Exact In-Network Aggregation with Integrity and Confidentiality

Stavros Papadopoulos, Aggelos Kiayias, Dimitris Papadias

Abstract—*In-network aggregation* reduces the energy cost of processing aggregate queries (such as SUM, MAX, etc.) in wireless sensor networks. Recently, research has focused on *secure* in-network aggregation, motivated by the following two scenarios: (i) the sensors are deployed in open and unsafe environments, and (ii) the aggregation process is outsourced to an *untrustworthy* service. Despite the bulk of work on the topic, there is currently *no* solution providing *both* integrity and confidentiality in the above scenarios. Moreover, existing solutions either return *approximate* results, or have limited applicability to certain types of aggregate queries. Our paper is the *first* work that provides *both integrity and confidentiality* in the aforementioned scenarios, while covering a *wide range* of aggregates and returning *exact* results. We initially present SIES, a scheme that solves exact SUM queries through a combination of *homomorphic encryption* and *secret sharing*. Subsequently, we show how to adapt SIES in order to support many other exact aggregate queries (such as MAX, MEDIAN, etc.). Finally, we augment our schemes with a functionality that *identifies* malicious sensors, preventing *denial of service* (DoS) attacks and attributing robustness to the system. Our techniques are lightweight and induce very small bandwidth consumption. Therefore, they constitute ideal solutions for resource-constrained sensors.

Index Terms—Sensor Networks, Aggregation, In-network, Security, Integrity, Confidentiality.

1 Introduction

Wireless sensor networks are nowadays deployed in a plethora of applications, such as factory monitoring, wildlife surveillance, environmental monitoring, battlefield operations, fire and burglar alarms, etc. The sensor nodes form a network topology by connecting to other sensors within their vicinity. Communication between nodes is dictated by a multi-hop routing protocol. The sensors generate and transmit stream data (e.g., environmental readings, information about moving objects, etc.). A querier (e.g., a corporate organization, a laboratory, etc.) poses long-running queries on the sensor readings, and periodically receives data from the network (typically via a single node, called the *sink*).

Aggregate queries (e.g., SUM, MAX, etc.) constitute a wide and important query class in sensor networks. In the naive case, the querier collects all the raw data from the sensors and performs the aggregation locally. Although this may be a viable solution in small networks, it leads to an excessive energy expenditure in large-scale networks. Specifically, the nodes situated closer to the querier route a considerable amount of data, which originate from farther nodes in the network topology. Therefore, their battery is depleted fast, since its lifespan is mainly impacted by data transmission. Moreover, the above solution introduces a significant bandwidth consumption and computational cost at the querier. *In-network aggregation* [1], [2] is a popular paradigm that tackles these drawbacks, by spreading more computation within the network. In particular, some sensors play the role of *aggregators*, which fuse the data as they flow in the network. The querier eventually receives only the final result from a single aggregator.

Recently, research has focused on *secure* in-network aggregation, which is mainly motivated by the following two scenarios (or their combination):

Scenario 1 *The sensors are deployed in open and hostile environments (e.g., in battlefield grounds), or in security-critical applications (e.g., in factory monitoring, burglar alarms), where adversarial activity must be averted (examples include [3], [4], [5], [6]).*

Scenario 2 *Under the new trend of outsourced aggregation [7], [8], the tasks of organizing/tuning the aggregation network and conducting the aggregation process are delegated to a third-party service provider with a well provisioned distributed infrastructure (e.g., Microsoft's SenseWeb [9]). Nevertheless, the provider may be untrustworthy and possibly malicious.*

Secure in-network aggregation mandates the following key security properties:

- **Data confidentiality** The adversary must not be able to read the raw data transmitted by the sensors.

- *S. Papadopoulos is with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. E-mail: stavrosp@cse.ust.hk*
- *A. Kiayias is with the Department of Informatics and Telecommunications, National and Kapodistrian University of Athens. E-mail: aggelos@di.uoa.gr*
- *D. Papadias is with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. E-mail: dimitris@cse.ust.hk*

- **Data integrity** The adversary must not be able to alter the result, i.e., the querier should be able to verify that all the raw data were included in the aggregation process, and no spurious data were injected.
- **Data authentication** The adversary must not be able to impersonate the querier and/or the sensors, i.e., these parties must be able to verify the origin of a received message.
- **Data freshness** The reported result must reflect the most recent instance of the system, i.e., the adversary must not be able to replay old results to the querier.

Initially, we provide a survey on the bulk of work targeting secure in-network aggregation, and clarify the various threat models and assumptions. Through this survey we identify an important gap in the literature: there is *no* solution to date that supports *both integrity and confidentiality*, capturing *both* Scenarios 1 and 2. Moreover, we point out that every existing technique either returns *approximate* results, or supports only few types of *exact* aggregates (e.g., it may handle exact `SUM`, but not exact `MEDIAN`).

Next we introduce SIES, a scheme that provides secure in-network processing of exact `SUM` queries. SIES satisfies *both* integrity and confidentiality (along with authentication and freshness) in *both* Scenarios 1 and 2. It achieves this goal through a combination of *homomorphic encryption* and *secret sharing*, which leads to *strong* (cryptographic) security guarantees. It is scalable, and entails small (practically constant and in the order of a few bytes) communication cost per network edge. Moreover, it requires few and inexpensive operations (hashes and modular additions/multiplications) at each party involved. The above render SIES lightweight and, thus, an ideal solution for resource-constrained sensor networks.

Subsequently, we illustrate how SIES can be adapted in order to efficiently solve `COUNT`, `AVG`, `VARIANCE`, `STDDEV`, and `q-QUANTILE` queries (including the special cases of `MIN`, `MAX`, and `MEDIAN`), while returning *exact* results and retaining high security and performance. Therefore, our work is the *first* to support integrity and confidentiality for a wide range of exact aggregate queries in the aforementioned scenarios.

In addition, we augment our schemes with a functionality that *identifies* sensors exhibiting malicious activity. This functionality prevents corrupted nodes from launching certain *denial of service* (DoS) attacks and, thus, attributes robustness to our system. Finally, we experimentally demonstrate the efficiency and practicality of our schemes.

The rest of the paper is organized as follows. Section 2 surveys the related work. Section 3 formulates our targeted problem. Section 4 describes SIES. Section 5 explains how to use SIES to solve aggregate queries beyond `SUM`. Section 6 presents the malicious node identification functionality. Section 7 evaluates the performance of our techniques. Finally, Section 8 concludes our paper.

2 BACKGROUND

Section 2.1 describes traditional (i.e., non-secure) in-network aggregation techniques. Section 2.2 provides some useful cryptographic tools. Section 2.3 surveys the methods on secure in-network aggregation.

2.1 In-network Aggregation

Here we provide necessary information on in-network aggregation. Initially, we present the system architecture and query model that we will be focusing on. Subsequently, we explain how existing methods handle a variety of exact aggregate queries. Finally, we briefly outline approximate techniques. We refer the reader to [10] for a more detailed survey on the literature of in-network aggregation.

System architecture A set of sensors forms a wireless network, and complies with a multi-hop routing protocol. The network topology is a *tree*. Each sensor may be either a *source*, or an *aggregator*, or both. However, without loss of generality, we assume that the leaves are only sources, whereas the internal nodes are only aggregators. A source generates environmental readings (e.g., temperature), and an aggregator performs computations. A *querier* interfaces with the network through the root aggregator, called the *sink*. In the following, N is the number of sources in the network, and h is the tree height.

The sensors are *resource-constrained*, while their battery is depleted fast when they perform computations and, more importantly, when they transmit/receive messages. Moreover, we assume that it takes some time for a sensor to communicate a message to another sensor over the wireless channel (typically $\sim$10 ms). Finally, each sensor is *loosely synchronized* with its children and its parent, such that it knows when to transmit/receive a message [1].

Time is subdivided into disjoint intervals of equal length T, called *time epochs*. Every source senses a value during an epoch. For simplicity, we hereafter perceive every epoch as a distinct time instant t.

Query model The querier registers a *continuous* query at the sources, either by broadcasting the query, or by manually installing it during a setup phase. A query may have the following form:

Query template
```
SELECT AGG(attr) FROM Sensors
WHERE pred
EPOCH DURATION T
```

`AGG` is the aggregate function (e.g., `SUM`) computed on a certain attribute `attr` (i.e., reading type) of `Sensors`, `pred` is a predicate, and T is the length of the epoch. Note that the query may have additional clauses, such as `GROUP BY` and `HAVING`, but we omit them for simplicity.

The network is responsible for *pushing* the query result to the querier in every epoch. This query model is called *push-based*, and is usually preferable to the *pull-based* approach where the querier broadcasts the query every time it wishes to collect the result on demand. The main reason is that the sources must be always on to receive potential queries in the pull-based model, whereas in the push-based they may preserve power by turning on periodically.

The aggregate functions are distinguished into *distributive* (or *decomposable*) and *non-distributive*. Let S, S_1 and S_2 be sets such that $S = S_1 \cup S_2$ and $S_1 \cap S_2 = \varnothing$. An aggregate function f is distributive if $f(S) = f(\{f(S_1), f(S_2)\})$, and non-distributive otherwise. Examples of distributive aggregates are `SUM` and `MIN`/`MAX`, whereas examples of non-distributive aggregates include `q-QUANTILE` (excluding the special cases of `MIN`/`MAX`).

Distributive aggregates We first explain the case of `SUM` [1]. In every epoch, each source generates a value and, if `pred` is satisfied, transmits it to its parent aggregator. Every aggregator sums the values collected from its children and forwards the yielded result to its parent. A partial sum is called a *partial state record* (PSR). Eventually, the querier receives the final result from the tree root. Observe that the correctness of the protocol stems from the distributive property. The communication cost *per tree edge* is $O(1)$.

Using the above `SUM` protocol we can answer `COUNT` queries; a source transmits 1 if it satisfies `pred`, and nothing otherwise. Combining `SUM` and `COUNT` results we can handle `AVG`, `STDDEV` and `VARIANCE`, e.g., `AVG` = $\frac{\texttt{SUM}}{\texttt{COUNT}}$. Finally, following a slightly modified protocol we can support `MIN`/`MAX` queries; the aggregator forwards to its parent the min/max value among those collected from its children, instead of their sum.

Non-distributive aggregates Due to the lack of the distributive property, we cannot solve exact non-distributive aggregates by performing in-network aggregation in a *single round*, i.e., with a single flow of messages from the leaves to the root. We next describe a common technique that uses the `SUM` protocol as a subroutine and concludes in $O(\log |D|)$ rounds, where $|D|$ is the value domain size. The communication complexity per tree edge is $O(\log |D|)$. It is noteworthy to say that in practice $\log |D| \ll N$, which renders this scheme much more communication-efficient than the trivial case where no in-network aggregation is performed (which implies $O(N)$ communication cost). Note also that if the longevity of the network is of primary importance, then sacrificing query delay (due to the multiple rounds) to save communication is acceptable.

We will describe `MEDIAN` computation through an example. Let the domain be $D = [0, 100]$, and suppose we have 11 sources. The values generated by the sources in an epoch are {34,13,15,8,81,64,54,80,77,90,92}, and the median is 64. Figure 1 illustrates these values ordered over D as dots, and the median is the 6th dot in the order (depicted hollow).

The querier initially broadcasts to the entire network *two* intervals, $[0, 50]$ and $(50, 100]$. Every source receiving these intervals sends a pair (v_1, v_2) to its parent, where v_1 (v_2) equals 1 if the sensed value is in $[0, 50]$ ($(50, 100]$), and 0 otherwise. For example, if the reading is 34, the source

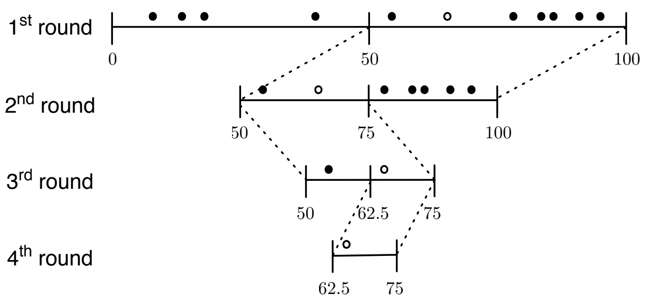


Fig. 1. `MEDIAN` computation example

sends pair $(1, 0)$. The aggregators then follow the described `SUM` protocol individually for the v_1 and v_2 values. In the end of the first round, the querier receives $(4, 7)$, which indicates that there are 4 values in $[0, 50]$ and 7 in $(50, 100]$. This means that the median (i.e., the 6th element) is in $(50, 100]$. Moreover, it reveals that the median is the 2nd element among those falling in $(50, 100]$.

Subsequently, the querier initiates a second round, broadcasting ranges $(50, 75]$ and $(75, 100]$. Following the above process, the querier eventually receives $(2, 5)$, which means that the median resides in $(50, 75]$. Continuing for two more rounds, the querier discovers that the median is the only value in $(62.5, 75]$. Therefore, it finally broadcasts $(62.5, 75]$ with a flag that it wishes to receive the actual value in this interval, and the corresponding source forwards it in the network.

Observe that arbitrary `q-QUANTILE` queries (including the special cases of `MIN`/`MAX`) are trivial to handle using the above *bucketization* technique. Furthermore, note that the querier could alternatively decompose the domain into B ranges/buckets, forcing the sources to create tuples of size B. The communication cost per edge would then change to $O(B \cdot \log_B |D|)$.

Approximate solutions Several methods focus on returning *approximate* results, while providing theoretical error bounds (examples include [11], [12], [13]). Their main motivation is to sacrifice accuracy in order to (i) enhance the robustness of the system in a communication-efficient manner, in case there are sensor failures that highly influence the value of the exact final result, and/or (ii) eliminate or reduce the multiple rounds of the `q-QUANTILE` query protocol to improve the query delay. Although we acknowledge the significance of approximate in-network aggregation, we note that the main focus of this paper is *exact* aggregation. Therefore, we omit a more detailed survey on the existing approximation techniques.

2.2 Cryptographic Tools

Homomorphic encryption Let m_1 and m_2 be two plaintexts, and $\odot$ a binary operation over the plaintext space. An encryption function $\mathcal{E}$ is homomorphic if it allows the generation of $\mathcal{E}(m_1 \odot m_2)$, given only ciphertexts $\mathcal{E}(m_1)$ and $\mathcal{E}(m_2)$ and without requiring their decryption. For example, the RSA cryptosystem [14] is homomorphic; supposing that the public key is (e, n), it holds that $\mathcal{E}_{RSA}(m_1) \cdot \mathcal{E}_{RSA}(m_2) \mod n = m_1^e \cdot m_2^e \mod n = (m_1 \cdot m_2)^e \mod n = \mathcal{E}_{RSA}(m_1 \cdot m_2)$. This scheme is also called *multiplicatively homomorphic*, since operation $\odot$ is modular multiplication. The methods that support the modular addition are called *additively homomorphic*. For instance, in the symmetric encryption setting, one can use a variant of the one-time pad to achieve an additively homomorphic encryption: we define $\mathcal{E}_K(m) = K \cdot m$, where the plaintext space is a finite field and keys are assumed to satisfy $K \neq 0$. It is easy to verify that $\mathcal{E}_K$ is homomorphic with respect to the field addition.

HMAC The HMAC (*hash-based message authentication code*) is a short piece of information used to prove the origin of a message m, as well as its integrity [15]. It is implemented by combining a one-way, collision-resistant hash function $H(\cdot)$ with a secret key K. It entails two applications of $H(\cdot)$, and consumes the same space as the hash digest. We use $HM_1(K, m)$/$HM_{256}(K, m)$/$HM_{384}(K, m)$ to denote the HMAC of m using key K, assuming that the underlying hash function is SHA-1/-256/-384 [16], [17] that produces 20-/32-/48-byte digests, respectively.

Pseudo-random function (PRF) A PRF takes as input a secret random key K, and a variable m that can be an arbitrary string. Its output is distinguished from that of a truly random function with *negligible* probability, as long as K is hidden. HMACs have been widely used as PRFs in the literature [17]. In our work, we assume that the PRFs are implemented as HMACs.

Authenticated broadcasting An authenticated broadcasting protocol allows the querier to transmit a message (e.g., a query) to the entire network, in a way such that every sensor can verify that the message has been indeed sent by the querier. A trivial solution would be for the querier to *sign* the message using an *asymmetric* cryptosystem like RSA. A sensor could then verify the signature using the public key of the querier. Nevertheless, asymmetric schemes have been criticized as being inefficient for sensor networks [18]. Alternatively, the querier could use a *symmetric* key (known to all sensors) and produce an HMAC on the message. A sensor could then efficiently verify the message using the common key. However, a compromised sensor would be able to *impersonate* the querier.

The state-of-the-art authenticated broadcasting scheme is μTESLA [18], which avoids these problems. Specifically, the querier first broadcasts the message to the network, along with an HMAC created with a *symmetric key*. Next, it broadcasts the key used in the HMAC to the network *after a certain delay*. This delay ensures that the sensors receive the message and HMAC *before* the key is disclosed to any party. Therefore, the sensors are certain that the HMAC was produced with a key that an adversary could not possess. Each sensor verifies the key with some stored secret information, and then the message with the key.

2.3 Secure In-network Aggregation

We first describe techniques supporting *exact* aggregation results, categorizing them into those that target at (i) only integrity, (ii) only confidentiality, and (iii) both integrity and confidentiality. Subsequently, we briefly discuss *approximate* schemes. Finally, we explain methods that attempt to identify malicious nodes, after an integrity violation is detected. We refer the reader to [19] for a more detailed survey on secure in-network aggregation.

Only integrity Du et al. [20] necessitate the involvement of certain *trusted* aggregators, called *witnesses*. Hu and Evans [3] protect against the case where only a *single* aggregator is malicious. Both the above methods cannot work under Scenario 2, where all the aggregators are untrustworthy (and, thus, possibly malicious).

The schemes in [21], [22] support exact `SUM` queries, following the *commit-and-attest model*. The latter consists of two rounds. During the *commitment round*, the aggregators are forced to commit to the partial results they produce, by constructing a cryptographic structure like the Merkle Hash Tree [23] and sending the root digest to the querier. In the *attestation round*, the querier broadcasts the aggregate result and the root digest it received from the network to all the sensors using μTESLA [18]. Each sensor then individually audits its contribution to the result using the commitment structure. This model inflicts considerable communication cost per network edge (equal to $O(\log N)$ *hash digests*), and increased query latency due to the extra attestation round.

SECOA [8] handles exact `MAX` queries, through the use of the one-wayness of the RSA encryption function. Its major downside is that it inflicts excessive computational cost to the sensors and the querier. Frikken et al. [24] present a scheme for general aggregation queries, focusing on attributing resilience in the presence of malicious nodes. This method cannot be applied in Scenario 2 where all aggregators may be malicious, since the communication complexity increases prohibitively with the number of malicious nodes tolerated. Moreover, [24] describes a scheme for `SUM` queries with $O(1)$ communication cost when no malicious nodes are tolerated. However, it involves a pipeline of $O(N)$ steps, where at each step a node performs $O(N)$ modular exponentiations. Consequently, it features an excessive computational cost at the sensors and a prohibitive query delay.

Only confidentiality LEAP [25] is a key management protocol that allows in-network aggregation, while restricting the impact of any malicious nodes *within their network neighborhood*. [26] satisfies confidentiality via a homomorphic encryption scheme. However, every source shares a *common* key, which means that the system security collapses when a single source is compromised. This problem is fixed in [5], which uses another efficient additively homomorphic scheme and a different key for every source. Finally, the `SUM` approaches described in

[27], [28] protect individual readings, but disclose partial sums and the final result, thus providing a low level of confidentiality.

Both integrity and confidentiality [29] makes use of HMACs (for integrity) and symmetric encryption (for confidentiality), assuming though that all the aggregators are *trusted* and *uncompromisable*. Jadia and Mathuria [4] extend [3] to support confidentiality in addition to integrity. Nevertheless, their technique inherits the weakness of being vulnerable against *multiple* malicious aggregators.

ABBA [30] is a bucketization approach (similar to the one we described in Section 2.1) where the messages incorporate secret keys (for confidentiality) and checksums (for integrity). This scheme suffers from three serious drawbacks: (i) the ids of all the sources replying to the query must be sent to the querier, inflicting the prohibitive communication cost of $O(N)$ per edge, (ii) a malicious aggregator may simply *drop* messages and the corresponding ids, thus affecting the final result, and (iii) it is not apparent how ABBA efficiently answers SUM queries.

[31] makes the assumption that certain *trusted* aggregators *monitor* the actions of other aggregators. Frikken and Zhang [32] opt for satisfying a weaker (i.e., non-cryptographic) notion of integrity and privacy. More importantly, they do not protect against malicious aggregators. Finally, they cover only SUM queries.

Approximate solutions There is a considerable number of methods ([6], [33], [34], [35], [7], [36], [37], [8]) that return approximate results securely, while *tolerating* malicious activity and preventing DoS attacks. Once again, although secure approximate in-network aggregation is a very interesting and challenging domain, it is orthogonal to our work.

Malicious node identification There exist two techniques [38], [39] that attempt to locate a corrupted node, in case an integrity violation is detected by the querier. They are both based on SHIA [21], a commit-and-attest protocol that provides only integrity. The first by Haghani et al. [38] leads to $O(N)$ communication complexity, and is subsumed by the more efficient approach in [39]. The latter employs a divide-and-conquer algorithm; the sensors are recursively divided into smaller groups, and SHIA is re-initiated in each new group. Eventually, a group contains a single sensor, and the algorithm can identify if this sensor is malicious.

Summary: An Important Research Gap Despite the attention given to the topic of secure in-network aggregation, very few approaches (namely, [29], [4], [30], [31], [32]) can support *both* integrity and confidentiality, while returning *exact* answers. Nevertheless, observe that *all* these schemes make some strong assumption about the trustworthiness of the aggregators. Although this could be viable in Scenario 1, it is not realistic in Scenario 2 where the entire aggregation infrastructure may be malicious. Furthermore, these methods fail to handle *both* sum-based and quantile queries. The above facts constitute the main motivation behind our work, and justify the importance of our contribution.

3 Problem Definition

In this section we rigorously define our problem. We describe in turn: (i) the underlying architecture, (ii) the query model, (iii) the security objectives, (iv) the performance objectives, and (v) additional issues we consider as orthogonal.

System architecture We assume the architecture described in Section 2.1. We denote a source as $\mathcal{S}$, an aggregator as $\mathcal{A}$, and the querier as $\mathcal{Q}$. The querier must either be the *owner* of the sources, or an authorized entity that possesses all the necessary keys. Finally, the sources are regarded as *trusted* and store key materials, but the aggregators are *not* and should not be given any sensitive data.

Query model We consider the *push-based* data collection model explained in Section 2.1. We design schemes for answering *exact* SUM, COUNT, AVG, VARIANCE, STDDEV, MIN, MAX, MEDIAN, and general q-QUANTILE. Without loss of generality, we assume that all data values are positive integers (we can always encode other types as positive integers via translation/scaling operations [8]).

Security objectives The adversary may either compromise a sensor node (source or aggregator) and thus take its full control, or eavesdrop in the wireless channel. We aim at providing strong *cryptographic security*. Our primary goal is to satisfy data confidentiality, integrity, authentication, and freshness, as they were defined in Section 1. Our secondary objective is to identify the potential malicious nodes. In the following, we first elaborate on our goals and assumptions concerning specifically integrity and confidentiality, and then explain the rationale of our malicious node identification scheme.

For data integrity, we mandate *detection* of any alteration of the result *during the aggregation process*. This is equivalent to the *optimal security* defined in [21], which essentially protects only against compromised/malicious *aggregators*. In case the adversary compromises a *source*, then it can extract the stored key materials, and report a *fake* value by properly participating in the protocol. The querier would then admit the result as correct, without detecting the malicious activity. Similar to [21], we cannot tackle this situation and integrity is always violated.

Regarding confidentiality, we consider two cases. If no source is compromised, an eavesdropper should neither be able to infer the individual readings of the sources, nor the partial/final aggregation results. On the other hand, if a source is compromised, we must limit the information the adversary can discover based on the query. Specifically, in distributive queries, we will show that the adversary learns neither the readings of other sensors, nor the partial/final results. In the non-distributive queries though, although it can still discover nothing about the readings of other sensors, it is able to constrain the final result within a certain range. We state that this is unavoidable following the philosophy of our scheme, and pose it as an open problem.

Finally, the motivation behind our malicious node identification scheme is to tackle certain *active DoS attacks*, which prevent the querier from receiving the final result. Our technique enables the querier to locate the corrupted nodes in the network, so that it can physically replace them. This functionality enhances the system robustness.

Performance objectives We target at achieving (i) low computational cost at the sources, the aggregators and the querier, (ii) low communication cost, which we measure as the consumed bandwidth *per network edge*, and (iii) low query delay (i.e., time that elapses from the instant the sources sense their reading, to the instant the querier receives the final result). Note that we make no strong assumptions on the computational capabilities of the sensors (e.g., unlike [8], [24]). On the contrary, we pay particular attention so that the entailed computations are lightweight and, thus, suitable for resource-constrained sensor networks.

Orthogonal issues The construction and fine-tuning of the tree topology, as well as its re-organization due to node failures, are issues orthogonal to our work. Moreover, we do not address access control at the querier. Additionally, we do not try to tackle *passive DoS attacks*, e.g., when a compromised sink does not report at all the result within one or more time epochs. This attack is trivially detected if the querier does not receive any data. Finally, we do not seek to protect against physical manipulation of the sources, e.g., when the adversary places heaters nearby sensors measuring temperatures to alter the real readings.

4 Answering SUM Queries - SIES

Section 4.1 contains the building blocks of SIES. Section 4.2 presents the SIES protocol. Section 4.3 proves the security of SIES. Table 1 summarizes the notation used throughout the section.

4.1 Building Blocks

SIES is based on a combination of a symmetric additively homomorphic scheme and a simple secret sharing technique. Below we describe in detail these two basic components.

Symmetric additively homomorphic scheme Let p be a prime, $m_i < p$ the message to be encrypted, and K, k_i two random secret keys with $K \neq 0$ and $K, k_i < p$. We define encryption as

$$c_i = \mathcal{E}(m_i, K, k_i, p) = (K \cdot m_i + k_i) \mod p$$

and decryption as

$$m_i = \mathcal{D}(c_i, K, k_i, p) = [(c_i - k_i) \cdot K^{-1}] \mod p$$

TABLE 1
Summary of Symbols

Symbol	Meaning
$\mathcal{S}/\mathcal{A}/\mathcal{Q}$	Source/Aggregator/Querier
N	Number of sources
K	Key known to $\mathcal{Q}$ and every source
k_i	Key known to $\mathcal{Q}$ and $\mathcal{S}_i$
p	Public prime modulus
t	Time epoch
K_t	Key generated by all sources at epoch t
$k_{i,t}$	Key generated by $\mathcal{S}_i$ at epoch t
$ss_{i,t}$	Secret share generated by $\mathcal{S}_i$ at epoch t
$v_{i,t}$	Value generated by $\mathcal{S}_i$ at epoch t
$m_{i,t}$	Plaintext of $\mathcal{S}_i$ to be encrypted at epoch t
$PSR_{i,t}$	PSR generated by $\mathcal{S}_i$ at epoch t
s_t	Secret verifiable by $\mathcal{Q}$ at epoch t
res_t	SUM result at epoch t
$HM_1(\cdot)$	HMAC implemented with SHA-1
$HM_{256}(\cdot)$	HMAC implemented with SHA-256

where K^{-1} is the multiplicative inverse of K modulo p. Note that K^{-1} always exists since p is prime.

Now consider two ciphertexts c_1 and c_2 corresponding to plaintexts m_1 and m_2, respectively. Observe that we can compute the encryption of SUM $m_1 + m_2$ as

$$\begin{aligned} c_1 + c_2 &= \mathcal{E}(m_1, K, k_1, p) + \mathcal{E}(m_2, K, k_2, p) = \\ &= [K \cdot (m_1 + m_2) + (k_1 + k_2)] \mod p = \\ &= \mathcal{E}(m_1 + m_2, K, k_1 + k_2, p) \end{aligned}$$

which can be decrypted using keys K and $k_1 + k_2$ as

$$m_1 + m_2 = \mathcal{D}(c_1 + c_2, K, k_1 + k_2, p)$$

In general, $\Sigma_{i=1}^{N} m_i$ can be extracted from $\Sigma_{i=1}^{N} c_i$ using keys K and $\Sigma_{i=1}^{N} k_i$ in the decryption function. In the sequel, $\mathcal{E}(\cdot)$ and $\mathcal{D}(\cdot)$ refer to the encryption and decryption functions of our homomorphic scheme, respectively. Observe that this type of encryption is secure in an *information-theoretic* sense, i.e., even against a computationally unbounded adversary. This holds since lacking knowledge of k, the value $\mathcal{E}(m, K, k, p)$ preserves no information whatsoever about m (for any value of K, p).

Secret sharing [16] Let s be a *secret*. Suppose that we wish to distribute s amongst N parties, in a way such that s can be re-constructed only when *all* N parties contribute. We first generate $N-1$ random values $ss_1, ss_2, \ldots, ss_{N-1}$, and distribute one ss_i to each party except for one. We then set $ss_N = s - \Sigma_{i=1}^{N-1} ss_i$ and give it to the last party. Each ss_i value is called a *secret share*. The secret is then equal to $s = \Sigma_{i=1}^{N} ss_i$. Observe that the adversary cannot compute s without knowing all N secret shares. This simple secret sharing technique is secure in an *information-theoretic* sense.

Homomorphic encryption enables the aggregators to perform aggregation directly on ciphertexts through its additive property, thus achieving data confidentiality. It should be noted that this scheme cannot guarantee the integrity of the aggregation result by itself. For example, a compromised aggregator may trivially drop the ciphertext from any source without being detected. We overcome this problem by incorporating secret shares into the plaintext values to be encrypted. The querier can then verify that all the ciphertexts have been involved in the aggregation process and no spurious ones have been added, by extracting the complete secret from the final ciphertext.

4.2 Protocol

The aggregation process consists of three phases: the *initialization phase* I, the *merging phase* M, and the *evaluation phase* E. I takes place at each source and operates on the generated raw data. The output is a partial state record (PSR), which integrates the raw data with other security information. M takes place at each aggregator; it combines the PSRs received from its children into a single one, which is subsequently forwarded to the respective parent. Finally, E occurs at the querier, and has as input a single PSR collected from the sink; it extracts the final aggregation result form the PSR, and verifies its correctness. Figure 2 illustrates a simple example architecture.

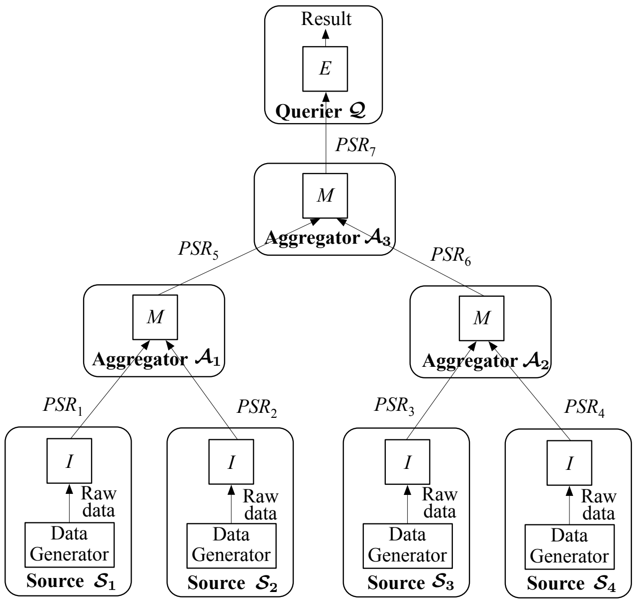


Fig. 2. SIES example architecture

Setup phase The querier $\mathcal{Q}$ first generates random keys K, and $k_1, k_2, \ldots, k_N$, each having an appropriate size that diminishes the probability of a random guess (e.g., at least 20 bytes). $\mathcal{Q}$ also produces an arbitrary prime p, which is used as the modulus of our homomorphic encryption scheme. As we shall see, in our implementation the size of p is 32 bytes. Subsequently, it *manually* registers (K, k_i, p) to every source $\mathcal{S}_i$, and provides each aggregator $\mathcal{A}_j$ with p. Observe that K is commonly known to all sources. Nevertheless, k_i is only known by source $\mathcal{S}_i$. Finally, $\mathcal{Q}$ issues the continuous query to the system. To do so, it broadcasts the query in an authenticated way with μTESLA [18]. After the sources receive the query, the aggregation process commences. Whenever $\mathcal{Q}$ wishes to change the query, it simply broadcasts a new query with μTESLA in the network, without re-establishing any keys.

Initialization Phase Let t be the current time epoch. Every source $\mathcal{S}_i$ first generates its data value $v_{i,t}$ (involved in the aggregation query), which is 4 bytes long. Moreover, it computes (pseudo-) random key $K_t = HM_{256}(K, t)$, using the HMAC PRF $HM_{256}(\cdot)$, which is implemented with SHA-256. In addition, $\mathcal{S}_i$ generates $k_{i,t} = HM_{256}(k_i, t)$. Subsequently, it calculates *secret share* $ss_{i,t} = HM_1(k_i, t)$, where $HM_1(\cdot)$ is the HMAC PRF that uses SHA-1. K_t and $k_{i,t}$ are 32 bytes long, whereas $ss_{i,t}$ is 20 bytes long. Note that K_t is known to all sources, whereas $k_{i,t}$ is only known by $\mathcal{S}_i$. Moreover, K_t, $k_{i,t}$, and $ss_{i,t}$ are *temporal*, as they all depend on t. As we shall see, this is important for providing data freshness. Next, $\mathcal{S}_i$ produces a binary message $m_{i,t}$ with the form depicted in Figure 3.

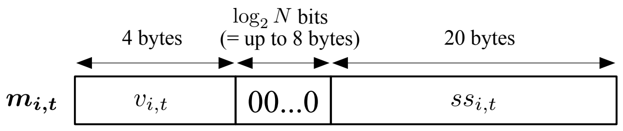


Fig. 3. Format of $m_{i,t}$ in SUM queries

The purpose of adding $\log_2 N$ zeros in $m_{i,t}$ will be clarified soon. Finally, $\mathcal{S}_i$ creates a PSR as $PSR_{i,t} = \mathcal{E}(m_{i,t}, K_t, k_{i,t}, p)$, and sends it to its parent aggregator. Since the size of p is determined by the size of K_t and $k_{i,t}$, the resulting ciphertext is 32 bytes long.

It is important to stress that *every* source must create and send a message m as above, whether it satisfies the query condition pred or not; if it does not satisfy it, it simply incorporates a zero v value into m, which does not influence the final SUM result. The reason for the above is twofold: (i) it proves that the source does not contribute to the result, obviating the need for explicitly informing the querier about which source replied to the query and, thus, avoiding the problems of [30] (see Section 2.3), and (ii) as an additional privacy benefit, an eavesdropper will never know if a source satisfies the query or not.

Merging Phase During this phase, an aggregator $\mathcal{A}_i$ receives the PSRs from its children and combines them into a single one, which is then forwarded to its parent. Supposing that $\mathcal{A}_i$ receives $PSR_{1,t}$ and $PSR_{2,t}$, it simply computes the new PSR as $PSR'_t = PSR_{1,t} + PSR_{2,t} \mod p$ (recall that $\mathcal{A}_i$ possesses p). The resulting PSR is also 32 bytes long.

Evaluation Phase Eventually, $\mathcal{Q}$ receives a single final PSR in time epoch t, denoted by $PSR_{f,t}$, which represents the modular addition of all the PSRs generated by the sources. It then computes $m_{f,t} = \mathcal{D}(PSR_{f,t}, K_t, \Sigma_{i=1}^{N} k_{i,t}, p)$. Note that $\mathcal{Q}$ can calculate K_t and all $k_{i,t}$ because it possesses K and all k_i. Due to the homomorphic property of our scheme, $m_{f,t}$ is equal to the sum of all $m_{i,t}$ produced by the sources.

Consequently, the first 4 bytes of $m_{f,t}$ constitute the result (res_t) of the SUM query[1]. The remaining $(\frac{\log_2 N}{8} + 20)$ bytes represent the secret $s_t = \Sigma_{i=1}^{N} ss_{i,t}$. Due to overflow during the summation, the extra bits required cannot be more than $\log_2 N$ when N numbers are added. This justifies our choice to pad $\log_2 N$ zeros before $ss_{i,t}$ in every $m_{i,t}$. Moreover, observe that our implementation can support up to $N = 2^{64}$ sources, since the padding in $m_{i,t}$ can be up to 8 bytes (so that the size of $m_{i,t}$ does not exceed 32 bytes). Figure 4 shows $m_{f,t}$ retrieved by $\mathcal{Q}$ for the topology depicted in Figure 2, where $PSR_{i,t}$ was generated by $\mathcal{S}_i$.

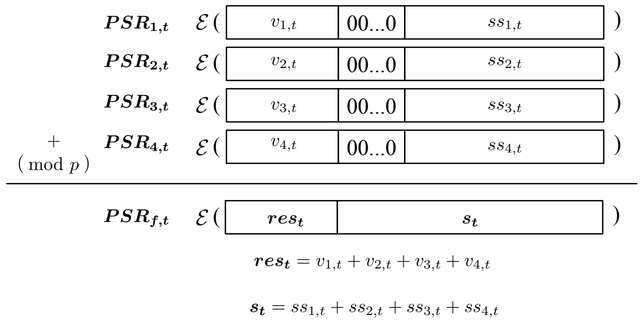

$PSR_{1,t}$ $\mathcal{E}($ $v_{1,t}$ 00...0 $ss_{1,t}$ $)$
$PSR_{2,t}$ $\mathcal{E}($ $v_{2,t}$ 00...0 $ss_{2,t}$ $)$
$PSR_{3,t}$ $\mathcal{E}($ $v_{3,t}$ 00...0 $ss_{3,t}$ $)$
$+$ (mod p) $PSR_{4,t}$ $\mathcal{E}($ $v_{4,t}$ 00...0 $ss_{4,t}$ $)$
$PSR_{f,t}$ $\mathcal{E}($ res_t s_t $)$
$res_t = v_{1,t} + v_{2,t} + v_{3,t} + v_{4,t}$
$s_t = ss_{1,t} + ss_{2,t} + ss_{3,t} + ss_{4,t}$

Fig. 4. Example aggregation in SIES

$\mathcal{Q}$ first extracts the result res_t and value s_t as explained above. Subsequently, it computes the secret share $ss_{i,t}$ of each source $\mathcal{S}_i$ as $HM_1(k_i, t)$. Next, it derives $\Sigma_{i=1}^{N} ss_{i,t}$. Finally, $\mathcal{Q}$ confirms the integrity and freshness of res_t, if and only if the $\Sigma_{i=1}^{N} ss_{i,t}$ value it computed is equal to the extracted s_t from $PSR_{f,t}$. To summarize, (i) confidentiality is ensured through our additively homomorphic scheme, which at the same time allows efficient aggregation, (ii) integrity is guaranteed through the embedding of the secret shares into the PSRs, which eventually sum up to a secret verifiable by the querier, and (iii) freshness is provided because the keys and shares used by the sources integrate t, which is different for different epochs.

4.3 Security

In this section we explain the security of SIES against various attacks in terms of data confidentiality, integrity, authentication, and freshness.

Data confidentiality We distinguish two scenarios: (i) the adversary does not compromise any source and simply eavesdrops the wireless channel, and (ii) the adversary compromises at least one source. Note that compromising an aggregator is equivalent to eavesdropping the channel, since the aggregators do not possess any keys and do not perform encryption. We focus on the case where the PSR is generated by a source, since the case where the PSR is a result of aggregation can be proven similarly.

In the first scenario, the adversary possesses neither K_t nor $k_{i,t}$, whereas in the second scenario it obtains global key K_t from a compromised source $\mathcal{S}_j$, with $k_{i,t}$ though remaining unknown for every $i \neq j$. In order to prove data confidentiality, it suffices to focus only on the second scenario, since the first scenario is more difficult for the adversary to attack.

Our attacker is a probabilistic polynomial-time (PPT) *eavesdropper adversary*, who can only *observe* the ciphertexts, without being able to request encryptions of plaintexts of its choice. The reason is that every $k_{i,t}$ is used to encrypt only *one* plaintext, since it changes in every epoch, and all keys are *unique* with an immense probability (because they are pseudorandomly drawn from a huge domain).

A scheme is secure against such an adversary if, given two plaintexts m_0, m_1 and a ciphertext c, the adversary can distinguish whether c encrypts m_0 or m_1 with probability negligibly higher than $\frac{1}{2}$ (i.e., than a random guess). We refer the reader to [40] for the rigorous definitions of what we discussed here.

Theorem 1: SIES has indistinguishable encryptions in the presence of a PPT eavesdropper adversary.

Proof: The encryption technique in SIES is a slight modification of the scheme described in Construction 3.15 in [40], which has indistinguishable encryptions in the presence of a PPT eavesdropper adversary. The only difference is that we use modular addition instead of the XOR operator (noting that we implement the pseudorandom generator with HMAC). It is easy to see that this modification does not negate the security of Construction 3.15, since the modular addition of a pseudorandom number to a plaintext has the same probabilistic effect as the XOR operator. □

The above discussion implies that K_t is not necessary for providing confidentiality, since the latter is satisfied solely by key $k_{i,t}$. Below we explain that K_t is important for guaranteeing data integrity.

Data integrity Recall from Section 3 that we focus on the case where the adversary does not compromise *any* source. We say that SIES guarantees data integrity, if the querier admits a PSR incorporating a *false* aggregation result as legitimate with *negligible probability*. For ease of exposition, we simply state that any probability smaller than 2^{-128} is negligible [40][2].

Theorem 2: SIES satisfies data integrity in the absence of a compromised source.

Proof: Deflation of the result (i.e., in case the adversary drops some PSRs) is directly tackled by the secret sharing scheme; if a share is missing, the secret verification fails. In the sequel, we focus on result *inflation*. In order to inflate $PSR_{f,t}$ without getting caught, the adversary must

1. We consider here that the final result cannot exceed $2^{32}-1$. However, if the application requires longer numbers we can use an 8-byte field in $m_{i,t}$ during the initialization phase.

2. We note that this is a *non-standard* definition, as we call negligible a *function* (with respect to some security parameter, such as the key/share size) that grows smaller than any inverse polynomial for sufficiently large values. Although our scheme complies also with the standard definition, we prefer a more intuitive security proof in this work.

add a value of the form $X = (v' \cdot 2^{224}) \cdot K_t$. This results in adding to the respective plaintext $m_{f,t}$ a 32-byte value with v' in the most significant bits and the rest of the bits as zeros. This injection does not alter the final secret. Result res_t is then inflated by v', while verification succeeds. Since K_t is unknown, produced with a PRF and used once, the adversary can only *guess* an X value with the above property. The probability that the adversary finds such an X is then at most $\frac{2^{32}}{2^{256}} = 2^{-224}$ (note that we do not consider sum overflows), which is *negligible*. Moreover, observe that we can satisfy any constant definition of negligibility by simply increasing the size of key K_t. □

Data authentication In another attack, the adversary could *impersonate* $\mathcal{Q}$ during the dissemination of the query, and provide the sources with a false query (which has a different result than the desired one). In this case, $\mathcal{Q}$ would accept the final collected result as correct, since the actual aggregation procedure is not altered.

SIES is secure against querier impersonation. This is directly ensured by the μTESLA protocol (Section 2.2), which enables each source to verify that the message (i.e., the query) indeed originated from $\mathcal{Q}$. In addition, note that source impersonation is covered by data integrity.

Data freshness A result is *fresh* if it reflects the current time epoch t. We say that an adversary violates data freshness if it presents to the querier *any* legitimate final PSR $PSR_{f,t'}$, which however corresponds to a *previous* time epoch t'. This is called a *replay attack*. We say that SIES satisfies data freshness, if the adversary can mount a replay attack with negligible probability.

Theorem 3: SIES provides data freshness.

Proof: Let $PSR_{f,t}$ be the legitimate final PSR at current epoch t. The adversary succeeds in breaking freshness, if it can find a legitimate PSR that encrypts secret $s_{t'}$ at time $t' < t$, such that $s'_t = s_t$. In order for this to happen (i) there must exist such a PSR among all the possible ℓ legitimate PSRs created before t and (ii) the adversary successfully identifies this PSR (recall that s_t and $s_{t'}$ are encrypted). Due to the birthday paradox [40], the probability for (i) to occur is at most $\frac{\ell^2}{2 \cdot 2^{160}}$, whereas for (ii) it is $\frac{1}{\ell}$ (by guessing). Therefore, the adversary attacks freshness with probability at most $\frac{\ell}{2^{161}}$. Even with $\ell = 2^{33}$, this probability is negligible. We can satisfy any constant definition of negligibility by increasing the secret share size. □

Discussion A final remark concerns *node failures*, i.e., situations where either a source does not produce a PSR or an aggregator fails to fuse its children's PSRs in a time epoch, due to an internal problem. In this case the failed node must be reported to the querier. However, $\mathcal{Q}$ must also manually check the corresponding node, since a compromised node may falsely report the failure. Then, during result verification, $\mathcal{Q}$ produces $s_t = \Sigma_i ss_{i,t}$ considering only the shares of the sources contributing to the result.

5 Answering Queries Beyond SUM

So far we have explained how to answer exact SUM queries with SIES. Here we discuss how to adapt SIES in order to support a wider range of aggregate queries, namely COUNT, AVG, VARIANCE, STDDEV, MIN, MAX, MEDIAN, and general q-QUANTILE.

Count COUNT queries can be trivially handled by directly using the SIES protocol without any alteration; every source incorporates into its message m (to be encrypted) a value $v = 1$ if it satisfies the query predicate, and $v = 0$ otherwise. Clearly, this approach inherits the performance and security properties of SIES.

Average It holds that AVG = $\frac{\text{SUM}}{\text{COUNT}}$. Therefore, we could run SIES independently for SUM and COUNT, and then combine the two results to yield AVG. Since SIES is executed *twice*, the communication and computational cost at all parties involved *doubles*. In the following, we present an alternative solution that enables AVG computation with a *single* execution of SIES.

Every source $\mathcal{S}_i$ creates a slightly different message $m_{i,t}$ in epoch t than in Figure 3 (Section 4.2), whose format is illustrated in Figure 5. $m_{i,t}$ now contains (from right to left) a 20-byte secret share $ss_{i,t}$, 4 bytes of zeros, and two values $v^C_{i,t}$ and $v^S_{i,t}$; the former (latter) value is what $\mathcal{S}_i$ would transmit in the COUNT (SUM) protocol. Observe also that the number of zeros implies that the system can now afford $N = 2^{32}$ sources instead of 2^{64}, which is though still sufficient under any practical application.

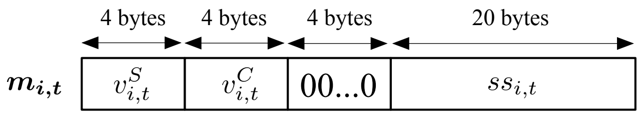


Fig. 5. Format of $m_{i,t}$ in AVG queries

The rest of the SIES protocol is executed *once* without modifications. Eventually, $\mathcal{Q}$ extracts a final message $m_{f,t}$ which contains (from right to left) a 24-byte secret, a 4-byte COUNT result, and a 4-byte SUM result (always assuming no overflows in the results). $\mathcal{Q}$ can then compute AVG using the two results, and verify integrity through the secret. It is easy to see that the slight modification in the format of $m_{i,t}$ does not negate the correctness of any argument and proof included in Section 4.3.

Variance/Standard deviation Observe that VARIANCE = $\frac{\text{SUM}'}{\text{COUNT}} - (\frac{\text{SUM}}{\text{COUNT}})^2$, where the result of SUM′ is $\Sigma_{i=1}^N v^2_{i,t}$. Consequently and following the technique we described above for AVG, we need to alter the format of $m_{i,t}$ in a way such that the SIES protocol eventually produces a final $m_{f,t}$ incorporating SUM′, SUM, and COUNT. Given this information, $\mathcal{Q}$ can compute VARIANCE as above. Moreover, $\mathcal{Q}$ can calculate STDDEV simply as the square root of VARIANCE.

We next describe the format of $m_{i,t}$ when answering VARIANCE queries, which is depicted in Figure 6. The message is now 48 bytes long (the reason will become clear soon). From right to left it encompasses a 20-byte secret share, 16 bytes of zeros, and three 4-byte values $v^C_{i,t}$, $v^S_{i,t}$, and $(v^S_{i,t})^2$; $v^C_{i,t}$ and $v^S_{i,t}$ are defined as above for `AVG`, whereas $(v^S_{i,t})^2$ is simply the square of $v^S_{i,t}$.

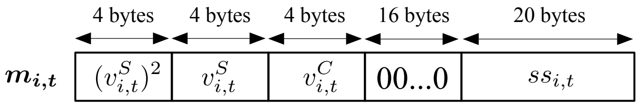


Fig. 6. Format of $m_{i,t}$ in `VARIANCE` queries

The SIES protocol is then slightly modified such that all the keys are 48 bytes long. To generate the keys, the querier employs the SHA-384 function $HM_{384}(\cdot)$ as the basis of the HMAC PRF, which produces 48-byte digests. This length is the next smallest after the 32-byte digests produced by SHA-256 in our original scheme, which unfortunately were not enough to contain the three desired values. The rest of the protocol remains the same, and $\mathcal{Q}$ eventually extracts an $m_{f,t}$ that contains (from left to right) the result of `SUM'`, `SUM` and `COUNT`, as well as the final secret. The security of the scheme still holds (in fact, it is even strengthened due to the larger keys).

Quantiles All the previous queries can be answered executing SIES only *once*. Unfortunately, this is not possible for quantile queries, which include `MEDIAN`, `MIN`, `MAX`, and general `q-QUANTILE`. To solve these queries, we will apply SIES and the μTESLA [18] authenticated broadcasting scheme (see Section 2.2) to the quantile computation approach described in Section 2.1.

Recall that in every round of this protocol, the querier must broadcast a pair of ranges. We assume that $\mathcal{Q}$ employs μTESLA to do so, which guarantees the integrity of the query. To ensure confidentiality, $\mathcal{Q}$ and the sources share a key k^Q. For epoch t and round j of the protocol, $\mathcal{Q}$ computes key $k^Q_{t,j} = HM_1(k^Q, t||j)$, and encrypts the query ranges using this key (e.g., through the simple variant of the one-time pad discussed in Section 2.2). Every source can also compute $k^Q_{t,j}$ and efficiently decrypt the query.

In addition, in every round a source $\mathcal{S}_i$ must forward a message to the network, based always on its reading and the query ranges of this round. $\mathcal{S}_i$ creates a message $m_{i,t}$ with the format of Figure 5, where values $v^C_{i,t}$ and $v^S_{i,t}$ are now substituted by $v^L_{i,t}$ and $v^R_{i,t}$. It sets $v^L_{i,t} = 1$ ($v^R_{i,t} = 1$) if the reading of $\mathcal{S}_i$ falls in the left (right) range collected in this round, and $v^L_{i,t} = 0$ ($v^R_{i,t} = 0$) otherwise. Finally, all parties follow the SIES protocol and the round concludes. No other modification is performed. $\mathcal{Q}$ eventually computes the desired quantile result as described in Section 2.1.

We next analyze the security of the above scheme. During the message forwarding from the sources to the querier, every party follows SIES and, therefore, all the security guarantees are inherited. During the phase of range broadcasting though, we must heed a small subtlety. In case there is no compromised source, the query confidentiality, integrity, authentication and freshness are protected. However, if an adversary hacks a source, it can retrieve k^Q and decrypt all ranges sent by the querier. Consequently, although the confidentiality of the rest of the messages in the network is protected, the adversary will be able to learn the final range the quantile falls into. This is the only weakness of our scheme, which we pose as an open problem and the focus of our future work.

6 Malicious Node Identification

Objective and basic idea So far we have explained that we can *detect* an integrity violation through SIES, which is the building brick of all described queries. In this section we present an additional functionality that *identifies* misbehaving sensors. Our functionality is *generic* and can be integrated also in other in-network aggregation schemes.

A misbehaving sensor aims at launching the following *active denial-of-service* (DoS) attack. It attempts to forward *falsified* PSRs into the system, such that (i) the querier $\mathcal{Q}$ never receives the correct aggregation result, and (ii) the sensor avoids being identified as the malicious one. Clearly, if such a DoS attack is not prevented, misbehaving sensors can disturb the aggregation process indefinitely. Note that we do not consider *passive* DoS attacks (e.g., when a misbehaving sensor simply shuts down), since the querier can trivially detect these attacks and handle them in a similar manner to sensor failures.

Recall from Section 3 that the source sensors are considered as trusted, and SIES cannot detect an integrity violation on their part if they are compromised. Therefore, here we focus only on *misbehaving aggregators*. Moreover, a misbehaving aggregator is a result of *physical corruption* in both Scenarios 1 and 2 (note that, in Scenario 2, it does not make sense for the service provider to *intentionally* install misbehaving aggregators launching such a DoS attack, since this would contradict the motivation of outsourcing).

The goal of our functionality is to *identify* the corrupted aggregators, so that it is possible for the querier to physically replace them with properly functioning ones. The main idea is that, whenever an integrity violation is detected in some epoch, the querier *challenges* the aggregators one by one and top-down. Every challenged aggregator must prove to the querier that it did not cheat, by presenting proofs that its descendants cheated instead.

Algorithm and example Before embarking on the description of the identification algorithm, we must perform a minor modification on the SIES scheme. We assume that every sensor possesses a public and private RSA key pair [14]. The private key is known only to the sensor, whereas the public key is stored at the parent of the sensor and at the querier. Upon forwarding a PSR to its parent, every sensor *signs* this PSR with the RSA digital signature cryptosystem, and transmits the signature along with the PSR. The parent is responsible for verifying this signature before proceeding in the SIES protocol, and for reporting the children that presented a non-verifiable signature to the querier. Moreover, the parent *temporarily caches* the children's PSRs (typically for a few milliseconds), until it is certain that the querier has not detected an integrity violation in this epoch. Note that, although the public-key operations add an extra overhead to the system, they are vital for the identification scheme. As such, our functionality achieves a *trade-off* between performance and robustness. Figure 7 presents the identification algorithm.

```
1. Set $L = \{\mathcal{A}_r\}$, where $\mathcal{A}_r$ is the root aggregator
2. While ($L \neq \emptyset$)
3.   Pick any $\mathcal{A}_i \in L$ and remove it from $L$
4.   Get the PSRs and signatures of $\mathcal{A}_i$'s children
5.   Verify the PSRs using the signatures and secret shares
6.   If a signature fails or all the PSRs are verified
7.     $\mathcal{A}_i$ is malicious and should be replaced
8.   Else
9.     Insert $\mathcal{A}_i$'s "malfunctioning" children in $L$
```

Fig. 7. The malicious node identification algorithm

We explain the procedure utilizing the example of Figure 8, where $\mathcal{A}_3$ is a corrupted aggregator in some epoch who sends a falsified PSR_3. The querier initiates the identification algorithm as soon as it detects the integrity violation, and executes line 1. Next, it requests PSR_5 and PSR_6 from $\mathcal{A}_7$, along with their signatures (lines 2-4). Suppose that the signatures of these PSRs are verified successfully using the public keys of $\mathcal{A}_5$ and $\mathcal{A}_6$ (line 5). The querier then discovers that PSR_5 is valid (using the secret shares), but PSR_6 is problematic. Subsequently, $\mathcal{Q}$ adds $\mathcal{A}_6$ in L (lines 8-9) and returns to line 2.

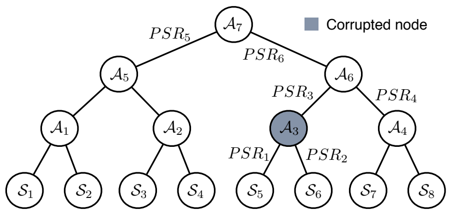


Fig. 8. Illustrating the identification scheme

The procedure continues similarly. $\mathcal{Q}$ extracts $\mathcal{A}_6$ from L, requests PSR_3 and PSR_4, discovers that PSR_3 is incorrect, and inserts $\mathcal{A}_3$ into L. Finally, it retrieves PSR_1 and PSR_2, identifies that both these PSRs are correct, and concludes that $\mathcal{A}_3$ is the malicious node (lines 6-7).

Note that the use of the RSA signatures is necessary. In every step of the algorithm, the querier essentially *challenges* an aggregator about its integrity. In order for the aggregator to convince the querier that it did not cheat, it must present the PSRs of its children along with the corresponding RSA signatures. These signatures constitute a *commitment* of the children that they indeed sent these PSRs during the SIES protocol. Recall also that the aggregator verifies these signatures upon collection, and must report a violation to the querier immediately (otherwise, the aggregator is held responsible for not doing so and is reported as malicious in lines 6-7). Therefore, it is impossible for a child to claim that it did not send a falsified PSR during the SIES protocol. Without the RSA signatures, a child could send an initially incorrect PSR to attack integrity, and then present the correct PSR upon the identification algorithm. In that case, the querier would not be able to locate the malicious aggregator.

Finally, observe that the algorithm can handle even *multiple* malicious aggregators. For instance, in our example, if $\mathcal{A}_5$ was also malicious, the algorithm would include it in L along with $\mathcal{A}_6$ in the first loop iteration. It would then dedicate a separate loop for this aggregator, retrieving the PSRs of $\mathcal{A}_1$ and $\mathcal{A}_2$, discovering that both are valid and, thus, identifying $\mathcal{A}_5$ as a malicious node. The only complication arises when two malicious aggregators appear on *the same path*, e.g., $\mathcal{A}_3$ and $\mathcal{A}_6$. In this case, the querier would identify the deepest node (i.e., $\mathcal{A}_3$) first, without being able to locate $\mathcal{A}_6$ at the same time. The querier would have to identify $\mathcal{A}_6$ in a subsequent run, after $\mathcal{A}_3$ is replaced and another integrity violation is detected.

7 Experiments

Competitors Recall that there is no direct competitor to our work, as no solution can provide in-network processing of exact aggregate queries satisfying both confidentiality and integrity without making strong assumptions. Nevertheless, we select [5] (hereafter referred to as CMT after its authors' initials) and SECOA [8] as benchmark solutions to assist our experimental evaluation, although they offer only partial solutions to our targeted problem. Specifically, CMT is the state-of-the-art method for confidentiality. This method is extremely lightweight because it is based on simple and cheap modular operations. However, its simplicity and performance come with the cost of sacrificing data integrity. SECOA is the state-of-the-art technique for integrity, which is more scalable than the commit-and-attest methods. Nevertheless, it does not provide confidentiality, and handles only approximate `SUM` queries (it is primarily designed for exact `MAX` queries).

Setup We implemented all methods in C++ using the GNU MP[3] and OpenSSL[4] libraries. We experimented with real dataset Intel Lab[5], which contains (among other data) sensor temperature readings (in degrees Celcius) represented as float numbers with precision of four decimal digits. Each source generates values v that are randomly drawn from the above dataset and fall in the range $[18, 50]$. The sources and aggregators form a complete tree.

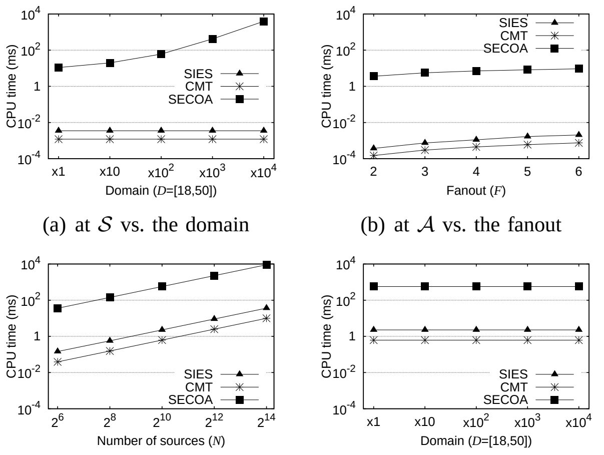
Since we do not possess a real wireless sensor network infrastructure, similar to the majority of the previous work on the topic (including our competitors), we performed our experiments on a single machine (specifically, a 2.66 GHz Intel Core i7 with 4GB RAM, running Mac OS X ver. 10.6.4). Although this hardware is much more powerful than that of a sensor, it demonstrates the relative performance of the various methods. Moreover, as we will show soon, our scheme incurs very small CPU and

3. `http://gmplib.org/`
4. `http://www.openssl.org/`
5. `http://db.csail.mit.edu/labdata/labdata.html`

bandwidth costs; therefore, it can perform exceptionally even on sensors that are several orders of magnitude slower than our processor. Finally, since the scheme involves non-hidden costs, heuristics or parameters, its implementations in different systems would yield consistent performance.

Sum-based queries We measure the costs of sum-based queries, varying: (i) the number of sources (N), (ii) the fanout of the aggregators (F), and (iii) the dataset domain. In order to vary D, each source multiplies its drawn value with powers of 10, and then truncates it (i.e., D takes values $[18, 50]$, $[180, 500]$, etc.). Scaling the domain in this manner is equivalent to changing the decimal precision of the readings and, thus, of the result (the querier divides the extracted integer result with the respective power of 10 to derive the final float result). Following [8], we fix the number of sketch instances of SECOA to 300, in order to bound the relative approximation error within 10% with probability 90%. We ran every experiment over 20 epochs and reported the average cost per epoch.

We first focus on `SUM` queries, and later discuss the effect of the rest of the sum-based queries on performance. Figure 9(a) shows the CPU time at the source $\mathcal{S}$ as a function of D, when $N = 1024$ and $F = 4$. SIES outperforms SECOA by more than two orders of magnitude. The reason is that SIES involves few and cheap HMAC operations and modular additions, whereas SECOA involves generating an excessive number of sketches and performing several RSA encryptions. SIES also retains a comparable performance to CMT, which is in the order of few microseconds. Furthermore, contrary to SECOA, the computational cost of SIES and CMT is independent of D.



(a) at $\mathcal{S}$ vs. the domain
(b) at $\mathcal{A}$ vs. the fanout
(c) at $\mathcal{Q}$ vs. the number of sources
(d) at $\mathcal{Q}$ vs. the domain

Fig. 9. Computational cost of `SUM` queries

Figure 9(b) demonstrates the CPU time at the aggregator $\mathcal{A}$ when varying its fanout F, and setting $N = 1024$ and $D = [1800, 5000]$. Once again, SIES outperforms SECOA by approximately two orders of magnitude, because $\mathcal{A}$ must perform several RSA encryptions in SECOA, whereas it performs $F - 1$ modular additions in SIES. The CPU time in SIES is in the order of microseconds and marginally close to that of CMT. As expected, the overhead of all solutions linearly increases with the fanout.

Figure 9(c) depicts the computational time at the querier $\mathcal{Q}$ versus N ($F = 4$, $D = [1800, 5000]$). This overhead is linearly dependent on N in all methods. SIES outperforms SECOA by more than one order of magnitude on all values. This happens because in SECOA the querier performs numerous RSA encryptions and modular multiplications during verification. On the other hand, SIES involves only the computation of the keys and shares with the efficient HMAC, plus a number of cheap modular additions. The CPU time in SIES is 0.15-36 ms. Additionally, the performance of SIES is comparable to that of CMT. Their difference mainly stems from the fact that in SIES the querier must also compute the shares that are used for integrity verification, a process missing from CMT.

In Figure 9(d) we assess the CPU time at $\mathcal{Q}$, varying D ($N = 1024$, $F = 4$). The overhead in SIES and CMT is independent of D and more than one order of magnitude lower than that of SECOA. Moreover, the cost in SECOA is practically unaffected by D because it is dominated by the numerous (i) HMAC operations to create the temporal seeds, and (ii) modular multiplications to fold these seeds during the creation of the reference SEAL.

We next evaluate the communication cost per network edge in `SUM` queries. The overheads are shown in Table 2, when $N = 1024$, $F = 4$ and $D = [1800, 5000]$. Note that all costs except for that corresponding to pair aggregator-querier in SECOA are invariant of our system parameters, whereas the latter cost is marginally affected by D and N. The benefit of SIES over SECOA is clear, reaching more than 3 orders of magnitude. Additionally, the difference between SIES and CMT is negligible.

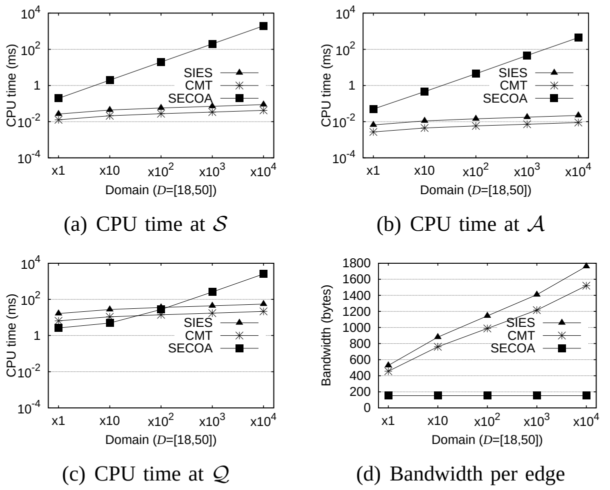
TABLE 2
Communication cost of `SUM` queries

Nework edge	CMT	SECOA	SIES
$\mathcal{S}$-$\mathcal{A}$	20 bytes	37.8 KB	32 bytes
$\mathcal{A}$-$\mathcal{A}$	20 bytes	37.8 KB	32 bytes
$\mathcal{A}$-$\mathcal{Q}$	20 bytes	832 bytes	32 bytes

Finally, we discuss the performance of the rest of the sum-based queries. Recall that `COUNT` queries can be answered trivially using the `SUM` protocol in all methods. `AVG` in SIES is handled using a variation of the `SUM` protocol without inflicting any additional costs, because the size of the plaintexts and keys remains the same. CMT can employ a similar technique and retain its performance as well. On the other hand, SECOA must run its `SUM` protocol *twice*; once for computing $v^S_{i,t}$ and once for $v^C_{i,t}$. Therefore, all its overheads double. For `VARIANCE` and `STDDEV`, SIES must use slightly larger plaintexts and keys. Nevertheless, the performance degradation is very small; for `VARIANCE` SIES utilizes SHA-384 that takes 1.34 μs and consumes 48 bytes, whereas for `SUM` it employs SHA-256 that takes 1.02 μs and consumes 32 bytes. CMT does not need to change the key sizes, since $v^S_{i,t}$, $(v^S_{i,t})^2$ and $v^C_{i,t}$ can fit in a 20-byte plaintext. Therefore, its performance remains intact. However, SECOA must run its SUM protocol separately for each $v_{i,t}^S$, $(v_{i,t}^S)^2$ and $v_{i,t}^C$ and, hence, its performance deteriorates by a factor of 3. We omit the relevant charts because they are straightforward.

Quantile queries Recall that we answer quantile queries combining our SUM protocol with a simple multi-round technique. Both SECOA and CMT can employ a similar approach. Nevertheless, SECOA can handle the special case of *exact* MIN/MAX queries with a more efficient, *one-round* protocol. In the following we focus only on MAX queries, since we have already explained the performance gains of SIES over SECOA in SUM, which are inherited also in the rest of the quantile queries (e.g., MEDIAN).

Figure 10 contains our experimental results for MAX queries, varying the domain D while setting $N = 1024$ and $F = 4$. Figure 10(a) plots the CPU time at the source. SECOA entails an excessive computational cost, which increases linearly with D. The reason is that the source performs a linear number of RSA encryptions in the value of the sensor reading. On the other hand, the overhead at SIES and CMT is not substantially affected by D. This is due to the fact that the number of SUM rounds entailed in these schemes is *logarithmic* in the domain size. Moreover, the operations incurred by μTesla are lightweight and, thus, the CPU time in both approaches is in the order of microseconds and comparable.



(a) CPU time at $\mathcal{S}$ (b) CPU time at $\mathcal{A}$

(c) CPU time at $\mathcal{Q}$ (d) Bandwidth per edge

Fig. 10. Performance of MAX queries vs. D

Figure 10(b) depicts the CPU time at the aggregator, which is analogous to Figure 10(a) for similar reasons. The only difference is that, in SECOA, the number of RSA encryptions performed by the aggregator is linear to the difference between the minimum and maximum values reported by the aggregator's children. As such, the overhead is lower than that at the source. Once again, SIES outperforms SECOA by up to two orders of magnitude. Figure 10(c) assesses the respective overhead at the querier. In SIES and CMT, the cost is comparable and remains practically unaffected by D. The reason is that the cost inflicted by the multiple rounds and μTesla is logarithmic and very low. On the other hand, the effect of D in SECOA is more evident. The cost is smaller than SIES and CMT when the domain involves small numbers. However, it rises quickly with D, and eventually it becomes up to approximately one order of magnitude higher than in SIES.

The communication cost is presented in Figure 10(d). SECOA has the attractive feature that it consumes a constant bandwidth with respect to the domain. On the other hand, SIES and CMT entail a logarithmic cost due to the multiple rounds. This is the only disadvantage of our approach as compared to the MAX protocol of SECOA. However, our cost is still better than sending the raw data of 1024 sources directly to the querier ($>$ 4 KB).

Summary In addition to covering all security properties, and offering exact results for a wide range of aggregate queries, SIES features an impressive performance advantage over SECOA in the vast majority of settings, especially considering (i) the approximate nature of SUM queries in SECOA, and (ii) its unsuitability to support confidentiality. Furthermore, SIES has comparable performance to CMT, despite the simplicity and efficiency of the encryption scheme of CMT due to its lack of the data integrity property. In overall, SIES is lightweight as it features small communication cost, and CPU times that most of the times range from a few microseconds to a few milliseconds in the worst-case. More notably, the processing cost at a sensor (source or aggregator) is always up to a couple of microseconds. Consequently, SIES would offer ideal performance even if it were deployed on a sensor CPU with several orders of magnitude smaller computational capabilities than our benchmark CPU. In that sense, SIES is a suitable technique for resource-constrained sensor networks.

8 Conclusion

This is the first work that addresses secure in-network aggregation supporting both integrity and confidentiality, while covering a wide variety of exact aggregate queries, and capturing application scenarios such as sensor deployment in hostile environments and outsourced aggregation. We initially presented SIES, a novel and efficient scheme for SUM queries. SIES relies on a combination of homomorphic encryption and secret sharing. These techniques are lightweight, leading to a very small bandwidth consumption and a very low CPU cost for all parties involved. This fact renders SIES a powerful security tool for resource-constrained sensor networks. Subsequently, we demonstrated how SIES can be adapted to handle several other sum-based and quantile queries, while retaining high security and performance. We also introduced a novel functionality that identifies malicious sensors, enhancing the robustness of our system. Finally, we experimentally confirmed our performance claims.

Acknowledgments

Dimitris Papadias was supported by grant 6181/08 from Hong Kong RGC. Aggelos Kiayias was partly supported by NSF awards 0831304, 0831306 and EU projects RECUP and CODAMODA.

References

[1] S. Madden, M. J. Franklin, J. Hellerstein, and W. Hong, “TAG: A Tiny AGgregation service for ad-hoc sensor networks,” in *OSDI*, 2002.

[2] Y. Yao and J. Gehrke, “The COUGAR approach to in-network query processing in sensor networks,” *SIGMOD Rec.*, vol. 31, no. 3, pp. 9–18, 2002.

[3] L. Hu and D. Evans, “Secure aggregation for wireless networks,” in *SAINT-W*, 2003.

[4] P. Jadia and A. Mathuria, “Efficient secure aggregation in sensor networks,” in *HiPC*, 2004.

[5] C. Castelluccia, E. Mykletyn, and G. Tsudik, “Efficient aggregation of encrypted data in wireless sensor networks,” in *MobiQuitous*, 2005.

[6] B. Przydatek, D. Song, and A. Perrig, “SIA: Secure information aggregation in sensor networks,” in *SenSys*, 2003.

[7] M. Garofalakis, J. M. Hellerstein, and P. Maniatis, “Proof sketches: Verifiable in-network aggregation,” in *ICDE*, 2007.

[8] S. Nath, H. Yu, and H. Chan, “Secure outsourced aggregation via one-way chains,” in *SIGMOD*, 2009.

[9] SenseWeb, Microsoft Research. [Online]. Available: http://research.microsoft.com/en-us/projects/senseweb/

[10] E. Fasolo, M. Rossi, J. Widmer, and M. Zorzi, “In-network aggregation techniques for wireless sensor networks: A survey,” *IEEE Wireless Communications*, vol. 14, no. 2, pp. 70–87, 2007.

[11] J. Considine, F. Li, G. Kollios, and J. Byers, “Approximate aggregation techniques for sensor databases,” in *ICDE*, 2004.

[12] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri, “Medians and beyond: New aggregation techniques for sensor networks,” in *SenSys*, 2004.

[13] M. B. Greenwald and S. Khanna, “Power-conserving computation of order-statistics over sensor networks,” in *PODS*, 2004.

[14] R. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[15] M. Bellare, R. Canetti, and H. Krawczyk, “Keying hash functions for message authentication,” in *CRYPTO*, 1996.

[16] A. J. Menezes, P. C. V. Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, Inc., 1996.

[17] J. Kim, A. Biryukov, B. Preneel, and S. Hong, “On the security of HMAC and NMAC based on HAVAL, MD4, MD5, SHA-0 and SHA-1,” in *SCN*, 2006.

[18] A. Perrig, R. Szewczyk, V. Wen, D. Culler, and J. D. Tygar, “SPINS: Security protocols for sensor networks,” in *Mobile Computing and Networking*, 2001.

[19] H. Chan, H.-C. Hsiao, A. Perrig, and D. Song, “Secure distributed data aggregation,” *Journal of Foundations and Trends in Databases*, vol. 3, no. 3, pp. 149–201, 2011.

[20] W. Du, J. Deng, Y. S. Han, and P. K. Varshney, “A witness-based approach for data fusion assurance in wireless sensor networks,” in *GLOBECOM*, 2003.

[21] H. Chan, A. Perrig, and D. Song, “Secure hierarchical in-network aggregation in sensor networks,” in *CCS*, 2006.

[22] K. B. Frikken and J. A. Dougherty, IV, “An efficient integrity-preserving scheme for hierarchical sensor aggregation,” in *WiSec*, 2008.

[23] R. Merkle, “A certified digital signature,” in *CRYPTO*, 1989.

[24] K. B. Frikken, K. Kauffman, and A. Steele, “General secure sensor aggregation in the presence of malicious nodes,” in *ICDE*, 2011.

[25] S. Zhu, S. Setia, and S. Jajodia, “LEAP: Efficient security mechanisms for large-scale distributed sensor networks,” in *CCS*, 2003.

[26] J. Girao, D. Westhoff, and M. Schneider, “CDA: Concealed data aggregation for reverse multicast traffic in wireless sensor networks,” in *ICC*, 2005.

[27] W. He, X. Liu, H. Nguyen, K. Nahrstedt, and T. Abdelzaher, “PDA: Privacy-preserving data aggregation in wireless sensor networks,” in *INFOCOM*, 2007.

[28] M. Conti, L. Zhang, S. Roy, R. Di Pietro, S. Jajodia, and L. V. Mancini, “Privacy-preserving robust data aggregation in wireless sensor networks,” *Security and Communication Networks*, vol. 2, no. 2, pp. 195–213, 2009.

[29] T. Dimitriou and D. Foteinakis, “Secure and efficient in-network processing for sensor networks,” in *BroadNets*, 2004.

[30] C. Castelluccia and C. Soriente, “ABBA: A balls and bins approach to secure aggregation in WSNs,” in *WiOpt*, 2008.

[31] R. Di Pietro, P. Michiardi, and R. Molva, “Confidentiality and integrity for data aggregation in WSN using peer monitoring,” *Security and Communication Networks*, vol. 2, no. 2, pp. 181–194, 2009.

[32] K. B. Frikken and Y. Zhang, “Confidentiality and integrity for SUM aggregation in sensor networks,” in *SECRYPT*, 2010.

[33] A. Mahimkar and T. S. Rappaport, “SecureDAV: A secure data aggregation and verification protocol for sensor networks,” in *GLOBECOM*, 2004.

[34] S. Roy, S. Setia, and S. Jajodia, “Attack-resilient hierarchical data aggregation in sensor networks,” in *SASN*, 2006.

[35] Y. Yang, X. Wang, S. Zhu, and G. Cao, “SDAP: A secure hop-by-hop data aggregation protocol for sensor networks,” in *MobiHoc*, 2006.

[36] S. Roy, M. Conti, S. Setia, and S. Jajodia, “Securely computing an approximate median in wireless sensor networks,” in *SecureComm*, 2008.

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

[38] P. Haghani, P. Papadimitratos, M. Poturalski, K. Aberer, and J.-P. Hubaux, “Efficient and robust secure aggregation for sensor networks,” in *NPSec*, 2007.

[39] G. Taban and V. D. Gligor, “Efficient handling of adversary attacks in aggregation applications,” in *ESORICS*, 2008.

[40] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*. Chapman Hall/CRC, 2008.

Stavros Papadopoulos Stavros Papadopoulos is a Visiting Assistant Professor of Computer Science and Engineering, at the Hong Kong University of Science and Technology (HKUST). Prior to joining HKUST in 2011, he was a post-doctoral researcher at the Chinese University of Hong Kong. He holds a Ph.D. degree in Computer Science from HKUST. His research interests include authenticated query processing, PIR, searchable encryption, and access control on encrypted data.

Aggelos Kiayias Aggelos Kiayias teaches and does research in cryptography and computer security. His work has been funded by a number of agencies including, the United States DoD, DHS, NIST and NSF including a CAREER award as well as the European Union ERC and Marie Curie programs. In 2011, he served as the program chair of the Cryptographer’s Track of the RSA Conference. Currently he serves as the General Chair of EUROCRYPT 2013. He holds a Ph.D. in Computer Science from the City U. of New York that was funded by a Fulbright fellowship and is a graduate of the University of Athens, Department of Mathematics.

Dimitris Papadias Dimitris Papadias is a Professor of Computer Science and Engineering, at the Hong Kong University of Science and Technology (HKUST). Before joining HKUST in 1997, he worked and studied at the German National Research Center for Information Technology (GMD), the National Center for Geographic Information and Analysis (NCGIA, Maine), the University of California at San Diego, the Technical University of Vienna, the National Technical University of Athens, Queen’s University (Canada), and University of Patras (Greece). He serves or has served in the editorial boards of the VLDB Journal, IEEE Transactions on Knowledge and Data Engineering, and Information Systems.