# Efficient Handling of Adversary Attacks in Aggregation Applications

Gelareh Taban[1] and Virgil D. Gligor[2,⋆]

[1] ECE, University of Maryland, College Park
gelareh@umd.edu
[2] ECE and CyLab, Carnegie Mellon University
gligor@cmu.edu

**Abstract.** Current approaches to handling adversary attacks against data aggregation in sensor networks either aim exclusively at the detection of aggregate data corruption or provide rather inefficient ways to identify the nodes captured by an adversary. In contrast, we propose a distributed algorithm for efficient identification of captured nodes over a constant number of rounds, for an arbitrary number of captured nodes. We formulate our problem as a combinatorial group testing problem and show that this formulation leads not only to efficient identification of captured nodes but also to a precise cost-based characterization of when in-network aggregation retains its assumed benefits in a sensor network operating under persistent attacks.

## 1 Introduction

Data aggregation is generally believed to be a fundamental communication primitive in resource-constrained, wireless sensor networks. In principle, in-network aggregation of sensor data can drastically reduce network communication. To accomplish this, nodes are logically organized as a tree—called the "aggregation tree"—that is rooted at a Base Station (BS). In response to BS queries, nodes aggregate the critical data they receive from their descendents together with their own data, and forward their partial aggregates to their ancestor nodes in the aggregation tree.

*Motivation.* A significant risk of aggregation is that a node that is captured by an adversary could report arbitrary values as its aggregation result, thereby corrupting not only its own measurements but also that of all the nodes in its entire aggregation sub-tree. As a consequence, an adversary who captures nodes selectively and strategically (e.g., close to the BS) can corrupt the entire

⋆ This research was supported in part by US Army Research Laboratory and the UK Ministry of Defence under Agreement Number W911NF-06-3-0001 and by the US Army Research Office under Contract W911NF-07-1-0287 at the University of Maryland. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the US Army Research Laboratory, US Army Research Office, the U.S. Government, the UK Ministry of Defense, or the UK Government.

S. Jajodia, and J. Lopez (Eds.): ESORICS 2008, LNCS 5283, pp. 66–81, 2008.

network aggregation process, while incurring minimal cost and effort. Therefore, to achieve reliable aggregation, and, in particular, to assure the integrity of aggregation process, it is important (i) to detect an adversary's presence in the network (i.e., by discovering aggregated-data corruption) and (ii) to identify and remove (i.e., revoke [2]) the captured nodes which corrupt data aggregates.

Most recent work on secure data aggregation has focused exclusively on efficient *detection* of integrity breach in the aggregation process (e.g., [9,3,14,10,7]). While detection of integrity breach is the first necessary step to achieving secure data aggregation, it does not provide a fully adequate response to malicious-node behavior; i.e., detection of integrity breach alone does not unambiguously identify and remove specific malicious nodes from the network. Exclusive reliance on detection of corrupt aggregate results would leave the network unprotected against repeated attacks that deny service to the BS. An effective approach to handling this problem would (i) identify corrupted nodes and remove them from the aggregation tree (e.g., by node revocation), and (ii) ensure continued, but gracefully degraded aggregation services, even during an attack period. Identification and removal of corrupted nodes has the added benefit of acting as a deterrent against some potential adversaries who might avoid the risk of being identified.

*Problem.* We consider an aggregation scenario where a subset of nodes is corrupted by an adversary. A corrupted node can (i) insert a false data into the network or (ii) if it is an aggregating node, output a false aggregation result. The goal of the corrupted node is to convince the base station to accept an invalid value. Since the network cannot protect against the insertion of incorrect aggregation values without assuming specific distributions on the environmental data [13,14], we simply assume that all valid sensor inputs $r$ must be within a given range $r_1 < r < r_2$. Our objective is to (i) detect an attack in the network, (ii) identify malicious nodes, (iii) ensure graceful degradation of the aggregate with respect to the number of corrupted nodes in the network, while retaining the *efficiency advantages* of data aggregation.

A straight-forward method of achieving the first three stated objectives *without* retaining in-network aggregation, henceforth called the *baseline scheme*, would be to detect the presence of malicious behavior in the network [7,3], and then require each node to directly transmit their data *without* aggregation along with a message authentication code (MAC) to the BS. By eliminating in-network aggregation, we would trivially remove any attacks on the aggregation process. The BS could then identify any malicious nodes that inject false data by range testing the received data. If the corrupted nodes are persistently malicious, the BS could identify *all* corrupted nodes. Furthermore, the BS itself could reconstruct the network aggregate by disregarding the data received from all malicious nodes and finally guarantee the correctness of the reconstructed aggregate based on the security of the MAC protocol and data-validity verification. Although the baseline scheme would satisfy the first three objectives mentioned above, it would do so at the cost of removing in-network data aggregation and its associated communication efficiency. For this reason, we do *not* consider the baseline scheme to be a useful solution. Nevertheless,

it constitutes a practical lower bound on the performance of any secure aggregation solution satisfying our three objectives above. That is, an efficient solution must have better performance than the baseline scheme; otherwise, the baseline scheme becomes preferable, and the entire notion of in-network data aggregation ceases to be useful, in hostile environments.

*Related Work: In-network Aggregation.* Chan et al. [3] propose a fully distributed aggregation verification algorithm, called the Secure Hierarchical In-network Aggregation (SHIA), which detects the existence of any misbehavior in the aggregation process. The algorithm perfectly satisfies its objective as a detection mechanism; however it is not intended to address our problem as it aims neither at the identification and removal of adversary nodes nor at providing continuous, but gracefully degraded, service under attack. Similarly, the work of Frikken and Dougherty [7], which improves the performance of SHIA, aims only at the detection of attacks against the aggregation process.

In contrast to SHIA, Hu and Evans [9] and Yang et al. [14] propose detection algorithms that also allow identification of corrupted nodes. However because both approaches use centralized verification, the incurred communication cost approaches that of the baseline scheme—$\mathcal{O}(n)$ for a network of size $n$—when in-network data aggregation ceases to be useful. In contrast, the cost of our scheme is logarithmic in $n$.

Another solution which uses a centralized approach is proposed by Haghani et al. [8] who extend SHIA. A corrupted node is detected via successive polling of the layers of a commitment tree (generated during the aggregation process) by the BS. Although this work is closest to ours in spirit, it differs in three fundamental ways. First, it incurs a high cost as it not only relies on centralized identification but also each run of the algorithm identifies only one malicious node at a time. In the worst case, to detect $c$ malicious nodes in a network of size $n$, $\mathcal{O}(nc)$ messages are generated per link. Second, the performance analysis and adversary model presented [8] does not include a comparison with the baseline scheme where identification of adversary nodes incurs a cost of only $\mathcal{O}(n)$. Hence, it is unclear at what point the proposed scheme ceases to be useful and the baseline scheme becomes preferable. Finally, Haghani *et al.* do not provide network service during the period of the attack.

*Related Work: Group Testing.* The identification of corrupted nodes is directly related to the problem of group testing, which strives to identify defective items of a given set through a sequence of tests. Each test is performed on a subset of all items and indicates whether the subset contains a defective item. In combinatorial group testing, it is assumed that the number of defectives in a set is constant. This number can be either known or unknown at the time of testing. Group testing is efficient when the number of defectives in a sample space is small compared to the total number of samples [5]. This is an analogous setup to untrusted sensor networks which are characterized as large, densely packed network of sensor nodes.

*Our Contributions.* We propose a *divide-and-conquer* approach to tracing and removing malicious nodes from the network which achieves the three objectives stated above. Briefly, our approach recursively (i) partitions suspicious subsets of the network, (ii) runs a given 'test' in each partition to check the correctness of the sub-aggregation values, (iii) if the result reveals possible node corruption, the set is tagged as suspicious; otherwise, it is considered to be good and the associated sub-aggregate value is retained. Hence, our algorithm allows for the incremental reconstruction of lost data from sub-aggregated value, over the course of its execution. The algorithm terminates when it has isolated all the malicious nodes in the network. The partition test is a primitive which we use in secure aggregation. The identification algorithm is designed and optimized with respect to the communication cost for an arbitrary number of malicious nodes. We prove the correctness of the algorithm and evaluate its performance using an analysis method inspired by the field of combinatorial group testing [5]. Our results illustrate the relationship between the efficiency of malicious-node identification and the number and distribution of these nodes. In particular, we define a precise cost-based threshold when in-network data aggregation ceases to be useful in hostile environments.

## 2 Preliminaries

*System Model.* Consider a multihop network of untrusted sensor nodes and a single trusted BS. The system administrator or user that resides outside the network interacts with the network through the BS interface. For brevity, subsequently we refer to any requests made by this external entity via the BS, as simply requests by the BS. We assume that each sensor has a unique identifier $v$ and a unique secret key shared with the BS, $K_v$. The sensor network continuously monitors its environment and measures some environmental data. We divide time into epochs; during each time epoch, the BS broadcasts a data request to the nodes in the network and nodes forward their data response back to the BS. Data can be forwarded individually or as an aggregate.

We model node corruption in the network as a function of the number $c$ and the distribution of the corrupted nodes. Each sensor node $v$ belongs either to the good set $G$ or the malicious or corrupted set $M$. A network instance is defined as $N = \{\forall v \text{ in network} : v \in G \vee v \in M\}$ where $|M| = c$ and $G = N \setminus M$. The collection of all $N$ for a given $c$, constitutes a family of networks $\mathcal{N}_c$.

For the purpose of computing the aggregate, we assume that the sensed environment (e.g., temperature) changes minimally with respect to the duration of the identification algorithm. This is a practical assumption as once malicious activity is detected, the identification algorithm is promptly executed. Moreover the algorithm terminates after a small, constant number of rounds.

*Adversary Model.* We assume that the network is deployed in an adversarial environment where the adversary can corrupt an arbitrary number of nodes. Once a node is corrupted, the adversary has total control over the secret data of the node as well as the subsequent behavior of the sensor node. We assume

that a corrupted node *persistently* misbehaves by inducing the BS to accept an 'illegal' value. An illegal value is defined based on the adversary objectives which is to induce the BS to accept a data value which is not already achievable by direct data injection.

A direct data injection attack occurs when an adversary modifies the data readings reported by the nodes under its direct control, under the constraint that only legal readings in $[r_1, r_2]$ are reported [3]. In the case of a single data values, this means that the data value transmitted is outside the legal reading of $[r_1, r_2]$. This is called a *false data injection attack*. In the case of data aggregation, the objective of the adversary is to tamper with the aggregation process such that the BS accepts an aggregation result which is not achievable by the direct data injection. We refer to this type of attack as a *false aggregation attack*. An aggregation protocol is considered secure if the adversary cannot successfully launch such an attack [3].

*Performance Measure.* We use link cost as a metric to analyze our algorithm. Link cost is defined as the total number of messages transmitted over a particular link in the network and is important as it determines how quickly nodes in the network exhaust their energy supply. Such nodes are often core to the connectivity or the functionality of the network and their loss can lead to network partitioning or denial of service.

## 3 Identification Algorithm

The main objective of our algorithm is to recursively isolate the malicious nodes in the network and thus render the adversary inoperative. The algorithm is initiated once misbehavior is detected in the network (e.g., via [3]) and is executed over a number of rounds, following an intuitive divide-and-conquer approach. In

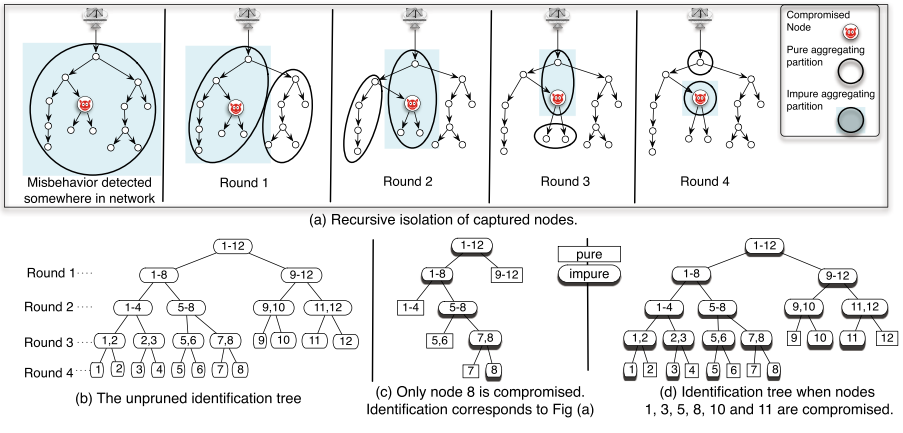



**Fig. 1.** Identification algorithm on an input of 12 nodes, $m = 2$

each round the algorithm partitions the suspicious subsets of the network and performs a partition test on the newly formed groups. The number of subsets a suspicious group is partitioned into is called the *partition degree*. The partition test consists of nodes aggregating their data and verifying the integrity of their aggregation process. The test has two outputs: 'pure' if all the nodes in the partition are good and 'impure' if there is at least one malicious node in the group. The algorithm terminates when there are no remaining impure groups.

By distributing the localization of the malicious nodes, the scheme simply keeps track of the lower bound on the number of malicious nodes in the network and increases the bound only when the findings of the scheme up to that point imply that this is valid.

**Algorithm 1.** ***Identification***
***Input:*** *All the nodes in the network $N \in \mathcal{N}_c$, partition degree $m > 1$, where integer $m$ is the number of partitions a group divides into in each iteration.*
***Output:*** *A result set $M$ of malicious nodes and a result set $G$ of good nodes, such that $M \cup G = N$*

*Let $t = 1$ be the lower bound on the number of malicious nodes in the network and $S = \cup_{i=1}^{t} S_i$ denote the current set of suspicious nodes, $S_1 = N$.*

*1. For $j = 1, \cdots, t$, BS requests partition $S_j$ to be divided into $m$ disjoint partitions (using partition rule viz. Algorithm 2). The collection of subdivided sets form the current collection $S$. Set $t$ to be the cardinality of set $S$.*

*2. For $j = 1, \cdots, t$, if $|S_j| > 1$, the nodes in partition $S_j$ partition themselves into groups of size $\frac{n}{m}$ and execute partition test. BS verifies the purity of each partition.*

*3. The BS learns the status of each node for the following round (details are provided in the next section). For $j = 1, \cdots, t$, if $S_j$ is pure (i.e., all the nodes are good), then $G = G \cup S_j$; else if $S_j$ is impure (i.e., there is at least one misbehaving node) and a singleton set, then $M = M \cup S_j$ and decrement $t$. Adjust the indices of the remaining sets, $\{S_j\}$ appropriately, to include only sets that are impure and non-singleton. If $t > 0$, go to step 1 (next round), else quit as all malicious nodes have been traced.*

We can model the divide-and-conquer approach of Algorithm 1 as the pruning process of an $m$-ary tree $T$ where each tree vertex is associated with a partition test. The root of tree $T$ is associated with the input set $N$ and each round $i$ is associated with level $(i + 1)$ of the tree. This is because the identification algorithm is initiated when misbehavior is detected in the network and therefore the test at level 1 has been already executed. If a partition $X$ is tested pure, then all the descendants of the associated vertex are pruned; otherwise the set $X$ is re-partitioned. Fig. 1(b) presents an unpruned identification tree for a network of 12 nodes and partition degree $m = 2$. Fig. 1(c) and (d) show how the tree can be pruned when the network contains one and six corrupted nodes respectively. Fig. 1(a) shows how the identification tree corresponds to the recursive isolation of the captured nodes on the physical network.

Next we define a novel partition rule inspired by Du and Hwang [4]. This algorithm partitions the network such that the identification tree contains at most one incomplete subtree. Intuitively a complete tree of $n$ nodes executes

less or equal number of tests than an incomplete tree of $n$ nodes as the complete tree contains less vertices (where each vertex corresponds to one test).

**Algorithm 2.** ***Partitioning Rule***
***Input:*** *Set $X$, maximum number of partitions $m$.*
***Output:*** *Result sets $\{X_i\}$, such that $\cup X_i = X$.*

*Let $i = 1$ denote the new subset $(X_i)$ to be determined.*
*1. Choose $X_i$ to contain $m^{\lceil \log_m |X| \rceil - 1}$ nodes.*
*2. Update set $X = X \setminus X_i$ to exclude the newly formed subset. If less than $m$ subsets are formed and $X$ has more than $m - 1$ nodes, then increment $i$ and go to Step 1.*
*Else if $X$ is not a singleton set, increment $i$ and add the remaining nodes in $X$ to $X_i$.*
*Else if $X$ is a singleton set, then $X$ cannot be partitioned anymore.*

### 3.1 Partition Test

The test that nodes perform in each newly formed partition is a fundamental step in our algorithm. There are two types of tests depending if the partition is a singleton or otherwise.

**Tests for Non-singleton Partitions.** In all non-singleton partitions (partitions containing more than one node), data is aggregated and the partition leader directly transmits the partition aggregate (via multi-hop) to the BS, which verifies the integrity of the aggregation process and hence the integrity of the nodes within that partition. In the general case, Algorithm 1 can be composed with any aggregation-verification algorithm that does not depend on a fixed partition and provides provable guarantees. Next we show how we can modify SHIA to satisfy these conditions.

SHIA extends the aggregate-commit-prove framework of [10]. In the aggregate-commit phase of the algorithm, a cryptographic commitment tree (hash tree) is built based on the sensor readings and the aggregation process. This forces the adversary to choose a fixed aggregation topology and set of aggregation results. In the prove phase of the algorithm, each sensor independently verifies that the final aggregate has incorporated its sensed reading correctly. Specifically each sensor reconstructs the commitment structure and ensures that the adversary has not modified or discarded the contributions of the node.

SHIA cannot be used as is because it assumes that the BS knows the exact set of nodes which are alive and reachable. Instead, we propose a new algorithm Group SHIA (GSHIA) which includes two additional properties. First, nodes can organize themselves into groups of size $g$, where $g$ is arbitrarily defined by the BS. This can be easily achieved as the 'delay aggregation' approach of SHIA develops an aggregation tree one node at a time. Since the root node of the aggregation tree knows the size of its subtree, it can declare a partition complete when it has $g$ nodes or it cannot add any more nodes to its partition.

In GSHIA, the BS can also verify the integrity of the aggregation process for a group of unknown size and membership set. This property can be implemented through the use of a Bloom filter [1] that summarizes the membership information of the partition. The BS then verifies the membership set by exhaustively

searching through the possible nodes. The change we propose places most of the membership resolution burden on the BS, which is generally assumed to be powerful. However we can reduce the computation burden by noting that Algorithm 1 is *nested* (i.e., each new partition is a proper subset of an older impure partition) and therefore the space of possible partitions in each round is reduced by a factor of $m$. Further improvements can be made if the BS knows the topology of the network a priori, using efficient schemes such as [11]. For protocol details as well as analysis and further improvement strategies, we refer the reader to [12].

An alternative approach to the above modification is to use the original SHIA algorithm and make the additional assumption that the BS knows the topology of the network prior to the detection period. The BS can then deterministically partition the network for a given $m$ and transmit this information to each sensor. When an impure group is detected, nodes divide themselves according to the specified partitioning. Although this method is simpler and more efficient, the additional assumption is not always practical as sensor networks often have dynamic topologies due to the short life span of the sensors.

**Tests for Singleton Partitions.** If a partition contains exactly one sensor node, the node $v$ transmits its measured data $x_v$ along with a MAC tag $\sigma_v$ computed using $K_v$. Upon receiving $\langle v, x_v, \sigma_v \rangle$, the BS verifies the tag and ensures that $x_v$ is valid. The BS assumes node $v$ has misbehaved if $x_v$ is not in the correct range but the tag verifies correctly.

### 3.2 Computing Aggregate

An important feature of our algorithm is that the network aggregate can tolerate malicious nodes and in fact, the aggregate degrades gracefully with the attack. In particular, dual to the intuition that the algorithm recursively isolates the corrupted nodes, is that the algorithm also increasingly identifies the uncorrupted nodes in the network. The BS can then use the data from the nodes determined to be uncorrupted to reconstruct the network service.

Recall our assumption that the sensed environment of the network does not change during the protocol execution. Thus we can improve the quality of the network aggregate in each successive round by incorporating the aggregates of newly found pure groups. Algorithm 3 shows how the aggregate is updated when the aggregation function is sum. We can easily extend this to other low-order statistics functions, such as min/max, averaging, etc.

**Algorithm 3.** ***Aggregate Update in Round*** $i$
***Input:*** *Aggregate* $\Psi_{i-1}$ *from round* $i-1$*, set* $\{\Psi[j]\}$ *of the aggregates of all pure partitions from round* $i$*.*
***Output:*** *Aggregate* $\Psi_i$ *of round* $i$*, where* $\Psi_i = \Psi_{i-1} + \sum_j \Psi[j]$*.*

### 3.3 Security Analysis

In the following, we first show the correctness of the proposed algorithm and in Section 4, we propose a mathematical framework to analyze the communication cost associated with providing our security solution.

**Theorem 1.** *Given an input set of nodes $N$ and partition degree $m$, Algorithm 1 outputs two resulting sets of corrupt nodes $M$ and of good nodes $G$, $M \cup G = N$.*

- *(Completeness) If corrupt node $v \in N$, then $v \in M$, i.e. no false negatives.*
- *(Soundness) If node $v \in M$, then $v$ is corrupt, i.e. no false positives.*

*Proof.* Let $T$ be the identification tree that Algorithm 1 generates. For any corrupted node $v \in N$, any vertex $u$ in $T$ which contains $v$, tests impure. This is because a corrupted node is persistently malicious and the partition test $t(\cdot)$ is perfect (i.e., the test result is always correct). Each impure vertex in $T$ is either divided into smaller partitions if it is a non-singleton set, or is added to the set $M$ if it is a singleton set. Since the algorithm converges when $t = 0$ or when there are no more impure non-singleton partitions, then by convergence time the algorithm must have found all corrupt nodes and added them to set $M$. Thus the algorithm is complete.

Additionally, the algorithm is sound since if node $v \in M$, then there exists a vertex $u$ in identification tree $T$ which is associated with a singleton set $\{v\}$ and that $\{v\}$ is impure. Thus $v$ must be malicious.

**Corollary 1.** *Algorithm 1 isolates all $c$ corrupt nodes within $\lceil \log_m |N| \rceil$ rounds.*

We refer the reader to [12] for details of the proof.

## 4 A Theoretical Model for Cost Analysis

In this section, we derive the cost associated with the security guarantees of the proposed protocol. We first formulate the communication cost in terms of an optimization problem. We then analytically solve this problem by introducing a novel mathematical framework, inspired by [5,6] and evaluate our results using an example network of 4096 nodes. For a complete analysis of the problem, finally we look at the best and average case cost of the system.

The link cost of the algorithm is a function of the number of partitions that are generated in each round (referred to as *partition* cost) as well as the aggregation-verification cost of each partition (referred to as the *test* cost of each partition). It is important to distinguish between the two costs because partition cost is characterized solely by the identification algorithm, whereas test cost is a function of the aggregation-verification primitive adopted and can be improved upon. We emphasize that the total cost derived in this section are based on the use of GSHIA as our primitive.

### 4.1 Cost Upper Bound Definition

Let $N$ be a network instance with $c$ corrupted nodes, $N \in \mathcal{N}_c$, input to the algorithm and let the algorithm terminate in $\tau = \lceil \log_m |N| \rceil$ rounds. Let $P(i, m, N)$ denote the number of partitions in round $i$ where each partition is of size

$T(j, m, N)$, $j = 1, \cdots, P(i, m, N)$. We formulate the total communication cost $G(m, N, c)$ of the algorithm as:

$$G(m, N, c) = \sum_{i=0}^{\tau} \sum_{j=1}^{P(i,m,N)} T(j, m, N) \tag{1}$$

*Worst case* cost of the algorithm is the maximum cost of the algorithm for all distributions of $c$ corrupt nodes in the network:

$$G(m, c) = \max_{N \in \mathcal{N}_c} C(m, N, c) \tag{2}$$

The optimization problem for the identification algorithm is defines as:

$$G(c) = \min_{m>1} G(m, c) \tag{3}$$

The parameters which achieve $G(c)$ are called the minimax parameters of the identification algorithm. *The goal of the network administrator is to find the minimax parameter $m$ for a given network $N$ without knowing the number of corrupt nodes $c$.*

In the following, we present some results relating $m$ with partition cost. This is of particular interest as our results can be applied to other divide-and-conquer algorithms. In fact the isolated problem of optimizing partition cost is equivalent to an instance of combinatorial group testing problem, where the number of defectives is unknown and we optimize the algorithm to minimize the number of tests performed. Inspired by group testing results for $m = 2$ [4], we extend the results for the general $m$-ary case. To the best of our knowledge this is the first time the $m$-ary case has been considered.

### 4.2 Results

**Upper Bound When $n$ is a Power of $m$.** We first prove the upper bound of the partition cost for different $m$-ary identification algorithms, where the number of nodes in the network $n$ is a power of $m$ and then compute the upper bound of the total cost of the identification scheme when $n$ is a power of $m$.

**Theorem 2.** *Let $n$ be a power of $m > 1$. Then for $c$ corrupted nodes in $n$ nodes, $1 \leq c \leq n$, the number of partitions generated is tightly upper bounded by* $\frac{m}{1-m} + mc(\log \frac{n}{c} + \frac{1}{m-1})$.

*Proof.* Let $T$ be the $m$-ary identification tree whose root vertex is associated with a set of size $n$, which is a power of $m$. According to the algorithm, every internal vertex must be associated with an impure set and there must exist exactly $c$ impure leaves. We sum up the total number of pure leaves in $T$ as follows. Let $u$ denote the height of tree $T$, $u = \log_m n$. Each level $i$ has $m^{i-1}$ vertices, where at most $c$ are impure. Level $v = \lceil \log c \rceil$ is the first level with at least $c$ vertices and let $w = v - \log c$. The total number of impure nodes $\gamma$ in $T$ is:

$$\gamma = \sum_{i=1}^{v} m^{i-1} + c(u - v + 1) = \frac{(1 - m^v)}{1 - m} + c(\log n - (w - \log c) + 1)$$

$$= \frac{1}{1-m} + c(\log \frac{n}{c} - w + 1 - \frac{m^w}{1-m}) \leq \frac{1}{1-m} + c(\log \frac{n}{c} + \frac{m}{m-1})$$

since $0 \leq w < 1$ and $f(w) = -w - \frac{m^w}{1-m}$ is a convex function. For $0 \leq w \leq 1$, $f(w)$ is maximized at $w = 0, 1$, where $f(0) = f(1) = \frac{1}{m-1}$. Thus there are at most $\gamma - c = \frac{1}{1-m} + c(\log \frac{n}{c} + \frac{1}{m-1})$ impure internal nodes in $T$. Each internal node has exactly $m$ children, so $T$ has at most $\frac{m}{1-m} + mc(\log \frac{n}{c} + \frac{1}{m-1})$ nodes.

**Theorem 3.** *Let $n$ be a power of $m > 1$. Then for $c$ corrupted nodes in the $n$ nodes, $1 \leq c \leq n/m$, the total cost $G(m, n, c)$ of the identification algorithm is upper bounded by $\sum_{i=1}^{u} H[i]$ where $H$ is a sequence of length $u = \lceil \log_m n \rceil$:*

$$H[i] = \begin{cases} m^{i-1}(\log \frac{n}{m^{i-1}} + 1) \text{ if } i < v \\ mc\ (\log \frac{n}{m^{i-1}} + 1), \text{ if } i \geq v \end{cases} \tag{4}$$

*where $v = \lceil \log_m c \rceil$ and* log *denotes* $\log_2$.

*Proof.* Let tree $T$ be the $m$-ary identification tree whose root vertex is associated with a set of size $n$. Let sequence element $H[i]$ represent the total cost of the identification algorithm in level $i$ of the identification tree $T$. Each level $i$ of $T$ has $m^{i-1}$ vertices, where at most $c$ are impure. Also each vertex at level $i$ has exactly $\frac{n}{m^{i-1}}$ nodes. Level $v$ of $T$ is the first level where $T$ has at least $c$ vertices. Therefore at level $i < v$, all $m^{i-1}$ vertices are impure. Since each test has a cost of at most $(\log p + m)$, where $p$ is the number of nodes tested, the total cost of each level $i < v$ is upper bounded by $m^{i-1}(\log \frac{n}{m^{i-1}} + m)$. Now consider level $i \geq v$. Then each level has at most $mc$ impure nodes of size $m^{i-1}$. Therefore total cost of each level $i \geq v$ is upper bounded by $mc(\log \frac{n}{m^{i-1}} + m)$.

**Upper bound when $n$ is not a power of $m$.** In the general case when $n$ is not a power of $m$, we cannot use the approach of [4] (solved for $m = 2$) as the number of possible ways the corrupted nodes are distributed within each subtree explodes (analogous to the combinatorial, ball in the bucket problem). Instead we propose a novel model, inspired by the work of Fiat and Tassa [6] in the context of dynamic traitor tracing (DTT)[1]. We introduce the notion of a *path trace*, defined with respect to a particular corrupted node. The path trace traces the identification path of that node in the identification tree $T$. Informally we say a path trace $D$ for corrupted node $u$ is rooted at the vertex $v$ in tree $T$ that the identification algorithm separates it from the other corrupted nodes in the network. The trace includes all the vertices in the path between $v$ and the leaf vertex associated with set $\{u\}$. Therefore each time an impure vertex $v'$ in $T$ has more than one impure child, then the algorithm learns that the node set at $v'$ contained more than one corrupted node, and thus a new tree trace $D'$ is generated.

Fig. 2 shows the paths generated for an example identification tree. Note that although a path trace is not unique to a given node, the set of path traces generated is unique. We can therefore determine the set of path traces in a network without associating them to a particular corrupted node.

**Lemma 1.** *A path trace of length $\ell$ generates $m\ell$ partitions where there are $m$ partitions of sizes $\{m^{\ell-i}\}$, $i = 1, \cdots, \ell$.*

[1] The DTT model differs to ours as in each of its rounds, only one node misbehaves.

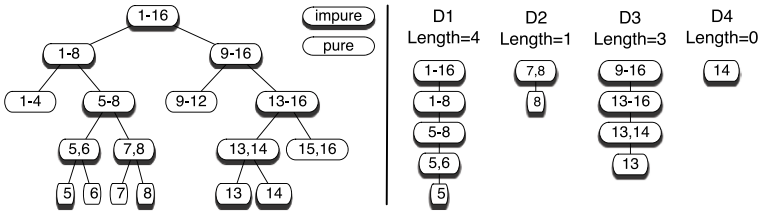



**Fig. 2.** Path traces for corrupted nodes 5,8,13 and 14

*Proof.* The path trace is a path on an $m$-ary tree and each internal vertex on the path has $(m-1)$ other siblings that are also tested. Also a path trace of length $\ell$ has a root vertex associated with $m^\ell$ nodes. Thus at level $i$ of the path, the vertex is associated with $m^{\ell-i}$ nodes.

Consider identification tree $T$ generated by Algorithm 1, for $n$ nodes.

**Lemma 2.** *Let $n = m^h$ where $h \in \mathbb{Z}^+$. Then define sequence $P$ as:*

$$P = \{h, \{h-1\}^{m-1}, \{h-2\}^{m(m-1)}, \{h-3\}^{m^2(m-1)}, \cdots\} \tag{5}$$

*where $\{y\}^x$ denotes the value $y$ repeated $x$ times. The first $c$ elements in $P$ represent the tight upper bound on the length of the path traces generated when $n$ contains $c$ corrupted nodes.*

*Proof.* Since $n$ contains at least one corrupted node, the first path trace $D_1$ is rooted at the root of $T$ and thus has length $h$. A new path trace is generated any time a vertex in $T$ contains more than one impure child. To find the upper bound on the length of the path traces, each trace should be generated in as early a round as possible. On level 2 of $T$ (round 1), up to $(m-1)$ path traces can be generated of length $(h-1)$; at level 3, up to $m(m-1)$ path traces can be generated of length $(h-2)$, and so on. In general, in level $i$, up to $m^i(m-1)$ path traces of length $(\log_m n - (i-1))$ can be generated. Since one path trace is associated with each corrupted node and there are $c$ corrupted nodes, the set of lengths associated with the generated path traces can be represented by the first $c$ elements of $P$.

**Theorem 4.** *Let $m^{h-1} < n < m^h$ where $h \in \mathbb{Z}^+$. Then define sequence $P'$ as:*

$$P'[i] = \begin{cases} P[i] & \text{if} \quad i < x \\ P[i] - 1 & \text{if} \quad (i > x \;\&\; P[i] > 0) \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

*where $P$ is the sequence defined in Equation 5 and the index $x = \lceil \frac{n - m^{h-1}}{m-1} \rceil$. The first $c$ elements in $P'$ represent the tight upper bound on the length of the path traces generated when $n$ contains $c$ corrupted nodes.*

*Proof.* It is trivial to show that the number of internal nodes at level $h-1$ is defined by $x$. Then there are $(m^{h-1}-x)$ leaves in level $(h-1)$ and $(n-m^{h-1}+x)$ leaves in level $h$. To find the upper bound of the lengths of the path traces, $\min(x, c)$ of the path traces are associated with a corrupted node at level $(h+1)$ and thus they correspond to the identification tree for $m^h$ nodes. The path traces corresponding to the remaining corrupted nodes have leaves at level $h$ and correspond to a identification tree for $m^{h-1}$ nodes.

We can use Theorem 4 to derive the tight upper bound on the total cost of the identification algorithm. Consider identification tree $T$ generated by algorithm 2, for a network of $n$ nodes. Let $T$ contain $\alpha$ complete $m$-ary trees and one incomplete $m$-ary tree, with respective depths $d_1, \cdots, d_{\alpha+1}$. Let $P_1, \cdots, P_{\alpha+1}$ correspond to the set of potential path traces for each of the $(\alpha+1)$ respective subtrees using Lemmas 1 and 2. Then let sequence $P$ be composed of the non-increasing ordered set of the path traces $\{P_i, \cdots, P_{\alpha+1}\}$, i.e. $P = \{h, (h-1)^{a_1}, (h-2)^{a_2}, \cdots\}$, where $a_1, a_2, \cdots$ are dependent on the size of the subtrees. If $n$ contains $c$ corrupted nodes, then the length of the generated path traces are bounded by the first $c$ elements in $P$. We can use Lemma 1 and sequence $P$ to compute the size and number of the partitions that Algorithm 2 generates in the worst case and derive total cost by summing the cost of the $c$ path traces.

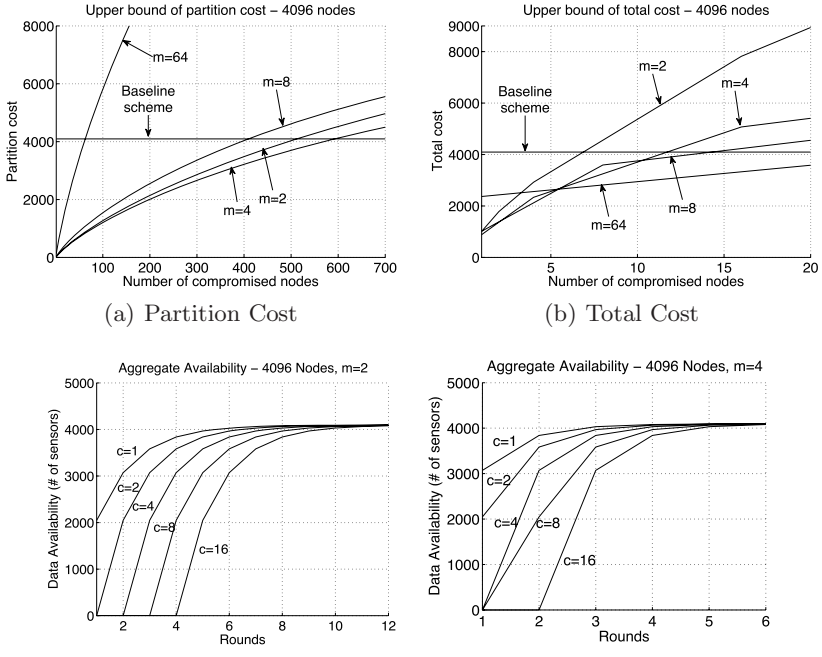
Finally we derive a closed form expression for the *loose* upper bound on the total cost of the network. This is purely for the purposes of comparison of our work with existing solutions. We refer the reader to [12] for details of the proof.

**Theorem 5.** *For $c$ corrupted nodes in a network of size $n$, the identification algorithm has a communication link cost of $O(c^2 \log^3 n)$.*

### 4.3 An Example

To gain a better intuition of the results, we compute the cost associated with handling an adversary attack in a network of 4096 nodes (where $c$ nodes are compromised) and analyze the graceful degradation of the network service. For test cost, we use the cost derived by [3] as the more efficient bound of [7] is not a fixed cost characteristic.

Fig. 3(a) and 3(b) graph the maximum partition cost and total cost of the $m$-ary identification scheme, for different $m$. The baseline scheme is used in the graphs as a lower bound for when the proposed identification scheme is effective and efficient. Fig. 3(a) verifies the intuition that for a fixed number of corrupted nodes in the network $c$, the number of generated partitions increases with the partition degree $m$. This increase plateaus when the the identification scheme needs to test every single vertex on the identification tree. The performance of the $m$-ary identification scheme is best shown in Fig. 3(b) where the link cost for different $m$ is compared with the baseline. It is clear that to optimize total cost, a network administrator choose an appropriate partition degree depending




(a) Partition Cost (b) Total Cost

(c) Aggregate Availability, m=2 (d) Aggregate Availability, m=4

**Fig. 3.**

on probability of attack, vulnerability of the network as well as the necessary rapidity of the response (as response time is $\mathcal{O}(\log_m n)$). Fig. 3(b) also shows that test cost is the dominant term in total cost. This is promising as test cost is only dependent on the cost of the aggregation-verification primitive. More efficient primitives yield better results.

Fig. 3(d) and 3(d) shows the rate of improvement of the network service over the course of identification. Data is normalized by only looking at the number of nodes that contribute to the aggregate in a particular round. The maximum available data for a network of size $n$ with $c$ corrupted nodes, is $n-c$. In particular we note that if we fix $c$, as $m$ is reduced, data becomes available in later rounds. This is because in the worst case, the first $c$ partitions generated are corrupt.

## 5 Conclusion

Adversary attacks against data aggregation in ad hoc networks can have disastrous results, whereby a single corrupted node can affect the perceived measurements of large portions of the network by the BS. Current approaches to handling such attacks either aim exclusively at the detection of attack or provide inefficient

ways of identifying corrupted nodes in the network, with respect to the baseline scheme; i.e., it is more efficient if sensor data is not aggregated at all. In this work, we presented a group-based approach to handling adversary attacks in aggregation applications, that identifies corrupted nodes while ensuring continuous, but gracefully degraded service during the attack period. Our analysis results in a precise cost-base characterization of when in-network aggregation retains its assumed benefits in a sensor network operating under persistent attacks. Our scheme is most effective when the adversary has corrupted a small fraction of the nodes in the network. Although our work provides promising results in divide-and-conquer handling of attacks in aggregation applications, we have assumed a simplified adversary model. In the future, we plan on generalizing our model to account for non-persistent adversaries as well as allowing for identification error to decrease identification efficiency.

## References

1. Bloom, B.: Space/time trade-offs in hash coding with allowable errors. Communications of the ACM 13(7), 422–426 (1970)
2. Chan, H., Gligor, V.D., Perrig, A., Muralidharan, G.: On the distribution and revocation of cryptographic keys in sensor networks. IEEE Transactions on Dependable and Secure Computing 2(3), 233–247 (2005)
3. Chan, H., Perrig, A., Song, D.: Secure hierarchical in-network aggregation in sensor networks. In: Proceedings of the 13th ACM Conference on Computer and Communications Security, pp. 278–287 (2006)
4. Du, D.-Z., Hwang, F.K.: Competitive group testing. Discrete Applied Mathematics 45(3), 221–232 (1993)
5. Du, D.-Z., Hwang, F.K.: Combinatorial Group Testing and Its Applications, 2nd edn. On Applied Mathematics, vol. 12. World Scientific, Singapore (2000)
6. Fiat, A., Tassa, T.: Dynamic traitor tracing. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666. Springer, Heidelberg (1999)
7. Frikken, K., Dougherty IV, J.A.: Efficient integrity-preserving scheme for hierarchical sensor aggregation. In: Proceedings of the 1st ACM conference on Wireless Network Security (2008)
8. Haghani, P., Papadimitratos, P., Poturalski, M., Aberer, K., Hubaux, J.-P.: Efficient and robust secure aggregation for sensor networks. In: 3rd IEEE Workshop on Secure Network Protocols, October 2007, pp. 1–6 (2007)
9. Hu, L., Evans, D.: Secure aggregation for wireless networks. In: Workshop on Security and Assurance in Ad Hoc Networks, pp. 384–392 (2003)
10. Przydatek, B., Song, D., Perrig, A.: SIA: Secure information aggregation in sensor networks. In: Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (2003)
11. Staddon, J., Balfanz, D., Durfee, G.: Efficient tracing of failed nodes in sensor networks. In: Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications, pp. 122–130 (2002)

12. Taban, G., Gligor, V.D.: Efficient handling of adversary attacks in aggregation applications. Technical Report TR 2008-11, Institute of Systems Research (2008)
13. Wagner, D.: Resilient aggregation in sensor networks. In: Proceedings of the 2nd ACM Workshop on Security of Ad Hoc and Sensor Networks (2004)
14. Yang, Y., Wang, X., Zhu, S., Cao, G.: SDAP: A secure hop-by-hop data aggregation protocol for sensor networks. In: ACM International Symposium on Mobile Ad Hoc Networking and Computing, pp. 356–367 (2006)