is less than $(n/2) \cdot \log R$, then there is an adversary that succeeds in attacking the protocol in the sense of Definition 1. The main idea is the following: The verifier receives the aggregate message and an aggregate tag. If the total size of the information that it receives is less than $(n/2) \cdot \log R$, then by simple pigeonhole principle, this means that there are lots of pairs of message tuples $\bar{m} = (m_1, ..., m_n)$ and $\bar{m}' = (m'_1, ..., m'_n)$ [4] such that the aggregate tag for both these message tuples are the same and the message aggregates for these tuples are also the same. Consider any such pair of tuples. Since the message aggregates are the same, there exists an index $j \in [n]$ such that $m_j > m'_j$. Now, just by knowing the secret key of the j^{th} node and tags of messages $m_1, ..., m_{j-1}, m_{j+1}, ..., m_n$, an adversary would be able to compute the aggregate tag for the message tuple $(m'_1, ..., m'_{j-1}, R-1, m'_{j+1}, ..., m'_n)$ without even knowing $m'_1, ..., m'_{j-1}, m'_{j+1}, ..., m'_n$ (by doing some

[4] m_i and m'_i denotes the message of the i^{th} node

simple group operations). Note that the aggregate of this message tuple is just the ((aggregate of $\bar{m}$) - $m'_j + (R-1)$). This is appropriately large for the attack to work. So, an adversary just guesses the index j, $\bar{m}$, and m'_j and mounts an attack. What remains to show is that the probability that this succeeds, is large. This is precisely what we discuss in the remaining section.

THEOREM 5. *Consider any in-network aggregation protocol P. If P is insider secure, then the worst-case congestion of this protocol is $\Omega(n)$.*

PROOF. For the sake of contradiction, assume that the worst case congestion is at most $(\frac{n}{2} \cdot \log R)$. Let $H : \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{T}$ be the function that maps the key and message space to the tag space (as specified by the GenerateTag algorithm of P) and $\oplus$ be the aggregation function for tags as specified in AggregateTag. For a fixed value of the session keys $(k_1, ..., k_n)$, consider the mapping f of message tuples to valid (M, Δ) pairs. By valid, it is meant that

$$f(m_1, ..., m_n) = \left(\sum_i m_i, \oplus_i H_{k_i}(m_i) \right)$$

Now since maximum number of bits that the verifier receives is $(n/2) \log R$, the maximum number of distinct (M, Δ) pairs is $R^{n/2}$. Let $f^{-1}(pair)$ denote the set of message tuples that map to *pair*. Let B be the set of bad tuples in the sense that for any tuple $\bar{m} \in B$, $|f^{-1}(f(\bar{m}))| = 1$. Note that $|B| \leq R^{n/2}$. For a randomly chosen message tuple, the probability that it belongs to B is $\leq R^{-n/2}$.

Now conditioned on a message tuple $(m_1, ..., m_n)$ not belonging to B (this happens with probability at least $(1 - R^{-n/2})$), we know that there is another message tuple $(m'_1, .., m'_n)$ such that:

(i) $\sum_i m_i = \sum_i m'_i$

(ii) $\oplus_i H_{k_i}(m_i) = \oplus_i H_{k_i}(m'_i)$

Now because of the first property above we know that there is an index j such that $m'_j < m_j$. We also know that

$$\begin{aligned} & \oplus_{i \neq j} H_{k_i}(m'_i) \oplus H_{k_j}(m'_j) = \oplus_i H_{k_i}(m_i) \\ \implies & \oplus_{i \neq j} H_{k_i}(m'_i) = \oplus_i H_{k_i}(m_i) \oplus H_{k_j}(m'_j)^{-1} \\ \implies & \oplus_{i \neq j} H_{k_i}(m'_i) \oplus H_{k_j}(R-1) = \\ & \qquad \oplus_i H_{k_i}(m_i) \oplus H_{k_j}(m'_j)^{-1} \oplus H_{k_j}(R-1) \quad (1) \end{aligned}$$

Now given this, we construct the following adversary that attempts to win the game $G_P^{insider}$:

Adversary A

1. Pick j randomly from the set $\{1, ..., n\}$.
2. Pick two distinct messages l_1 and l_2 randomly from $\{0, ..., (R-1)\}$. Let $l_1 < l_2$.
3. Pick $n-1$ messages $m_1, ..., m_{n-1}$ randomly from the set $\{0, ..., (R-1)\}$.
4. Query the procedure **RevealKeys** with the set $\{S_j\}$ and get back the session key k_j for S_j.
5. Query the procedure **RevealTags** with input $((m_1, S_1), ..., (m_{j-1}, S_{j-1}), (m_j, S_{j+1}), ..., (m_{n-1}, S_n))$ to obtain tags t_i for every $i \neq j$ and compute the aggregate tag $T = H_{k_j}(l_2) \oplus (\oplus_{i \neq j} t_i)$.
6. Compute $T' = H_{k_j}(l_1)^{-1} \oplus T \oplus H_{k_j}(R-1)$.
7. Output $((\sum_{i=1}^{n-1} m_i + l_2 - l_1 + R - 1), T')$ as an attack for the protocol P

Next, we show that the advantage of the above adversary over the protocol is large.

LEMMA 6. *The following is true for the above adversary A:*

$$\begin{aligned} \mathbf{Adv}_P^{insider}(A) &\geq \frac{1}{n \cdot R} \cdot \left(1 - R^{-n/2}\right) \\ &\geq \frac{1}{poly(k)}. \end{aligned}$$

PROOF. Let $\bar{\alpha} = (\alpha_1, ..., \alpha_n)$ be a randomly chosen message tuple and let $M_{\bar{\alpha}} = \sum_i \alpha_i$ and $\Delta_{\bar{\alpha}} = \oplus_i H_{k_i}(\alpha_i)$. Now if the description size of $(M_{\bar{\alpha}}, \Delta_{\bar{\alpha}})$ is at most $(\frac{n}{2} \cdot \log R)$, then we know that with probability at least $(1 - R^{-n/2})$ there is another tuple of messages $(\beta_1, ..., \beta_n)$ such that for at least one $j \in [n]$, $\beta_j < \alpha_j$. Furthermore, $\sum_i \beta_i = \sum_i \alpha_i$ and $\oplus_i H_{k_i}(\alpha_i) = \oplus_i H_{k_i}(\beta_i)$.

Conditioned on the existence of this tuple, in the case of our adversary, $(m_1, ..., m_{j-1}, l_2, m_j, ..., m_{n-1})$ may be interpreted as $(\alpha_1, ..., \alpha_n)$ above, and l_1 may be interpreted as β_j. Since j and l_1 are chosen randomly, we get that the probability that A gets them correct is at least $\frac{1}{n \cdot R}$. Conditioned on this event happening, we have that the verifier accepts on the input (M, Δ), where

$$M = \left(\sum_{i=1}^{n-1} m_i + l_2 - l_1 + R - 1 \right), \text{and}$$

$$\Delta = \oplus_{i \neq j} H_{k_i}(m_i) \oplus H_{k_j}(l_2) \oplus H_{k_j}(l_1)^{-1} \oplus H_{k_j}(R-1).$$

This is due to equation (1) and that there is only one compromised node and we have

$$M = \sum_{i=1}^{n-1} m_i + l_2 - l_1 + R - 1 > \sum_{i=1}^{n-1} m_i + 1 \cdot (R-1).$$

So, we get that A succeeds with probability at least

$$(1 - R^{-n/2}) \cdot \frac{1}{n \cdot R} \geq \frac{1}{poly(k)}.$$

□

This completes the proof of Theorem 5.

3.3 Security against outsider adversary

The outsider case is similar to the insider case except that here the adversary cannot get access to the session keys of any of the sensor nodes. In terms of the security game, here the adversary does not have access to the **RevealKeys** method and $U = \{\}$. For completeness we give the security game for the outsider case.

Game $G_P^{outsider}$

procedure Initialize
1. Let $\mathcal{S} = \{S_1, ..., S_n\}$
2. $M_h \leftarrow 0$
3. Run Setup to fix the session keys $k_1, ..., k_n$

procedure RevealTags$((m_1, S_{i_1}), ..., (m_k, S_{i_k}))$
1. Generate and return tags $\sigma_1, ..., \sigma_k$ for messages $m_1, ..., m_k$ corresponding to nodes $S_{i_1}, ..., S_{i_k}$.
2. $M_h \leftarrow \sum_{j, S_{i_j} \in \mathcal{S}} m_j$

procedure Finalize(M, Δ)
1. If (Verify$(M, \Delta) = 1$) and (($M > M_h$) or ($M < M_h$)) then return true else return false.

3.3.1 An aggregation protocol secure against outsiders

We will now describe an aggregate protocol which is secure against outsider adversaries. The intuition behind the protocol is as follows: Each node shares one long-term key (k_i) with the verifier. The verifier also shares one common key s with all the nodes. A session key is derived from s and the session identifier *sid* and is the same for all the sensor nodes. The session key is used to aggregate the messages of each node. The protocol is formally defined below. We will call this protocol P_3.

- Let p be a large prime, $H : \mathcal{K}_1 \times \mathcal{S} \rightarrow \mathbb{Z}_p$ be a family of pseudorandom functions, and $I : \mathcal{K}_2 \times \mathbb{Z}_p \rightarrow \mathbb{Z}_p$ be another such family. Here $\mathcal{K}_1$ is the keyspace of H and $\mathcal{K}_2$ is the keyspace of I. We will assume that the verifier shares the pair of keys (k_i, s) with node S_i. Here $k_i \xleftarrow{\$} \mathcal{K}_1$ and $s \xleftarrow{\$} \mathcal{K}_2$.
- Setup: Each node and the verifier generates common session key k_{sid} by applying the function I on the common shared key s and the session identifier *sid* (which in this case may be thought of as the aggregation round number). So, $k_{sid} \leftarrow I_s(sid)$.
- GenerateTag: The i^{th} node generates the following tag for its messgae m_i: $\sigma_i \leftarrow m_i \cdot k_{sid} + H_{k_i}(k_{sid})$.
- AggregateTag: The i^{th} node just takes the sum of the all the tags its receives and its own tag.
- Verify: The verifier in this case receives an aggregate message M and an aggregate tag Δ. It checks if

$$\Delta = M \cdot k_{sid} + \sum_i H_{k_i}(k_{sid}).$$

If this check succeeds, it accepts else it rejects.

Security.

For showing that the above protocol is secure from outside adversaries, we will show that an attack on the protocol leads to an attack on the underlying pseudorandom functions. So, as long as we use secure psudorandom functions in our protocol, it will be secure.

For the sake of contradiction, we will start with an adversary that attacks our protocol, i.e,

$$\mathbf{Adv}_{P_3}^{outsider}(A) \geq 1/poly(k).$$

We will assume that the adversary makes **RevealTags** queries for all nodes. This is because if there is an adversary that makes tag queries for only a subset of nodes, then there is another adversary with higher chance of success that makes tag queries for all nodes. The next lemma shows that if any such adversaries exist then there will be an adversary that can compute the value of the session key k_{sid}.

LEMMA 7. *Let A be an adversary for protocol P_3. There is an adversary A' such that A' determines the challenge session key $k_{sid} = I_s(sid)$ with probability at least $\mathbf{Adv}_{P_3}^{outsider}(A)$.*

PROOF. Let (M, Δ) be the forgery by A. Let m_i be the message sent to the i^{th} node in the **RevealTags** query and let y_i be the response (this is the tag for m_i). Given that A succeeds, we have

$$M \neq \Sigma_{i=1}^n m_i.$$

Thus k_{sid} can be determined as follows:

$$\frac{\Delta - \Sigma_{i=1}^n y_i}{M - \Sigma_{i=1}^n m_i} = \frac{k_{sid} \cdot (M - \sum_{i=1}^n m_i)}{M - \sum_{i=1}^n m_i} = k_{sid}$$

□

We will use the following security amplification result from Dodis et. al. [5]. The result says that parallel repetition of pseudorandom functions leads to a stronger psedorandom function.

THEOREM 8 (THEOREM 7, [5]). *If $H : \mathcal{K}_1 \times \mathbb{Z}_p \rightarrow \mathbb{Z}_p$ is a secure pseudorandom family of functions then so is $H^n : \mathcal{K}_1^n \times \mathbb{Z}_p \rightarrow \mathbb{Z}_p^n$ where*

$$H^n((k_1, k_2 ... k_n), x) = H_{k_1}(x) | H_{k_2}(x) | ... | H_{k_n}(x).^5$$

We use the above result to show that if there is an adversary that attacks P_3, then we can construct an adversary that attacks H^n (in the PRF sense). The next theorem formalizes this.

THEOREM 9. *Let A be an adversary attacking the authenticated aggregation protocol P_3 and having a running time of t. Then there is an adversary B that attacks the pseudorandom function H^n such that:*

$$\mathbf{Adv}_{H^n}^{prf}(B) \geq \mathbf{Adv}_{P_3}^{outsider}(A) - 1/|\mathcal{K}_1|.$$

Furthermore, B makes only one query and has a running time of $O(t)$.

PROOF. We use A to construct an adversary B that distinguishes H^n from a random function f mapping $\mathbb{Z}$ to $\mathbb{Z}_p^n$. We will need the following simple lemma with respect to random functions.

[5] here | denotes concatenation.

LEMMA 10. *Let $f : \mathbb{Z}_p \to \mathbb{Z}^n$ be a random function. Consider a variant of game G'_{P_3} in which instead of returning tags $\sigma_i = m_i \cdot k_{sid} + H_{k_i}(k_{sid})$ in the* **RevealTags** *query, the following tags are returned: $\sigma_i = m_i \cdot k_{sid} + f(k_{sid})[i]$. The probability that A succeeds in computing k_{sid} in such a game $\leq 1/|\mathcal{K}_1|$.*

PROOF. This will be an information theoretic argument. For any fixed value of $m_1, ..., m_n$, and $\sigma_1, ..., \sigma_n$, the distribution

$$(\sigma_1 - m_1 \cdot K)|...|(\sigma_n - m_n \cdot K)$$

for randomly chosen $K \in \mathcal{K}_1$ is a uniform distribution over $\mathbb{Z}_p^n$. This means that for any key $K \in \mathcal{K}_1$, it is as likely to be the challenge session key k_{sid} as any other key. This means that the probability that A can guess the value of the session key is at most $1/|\mathcal{K}_1|$. □

We now give the construction of the adversary B that attacks H^n in the PRF sense.

Algorithm B^g
1. $k_{sid} \leftarrow I_s(sid)$
2. Run A
3. When A asks to reveal tags, i.e., it sends a query $(m_1, S_1), ..., (m_n, S_n)$:
4. B makes a query to its function oracle g and gets back $(y_1, ..., y_n) \leftarrow g(k_{sid})$
5. Tags $(\sigma_1, ..., \sigma_n)$, where $\sigma_i = m_i \cdot k_{sid} + y_i$ are returned to A
6. Suppose A halts and returns the key K
7. If $(K = k_{sid})$ then B outputs 1 else B outputs 0.

The analysis of B is simple. When $g = f$ (i.e., random function), then from Lemma 10 we know that

$$\mathbf{Pr}[B^f = 1] \leq 1/|\mathcal{K}_1| \tag{2}$$

Furthermore, we know that

$$\mathbf{Pr}[B^{H^n} = 1] = \mathbf{Adv}^{outsider}_{P_3}(A) \tag{3}$$

Finally, combining equations (2) and (3), we get the following:

$$\mathbf{Adv}^{PRF}_{H^n}(B) = |\mathbf{Pr}[B^{H^n} = 1] - \mathbf{Pr}[B^f = 1]|$$
$$\Rightarrow \mathbf{Adv}^{PRF}_{H^n}(B) \geq \mathbf{Adv}^{outsider}_{P_3}(A) - 1/|\mathcal{K}_1|.$$

This concludes the proof of Theorem 9.

4. CONCLUSION

We presented a lower bound of $\Omega(n)$ on congestion in a one-round end-to-end secure in-network aggregation protocol. To prove the lower bound, we formally define an in-network aggregation protocol and a strong security notion to study the security of the protocol against compromised nodes. Our security notion prevents any compromised node from modifying the contributed readings of the honest nodes. We show that for such a strict security notion, we cannot do better in terms of congestion than a trivial protocol which doesn't do any aggregation. The same security notion, though, can be met against a weaker adversary (an adversary that does not compromise nodes) with a better congestion.

In summary, our results can be used to argue that as far as single-round, end-to-end secure in-network aggregation is concerned, the most simple protocols (with linear congestion) are the only solutions. For lower congestion, we have to construct protocols that run in multiple rounds or are not end-to-end.

5. REFERENCES

[1] Mihir Bellare and Philip Rogaway, *Introduction to modern cryptography*, http://cseweb.ucsd.edu/ mihir/cse207/classnotes.html/.

[2] A.C.-F. Chan and C. Castelluccia, *On the (im)possibility of aggregate message authentication codes*, Information Theory, 2008. ISIT 2008. IEEE International Symposium on, july 2008, pp. 235 –239.

[3] Haowen Chan, Adrian Perrig, Bartosz Przydatek, and Dawn Song, *Sia: Secure information aggregation in sensor networks*, J. Comput. Secur. **15** (2007), 69–102.

[4] Haowen Chan, Adrian Perrig, and Dawn Song, *Secure hierarchical in-network aggregation in sensor networks*, Proceedings of the 13th ACM conference on Computer and communications security (New York, NY, USA), CCS '06, ACM, 2006, pp. 278–287.

[5] Yevgeniy Dodis, Russell Impagliazzo, Ragesh Jaiswal, and Valentine Kabanets, *Security amplification for interactive cryptographic primitives*, Lecture Notes in Computer Science, vol. 5444, pp. 128–145, Springer Berlin / Heidelberg, 2009.

[6] W. Du, J. Deng, Y.S. Han, and P.K. Varshney, *A witness-based approach for data fusion assurance in wireless sensor networks*, Global Telecommunications Conference, 2003. GLOBECOM '03. IEEE, vol. 3, 2003, pp. 1435 – 1439 vol.3.

[7] Keith B. Frikken and Joseph A. Dougherty, IV, *An efficient integrity-preserving scheme for hierarchical sensor aggregation*, Proceedings of the first ACM conference on Wireless network security (New York, NY, USA), WiSec '08, ACM, 2008, pp. 68–76.

[8] Lingxuan Hu and David Evans, *Secure aggregation for wireless networks*, In Workshop on Security and Assurance in Ad hoc Networks, IEEE Computer Society, 2003, p. 384.

[9] A. Mahimkar and T.S. Rappaport, *Securedav: a secure data aggregation and verification protocol for sensor networks*, Global Telecommunications Conference, 2004. GLOBECOM '04. IEEE, vol. 4, 2004, pp. 2175 – 2179 Vol.4.

[10] Mark Manulis and Jörg Schwenk, *Security model and framework for information aggregation in sensor networks*, ACM Trans. Sen. Netw. **5** (2009), 13:1–13:28.

[11] Alfred J. Menezes, Paul C. Van Oorschot, Scott A. Vanstone, and R. L. Rivest, *Handbook of applied cryptography*, 1997.

[12] Yi Yang, Xinran Wang, Sencun Zhu, and Guohong Cao, *Sdap: a secure hop-by-hop data aggregation protocol for sensor networks*, Proceedings of the 7th ACM international symposium on Mobile ad hoc networking and computing (New York, NY, USA), MobiHoc '06, ACM, 2006, pp. 356–367.