Secure and Efficient In-Network Processing of Exact SUM Queries

Stavros Papadopoulos [1], Aggelos Kiayias [2], Dimitris Papadias [3]

[1] *Department of Computer Science and Engineering*
The Chinese University of Hong Kong
stavros@cse.cuhk.edu.hk

[2] *Department of Informatics and Telecommunications*
National and Kapodistrian University of Athens
aggelos@di.uoa.gr

[3] *Department of Computer Science and Engineering*
The Hong Kong University of Science and Technology
dimitris@cse.ust.hk

***Abstract*—*In-network aggregation* is a popular methodology adopted in wireless sensor networks, which reduces the energy expenditure in processing aggregate queries (such as SUM, MAX, etc.) over the sensor readings. Recently, research has focused on *secure* in-network aggregation, motivated (i) by the fact that the sensors are usually deployed in open and unsafe environments, and (ii) by new trends such as *outsourcing*, where the aggregation process is delegated to an untrustworthy service. This new paradigm necessitates the following key security properties: *data confidentiality*, *integrity*, *authentication*, and *freshness*. The majority of the existing work on the topic is either unsuitable for large-scale sensor networks, or provides only *approximate* answers for SUM queries (as well as their derivatives, e.g., COUNT, AVG, etc). Moreover, there is currently no approach offering *both* confidentiality and integrity at the same time. Towards this end, we propose a novel and efficient scheme called SIES. SIES is the first solution that supports Secure In-network processing of Exact SUM queries, satisfying *all* security properties. It achieves this goal through a combination of *homomorphic encryption* and *secret sharing*. Furthermore, SIES is lightweight (it relies on inexpensive hash operations and modular additions/multiplications), and features a very small bandwidth consumption (in the order of a few bytes). Consequently, SIES constitutes an ideal method for resource-constrained sensors.**

I. 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 that reside 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 motivated by the following two facts. First, sensor networks are usually 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]). Second, *outsourced aggregation* [7], [8] has started to gain populararity. Under this new trend, 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.
- **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.

Next, we outline the weaknesses of existing work on secure in-network aggregation, and present our contributions.

Prior work. The majority of the schemes are either unsuitable for large-scale networks, or support only approximate answers to SUM queries. More specifically, several methods follow the *commit-and-attest model* [6], [10], [11], [12], [13] that involves an expensive broadcasting phase, during which the sensors actively participate in the verification process. The performance of these solutions deteriorates drastically with the number of sensors. On the other hand, more efficient methods [7], [8] associate small proofs of integrity with the transmitted raw data, which can be aggregated in-network and easily verified by the querier. Nevertheless, they are based on sketches and, thus, cannot offer exact results. Finally, currently no approach can support *both* confidentiality and integrity at the same time. For example, [5] provides confidentiality but not integrity, whereas [8] focuses on integrity without being able to support confidentiality.

Our contributions. Motivated by the shortcomings of the related work, we introduce SIES, the first solution for **S**ecure **I**n-network processing of **E**xact **S**UM queries (as well as their derivatives, e.g., COUNT, AVG, etc.), satisfying *all* four security properties. SIES achieves this goal through a combination of *homomorphic encryption* and *secret sharing*. It is scalable as it does not involve the participation of the sensors in the verification process. It entails a small constant communication cost per network edge (in the order of a few bytes). Moreover, it requires few and inexpensive cryptographic operations (hashes and modular additions/multiplications) at each party involved. The above render SIES lightweight and, thus, an ideal solution for resource-constrained sensors. We analytically and experimentally confirm our performance claims.

The rest of the paper is organized as follows. Section II surveys the related work, Section III contains preliminary information, Section IV explains SIES in detail, Section V includes cost models, Section VI experimentally evaluates our scheme and, finally, Section VII concludes our paper.

II. Related Work

In Section II-A we describe the basic cryptographic tools that are necessary for our presentation. In Section II-B we survey the prior work on secure in-network aggregation. In Section II-C we discuss the aggregation methods appearing in the Outsourced Database (ODB) model. Finally, in Section II-D we select suitable benchmark solutions for our experimental evaluation, and present them in detail.

A. Cryptographic Primitives

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 multiplication. The methods that support the addition operation, such as the Paillier cryptosystem [15], 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 [16]. 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. In the sequel, we use $HM_1(K, m)$ ($HM_{256}(K, m)$) to denote the HMAC of m using key K, assuming that the underlying hash function is SHA-1 (SHA-256) [17], [18] that produces 20-byte (32-byte) digests.

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 [18]. In our work, we assume that the PRFs are implemented as HMACs.

B. Secure In-network Aggregation

Several approaches follow the *commit-and-attest model* [6], [12], [13], [11], [10], which consists of two phases. During the *commitment phase*, the aggregators are forced to commit to the partial results they produce, by constructing a cryptographic structure like the Merkle Hash Tree [19] and sending the root digest to the querier. In the *attestation phase*, the querier broadcasts the aggregate result and the root digest it received from the network to all the sensors, using an authenticated broadcasting protocol like μTesla [20]. Each sensor then individually audits its contribution to the result using the commitment structure. The broadcasting inflicts considerable communication cost to the network and high query latency that increase with the number of sources, gravely impacting scalability.

In the context of *outsourced in-network aggregation*, Proof-Sketches [7] and SECOA [8] associate small proofs of integrity with the transmitted raw data, which can be aggregated in-network and easily verified by the querier. These approaches are more scalable than the above, since they do not require the active participation of the sensors in the verification process. However, they are both based on sketches and, thus, offer only *approximate* answers.

All the described methods so far focus on integrity, without being able to provide confidentiality. On the other hand, [5] supports aggregation directly on encrypted data via an additively homomorphic encryption scheme, satisfying data confidentiality. Nevertheless, this approach does not safeguard against data tampering. Currently, there is no solution that provides both confidentiality and integrity.

For completeness, we also present some methods that target at slightly different models than the above schemes and are orthogonal to our work. LEAP [21] is a key management protocol that allows in-network aggregation, while restricting the impact of any malicious nodes *within their network neighborhood*. [3] and [4] provide secure aggregation against a *single* malicious node. Finally, Yu [22] introduces a random sampling technique that enables aggregation queries to *tolerate* the adversarial nodes (instead of just detecting them), in order tackle denial of service (DoS) attacks.

C. Aggregation in the ODB Model

In the Outsourced Database (ODB) model [23], a data owner delegates the administration of its database to a specialized third-party service provider. Since the provider may be untrustworthy, security issues such as data confidentiality and integrity arise. Although there exist numerous approaches that follow this paradigm, here we discuss only those that focus specifically on secure processing of *aggregate queries*.

Li et al. [24] design authenticated index structures that incorporate hash digests similar to the Merkle Hash Tree [19], and are signed by the data owner. In addition to the aggregation results, the provider utilizes the indices to produce verification information that proves the answer integrity. In [25], the owner outsources the storage of a data stream to the provider, and subsequently asks aggregate queries on the stream. The owner monitors the stream and stores a compact authentication summary that helps in auditing the result integrity. Ge and Zdonik [26] focus on confidentiality instead of integrity. They assume that the database is encrypted with the Paillier additively homomorphic scheme [15]. The provider operates solely on the ciphertexts producing answers to SUM-based queries. This scheme, however, cannot guarantee integrity.

The above methods cannot be applied to the in-network aggregation model because they assume the existence of a *single* data owner. In particular, the signatures and ciphertexts are produced with a single key. In our setting, there are multiple sensors, each regarded as a separate data owner that encrypts/signs its data with its *unique* key (otherwise, compromising a single sensor would lead to the compromise of the entire system). Providing secure in-network aggregation in the presence of multiple keys is a more challenging task.

D. Benchmark Solutions

Since there is currently no scheme offering both integrity and confidentiality for in-network processing of SUM queries, there is no direct competitor to our work. However, in order to facilitate our experimental evaluation, we choose as benchmarks (i) the only method guaranteeing confidentiality, i.e., [5], henceforth referred to as CMT (after the authors' initials), and (ii) the best in-network solution providing integrity. Specifically, we select SECOA [8] because it is more scalable than the commit-and-attest approaches, and subsumes Proof-Sketches [7]. SECOA supports a wide range of aggregate queries (including MAX). We hereafter use SECOA$^{\mathrm{S}}$ to refer to the SUM algorithm of SECOA. Below we present these two methods in more detail.

CMT. We illustrate this method through a simple example. Suppose that sensor $\mathcal{S}_1$ ($\mathcal{S}_2$) shares a secret key k_1 (k_2) with the querier. Also let v_1 (v_2) be the reading of $\mathcal{S}_1$ ($\mathcal{S}_2$). $\mathcal{S}_1$ ($\mathcal{S}_2$) computes ciphertext $c_1 = v_1 + k_1 \mod n$ ($c_2 = v_2 + k_2 \mod n$), where $n > v_1, v_2, k_1, k_2$ is a publicly known integer. Now suppose that the sink $\mathcal{A}$ receives c_1 and c_2. It aggregates them into $c = c_1 + c_2 \mod n$ and forwards it to the querier. The latter can extract $v_1 + v_2 = c - (k_1 + k_2)$, since it knows k_1 and k_2. This is a simple additively homomorphic scheme that allows in-network processing of SUM queries on encrypted data, thus satisfying confidentiality. However, it does not guarantee integrity; the adversary can inject any integer v' to c, cheating the querier to extract $v_1 + v_2 + v'$ and admit it as a correct result.

SECOA$^{\mathrm{S}}$. This scheme is a combination of the MAX protocol of SECOA, denoted by SECOA$^{\mathrm{M}}$, and the AMS sketches [27] for distinct item estimation. We first describe SECOA$^{\mathrm{M}}$. Each sensor $\mathcal{S}_i$ sends to its parent aggregator $\mathcal{A}$ (i) the generated data value v_i, (ii) an *inflation certificate*, and (iii) a *deflation certificate*. The inflation (deflation) certificate guarantees that v_i has not been inflated (deflated) by an adversary.

The inflation certificate of v_i is simply $HM_1(K_i, v_i)$, where K_i is a unique key shared by $\mathcal{S}_i$ and the querier. The deflation certificate, called a SEAL, is a value produced by applying v_i times the RSA encryption function on a seed sd_i known only by $\mathcal{S}_i$ and the querier. For example, if $v_i = 3$, then the SEAL is equal to $\mathcal{E}_{RSA}(\mathcal{E}_{RSA}(\mathcal{E}_{RSA}(sd_i)))$ and denoted by $\mathcal{E}^3_{RSA}(sd_i)$. A SEAL can be perceived as a *one-way chain*; from $\mathcal{E}^{v_1}_{RSA}(sd_i)$ one can produce $\mathcal{E}^{v_2}_{RSA}(sd_i)$ for any $v_2 > v_1$, but not for $v_2 < v_1$.

Aggregator $\mathcal{A}$ first chooses the MAX of the received values and forwards it to its parent aggregator, along with its inflation certificate. Subsequently, it combines all the collected SEALs. Let $v_1 = 3$ and $v_2 = 5$ be the values received from sensors $\mathcal{S}_1$ and $\mathcal{S}_2$, sd_1 and sd_2 the corresponding seeds and $\mathcal{E}^3_{RSA}(sd_1)$ and $\mathcal{E}^5_{RSA}(sd_2)$ the respective SEALs. $\mathcal{A}$ applies RSA encryption on $\mathcal{E}^3_{RSA}(sd_1)$ 2 times (i.e., $v_2 - v_1$), which yields $\mathcal{E}^5_{RSA}(sd_1)$. This process is called *rolling*. It then computes the modular product $\mathcal{E}^5_{RSA}(sd_1) \cdot \mathcal{E}^5_{RSA}(sd_2) \mod n = \mathcal{E}^5_{RSA}(sd_1 \cdot sd_2)$, where n is the public RSA modulus. This step is called *folding*. The product is the aggregate SEAL sent to the parent aggregator.

The described process continues recursively, until the querier eventually receives the MAX result res along with its certificates from the sink. It first verifies the inflation certificate using the corresponding shared key. Next, knowing all the secret seeds, it recreates the aggregate SEAL (this entails folding all seeds together and rolling them res times), and verifies it against the collected one.

To answer SUM queries, SECOA$^{\text{S}}$ necessitates each sensor $\mathcal{S}_i$ to generate $J \cdot v_i$ AMS sketches and merge them into exactly J ones, where J adjusts the accuracy of the method (with higher values leading to better accuracy). It then invokes SECOA$^{\text{M}}$ separately on each of these J sketches. As an optimization, the aggregators merge the inflation proofs into a single *aggregate HMAC* [28] by XOR-ing them. Furthermore, the sink folds the SEALs that are at the same "position" in the chain to reduce the number of SEALs sent to the querier. After verification, the querier *approximates* the SUM result as $2^{\bar{x}}$, where $\bar{x}$ denotes the average over the J collected sketches.

III. Preliminaries

Section III-A presents our system architecture, Section III-B describes our query model, Section III-C includes our threat model, and Section III-D contains the building blocks of SIES.

A. System Architecture

In the sequel, without loss of generality, we separate the roles of the sensor that generates data values, and the sensor that performs aggregation tasks. We call the former a *source* and denote it as $\mathcal{S}$, whereas we refer to the latter as *aggregator* and denote it as $\mathcal{A}$. For simplicity, we assume that the sensors are organized into a *tree topology*, with the sources being the leaves and the aggregators representing the internal nodes. The tree topology can be arbitrary, while its construction, fine-tuning and re-organization due to node failures are issues orthogonal to our work. A querier $\mathcal{Q}$ poses long-running queries, and communicates only with the root of the aggregation tree, i.e., the network *sink*. Figure 1 illustrates a simple example architecture. The topology configuration, the dissemination of the necessary information to the aggregators and sources by the querier, and the initiation of the continuous query at the sources occur before the aggregation process commences.

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 (we adopt the notation from [1], [7]), 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.

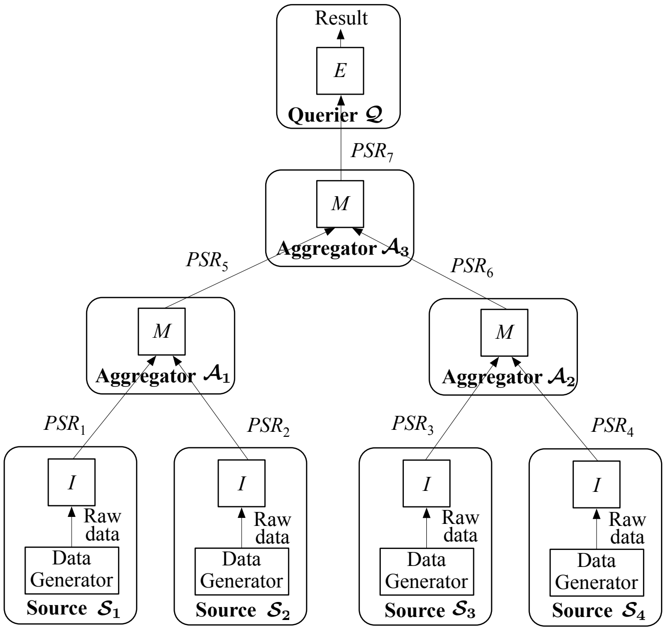


Fig. 1. System Architecture

A last remark concerns the computational capabilities of the sources, the aggregators and the querier. Unlike SECOA [8], we do not make the strong assumption that the sensors are capable of performing expensive computations (such as RSA encryption operations), or that they are attached to more powerful proxy machines. On the contrary, in our scheme, each party involved needs only to be able to compute inexpensive hash functions and modular additions/multiplications. In that sense, our work targets traditional resource-constrained sensor networks.

B. Query Model

We assume the *push-based* data collection model, where a continuous query is registered at the sources during a setup phase, and then each source periodically transmits its data to the network. This model is usually preferable to the *pull-based* approach (where the querier broadcasts the query to the sources in order to extract the result on demand) because (i) pulling the data incurs a large communication overhead and can be prohibitively slow for large-scale networks, and (ii) the sources must be always on to receive potential queries (whereas in the push-based model the sources may preserve power by turning on periodically).

All sources, aggregators and the querier are *loosely synchronized* in time epochs. The epochs are specified by the transmission period T of each source. In the sequel, for simplicity, we perceive every epoch as a distinct time instant t. We focus on exact SUM queries, which have the following form:

Query template

```
SELECT SUM(attr) FROM Sensors
WHERE pred
EPOCH DURATION T
```

If a source does not satisfy the WHERE predicate, it simply transmits 0. Without loss of generality, we consider that all data values are positive integers (we can always encode other data types as positive integers via simple translation and scaling operations [8]). Note that COUNT queries are trivially reduced to SUM (e.g., a source simply transmits 1 if it satisfies the query predicate). Moreover, SUM and COUNT results can be combined to answer other aggregate queries, e.g., the average as AVG = SUM/COUNT. In a similar manner we can derive other queries from SUM and COUNT, such as STDDEV and VARIANCE.

C. Threat Model

The adversary may either compromise a sensor node (source or aggregator) and thus take its full control, or infiltrate the wireless channel. We do not make any assumption about the computational capabilities of the adversary. Our main goal is to satisfy data confidentiality, integrity, authentication, and freshness, as they were defined in Section I. Particularly for data integrity, we mandate *detection* of any alteration of the result, and not *tolerance* or *error recovery*. Additionally, we do not try to tackle DoS attacks, e.g., when the (compromised) sink does not report at all the result within one or more time epochs. Such cases are trivially detected if the querier does not receive any data. Furthermore, 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.

Another remark concerns our goals in the presence of a compromised source. Note that a compromised source can arbitrarily alter its *own* data. In this case, the querier admits the (modified) result as correct, without detecting the malicious activity. Our scheme, as well as all the approaches in the literature, cannot tackle this situation. Nevertheless, the adversary should not be able to breach the security of *the rest of the system*, i.e., it must not be able to impersonate or decrypt the ciphertext transmitted by an *uncompromised* node. Note that this is important for the robustness of our scheme, since it is very likely that some sensors are hacked in open and unsafe environments. Furthermore, take into account that SUM and AVG results are resilient to a small number of “fake” readings, unlike MAX/MIN queries where a single compromised node suffices to significantly alter the result.

Finally, note that the querier must either be the *owner* of the sensors, or an authorized entity that possesses all the necessary keys. Access control issues are orthogonal to our work.

D. Building Blocks

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

Additively Homomorphic Scheme. Let p be a prime, $m_i < p$ the message to be encrypted, and $K \neq 0, k_i < p$ two secret keys. 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

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

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 [17]. 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 also in an information theoretic sense.

IV. SIES

Before embarking on the details of SIES we provide the main idea. We use the homomorphic encryption scheme described in Section III-D because it 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.

SIES complies with the architecture presented in Section III-A, i.e., it consists of a setup phase that occurs before setting the system into motion, and the three phases of the aggregation process (initialization, merging, and evaluation). Section IV-A explains these phases, and Section IV-B discusses the security of SIES. Table I provides our notation.

TABLE I
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

A. Phases

Setup phase. Suppose that the number of sources is N. 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 (in our implementation we set this size to 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 [20]. After the sources receive the query, the aggregation process commences. Whenever $\mathcal{Q}$ issues a new query, it simply broadcasts it 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-byte long, whereas $ss_{i,t}$ is 20-byte 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 2.

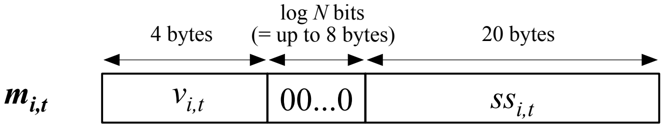


Fig. 2. Format of $m_{i,t}$

The purpose of adding $\log N$ 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.

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 $((\log N)/8 + 20)$ 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 $\log N$ when N numbers are added. This justifies our choice to pad $\log N$ 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 total size of $m_{i,t}$ does not exceed 32 bytes). Figure 3 illustrates the final $m_{f,t}$ retrieved by $\mathcal{Q}$ for the topology depicted in Figure 1, where $PSR_{i,t}$ was generated by $\mathcal{S}_i$.

$\mathcal{Q}$ 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

[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.

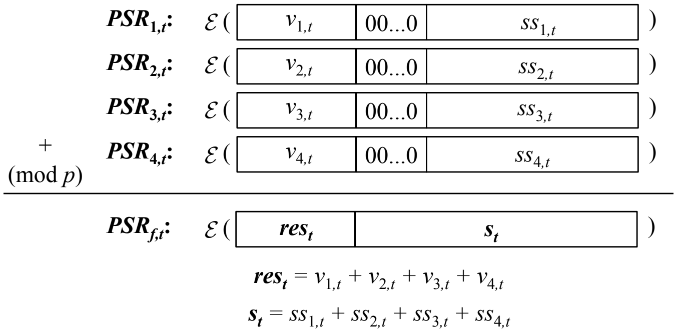

PSR1,t: ℰ(v1,t 00...0 ss1,t)
PSR2,t: ℰ(v2,t 00...0 ss2,t)
PSR3,t: ℰ(v3,t 00...0 ss3,t)
+ (mod p) PSR4,t: ℰ(v4,t 00...0 ss4,t)
PSRf,t: ℰ(rest st)
rest = v1,t + v2,t + v3,t + v4,t
st = ss1,t + ss2,t + ss3,t + ss4,t

Fig. 3. Example aggregation in SIES

(iii) freshness is provided because the keys and shares used by the sources integrate the temporal information t, which is different for different epochs. In the next sub section we discuss the security of SIES in detail.

B. 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. In both scenarios, the adversary attempts to extract the plaintext associated with a PSR. 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. Let $PSR_{i,t} = K_t \cdot m_{i,t} + k_{i,t}$ be the PSR targeted by the adversary, which originates from an *uncompromised* source $\mathcal{S}_i$ at epoch t. We say that SIES provides data confidentiality, if the adversary succeeds in extracting $m_{i,t}$ from $PSR_{i,t}$ with *negligible* probability.

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 $\mathcal{S}_j$, $j \neq i$, with $k_{i,t}$ though remaining unknown. 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.

Theorem 1: SIES satisfies data confidentiality.

Proof: Recall that $k_{i,t}$ is produced by pseudo-random function $HM_{256}(\cdot)$ with output size 32 bytes. This key is unknown to the adversary. Additionally, $k_{i,t}$ is used only *once*, i.e., no other plaintext is encrypted with $k_{i,t}$. Consequently, in order to extract $m_{i,t}$ and provided that K_t, p may be known, the adversary can only attempt to correctly *guess* $k_{i,t}$. However, this happens with probability 2^{-256}, which is negligible. Moreover, correctly guessing key k_i, used as input to $HM_{256}(\cdot)$ when generating $k_{i,t}$, occurs with probability 2^{-160} (since k_i is 20 bytes long in our implementation), which is also negligible. ■

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. We focus on the case where the adversary does not compromise *any* source. If the adversary hacks a source, it can obtain its keys and alter arbitrarily its own reading. Therefore, in this case data integrity is always breached. We say that SIES guarantees data integrity in the absence of a compromised source, if the querier admits a *corrupted* PSR (incorporating a false aggregation result) as legitimate with negligible probability.

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

Proof: Let $PSR_{f,t}$ be the final legitimate PSR, and $PSR'_{f,t}$ the corrupted PSR eventually presented to $\mathcal{Q}$. In order for data integrity to be violated, the last 28 bytes of value $(PSR_{f,t} - PSR'_{f,t}) \cdot K_t^{-1} \mod p$ must all be 0. Observe that this holds only when the secret s_t incorporated in $PSR_{f,t}$ (whose size is up to 28 bytes) is equal to s'_t contained in $PSR'_{f,t}$. This is necessary in order for the querier to admit $m'_{f,t}$ extracted from $PSR'_{f,t}$ as legitimate. Since K_t is unknown, this happens with probability $\frac{2^{32}}{2^{256}} = 2^{-224}$, which is negligible. ■

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.

Theorem 3: SIES is secure against querier impersonation.

Proof: This is directly ensured by the μTesla protocol, which enables each source to verify that the message (i.e., the query) indeed originated from $\mathcal{Q}$ (for details, we refer the interested reader to [20]). ■

Source impersonation is covered by data integrity discussed above.

Data freshness. A result is *fresh* if it reflects the current time epoch t. An adversary violates data freshness if it presents to the querier a *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 4: SIES provides data freshness.

Proof: Let $PSR_{f,t}$ be the legitimate final PSR at epoch t, and $PSR_{f,t'}$ the legitimate final PSR at epoch $t' < t$. The adversary succeeds in breaking freshness, if the secret s_t in $PSR_{f,t}$ is the same as s'_t in $PSR_{f,t'}$. Recall that all the secret shares are produced by pseudo-random function $HM_1(\cdot)$, which takes t as seed and has output length 20 bytes. Consequently, the probability that $s_t = s'_t$ is equal to 2^{-224} (similar to Theorem 2), which is negligible. ■

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 secret shares of the sources contributing to the result.

V. Cost Models

We analytically compare SIES against CMT [5] and SECOA$^{\text{S}}$ [8] in terms of the *computational cost* at each party, and the *communication overhead* at a network edge (i.e., between source-aggregator, aggregator-aggregator, and aggregator-querier). Table II summarizes the symbols used in the analysis, as well as their typical values (1 $\mu s = 10^{-6}$ seconds, 1 $ms = 10^{-3}$ seconds). These values were obtained based on the hardware and software settings of our experiments.

TABLE II
Symbols and Values in the Analysis

Symbol	Meaning	Typical Value
N	Number of sources	1024
J	Number of sketches	300
F	Aggregator fanout	4
v	Source value	$\in [1800, 5000]$
x_i	Value of sketch i	$\in [0, 23]$
rl_i	Rolling operations for SEAL i	$\in [0, 22]$
C_{sk}	Cost of sketch generation	0.037 μs
C_{RSA}	Cost of RSA encryption	5.36 μs
C_{HM_1}	Cost of $HM_1(\cdot)$	0.46 μs
$C_{HM_{256}}$	Cost of $HM_{256}(\cdot)$	1.02 μs
C_{A20}	Cost of 20-byte modular addition	0.15 μs
C_{A32}	Cost of 32-byte modular addition	0.37 μs
C_{M32}	Cost of 32-byte modular multiplication	0.45 μs
C_{M128}	Cost of 128-byte modular multiplication	1.39 μs
C_{MI32}	Cost of finding a 32-byte mult. inverse	3.2 μs
S_{sk}	Size of a sketch	1 byte
S_{inf}	Size of an inflation certificate	20 bytes
S_{SEAL}	Size of a SEAL	128 bytes

Computational cost. In CMT, the ciphertext calculation at a source involves a single modular addition of the plaintext with the secret key. Furthermore, in order to address data freshness, we must consider that a different key is used in every epoch t. Therefore, its creation time must be added to the sources's computational cost. We assume that the pseudo-random function used for key generation is $HM_1(\cdot)$ and, thus, the key size is 20 bytes. If C_{HM_1} and C_{A20} denote the costs of $HM_1(\cdot)$ and addition modulo a 20-byte integer, respectively, the total processing cost of CMT at a source is:

$$C^{\mathcal{S}}_{CMT} = C_{HM_1} + C_{A20} \tag{1}$$

In SECOA$^{\text{S}}$, a source first computes $J \cdot v$ sketches, where v is the source value and J is proportional to the desired accuracy. It then merges them (with negligible cost) to produce the final set of J sketches to be transmitted to the parent aggregator. Moreover, it creates an inflation certificate for each of the J sketches by applying HMAC $HM_1(\cdot)$. Subsequently, it produces J temporal seeds using the $HM_1(\cdot)$ function (similar to CMT to satisfy freshness). Finally, the source uses the seeds to derive a SEAL for each sketch. Let x_i denote a sketch value, C_{sk} the cost to generate a sketch, and C_{RSA} the time consumed by RSA encryption. Then, the total processing cost at a source in SECOA$^{\text{S}}$ is:

$$C^{\mathcal{S}}_{SECOA^S} = J \cdot (v \cdot C_{sk} + 2 \cdot C_{HM_1}) + \sum_{i=1}^{J} x_i \cdot C_{RSA} \tag{2}$$

In SIES, the computation at a source entails (i) two key generations by applying HMAC $HM_{256}(\cdot)$, (ii) a secret share creation with $HM_1(\cdot)$, and (iii) a multiplication and an addition modulo a 32-byte number. If $C_{HM_{256}}$ is the cost of $HM_{256}(\cdot)$, and C_{M32} and C_{A32} refer to the cost of 32-byte modular multiplication and addition, respectively, the CPU consumption at a source in SIES is:

$$C^{\mathcal{S}}_{SIES} = 2 \cdot C_{HM_{256}} + C_{HM_1} + C_{M32} + C_{A32} \tag{3}$$

Let F be the number of children (or the fanout) of an aggregator. In CMT, the aggregator simply adds the ciphertexts (modulo a 20-byte number). Hence, the computational cost at the aggregator is

$$C^{\mathcal{A}}_{CMT} = (F - 1) \cdot C_{A20} \tag{4}$$

In SECOA$^{\text{S}}$, the aggregator first merges the $F \cdot J$ sketches received from its children into J sketches with negligible cost. Subsequently, it combines the F SEALs of each of the final J sketches, by performing the appropriate folding and rolling operations. Next, it XOR-es the F inflation certificates, which involves a negligible cost. Assuming that the RSA modulus is 128 bytes long, rl_i is the number of rolling operations required for the i^{th} SEAL, and C_{M128} is the cost of a 128-byte modular multiplication, the computational cost at an aggregator in SECOA$^{\text{S}}$ is:

$$C^{\mathcal{A}}_{SECOA^S} = J \cdot (F - 1) \cdot C_{M128} + \sum_{i=1}^{J} rl_i \cdot C_{RSA} \tag{5}$$

In SIES, similar to CMT, the aggregator simply adds the ciphertexts received from its children (however, this time the modulus is 32 bytes long). Therefore, the processing overhead at an aggregator in SIES is:

$$C^{\mathcal{A}}_{SIES} = (F-1) \cdot C_{A32} \tag{6}$$

The querier in CMT first computes with $HM_1(\cdot)$ the temporal keys of all the N sources that participated in the aggregation. Then, it subtracts these keys from the final ciphertext to decrypt it. Thus, the processing cost of the querier in CMT is:

$$C^{\mathcal{Q}}_{CMT} = N \cdot (C_{HM_1} + C_{A20}) \tag{7}$$

The querier in SECOA$^{\mathrm{S}}$ aggregates the SEALs collected from the root aggregator, by performing the necessary rolling and folding operations. Additionally, it creates a reference SEAL by (i) computing the $J \cdot N$ seeds with the $HM_1(\cdot)$ function, (ii) folding the seeds together, and (iii) rolling them to the maximum collected sketch value x_{max}. Finally, it generates J inflation certificates, once again using $HM_1(\cdot)$. If $seals$ denotes the number of SEALs collected from the root aggregator, the CPU time at the querier in SECOA$^{\mathrm{S}}$ is given by:

$$\begin{aligned} C^{\mathcal{Q}}_{SECOA^S} = J \cdot N \cdot C_{HM_1} + (seals + J \cdot N - 2) \cdot C_{M128} \\ + (\sum_{i=1}^{seals} rl_i + x_{max}) \cdot C_{RSA} + J \cdot C_{HM_1} \end{aligned} \tag{8}$$

In SIES, the computational cost at the querier involves the generation of N secret shares with $HM_1(\cdot)$, $N+1$ keys with $HM_{256}(\cdot)$, the summation of all shares together (for verification), the subtraction of the keys from the final ciphertext, and a modular multiplication with the multiplicative inverse of K_t. Supposing that C_{MI32} is the time to produce the multiplicative inverse modulo a 32-byte number, the total computational cost at the querier in SIES is

$$\begin{aligned} C^{\mathcal{Q}}_{SIES} = N \cdot C_{HM_1} + (N+1) \cdot C_{HM_{256}} + \\ (2 \cdot N - 1) \cdot C_{A32} + C_{MI32} + C_{M32} \end{aligned} \tag{9}$$

Communication cost. In CMT, each party exchanges a single 20-byte ciphertext. Similarly, in SIES every party transmits a 32-byte PSR. Therefore, the communication cost in CMT and SIES at every network edge is always constant and equal to 20 and 32 bytes, respectively.

In SECOA$^{\mathrm{S}}$, every source and every aggregator (except for the root aggregator) sends J sketch values, J SEALs, and one (aggregated) inflation certificate. The difference at the root aggregator is that it folds the SEALs that correspond to the same chain position. The final number of SEALs that it sends to the querier is $seals$ instead of J. If S_{sk}, S_{SEAL} and S_{inf} denote the size of a sketch, a SEAL and an inflation certificate, respectively, the communication overhead between source-aggregator and aggregator-aggregator are given in Equation 10, whereas the cost between aggregator-querier is shown in Equation 11.

$$S^{\mathcal{S}-\mathcal{A}}_{SECOA^S} = S^{\mathcal{A}-\mathcal{A}}_{SECOA^S} = J \cdot S_{sk} + J \cdot S_{SEAL} + S_{inf} \tag{10}$$

$$S^{\mathcal{A}-\mathcal{Q}}_{SECOA^{\mathrm{S}}} = J \cdot S_{sk} + seals \cdot S_{SEAL} + S_{inf} \tag{11}$$

Formulae evaluation for typical values. Observe that the costs of SIES are independent of the dataset. On the other hand, some costs in SECOA$^{\mathrm{S}}$ depend on the dataset-specific variables v, x_i, x_{max}, $seals$ and rl_i. Supposing that the domain of v is $[D_L, D_U]$, x_i takes values from domain $[0, \log(N \cdot D_U)]$ [8]. By bounding v and x_i, we can also bound x_{max}, $seals$ and rl_i, since they are all derived from x_i. Consequently, we can find the minimum (*best-case*) and maximum (*worst-case*) costs for SECOA$^{\mathrm{S}}$, which hold for *any* dataset distribution in $[D_L, D_U]$.

Table III illustrates the costs calculated by inserting the typical values of Table II into Equations 1-11. Interestingly, in addition to its exact nature and security properties, SIES outperforms the best-case scenario of SECOA$^{\mathrm{S}}$ on all metrics, by up to 4 orders of magnitude. Moreover, it is marginally inferior to the lightweight scheme of CMT on all metrics, which though fails to support data integrity. In the next section we experimentally confirm our observations.

TABLE III
COSTS USING TYPICAL VALUES

Costs	**CMT**	**SECOA$^{\mathrm{S}}$ (min/max)**	**SIES**
Comput. cost at $\mathcal{S}$	1.17 μs	20.26 ms / 92.75 ms	3.46 μs
Comput. cost at $\mathcal{A}$	0.45 μs	1.25 ms / 36.63 ms	1.11 μs
Comput. cost at $\mathcal{Q}$	0.62 ms	568.46 ms / 568.63 ms	2.28 ms
Commun. cost $\mathcal{S}$-$\mathcal{A}$	20 bytes	38.72 KB / 38.72 KB	32 bytes
Commun. cost $\mathcal{A}$-$\mathcal{A}$	20 bytes	38.72 KB / 38.72 KB	32 bytes
Commun. cost $\mathcal{A}$-$\mathcal{Q}$	20 bytes	0.44 KB / 3.25 KB	32 bytes

VI. EXPERIMENTS

Recall that there is no direct competitor to SIES, as no solution can provide in-network processing of exact SUM queries satisfying both confidentiality and integrity. Nevertheless, we select CMT and SECOA$^{\mathrm{S}}$ (see Section II-D for their detailed description) as benchmark solutions to assist our experimental evaluation, although they offer only partial solutions to our targeted problem (CMT cannot offer data integrity, whereas SECOA$^{\mathrm{S}}$ does not provide confidentiality and supports only approximate answers).

We ran our experiments on a 2.66 GHz Intel Core i7 with 4GB RAM, running Mac OS X ver. 10.6.4. Admittedly, this hardware is much more powerful than that of a sensor and, thus, it solely facilitates the comparison of the methods. However, as we shall demonstrate soon, our scheme is lightweight and, therefore, it can perform exceptionally even on sensor CPUs, which may be several orders of magnitude slower than our processor. We implemented SIES, CMT and SECOA$^{\mathrm{S}}$ in

C++ using the GNU MP[2] and OpenSSL[3] libraries. We experimented with real dataset Intel Lab[4], 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 the aggregators form a complete tree. Finally, following [8], we fix the number of sketch instances (J) of SECOA$^{\mathrm{S}}$ to 300, in order to bound the relative approximation error within 10% with probability 90%.

We measure the costs of SUM queries, varying the following system parameters: (i) the number of sources (N), (ii) the fanout of the aggregators (F), and (iii) the dataset domain ($D = [D_L, D_U]$). Recall that all solutions handle aggregates only on integers. 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 temperature readings supported by the system and, thus, of the SUM result (the querier divides the extracted integer result with the respective power of 10 to derive the final float result). In every experiment we vary one parameter, setting the other two to their default values. We evaluate a SUM query over 20 epochs and report the average cost per epoch.

Table IV includes the system parameters, along with their ranges and default values. Sections VI-A, VI-B and VI-C evaluate the computational cost at the source, the aggregator and the querier, respectively. Section VI-D discusses the communication overhead at all parties. Finally, Section VI-E summarizes our results.

TABLE IV
SYSTEM PARAMETERS

Parameter	Default	Range
Number of sources (N)	1024	64, 256, 1024, 4096, 16384
Fanout (F)	4	2, 3, 4, 5, 6
Domain ($D = [18, 50]$)	$\times 10^2$	$\times 1$, $\times 10$, $\times 10^2$, $\times 10^3$, $\times 10^4$

A. Computational Cost at a Source

Figure 4 shows the computational cost at the source as a function of D, when $N = 1024$ and $F = 4$. The error bars on the curve of SECOA$^{\mathrm{S}}$ indicate its best- and worst-case scenario, as they were calculated by the cost models of Section V. SIES outperforms SECOA$^{\mathrm{S}}$ by more than two orders of magnitude. The reason is that SIES involves few and cheap HMAC operations and modular additions, whereas SECOA$^{\mathrm{S}}$ involves generating an excessive number of sketches and performing several RSA encryptions to produce the SEALs. SIES also retains a comparable performance to CMT, which is in the order of a couple of microseconds. Furthermore, contrary to SECOA$^{\mathrm{S}}$, the computational cost of SIES and CMT are independent of D. The overhead in SECOA$^{\mathrm{S}}$ increases rapidly with D because it is dominated by the time to produce the numerous sketches, whose number depends on the source value v (see also Equation 2 in Section V). Finally, note that the processing time at the source is unaffected when varying F and N and, thus, we omit the corresponding diagrams.

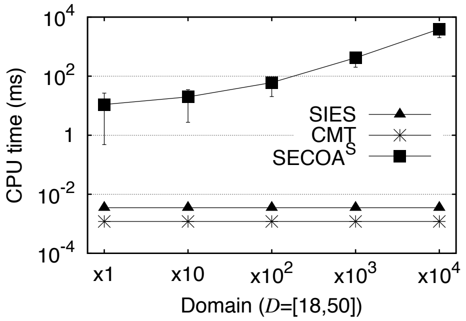


Fig. 4. Computational cost at the source vs. the domain

B. Computational Cost at the Aggregator

Figure 5 demonstrates the CPU time at the aggregator when varying its fanout F, and setting $N = 1024$ and $D = [1800, 5000]$. Once again, SIES outperforms SECOA$^{\mathrm{S}}$ by approximately two orders of magnitude, while featuring a marginal performance difference from CMT. Specifically, the cost in SIES is within 0.3-2 μs due to the inexpensive modular additions it involves. On the other hand, SECOA$^{\mathrm{S}}$ entails expensive folding and rolling operations (modular multiplications and RSA encryptions, respectively). As expected, the overhead of all solutions linearly increases with the fanout. In SECOA$^{\mathrm{S}}$ this is justified because each increase in the fanout causes the number of folding operations to rise, whereas in SIES and CMT the number of modular additions increase with F. We do not include experiments varying N because the performance of the aggregator in all schemes is independent of this parameter. Furthermore, D has no impact on SIES and CMT, whereas it negligibly affects SECOA$^{\mathrm{S}}$. Consequently, we also omit the corresponding diagram.

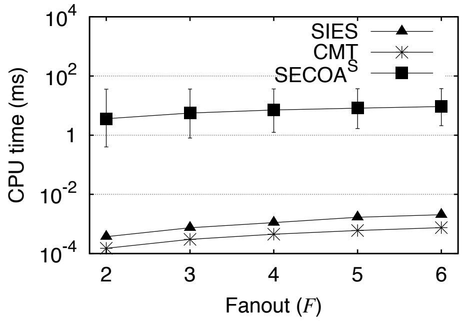


Fig. 5. Computational cost at the aggregator vs. the fanout

[2] http://gmplib.org/

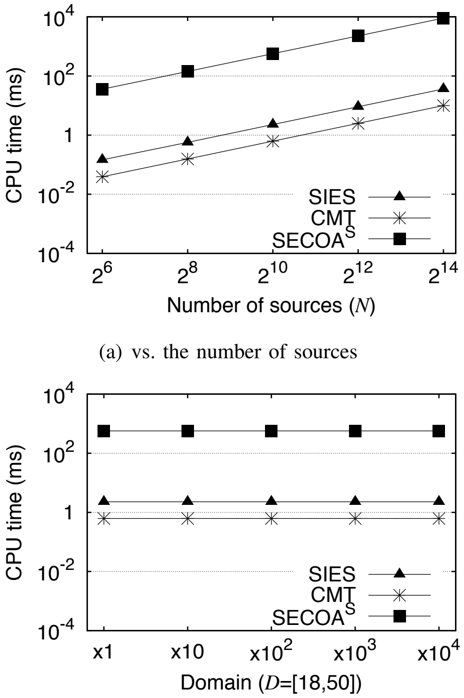
[3] http://www.openssl.org/

[4] http://db.csail.mit.edu/labdata/labdata.html

C. Computational Cost at the Querier

Figure 6 depicts the CPU time consumed by the querier ($D = [11800, 5000]$, $F = 4$). We omit the cost model of SECOA^S from the diagram because it bounds the actual values very accurately (within a 0.001 relative error). In Figure 6(a) we vary N ($F = 4$, $D = [1800, 5000]$). This overhead is linearly dependent on N in all methods. SIES outperforms SECOA^S by more than one order of magnitude on all values. This happens because in SECOA^S the querier performs numerous folding and rolling operations to compute the reference SEAL 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 consumption in SIES is within range 0.15-36 ms. Additionally, the performance of SIES is comparable to that of CMT. Their difference is mainly justified by 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 6(b) we vary 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^S. Moreover, the cost in SECOA^S 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. Finally, the performance of all solutions does not vary with F and, therefore, we omit the respective experiment from our discussion.



(a) vs. the number of sources

(b) vs. the domain

Fig. 6. Computational cost at the owner

D. Communication Cost

Table V provides the communication cost per network edge, when $N = 1024$, $F = 4$ and $D = [1800, 5000]$. We also include the minimum and maximum values derived by the models of SECOA^S. Note that all costs except for that corresponding to pair aggregator-querier in SECOA^S are invariant of our system parameters, whereas the latter cost is marginally affected by D and N. The benefit of SIES over SECOA^S is clear, reaching more than 3 orders of magnitude. Additionally, the difference between SIES and CMT is negligible.

TABLE V
COMMUNICATION COST

$N = 1024$, $F = 4$, $D = [1800, 5000]$

Nework edge	**CMT**	**SECOA (actual/min/max)**	**SIES**
$\mathcal{S}$-$\mathcal{A}$	20 bytes	37.8 KB / 37.8 KB / 37.8 KB	32 bytes
$\mathcal{A}$-$\mathcal{A}$	20 bytes	37.8 KB / 37.8 KB / 37.8 KB	32 bytes
$\mathcal{A}$-$\mathcal{Q}$	20 bytes	832 bytes / 448 bytes / 6.7 KB	32 bytes

E. Summary

In addition to covering all security properties and offering exact results, SIES features an impressive performance advantage over SECOA^S on all performance metrics, especially considering (i) the approximate nature of SUM queries in SECOA^S, 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 very small communication cost in the order of a few bytes, and CPU consumption that most of the times ranges 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 constitutes a suitable technique for resource-constrained sensor networks.

VII. CONCLUSION

In this paper we introduced SIES, a novel and efficient scheme for secure in-network processing of SUM queries (as well as their derivatives, e.g., COUNT, AVG, etc.). SIES is the only solution that offers exact query answers, satisfying all the necessary security properties of the targeted model, i.e., data confidentiality, integrity, authentication, and freshness. It achieves this goal through a combination of a homomorphic encryption scheme and a secret sharing method. These techniques are lightweight, leading to a very small bandwidth consumption for all parties involved (in the order of a few bytes), and a very low CPU cost because they entail a small number of inexpensive cryptographic operations (hashes and modular additions/multiplications). This fact renders SIES a powerful security tool for resource-constrained sensor networks. We confirm our performance claims through a detailed analytical and experimental evaluation.

ACKNOWLEDGMENTS

Dimitris Papadias was supported by grant 6181/08 from Hong Kong RGC.

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] A. Mahimkar and T. S. Rappaport, "SecureDAV: A secure data aggregation and verification protocol for sensor networks," in *Globecomm*, 2004.

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

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

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

[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] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *EUROCRYPT*, 1999.

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

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

[18] 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.

[19] R. Merkle, "A certified digital signature," in *CRYPTO*, 1989.

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

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

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

[23] H. Hacigumus, B. Iyer, and S. Mehrotra, "Providing database as a service," in *ICDE*, 2002.

[24] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin, "Authenticated index structures for aggregation queries," *TISSEC (to appear)*, 2010.

[25] K. Yi, F. Li, G. Cormode, M. Hadjieleftheriou, G. Kollios, and D. Srivastava, "Small synopses for group-by query verification on outsourced data streams," *TODS*, vol. 34, no. 3, pp. 1–42, 2009.

[26] T. Ge and S. Zdonik, "Answering aggregation queries in a secure system model," in *VLDB*, 2007.

[27] N. Alon, Y. Matias, and M. Szegedy, "The space complexity of approximating the frequency moments," *J. Comput. Syst. Sci.*, vol. 58, no. 1, pp. 137–147, 1999.

[28] J. Katz and Y. Lindell, "Aggregate message authentication codes," in *CT-RSA*, 2008.