Secure Aggregation with Malicious Node Revocation in Sensor Networks

Binbin Chen
Department of Computer Science
National University of Singapore
Email: chenbinb@comp.nus.edu.sg

Haifeng Yu
Department of Computer Science
National University of Singapore
Email: haifeng@comp.nus.edu.sg

***Abstract*—Sensor applications often leverage *in-network aggregation* to extract aggregates, such as *predicate count* and *average*, from the network. With in-network aggregation, a malicious sensor can easily manipulate the intermediate aggregation results and corrupt the final answer. Most existing secure aggregation schemes aim to defend against *stealth attacks* and can only raise an alarm when the final answer is corrupted, without being able to pinpoint and revoke the malicious sensors. While some recent protocols can pinpoint and revoke malicious sensors, they need to rely on expensive public key cryptography to be robust against certain attacks.**

Using only symmetric key cryptography, this paper aims to strictly diminish the capability of adversaries whenever they launch a successful attack, so that malicious sensors can only ruin the aggregation result for a small number of times before they are fully revoked. To this end, we propose VMAT (verifiable minimum with audit trail), a novel secure aggregation protocol with malicious sensor revocation capability. VMAT relies on symmetric key cryptography only, and provides provable guarantees that each execution can either produce the correct aggregation result efficiently, or revoke some key held by the adversary.

I. Introduction

Wireless sensor networks have seen numerous interesting applications in recent years, ranging from battlefield monitoring to emergency response. These sensor applications often leverage *in-network aggregation* to extract aggregate information, such as *predicate count* and *average*, from the network. Instead of blindly forwarding others' packets, each individual sensor peeks into the packets passing through it, and generates more compact records by combining its own reading with other relevant data, whenever possible [15]. Only forwarding such aggregated record can significantly improve performance and reduce sensors' power consumption. Unfortunately, naive in-network aggregation can easily be attacked in a hostile environment, where sensors may potentially be compromised due to physical tampering and become *malicious*. These malicious sensors can easily modify the intermediate aggregation results.

Previous results. Early efforts [2], [6], [12], [14], [16] on secure aggregation often tackle the problem under simplified settings that can either cope with only a single malicious sensor or support only single-level aggregation. These strong assumptions often do not hold in modern sensor networks. Later research efforts, such as the secure hierarchical in-network aggregation (SHIA) protocol [3] and its enhanced variation [9], as well as the SECOA secure aggregation protocol [19], have started to consider multi-level in-network aggregation setting where more than one sensor may be compromised. All these protocols [3], [9], [19] aim to deal with *stealth attacks*, where the adversary's goal is to influence the aggregation result without being detected. Consistent with such a goal, these protocols can only raise an alarm when the aggregation result is corrupted, without being able to pinpoint malicious sensors and diminish their attacking capability. As a result, even a single malicious sensor can keep failing the final result verification without exposing itself.

More recently, SHIA has been extended [11], [24] to allow pinpointing and revocation of malicious sensors. These approaches will need to rely on public key cryptography to be robust against *choking attacks* [5], [20], [25], [29], where malicious sensors inject spurious messages to "choke" the relaying capacity of other sensors and stall the propagation of desirable messages. On the other hand, given sensors' limited battery power and computational capacity, it is often desirable to avoid using public key cryptography. If the sensors cannot support expensive public key cryptography, then a single malicious sensor can still keep corrupting the final result without being revoked in these protocols. This can be rather serious, since the entire sensor network is effectively brought down by just a single malicious sensor.

To address this problem, Yu [29] proposes an alternative approach that requires symmetric key cryptography only and uses sampling (instead of in-network aggregation) to answer aggregation queries. This sampling-based protocol can *tolerate* malicious sensors in the sense that the malicious sensors can never prevent the system from generating a correct answer. Since malicious sensors are already tolerated, there is no need for pinpointing and revocation any more. Unfortunately, this protocol requires $\Omega(\log n)$

sequential *flooding rounds*, where n is the number of sensors and a flooding round is the time required for the base station to flood the entire sensor network. This clearly incurs much larger delay than typical in-network aggregation approaches that only need $O(1)$ flooding rounds.

Our results. This paper proposes a novel secure aggregation protocol called *VMAT* (*verifiable minimum with audit trail*), using only symmetric key cryptography. Each execution of VMAT incurs only $O(1)$ flooding rounds, and either produces the correct aggregation result or records proper (distributed) audit information. Based on the audit information, VMAT can always pinpoint and revoke at least one key held by the malicious sensors. Clearly, with such pinpointing and revocation functionality, malicious sensors can no longer stall the system forever. Compared to [29], VMAT incurs only $O(1)$ (instead of logarithmic) flooding rounds to produce aggregation result or generate proper audit trails.

While it may appear counter-intuitive, pinpointing and revocation are non-trivial in sensor networks, especially without public key cryptography. The lack of public key cryptography enables the malicious sensors to easily interfere with the pinpointing process (e.g., via choking attacks). Clearly, the pinpointing process must tolerate such malicious interference, instead of resorting to further pinpointing. To the best of our knowledge, VMAT is the first secure aggregation protocol that is able to pinpoint and revoke malicious nodes, even under choking attacks, without using any public key cryptography.

VMAT integrates several complementary techniques to achieve its end goal:

- Perhaps counter-intuitively, sensors in VMAT use timestamp instead of hop count to determine their levels on the aggregation tree. This novel design prevents malicious sensors from increasing the height of the resulting tree by manipulating the hop count.
- Similar to some previous efforts [2], [19], VMAT transforms COUNT query, SUM query, and AVERAGE query to MIN query. By broadcasting the minimum result from the MIN query back to the sensors, any silent drop of the true minimum reading will always trigger a *veto* from the sensor with that true minimum reading. Different from [2], [19], VMAT further maintains proper (distributed) audit information so that a pinpointing protocol can later pinpoint the culprit of the silent drop (if any).
- To further defend against choking attacks where malicious sensors inject spurious veto to stall the propagation of legitimate veto, we design a simple and elegant *Slotted One-time Flooding with Audit Trail (SOF)* protocol to propagate the veto. The protocol ensures that either a legitimate veto reaches the base station or the system records proper (distributed) audit information that can later be used to pinpoint the source of the choking attack. Choking attacks are also possible during the pinpointing process itself. We adopt an existing keyed predicate test protocol [29] to prevent choking attacks during pinpointing.
- Without public key cryptography, VMAT can only pinpoint and revoke the symmetric keys held by the malicious sensors that they use for communication with their neighbors. Since each malicious sensor may hold a large number (e.g., 250) of such keys, sequential revocation can be prohibitively expensive. VMAT instead will try to uniquely pinpoint a malicious sensor after just revoking a small number of its symmetric keys. We show that this can often reduce the number of keys that need to be individually revoked by over 90%.

II. Related Work

Section I has already discussed some related aggregation protocols [2], [3], [6], [9], [11], [12], [14], [16], [19], [24]. Among these protocols, the protocol of Haghani et al. [11] and the protocol of Taban and Gligor [24] are perhaps the most related to ours. Haghani et al. explicitly use public key cryptography in order to avoid choking attacks. Taban and Gligor's protocol requires certain sensors (partition leaders) to propagate (via multi-hop) the partial aggregate results to the base station. Without public key cryptography, malicious sensors can easily launch choking attack here to stall the propagation of such partial aggregate results. In addition to not relying on public key cryptography, our VMAT protocol offers other nice properties as compared to above protocols [11], [24]. For example, [11] requires the base station to know the entire sensor network topology while VMAT does not require such global knowledge. [24] imposes non-trivial restriction on the adversary and assumes that a malicious sensor *persistently* misbehaves, while we allow malicious sensors to behave arbitrarily and adaptively in a Byzantine way.

Other than these protocols, Roy et al. [23] and Garofalakis et al. [10] use verifiable FM synopses [8] for secure aggregation. In these protocols, a sensor generates a MAC [23] or a public key digital signatures [10] to vouch for the bit that it sets in the FM synopses. Pinpointing and revocation would be trivial with public key signatures. If one only uses symmetric key cryptography as in [23], however, the MACs will only be verifiable at the base station and not at the intermediate sensors forwarding the message. This makes the design vulnerable to choking attacks [5], [20], [25], [29]. More specifically, since the forwarding sensors do not have the needed information to verify the MACs of these spurious messages, they have to forward all of them to the base station. This in turn, can stall the propagation of legitimate messages. The protocol in [23] unfortunately does not provide mechanisms to either prevent such choking attacks or pinpoint the source.

Some researchers have also proposed various secure aggregation heuristics without end-to-end guarantees. For example, the SDAP secure aggregation protocol [27] assumes certain correlation among the readings of the sensors and performs anomaly detection directly by comparing multiple independent aggregation results. In comparison, our VMAT protocol does not require such assumption.

The pinpointing process in VMAT can be viewed as a secure "traceroute" protocol for sensor networks. Similar need for pinpointing also arises in secure sensor network routing [4], [22], [26], but none of these efforts provide full details on how to do so. Secure traceroute [21] has also been studied in the Internet under a different set of assumptions (e.g., direct point-to-point communication). Finally, a preliminary version of this work appeared as a Brief Announcement [28], which provides a sketch of the VMAT design without details or proofs.

III. System Model and Attack Model

System Model. We consider a sensor network with n wireless sensors. The sensor network is multi-hop in the sense that not all sensors can directly communicate with the base station. The base station is trusted. To unify terminology, we sometimes call the base station as a sensor as well. We assume that messages are reliable, after proper retransmissions if necessary.[*] We assume that the sensors and the base station have loosely synchronized clocks with bounded clock errors, which is a common assumption for in-network aggregation [2], [15], [18].

To avoid the (prohibitive) overhead of using public key cryptography on sensors, VMAT will only use symmetric key cryptography. In VMAT, each sensor shares a unique symmetric key (called *sensor key*) with the base station. For pair-wise authentication among sensors, we assume that the sensors use the simple key pre-distribution scheme by Eschenauer et al. [7], though VMAT also works with other schemes [1]. In Eschenauer et al.'s approach, each sensor is loaded with r keys (called its *key ring*) drawn uniformly randomly from a global pool of u symmetric keys. The value of r usually satisfies $r < n$, since otherwise it would be better for each sensor to hold a distinct key for every other sensor. When r reaches $c\sqrt{u}$, by the Birthday Paradox, any two sensors will have a common key with probability at least $1 - e^{-c^2}$. We call such a common key between two neighboring sensors as an *edge key*, since it corresponds to an edge in the sensor network topology. Every message in VMAT carries a MAC (called the *edge MAC*) created using the corresponding edge key. When forwarding a message to all its neighbors, a sensor will need to send different edge MACs individually to different neighbors. Edge MACs only provide limited authentication since more than one sensor may have a given edge key. To simplify discussion, we will not explicitly mention these edge MACs in the remainder of this paper.

[*]Since VMAT supports synopsis-diffusion style [18] multi-path aggregation, we expect the effect of message losses to be minimum.

Attack model. The adversary can see all messages in the network, and may further send messages to any sensor. We assume that the adversary compromises up to f sensors in the network, and these sensors become *malicious*. Adversary learns all the sensor keys of these malicious sensors, as well as the keys in their key rings. If a sensor is destroyed or radio-jammed by the adversary, we also pessimistically consider it as being malicious. In addition to modifying the partial results passing through them, the malicious sensors may further launch DoS-related attacks such as choking attacks [5], [20], [25], [29]. In a choking attack, the malicious sensors inject many spurious messages to "choke" the limited forwarding capacity of the network and stall the propagation of legitimate messages. Notice that without public key cryptography, the honest sensors cannot easily authenticate the origin of these messages and thus rate limiting will not help.

We assume that the malicious sensors do not partition the sensor network. Notice that it is fundamentally impossible to incorporate the readings from those sensors that are partitioned away. If the malicious sensors indeed partition the sensor network, then VMAT will simply compute an aggregate for those sensors that are in the same connected component as the base station. We assume that VMAT knows a rough upper bound on the *depth* (denoted as L) of the sensor network (excluding all malicious sensors). We define the *depth* of a sensor to be the length of the shortest path from that sensor to the base station. The *depth* of the sensor network is simply the maximum depth of all individual sensors. Finally, we define a *flooding round* as the amount of time required for the base station to flood the entire sensor network.

Aggregation query and approximation answer. Similar to most previous efforts [2], [3], [9], [10], [19], [23], [29], we focus on simple aggregation queries such as COUNT query (i.e., counting the number of sensors whose reading satisfies a certain predicate), SUM query, and AVERAGE query. VMAT produces approximate answers to these queries, with provable approximation error. Notice that since sensor readings inherently carry measurement errors and sensor failures can be common, even state-of-art aggregation algorithms [18] for *trusted* environments do not aim for exact answers. We use the standard notion of (ϵ, δ)-*approximation* to quantify approximation error. An (ϵ, δ)-*approximation* answer is guaranteed to be within $(1 \pm \epsilon)$ multiplicative factor of the *correct* answer with probability at least $1 - \delta$, where the probability is taken over the random coin flips in our randomized protocol. Same as the definition in most previous efforts [2], [3], [9], [10], [19], [23], [29], an answer is considered *correct* as long as the malicious sensors do not add additional fabricated readings or drop/change the reported readings

```
1. form an aggregation tree;
2. wait for the minimum in aggregation phase;
3. *if* (no minimum received)
       set minimum to be $\infty$;
4. *if* (get spurious minimum)
       invoke **junk-triggered pinpointing/revocation**; return;
5. broadcast the minimum, wait for veto;
6. *if* (no veto)
       return the minimum as correct result;
7. *if* (get spurious veto)
       invoke **junk-triggered pinpointing/revocation**; return;
8. *if* (get legitimate veto)
       invoke **veto-triggered pinpointing/revocation**; return;
```

Figure 1. VMAT protocol overview.

of honest sensors (i.e., they do not interfere with the aggregation process). The secure aggregation problem does not aim to prevent malicious sensors from reporting arbitrary readings on their own behalves.

IV. VMAT: Tree Formation, Aggregation, and Confirmation

To facilitate understanding, we will first describe the VMAT protocol for answering a MIN query that simply asks for the minimum reading among all sensors. Note that MIN query itself is not a robust aggregate query, in the sense that a single malicious sensor can change the final result by just modifying its own reading. Section VIII later will explain how more complex robust aggregate queries, such as COUNT query and SUM query, can be converted to MIN query while preserving their robustness. MIN query has the nice property that the result of this query (i.e., the minimum value) is generated by a single sensor, which can easily create a MAC to vouch for it. It also allows each sensor to independently check the correctness of the final result by comparing the final result to its own reading.

Overview. Figure 1 gives an overview of VMAT. VMAT first forms an aggregation tree, and then uses in-network aggregation to find out the minimum value among all sensors. The next confirmation phase serves to confirm that the true minimum value was not silently dropped by the malicious sensors during aggregation. To do so, the base station broadcasts the minimum value received and waits for potential *vetoes*. Audit trails are maintained in both the aggregation and the confirmation phase. VMAT returns a correct minimum value if and only if neither veto nor any spurious message is received in the two phases. Otherwise, VMAT invokes *veto-triggered pinpointing/revocation* or *junk-triggered pinpointing/revocation*, which will be described in Section VI.

A. Tree Formation Phase

Naive tree formation and aggregation. To better understand our design choices, we first briefly review traditional approaches [15] for forming an aggregate tree and doing in-network aggregation. The process begins with the base station broadcasts a tree-formation message containing a hop count of 0. Any sensor A receiving this message for the first time increases the hop count in the message and then forwards the message. Simultaneously, A sets its own *level* to be the hop count in the message that A sends out, and sets its *parent* as the sensor from which A received the message. Eventually, this tree-formation message will flood the entire network, and every sensor will have a level and a parent.

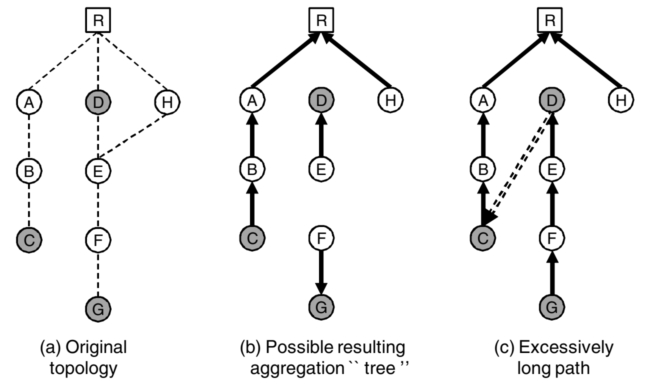

R
A
D
H
B
E
C
F
G
(a) Original topology
(b) Possible resulting aggregation `` tree ''
(c) Excessively long path

Figure 2. Possible attack on tree formation. R is the base station, C, D, and G are malicious sensors, while the rest are honest.

A sensor uses its level to determine when it should send a partial aggregation result to its parent. Specifically, the aggregation phase is divided into L intervals. A sensor at level i waits for messages from its children until the $(L-i)$th interval, and then sends an aggregated message to its own parent in the $(L-i+1)$th interval.

Sensors determine the current interval number based on their local clocks. Here accounting for bounded clock errors is simple. Let Δ denote the maximum clock error between any two honest sensors. When a sensor needs to send a message in an interval, it will avoid sending that message in the initial/final Δ time of the interval (according to its own clock). This is sufficient to ensure that the receiving sensor's clock is in that interval. This well-known technique applies in all our discussions later, and thus we do not discuss clock error any more.

Attacks on tree formation. Malicious sensors can easily interfere with the above traditional tree formation process. A malicious sensor may select multiple sensors as parents, or attract a large number of children. Thus the actual topology formed may not be a tree or may not even be connected (Figure 2(a) and 2(b)). However, if a malicious sensor prevents the true minimum value from being propagated to the base station, it can be properly pinpointed as described later.

A more subtle attack is for malicious sensors to manipulate the hop count to prevent honest sensors from getting a valid level in tree-formation. For example in Figure 2(a), the depth of the original topology (without the malicious sensors) is $L = 3$. However, during tree formation (Figure 2(c)), C and D establish a wormhole link [13] and concatenate two legitimate paths together, so that the concatenated path (with a length of 7) is longer than L. As a sensor with level greater than L cannot determine a valid transmission interval, this attack can fail the aggregation phase. Worse still, it is impossible to tell which sensors caused the long path to form.

Tree formation in VMAT. To address this problem, our design determines the level via timestamps instead of via hop counts. First, prior to the tree formation phase, the base station announces when the tree formation will start, using existing authenticated broadcast protocols such as [20]. This authenticated broadcast message serves to prevent DoS attacks where the malicious sensors keep initiating tree formation. At the specified starting time, the base station broadcasts the tree-formation message. VMAT divides the tree formation phase into L intervals. Sensors receiving the message in the first interval set their levels to 1. Instead of forwarding this tree-formation message immediately, they hold the message and forward it only in the next (i.e. second) interval, so that their children set their levels to 2. The process continues until all sensors set their levels. As the network depth is L, honest sensors that are not partitioned by malicious sensors are guaranteed to receive the tree-formation message by the Lth interval and will set a valid level. Any tree-formation message received after the Lth interval is ignored.

B. Aggregation Phase

Prior to the actual aggregation, the base station first uses authenticated broadcast to announce the query, the aggregation starting time, and a fresh nonce. When the aggregation starts, each sensor creates a message $\langle id, v, \text{MAC}_{id}(v||nonce)\rangle$, where id is the sensor's ID, v is its reading, and the MAC_{id} is generated using the sensor key (notice this is not the edge MAC). An ith level sensor waits until the $(L-i)$th interval to collect all the messages from its children (if any). It then selects and forwards the message with the smallest v among all these messages from its children and itself in the $(L-i+1)$th interval. The sensor further stores this forwarded message in the following tuple form:

⟨*level, message, sensor key, in-edge key, out-edge key*⟩

Here *level* is the level of the sensor, *message* is the forwarded message, *sensor key* is the sensor key of the sensor, and *in/out-edge key* is the edge key used to receive/forward the message. Some information in this tuple does not actually need to be explicitly recorded, but we include that in the tuple to simplify our discussion later. All these tuples stored at different sensors in the system comprise the distributed audit trails.

Malicious sensors may inject spurious messages (i.e., a minimum value without valid MAC_{id}) during this aggregation phase. For such attacks, the source of a spurious message can be pinpointed as later explained in Section VI. Malicious sensors can also launch *dropping attacks* to silently drop the true minimum value during aggregation phase. VMAT uses the confirmation phase as presented next to detect such dropping attacks, and then pinpoints the malicious sensor based on the audit trail recorded in this aggregation phase (as later explained in Section VI).

C. Confirmation Phase

If the final message containing the minimum value holds a valid MAC, the base station will use authenticated broadcast to announce the minimum value received, the confirmation phase starting time, and a fresh nonce. When the confirmation phase starts, any sensor with its own reading v smaller than the broadcast value is a *vetoer* with $\langle id, v, level, \text{MAC}_{id}(v||level||nonce)\rangle$ being the *veto* message. Here id is the sensor's ID, and $level$ is the sensor's level during the aggregation phase.

SOF protocol. Vetoes are propagated back using a *Slotted One-time Flooding with Audit Trail (SOF)* protocol. The basic idea in SOF is to let each vetoer flood its veto back toward the base station (recall that the honest sensors and the base station form a connected component). However, if the malicious sensors dropped many values during the aggregation phase, there can be numerous vetoers. Propagating all these vetoes back is neither feasible nor necessary: One legitimate veto is already sufficient to show that the minimum result may not be correct, and can trigger pinpointing. Thus our flooding protocol is *one-time* in the sense that a vetoer sends to all its neighbors the veto message, while a non-vetoer forwards to all its neighbors the *first* veto it receives and *ignores all other messages*. The sensor from which a non-vetoer A receives the first message is called A's parent in confirmation phase.

For similar reasons as in tree formation (Figure 2(c)), with the disruption from malicious sensors, the number of sensors that a veto traverses in the flooding protocol can be much larger than L. This is undesirable since it will result in excessively long audit trails, making later pinpointing inefficient. To avoid this problem, VMAT uses a *slotted* version of the flooding protocol. Specifically, the confirmation phase is partitioned into L intervals. All vetoers send out their vetoes in the first interval. If the first veto received by a non-vetoer A is received during the ith interval, then A will forward this veto during the $(i+1)$th interval. (A still ignores all future vetoes received.) This will elegantly ensure that the length of the audit trail is at most $L+1$ (including the tuple at the base station), without compromising any other desirable properties. We will present proofs later.

In the SOF protocol, each sensor (vetoer or non-vetoer) further records the veto message that it sends/forwards in the following tuple form:

⟨*interval, message, sensor key, in-edge key, out-edge key*⟩

Here *interval* is the index of the interval when the message is sent/forwarded. All other fields are similar to the fields in the tuple recorded during the aggregation phase.

Properties of SOF. Obviously malicious sensors may interfere with the SOF protocol by manipulating messages that pass through them, as well as introducing new veto messages themselves. One can easily show that if there is at least one honest vetoer in the network, the SOF protocol guarantees to forward at least one message (either a valid veto or some spurious veto) to the base station:

Lemma 1: In the SOF protocol, if some honest sensor generates a veto x, then the base station is guaranteed to receive some veto y by the end of the confirmation phase. Notice that y may or may not be the same as x.

Proof: To prove the lemma, it suffices to show that if an honest sensor with depth of i sends/forwards a veto during or before the $(L - i + 1)$th interval, then the base station is guaranteed to receive some veto by the end of the confirmation phase. Note that an honest vetoer sends a veto in the first interval, and this first interval must be the same or before the $(L-i+1)$th interval, where $i \leq L$ is the depth of this honest vetoer.

We use an induction on i. The induction base for $i = 1$ clearly holds. Consider any given honest sensor A with depth of $i+1$. Let B be the honest sensor on the shortest path from A to the base station and whose depth is i. Clearly such B exists. Since A sends/forwards a veto during or before the $(L - (i + 1) + 1)$th interval, B must receive that veto during or before that interval. If B has not previously sent/forwarded a veto, B will forward this veto during or before the $(L - i + 1)$th interval. Otherwise B must have previously sent/forwarded a veto before the $(L-i+1)$th interval. In either case, by inductive hypothesis, the base station is guaranteed to receive some veto by the end of the confirmation phase. □

This lemma implies that if the base station does not receive any veto, the minimum value received in the aggregation phase must be correct.

After receiving a veto. If the base station does receive a veto and if the veto is spurious, this spurious veto must have been injected by some malicious sensor. This can happen for example, if the malicious sensors aim to prevent the base station from receiving any legitimate veto. They may do so by launching a choking attack and injecting spurious vetoes (with valid edge MACs though) to "beat" the legitimate veto. In such a case, VMAT will invoke junk-triggered pinpointing/revocation based on the audit trails in SOF.

If the base station receives a valid veto, this valid veto can still originate from either an honest sensor or a malicious sensor. A malicious sensor can generate a valid veto if it purposely hid its value during the aggregation phase. The audit trail recorded in such a case will still be equivalent to the malicious sensor dropping that value. Thus regardless of the origin of the valid veto, VMAT will invoke veto-triggered pinpointing/revocation protocol. Conceptually this protocol will find out during the aggregation phase, between which neighboring sensors the value contained in the veto was dropped without an even smaller value being forwarded.

D. Supporting Multi-path Aggregation

So far our VMAT protocol uses tree-based single-path aggregation as in TAG [15]. State-of-art aggregation approaches such as synopsis-diffusion [18] often use multi-path ring-based aggregation. In multi-path aggregation for the MIN query, a sensor may send its partial aggregation result to multiple parents instead of a single one. This helps to route around failed parent or in our case, malicious parent.

Our VMAT protocol can be trivially adapted to such multi-path ring-based aggregation. The only modification needed is that when forming the tree (or more precisely, the *rings* [18] in multi-path aggregation), a sensor at level i may have multiple parents at level $i-1$. Later during the aggregation phase, a sensor should store a tuple for each of its parents, as the audit trail. The confirmation phase is the same as before. Junk-triggered pinpointing/revocation in multi-path aggregation will be the same as in single-path aggregation. For veto-triggered pinpointing/revocation, the protocol can start tracking from an arbitrary parent of the vetoer. Our claims in the next section will apply to multi-path aggregation as well.

V. Properties of VMAT Tree Formation, Aggregation, and Confirmation

We have discussed above how our design is robust against certain specific attacks. However, there are unlimited number of possible attack strategies that the (potentially colluding) malicious sensors may adopt, making it impossible to exhaustively enumerate. Thus to ultimately show the security of our protocol, we will derive a proof below to formalize its guarantee. Our proof will capture *all* possible attacks under our attack model in Section III.

We first define the notion of a *well-formed audit trail for veto-triggered pinpointing/revocation* (see Figure 3). Note that while the veto message itself is collected during the confirmation phase, the audit trail for veto-triggered pinpointing/revocation was previously recorded during the aggregation phase. Consider all the tuples stored and thus *owned* by the honest sensors in the aggregation phase. A *well-formed audit trail for veto-triggered pinpointing/revocation* is an ordered list of some stored tuples and additional special $\perp$*-tuples*, where:

- Each $\perp$-tuple is in the form of $\langle$*level, message,* $\perp$*, in-edge key, out-edge key*$\rangle$. We say that a $\perp$-tuple is *owned* by the (potentially colluding) malicious sensors.

$$\boxed{\langle 8, m, k_1\rangle,\ \langle 7, m', k_2\rangle,\ \langle 4, m', \perp\rangle,\ \langle 3, m', k_3\rangle,\ \langle 2, m', \perp\rangle}$$

Figure 3. Example of a well-formed audit trail for veto-triggered pinpointing/revocation. Here m and m' are individual stored messages, where m' contains an intermediate aggregation value that is smaller than the value contained in m. k_1, k_2, and k_3 are sensor keys of some honest sensors. The in/out-edge keys in the tuples are not show in this example.

- In the list, no two $\perp$-tuples are adjacent and the last tuple is a $\perp$-tuple.
- The *level* of any tuple is within $[0, L]$.
- The *level* of any normal tuple x is one smaller than the *level* of x's predecessor tuple.
- The *level* of any $\perp$-tuple x is smaller than the *level* of x's predecessor tuple.
- The partial aggregation value contained in any tuple x (or more specifically, contained in the *message* of tuple x) is smaller than or equal to the partial aggregation value contained in its predecessor tuple.
- For any two adjacent tuples xy in the list, x's *out-edge key* is the same as y's *in-edge key*. Furthermore, this key is held by both x's owner and y's owner.

A single $\perp$-tuple in the above definition captures a continuous segment of malicious sensors on the audit trail and thus there should be no two adjacent $\perp$-tuples. By the requirements on the levels of the tuples, a well-formed audit trail can have at most $L+1$ sensors. When tracking down the audit trail in veto-triggered pinpointing/revocation, we may encounter decreasing partial aggregation values. Thus we do not require the partial aggregation values in all tuples to be the same.

We similarly define *well-formed audit trail for junk-triggered pinpointing/revocation in the aggregation phase*. This is the same as a well-formed audit trail for veto-triggered pinpointing/revocation except that i) the *level* of a tuple should be *one larger* (for normal tuple) or *larger* (for $\perp$-tuple) than its predecessor, and ii) the *message* contained in all tuples are *identical*. Finally, a *well-formed audit trail for junk-triggered pinpointing/revocation in the confirmation phase* is similarly defined except that: i) *level* is replaced by *interval* (from the SOF protocol) in the definition, ii) the *interval* should be *one smaller* (or *smaller*) than its predecessor, and iii) the *message* contained in all tuples are *identical*.

Now we can prove the claims on the protocol so far:

Theorem 2: The VMAT tree formation, aggregation, and confirmation protocol, as well as its multi-path aggregation version, has the following properties:

- *It always terminates within $O(1)$ flooding rounds in the sense that within $O(1)$ flooding rounds, it always i) returns a minimum result, or ii) invokes veto-triggered pinpointing/revocation, or iii) invokes junk-triggered pinpointing/revocation.*
- *If it does return a minimum result, then the result returned is correct.*
- *If it invokes veto-triggered pinpointing/revocation, then the system must have a well-formed audit trail for veto-triggered pinpointing/revocation.*
- *If it invokes junk-triggered pinpointing/revocation during the aggregation or confirmation phase, then the system must have a well-formed audit trail for junk-triggered pinpointing/revocation in the aggregation or confirmation phase, respectively.*

Proof sketch: All steps below refer to steps in Figure 1.

- The termination property is trivial since there is a timeout in all steps where the base station needs to wait for an incoming message (i.e., Step 2 and 5).
- Prove by contradiction and assume that the protocol returns an incorrect minimum result w. Let x be the true minimum value among the readings of all the honest sensors, and y ($y \le x$) be the true minimum value among the readings of all the sensors. w being incorrect means that $w > x$ or $w < y$. If $w > x$, then the honest sensor with reading x would have generated a veto during the confirmation phase. By Lemma 1, the base station must have received some veto and thus not output a minimum result. For the case of $w < y$, notice that the minimum result must have passed the verification at Step 4, and carries a valid MAC. This contradicts to the fact that no sensor can generate such a valid MAC since $w < y$.
- The protocol may invoke veto-triggered pinpointing/revocation at Step 8. Notice that this legitimate veto may either originate from an honest sensor or a malicious sensor. Let that sensor be A. Consider any honest sensor B that receives (potentially indirectly) A's value during the aggregation phase. B will either forward A's value or forward another value that is smaller than A's value. This means that there is *always* another tuple in the audit trail after a tuple owned by an honest sensor B. Further because the length of the audit trail is finite (i.e., bounded by $L+1$), the audit trail must end with a $\perp$-tuple. The remaining properties needed for the audit trail to be well-formed are trivial to show and thus we omit the proof details.
- The protocol may invoke junk-triggered pinpointing/revocation at Step 4 and 7. In either case, if any honest sensor forwards a junk message, it must have previously received that message from some other sensor. Thus same as above, the audit trail must end with a $\perp$-tuple. The remaining properties needed for the audit trail to be well-formed are trivial to show and thus we omit the proof details. $\square$

VI. VMAT: Pinpointing and Revocation

We first describe the *veto-triggered pinpointing/revocation* protocol, and then briefly discuss the *junk-triggered pinpointing/revocation* protocol, which is similar. The guarantees of these protocols will later be proved in Section VII. Since only edge keys (instead of sensor keys) are used for sender authentication during message forwarding, tracking the audit trail only allows the base station to revoke some edge key. Instead of revoking all compromised edge keys one by one, VMAT pinpoints a malicious sensor after a small number of its edge keys are exposed, so that VMAT can straightaway revoke the rest of its edge keys even before it uses them to launch attacks. The last part of this section will elaborate on this design.

A. Veto-triggered Pinpointing/Revocation

Overview. Figure 4 presents the pseudo-code for veto-triggered pinpointing/revocation. Initially, the base station knows the ID of the sensor (A) that generated the veto. VMAT first finds out the edge key K_e between A and A's parent in the aggregation phase. Next VMAT tries to find a sensor B holding K_e that is willing to admit having received the minimum value from A at the given level during the aggregation phase[†]. If we do find B, we will repeat the above process from B. If such B cannot be found, then either A or A's parent is malicious and we can revoke K_e. To revoke an edge key K_e, the base station uses authenticated broadcast to announce the index of that key to all sensors, so that they will ignore messages authenticated using that key.

In a naive solution, for the base station to find out K_e, A would directly send back the index of K_e, together with a MAC generated using A's sensor key. (Remember that we intentionally avoid public key cryptography.) But since other sensors cannot verify this MAC, choking attacks can occur. We cannot simply record audit information to later pinpoint such choking attacks – otherwise we will fall into endless recursive pinpointing. To overcome this problem, we use an existing keyed predicate test protocol [29] to enable A to send back K_e resiliently despite malicious interference.

Keyed predicate test. The keyed predicate test protocol is not the contribution of this work. However for completeness, the following provides a concise review of the protocol. This protocol tests whether there is at least one sensor that i) holds a particular symmetric key K, and ii) satisfies a certain predicate. Any such sensor will send a "yes" reply in the form of a MAC generated using key K. Clearly, sensors not holding K cannot generate such MAC. The protocol further ensures that such a reply can be propagated from the sensors to the base station in a way that is resilient against choking attacks. To achieve this, it leverages existing solutions [20] to tolerate choking

[†]B may or may not be the real parent of A, and can be a malicious sensor that lies. But this will not cause any problem.

```
1. let $K_s$ and $level$ be the vetoer's sensor key and level;
2. repeat
3.     invoke the protocol in Figure 5 to find $K_e$;
4.     invoke the protocol in Figure 6 to find the $K'_s$ (the
       sensor key of the parent sensor);
5.     $K_s = K'_s$; $level = level - 1$;
```

Figure 4. Veto-triggered pinpointing/revocation protocol.

attacks for authenticated broadcast from the base station. The protocol uses such authenticated broadcast to first disseminate a one-way hash of the potential "yes" reply (i.e., one-way hash of the MAC), using a publicly-known one-way hash function H(). This one-way hash enables all the sensors to test whether a message received is a valid reply, without the need of public key cryptography or the need of knowing K.

Specifically in the protocol, the base station first uses authenticated broadcast to send the following to all sensors:

$$\langle \textit{index of } K\textit{, the predicate, nonce } N\textit{, } H(MAC_K(N)) \rangle$$

If a sensor holds K and satisfies the predicate, it will generate $\text{MAC}_K(N)$ as a "yes" reply and broadcast it locally. The hash $\text{H}(\text{MAC}_K(N))$ enables all sensors in the system to verify whether a message is a valid "yes" reply. If a sensor receives a valid "yes" reply for the first time, it will relay it by locally rebroadcasting the message. It ignores the message otherwise. Notice that since the only message that can propagate in the network is the valid reply, choking attacks are no longer possible.

The predicate test *succeeds* if the base station receives the valid "yes" reply $\text{MAC}_K(N)$, within two flooding rounds. The following theorem from [29] summarizes the guarantees of the protocol:

Theorem 3: [29] If at least one honest sensor holding K satisfies the predicate, then the keyed predicate test is guaranteed to succeed. On the other hand, if no honest sensor holding K satisfies the predicate and if no malicious sensor holds K, then the keyed predicate test is guaranteed not to succeed.

Pinpointing/revocation details. Now to retrieve the index of the edge key K_e that A shares with its parent, the base station uses $O(\log r)$ (or $O(\log n)$ since $r < n$) keyed predicate tests to do a simple binary search on the r edge keys held by A. See Figure 5 for the pseudo-code. Conceptually, if A is honest, after asking $O(\log r)$ "yes/no" questions or keyed predicate tests, the base station can uniquely pinpoint K_e in the sequence of the r edge keys. Otherwise, A is a malicious sensor, and the base station revokes A (i.e., revokes all edge keys held by A) at Step 7. Remember that the edge keys of a sensor are uniformly randomly selected from the global key pool. To revoke all of A's edge keys, the base station only needs to announce the associated random seed used for the selection.

```
1. let $z_1 < z_2 < ... < z_r$ be the index of the $r$ edge keys held by sensor $A$ (with sensor key $K_s$);
2. let $x = 1$, $y = r$;
3. repeat
4.    if $(x = y)$ { return the edge key $K_e$ corresponding to index $z_x$; }
5.    $i = (x + y)/2$;
6.    if keyed predicate test on $K_s$ with predicate $\langle z_x, z_y, v_{veto}, level\rangle$ succeeds { $y = i$; } else { $x = i + 1$; }
      (A sensor satisfies this predicate if i) it holds $K_s$, and ii) it received a message with value no greater than $v_{veto}$ from a child
      at the given $level$ during the aggregation phase, and iii) the index of the edge key for communication with its parent is
      between $z_x$ and $z_y$.)
7.    if $(x > y)$ { revoke all edge keys held by the sensor with sensor key $K_s$ and exit; }
```

Figure 5. Find the edge key to blame.

```
1. let $id_1 < id_2 < ... < id_t$ be the IDs of the $t$ sensors holding $K_e$; let $x = 1$ and $y = t$;
2. if keyed predicate test on $K_e$ with predicate $\langle id_x, id_y, v_{veto}, level\rangle$ fails { revoke $K_e$ and exit; }
3. repeat
4.    if $(x = y)$ {
5.       let $K'_s$ be the sensor key of sensor with the ID $id_x$;
6.       if keyed predicate test on $K'_s$ with predicate $\langle id_x, id_x, v_{veto}, level\rangle$ succeeds { return $K'_s$; }
7.       else { revoke $K_e$ and exit; }
8.    }
9.    $i = (x + y)/2$;
10.   if keyed predicate test on $K_e$ with predicate $\langle id_x, id_i, v_{veto}, level\rangle$ succeeds { $y = i$; }
11.   else if keyed predicate test on $K_e$ with predicate $\langle id_{i+1}, id_y, v_{veto}, level\rangle$ succeeds { $x = i + 1$; }
12.   else { revoke $K_e$ and exit; }
```

Figure 6. Find the sensor key of the parent sensor.

Next to find out the ID of A's parent B, we use keyed predicate test on the set of all sensors holding the edge key K_e. The base station knows the exact set of the t sensors holding K_e. Let their IDs be id_1, id_2, ..., id_t in increasing order. For the same reason as earlier, we cannot have B directly send back its ID. Rather, we use keyed predicate tests to do a binary search on the sequence of the t IDs (Figure 6). At each step, the base station encodes the predicate as $\langle id_x, id_y, v_{veto}, level\rangle$. Here $[x, y]$ is the window that the binary search is examining, and v_{veto} is the value from the vetoer, while $level$ is initialized to vetoer's level and decreases with every sensor tracked. The predicate asks "whether any sensor holding K_e with $id \in [id_x, id_y]$ received a report with value no greater than v_{veto} from a child at the given $level$ during the aggregation phase". Here we ask for "no greater than" instead of "equal" to allow the vetoer's value to be replaced by a smaller value. Also note that there may be multiple honest sensors satisfying the predicate test (in the presence of malicious interference), and the protocol only aims to find one of them.

Several steps in Figure 6 are worth highlighting. At Step 2, the keyed predicate test fails if no sensor is willing to admit having received A's value. This means that either A or A's parent is malicious, and we can safely revoke K_e. Next, it is possible for a malicious sensor holding K_e to behave inconsistently during the binary search. This may cause the keyed predicate test at both Step 10 and 11 to fail. But if this does happen, we know that K_e is held by some malicious sensor and Step 12 will revoke K_e. This also explains why it is important to use keyed predicate test on K_e and restrict ourselves to sensors holding K_e. Finally, a malicious sensor may frame an honest sensor with a different ID. Step 6 thus re-confirms using a keyed predicate test on the sensor key of the ID found.

B. Junk-triggered Pinpointing/Revocation

Junk-triggered pinpointing/revocation is invoked to pinpoint malicious sensors that injected spurious messages in either the aggregation phase or confirmation phase. The pinpointing is triggered immediately when a spurious message is received at the base station. Different from veto-triggered pinpointing/revocation, which tracks the audit trail from the vetoer (source) toward the base station, in junk-triggered pinpointing/revocation the base station tracks the audit trail toward the initially unknown source. The process begins with the edge key K_e that is used by the base station to receive the spurious message. Similar to Figure 6, VMAT then uses keyed predicate test on K_e to find the sensor A that is willing to admit having forwarded the message to the base station. If no such sensor is identified, K_e will be revoked. Otherwise, VMAT retrieves the edge key K'_e that is used by A to receive the message, similar to Figure 5. The process then continues after decreasing *interval* (for spurious veto) or increasing *level* (for spurious aggregation result). Since this is almost identical to veto-triggered pinpointing/revocation, we omit the details for clarity.

C. Effectiveness of Edge-key Revocation

Revoking a single edge key will not fully disable a malicious sensor, since it has r (e.g., 250) edge keys in its key ring. Thus it may appear that revoking edge keys is rather inefficient to counter attacks. However, notice that it is unlikely for an honest sensor to share too many edge keys with the malicious sensors. Thus we can immediately revoke a malicious sensor (i.e., revoke all keys in its key ring) whenever a given threshold θ of its keys have been revoked. Doing so enables us to revoke the malicious sensor's other edge keys even before they are used to launch attacks.

There is an obvious tradeoff in choosing θ. A smaller θ allows faster revocation, at the cost of potentially mis-revoking honest sensors that happen to share more than θ edge keys with the malicious sensors. In particular, notice that the adversary can use the edge keys held by different malicious sensors to frame honest sensors. A larger threshold is thus needed when the number of malicious sensors is large, to keep the mis-revocation probability low. Our numerical results later show that, even with 20 malicious sensors, a θ of 27 is already sufficient to ensure almost zero probability of mis-revocation. Such θ is only about 10% of the 250 edge keys held by a sensor.

VII. Properties of VMAT Pinpointing and Revocation

Lemma 4, 5, and Theorem 6 below prove the correctness and performance guarantees of veto-triggered pinpointing/revocation. The guarantees from junk-triggered pinpointing/revocation are exactly the same and thus are omitted for clarity.

Lemma 4: Assume that the system has a well-formed audit trail for veto-triggered pinpointing/revocation. Consider the protocol in Figure 5.

- *If sensor A is honest, then the protocol will always return some edge key K_e held by A, and A will never be revoked in the protocol.*
- *If sensor A is malicious, then the protocol will either return some edge key K_e held by A, or revoke A.*

Proof: Regardless of whether A is honest or malicious, obviously the protocol always either returns some edge key K_e held by A, or revokes A. Next for an honest sensor A, its tuple will never be the last one in a well-formed audit trail. This means that the binary search must succeed and the condition of $x > y$ at Step 7 can never be satisfied, which in turn means that A will never be revoked. □

Lemma 5: Assume that the system has a well-formed audit trail for veto-triggered pinpointing/revocation. Consider the protocol in Figure 6.

- *If no malicious sensor holds edge key K_e, then K_e will never be revoked in the protocol.*
- *If the protocol returns a sensor key K'_s, then either K'_s is held by some malicious sensor or K'_s is held by some honest sensor that received a message with value no greater than v_{veto} during the aggregation phase.*

Proof:

- The protocol may potentially revoke K_e at Step 2, 7 or 12. Since no malicious sensor holds K_e and since A holds K_e (by Lemma 4), A must be honest. Thus its tuple is not the last one in the well-formed audit trail, and A must have a parent B, which owns the successor tuple in the audit trail.
  We next prove that Step 2 will not revoke K_e. Assume instead that Step 2 revokes K_e, implying that the keyed predicate test at Step 2 fails and no sensor is willing to admit that it received the message from A at the given level during the aggregation phase. This can never happen if both A and B are honest. Since A is honest, B must be malicious. Thus K_e is held by some malicious sensor (i.e., B). Contradiction.
  Finally, the proof for Step 7 and 12 is trivial, since the condition for reaching Step 7 and 12 will never be satisfied if all sensors holding K_e are honest.
- Obvious from the condition at Step 6. □

We define *communication complexity* as the total number of bits sent and received by a sensor (including those bits forwarded for other sensors during multi-hop forwarding). The following theorem proves the correctness and performance guarantees of veto-triggered pinpointing/revocation.

Theorem 6: Suppose that the base station receives a legitimate veto during the confirmation phase. Then the pinpointing/revocation protocol in Figure 4:

- *Will always revoke at least one edge key, and any key revoked must be held by some malicious sensor.*
- *Incurs communication complexity of $O(Ld \log n)$ for each sensor, where d is the corresponding sensor's degree.*
- *Incurs time complexity of $O(L \log n)$ flooding rounds.*

Proof: Since the base station receives a legitimate veto, Theorem 2 already proved that the system has a well-formed audit trail for veto-triggered pinpointing/revocation. Lemma 4 and 5 above proved that any key revoked must belong to some malicious sensor. The length of a well-formed audit trail is at most $L + 1$. Thus the number of steps that the protocol will track before revoking some key is $O(L)$. Each step incurs at most $O(\log n + \log r) = O(\log n)$ flooding rounds, and $O(d \log n)$ communication complexity for a sensor with degree d. □

Putting it all together. Combining the properties we have proved so far for VMAT tree formation, aggregation, confirmation, pinpointing, and revocation in Theorem 2 and 6 yields the following main theorem, which summarizes the overall guarantee of VMAT.

Theorem 7: The VMAT protocol (including tree formation, aggregation, confirmation, pinpointing, and revocation) will either return a correct result within $O(1)$ flooding rounds, or revoke at least one edge key held by some malicious sensor within $O(L \log n)$ flooding rounds.

VIII. Generalizing to Count and Sum

Converting COUNT query or SUM query to MIN query is a well-known technique to facilitate secure aggregation [2], [19]. There are several ways to do so, and VMAT uses the following recent scheme [17] with strong error guarantees. Without loss of generality, we assume that sensor readings are integers within a certain domain. To compute sum (and predicate count as a special case), a sensor x with a reading $v > 0$ generates m independent random numbers (*synopses*) $a_{1,x}, a_{2,x} \cdots, a_{m,x}$ according to an exponential distribution with mean $\frac{1}{v}$. Let a_i^{min} be the minimum $a_{i,x}$'s across all sensors (i.e., $a_i^{min} = \min_x a_{i,x}$). Let $a^{min} = \sum_{i=1}^{m} a_i^{min}/m$. The sum is estimated to be $1/a^{min}$. It is proved [17] that when $m = \Theta(\frac{1}{\epsilon^2} \log \frac{1}{\delta})$ this estimator is an (ϵ, δ)-approximation of the real sum. In practice we will use $\Theta(\frac{1}{\epsilon^2} \log \frac{1}{\delta})$ parallel instances. Finally, average can be computed from predicate count and sum.

To support robust COUNT query and SUM query with the non-robust MIN query, as in [2], [19], we require that synopses be generated using a (deterministic) pseudo-random number generator seeded by the concatenation of some nonce specified by the base station and the sensor's ID. With this design, a (valid) synopsis produced by a malicious sensor must correspond to some possible sensor reading. This prevents the malicious sensors from reporting spurious synopses (e.g., always reporting 0). On the other hand, a malicious sensor can still enumerate all possible readings and pick an adversarial reading to generate an adversarial synopsis. However, this will have precisely the same effect as the malicious sensor reporting a fake reading for itself. The same applies when multiple malicious sensors coordinate and collectively choose adversarial readings. Since this well-known technique for securely converting predicate count and sum to minimum is not our contribution, we refer interested readers to [2], [19] for more details.

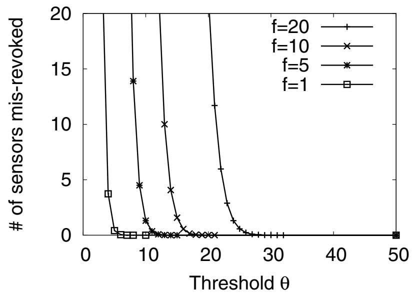
IX. Numerical Examples

We have proved VMAT's correctness and performance guarantees. In this section, we provide some numerical examples to further illustrate VMAT's properties in terms of revocation efficiency and approximation quality.

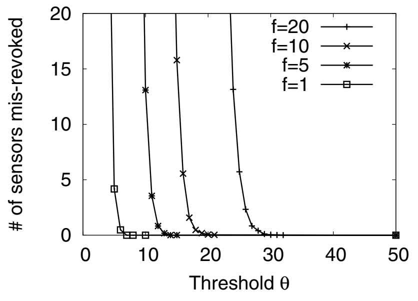
Effectiveness of edge-key revocation. We first aim to better understand how many edge keys we need to revoke before being able to revoke a malicious sensor entirely. As explained in Section VI-C, a malicious sensor can often be uniquely identified by a small number θ of its edge keys. We use simulation to quantify how many exposed edge keys are sufficient to uniquely identify a malicious sensor, while ensuring near-zero probability of mis-revocation of honest sensors. As in [7], we consider a sensor network with each sensor holding 250 keys randomly selected from a global key pool of size $100,000$. Under this setting, any two sensors can find at least one common edge key with probability around 0.5. We consider two network sizes with $1,000$ and $10,000$ sensors respectively. For each network size, we vary the number f of malicious sensors. Figure 7 plots the average number of mis-revoked honest sensors (out of 100 trials) for different θ thresholds.

The figure shows that under both network sizes, with a single malicious sensor, we can identify that malicious sensor after it exposes roughly 7 edge keys, while incurring close-to-zero probability of mis-revoking any honest sensor. As explained in Section VI-C, with more malicious sensors, a larger θ is needed to achieve the same mis-revocation probability. As shown in the figure, to keep the average number of mis-revoked honest sensors below 1, θ needs to be 27 for 20 malicious sensors. On the other hand, even this θ value is still roughly an order of magnitude smaller than the total number of edge keys held by a malicious sensor. Finally, for scenarios with much larger number of malicious sensors, the cost of revocation can become too high to be practical. However, in such cases, the adversary will likely have already acquired a large fraction of edge keys from the global key pool. Revoking all these edge keys, even if possible, will likely result in a disconnected network. Thus in such scenarios, directly tolerating the malicious sensors (e.g., as in [29]) will perhaps be more meaningful.

Approximation quality. We next use simulation to better understand the approximation quality by converting COUNT query to MIN query. Figure 8 presents the approximation error achieved by 100 synopses, for different predicate count values, each over 200 trials. The "average" is taken over all trials for a given predicate count value. The "x percentile" value means that $x\%$ of all trials have an error below that value. The figure shows that using only 100 synopses can give us an average relative error of below 10%. Even if we pessimistically assume that each synopsis takes 24 bytes (including all the MACs), 100 synopses will only take 2.4KB. Sending and receiving these synopses are quite affordable in today's common sensor platforms (e.g., ZigBee sensors can communicate at 250Kbps). In comparison, without the VMAT secure in-network aggregation protocol, a naive approach would need to transmit all individual readings back to the base station. Here each reading still needs to carry MACs to prevent the attacker from injecting additional fabricated readings. Assuming each MAC is 8 bytes, this naive approach would incur a communication complexity of at least 80KB for a network with $10,000$ sensors, which is



(a) 1,000 sensors



(b) 10,000 sensors

Figure 7. Avg # of honest sensors mis-revoked under various threshold θ.

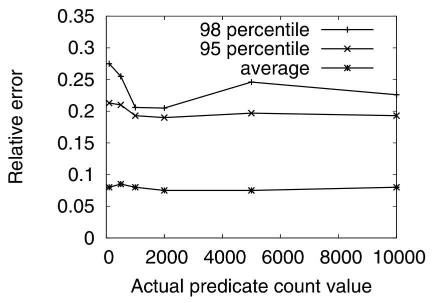


Figure 8. Relative estimation error for different predicate count values.

one to two orders of magnitude larger than VMAT.

X. Conclusion

This paper proposes VMAT, a secure in-network aggregation algorithm that strictly diminishes the capability of adversaries whenever they launch a successful attack, by properly pinpointing and revoking keys held by the adversary. To achieve such functionality using only symmetric key cryptography, VMAT integrates several complementary techniques, including the timestamp-based tree formation, SOF, edge-key-based revocation, and audit trail information retrieval with keyed predicate test. We have proved VMAT's correctness and performance guarantees.

Acknowledge

This work is partly supported by National University of Singapore Young Investigator Award R-252-000-334-101.

References

[1] S. Camtepe and B. Yener. Key Distribution Mechanisms for Wireless Sensor Networks: A Survey. Technical report, Rensselaer Polytechnic Institute, 2005.

[2] H. Chan, A. Perrig, B. Przydatek, and D. Song. SIA: Secure Information Aggregation in Sensor Networks. *Journal of Computer Security*, 15(1), 2007.

[3] H. Chan, A. Perrig, and D. Song. Secure Hierarchical In-network Aggregation for Sensor Networks. In *CCS*, 2006.

[4] J. Deng, R. Han, and S. Mishra. INSENS: Intrusion-Tolerant Routing For Wireless Sensor Networks. *Elsevier Journal on Computer Communications*, 29(2), 2005.

[5] J. Deng, R. Han, and S. Mishra. Limiting DoS Attacks During Multihop Data Delivery In Wireless Sensor Networks. *International Journal of Security and Networks*, 2006.

[6] W. Du, J. Deng, Y. Han, and P. K. Varshney. A Witness-based Approach for Data Fusion Assurance in Wireless Sensor Networks. In *GLOBECOM*, 2003.

[7] L. Eschenauer and V. Gligor. A Key-Management Scheme for Distributed Sensor Networks. In *CCS*, 2002.

[8] P. Flajolet and G.N. Martin. Probabilistic Counting Algorithms for Database Applications. *Journal of Computer and System Sciences*, 1985.

[9] K. Frikken and J. Dougherty. An Efficient Integrity-preserving Scheme for Hierarchical Sensor Aggregation. In *WiSec*, 2008.

[10] M. Garofalakis, J. Hellerstein, and P. Maniatis. Proof Sketches: Verifiable In-Network Aggregation. In *ICDE*, 2007.

[11] P. Haghani, P. Papadimitratos, M. Poturalski, K. Aberer, and J. Hubaux. Efficient and robust secure aggregation for sensor networks. In *NPSec*, 2007.

[12] L. Hu and D. Evans. Secure Aggregation for Wireless Networks. In *WSAAN*, 2003.

[13] Y. Hu, A. Perrig, and D. Johnson. Wormhole attacks in wireless networks. *Journal on Selected Areas in Communications*, 24(2):370–380, 2006.

[14] P. Jadia and A. Mathuria. Efficient Secure Aggregation in Sensor Networks. In *HiPC*, 2004.

[15] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. TAG: A Tiny AGgregation Service for Ad-Hoc Sensor Networks. In *OSDI*, 2002.

[16] A. Mahimkar and T. Rappaport. SecureDAV: A Secure Data Aggregation and Verification Protocol for Sensor Networks. In *GLOBECOM*, 2004.

[17] D. Mosk-Aoyama and D. Shah. Computing Separable Functions via Gossip. In *PODC*, 2006.

[18] S. Nath, P. B. Gibbons, S. Seshan, and Z. Anderson. Synopsis Diffusion for Robust Aggregation in Sensor Networks. In *SenSys*, 2004.

[19] S. Nath, H. Yu, and H. Chan. Secure Outsourced Aggregation via One-way Chains. In *ACM SIGMOD*, 2009.

[20] P. Ning, A. Liu, and W. Du. Mitigating DoS Attacks against Broadcast Authentication in Wireless Sensor Networks. *ACM Transactions on Sensor Networks*, 2008.

[21] V. Padmanabhan and D. Simon. Secure Traceroute to Detect Faulty or Malicious Routing. *SIGCOMM Computer Communication Review*, 33(1), 2003.

[22] B. Parno, M. Luk, E. Gaustad, and A. Perrig. Secure Sensor Network Routing: A Clean-Slate Approach. In *CoNEXT*, 2006.

[23] S. Roy, S. Setia, and S. Jajodia. Attack-resilient Hierarchical Data Aggregation in Sensor Networks. In *SASN*, 2006.

[24] G. Taban and V. Gligor. Efficient handling of adversary attacks in aggregation applications. In *ESORICS*, 2008.

[25] R. Wang, W. Du, and P. Ning. Containing Denial-of-Service Attacks in Broadcast Authentication in Sensor Networks. In *MobiHoc*, 2007.

[26] A. Wood, L. Fang, J. Stankovic, and T. He. SIGF: A Family of Configurable, Secure Routing Protocols for Wireless Sensor Networks. In *SASN*, 2006.

[27] Y. Yang, X. Wang, S. Zhu, and G. Cao. SDAP: A Secure Hop-by-Hop Data Aggregation Protocol for Sensor Networks. In *MobiHoc*, 2006.

[28] H. Yu. Brief Announcement: DoS-Resilient Secure Aggregation Queries in Sensor Networks. In *PODC*, 2007.

[29] H. Yu. Secure and highly-available aggregation queries in large-scale sensor networks via set sampling. In *IPSN*, 2009.