Secure Outsourced Aggregation via One-way Chains

Suman Nath
Microsoft Research
sumann@microsoft.com

Haifeng Yu
National University of Singapore
haifeng@comp.nus.edu.sg

Haowen Chan
Carnegie Mellon University
haowen@cs.cmu.edu

ABSTRACT

We consider the Outsourced Aggregation model, where sensing services outsource their sensor data collection and aggregation tasks to third-party service providers called *aggregators*. As aggregators can be untrusted or compromised, it is essential for a sensing service to be able to verify the correctness of aggregation results. This work presents SECOA, a framework with a family of novel and optimally-secure protocols for secure outsourced aggregation. Our framework is based on a unified use of one-way chains. It supports a large and diverse set of aggregate functions, can have multiple hierarchically organized aggregators, can deterministically detect any malicious aggregation behavior without communication with sensors, and incurs a small and workload-independent communication load on sensors. We also present extensive evaluation results to demonstrate the feasibility of our framework.

Categories and Subject Descriptors

H.2.0 [**Database Management**]: Security, integrity, and protection

General Terms

Algorithms, Security

Keywords

secure outsourced aggregation, wide-area sensing

1. INTRODUCTION

The paradigm of wide-area shared sensing [9, 12] has recently received considerable attention. For example, Microsoft's SenseWeb [12] provides a shared sensor portal where numerous sensors, owned by different entities, publish data and where users can query the sensors for the current state of the physical world, e.g., the warmest park of a city. Similarly, the portals at Weather Underground,[1] SciScope,[2] SensorBase,[3] etc. enable accessing a large number of live sensors (e.g., weather stations, geological sensors, etc.) all over the world. In all these cases, the portal needs to provide various kinds of aggregate results on live data provided by Internet-connected sensors. In contrast to a typical wireless sensing application, a wide-area shared sensing service has the following unique characteristics:

1) Diverse queries. Such shared service needs to support a wide and diverse set of query processing functionalities on sensor data. For example, important queries in SenseWeb include Max/Min, Sum and related queries, Uniform Samples, Quantiles, Top-k readings, Frequent Readings, etc. In contrast, typical wireless sensor networks are deployed for dedicated sensing where a relatively small set of aggregates can be sufficient in most cases.

2) Push-based data collection. In a wide-area sensing service, sensor data is generally pushed from sensors to the portal or to intermediate aggregators, without any explicit communication initiated by the portal and before queries are submitted. This is due to the following reasons. First, a typical service incorporates a large number of geographically dispersed sensors. For example, SciScope currently incorporates more than a million sensors worldwide. Pulling data from all sensors during query time is infeasible as it can be prohibitively slow in such a large network. Second, a sensor may be shared by multiple sensing services and hence the query rate can be substantially higher (e.g., multiple queries per second) than the data collection/reporting frequency of the sensor (e.g., once per hour). Third, some sensors may even connect only occasionally (e.g., once per hour) to the Internet for data reporting. Due to all these reasons, an on-demand pull-based data collection at query time is usually not feasible.

Instead of performing data collection and query processing tasks itself, a portal often delegates such tasks to a third party, called *aggregator*, that provides data aggregation services. For example, SenseWeb and SciScope currently use many such aggregators deployed by independent parties worldwide. In such *outsourced aggregation model*, aggregators collect sensor data and process aggregate queries on the data. The portal simply forwards users' queries to aggregators and returns answers from aggregators to users. Such outsourced aggregation provides multiple key bene-

[1]`http://www.wunderground.com/`
[2]`http://sciscope.org/`
[3]`http://sensorbase.org/`

fits. First, sensors periodically report data, and collecting all data to the centralized portal may incur prohibitively high network load on the portal. Second, some sensors generate large amounts of raw data, and sending such data to a nearby aggregator (instead of to the far away portal) can reduce network traffic. Finally, outsourcing makes a sensing service more affordable for parties with limited resources. Since database service providers (e.g., corporate-level services) have the advantage of expertise consolidation, they can potentially offer the service with much lower cost and also with better performance and availability guarantees.

In spite of its clear advantages, the wide adoption of outsourced aggregation in commercial contexts faces key security challenges. Namely, these third-party aggregators can be untrusted, compromised, or even malicious. Since an aggregator aggregates data from potentially a large number of sensors, a problematic aggregator can significantly affect query answers. To guard against this, the portal must be able to verify the correctness of answers provided by an aggregator. Such assurance is essential for outsourced aggregation to become a sound and viable alternative to in-house aggregation.

Previous research on outsourced database has provided techniques to authenticate results of (range) selection and join queries [13, 16, 20, 21, 22]. These techniques do not apply to aggregation queries. In the context of wireless sensor networks, several existing works address the problem of making aggregation query with distributed aggregators secure (e.g., SHIA [6] and set sampling [27]). However, when processing a query, all these approaches require communicating with all the sensors in the system (e.g., for verification of the query result). Such property makes these approaches rather unsuitable for large-scale wide-area sensing with push-based data collection. Moreover, these approaches support limited sets of aggregates (e.g., they do not support Max and Top-k queries), which is insufficient for general-purpose wide-area sensing. While proof-sketch [11] can be trivially modified to work in the push-based data collection model, it also supports only Count-related aggregates. Furthermore, proof-sketch is not optimally-secure [6] (i.e., it has false positives and false negatives in verification of the query result) and it allows an aggregator to cheat without being detected.

Our approach and contributions. This paper presents SECOA, the first unified framework with a family of novel, optimally secure (i.e., no false positive/negative) protocols that addresses the above challenges for secure outsourced aggregation in wide-area sensing. SECOA supports a rich and diverse set of aggregates, which is a strict superset of the aggregates supported by all previous approaches [6, 11, 27]. SECOA does not require the sensors to be involve in each query, and thus well-suits the push-based data collection in wide-area sensing.

All our protocols are based on a novel and unified use of one-way chains, where sensor readings and intermediate aggregation results are represented as positions on the chains. Using one-way chains, we first develop a novel, conceptually clean, and optimally secure protocol for verifying the answer to a Max query, without involving the sensors. In this protocol, one-way chains prevent the adversary from deflating (i.e., under-reporting) the Max. Preventing inflation is trivial via message authentication code.

We then show that many aggregates, such as Count, Count Distinct, Sum, and Average, can be readily reduced to Max. All our protocols are optimally secure, i.e., they are free of false positives and false negatives. Generalizing from the Max protocol, we further develop novel one-way-chain-based protocols for verifying the query answers for a large set of aggregates including Sum, Uniform Samples, Quantiles, Top-k Readings, Top-k Groups, Most Popular Readings, and Frequent Readings. To the best of our knowledge, no previous secure aggregation scheme supports Most Popular Readings or Frequent Readings.

One-way chains often incur substantial computational overhead, especially with a relatively large number of chains. Indeed, we find that with traditional one-way chains based on one-way hash functions such as MD5 or SHA1, our protocols would incur unacceptable overhead for certain aggregates (e.g., several minutes to verify a Count answer). To reduce such overhead, somewhat counter-intuitively, we use novel one-way chains based on RSA encryptions. To the best of our knowledge, ours are the first one-way chains based on encryptions. We leverage the homomorphic property of RSA encryption to develop aggressive optimization techniques. These techniques can help reduce computation overhead by multiple orders of magnitude (e.g., several milliseconds to verify a Count answer), despite RSA encryptions being expensive compared to hashing. Using RSA also helps us to address some other challenges in our protocols.

We have implemented all our protocols in C++. Our experimental evaluation shows that the computational overhead on the portal to verify a query answer is less than a few milliseconds, and corresponding communication overhead is less than a few kilobytes. The overheads on sensors and aggregators are rather small as well. Our results also demonstrate the importance of removing false positives and false negatives in SECOA—in one of our experiments with a Count query with proof-sketches [11], an adversary can use a safe cheating strategy [11] to report an incorrect Count of 0 in place of a correct answer of 20,000 without being detected.

2. RELATED WORK

Outsourced Database. Outsourced Database model, where entities outsource their data management needs to third-party service providers, has been recently studied in literature. This model allows data owners to create, store, and update their databases, and users to query the database. Our outsourced aggregation model is different, in the sense that the sensor list at the aggregator is created and updated by the portal, while the data to be aggregated come from sensors. For outsourced databases, many existing mechanisms can ensure that the query answer provided by the third party is correct (i.e., data has not been modified by the third party) and complete (i.e., all data within a given range are returned) [13, 15, 16, 19, 20, 21, 22]. The scheme by Li et al. [13] also supports the outsourced database to be dynamically updated by the data owner, and ensures data freshness (i.e., the latest version of data is returned). However, all these work mostly deal with (range) selection queries on ordered data, join queries, and various set operations. Such techniques are not sufficient for authenticating aggregate answers, which requires verifying the correct *processing* of an aggregate function on relevant data. Query execution proofs presented in [25] can be used to detect if an aggregator is lazy to compute the aggregate function over all sensor data, but it is not adequate to prevent the aggregator to report an incorrect result.

Secure in-network aggregation. In the context of wireless sensor networks, several previous works provide techniques for secure in-network aggregation. SIA [5] supports only a single aggregator, and is not suitable for a large number of sensors. SHIA [5] supports multiple aggregators organized hierarchically, but supports only Count and related queries. Moreover, it has a verification phase where the query result is broadcasted/disseminated to all sensors, so that every sensor has the opportunity to raise an alarm if it disagrees with the query result. Such on-demand pull-based techniques, however, are not simply feasible for wide-area sensing services with push-based data collection. Set sampling [27] has a similar property where all sensors are involved in the query processing, making it unsuitable for our push-based paradigm.

Proof-sketch [11] can be trivially modified to use in the push-based data collection model. However, it also supports only Count-related aggregates. Moreover, it relies on "complement predicates" to verify the query answer. For example, to count the number of sensors satisfying a certain predicate, proof-sketch estimates the approximate counts (using Flajolet-Martin (FM) sketches [10]) C_1 (C_2) of sensors satisfying (not satisfying) a given predicate. Then the protocol checks whether the sum (C_1+C_2) is within $(1\pm\epsilon)U$, where U is the total number of sensors and ϵ is some positive constant. Such an approach introduces both false positives and false negatives in verification. In particular, a malicious aggregator can use a safe cheating strategy [11] to always report substantially incorrect results without ever been detected.

Fundamentally, the above problem stems from the fact that proof sketches by themselves cannot detect the deflation attack, where the aggregators throw away some sensor readings and "under-report" FM sketches. As a result, the protocol must rely on "complement predicates" and then compare against U. It turns out that dealing with such deflation attack is the key challenge when sensors are not involved in the verification process (i.e., in the push-based model).[4] SECOA solves this problem via a novel use of one-way chains. Doing so enables SECOA to securely answer not only all aggregates in [11] without any false positives or false negatives, but also many other aggregates (such as Max, Top-k, Frequent Items, Popular Items) that proof sketches cannot deal with at all. Proof sketches cannot answer these queries because the "complement predicates" technique does not apply to these aggregates, and thus deflation attack cannot be detected.

3. PRELIMINARIES

3.1 Problem Formulation

System model. Motivated by Microsoft's SenseWeb, we consider the outsourced aggregation model as in Figure 1, where users query the portal for various aggregates on the data submitted by sensors. The queries may be qualified with predicates on static attributes of the sensors (e.g., sen-

[4]For wireless sensor networks where we can afford to involve all sensors, preventing deflation attack is trivial. We can simply disseminate the final result to all sensors. Any sensor with a synopsis "larger" than the final result can raise an alarm.

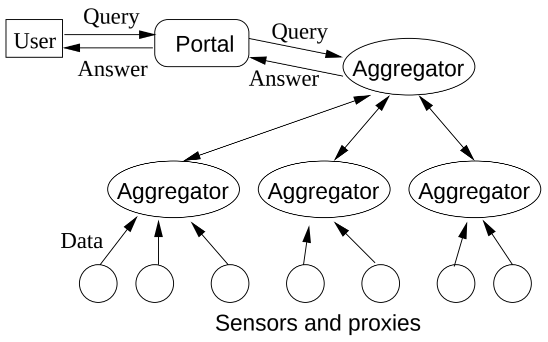


Figure 1: Model of secure outsourced aggregation.

sors within a given geographic region). The portal forwards user queries to the root aggregator. The root aggregator computes query answer (potentially after communicating with other aggregators in the system), and then sends the answer back to the portal. The portal verifies the correctness of the answer and returns the answer to the user if it passes verification. It is possible to have multiple independent portals (not shown in Figure 1), e.g., for load balancing.

In all real-world sensing services we mentioned in Section 1, a new sensor must first register itself (with all static metadata) before publishing sensor data through the portal. Therefore, we assume that the portal knows the complete list of sensors in the system, as well as their static attributes (e.g., locations).[5] Each aggregator knows the same information for all sensors within its subtree. Given all such information, it is trivial for the portal and the aggregators to determine those sensors qualified by a query predicate on static attributes, and exclude other sensors in the query processing and verification protocol. Thus we will not explicitly discuss static predicates when the context is clear. We assume that each sensor shares a distinct secret symmetric key with the portal (e.g., established when the sensor is registered to the portal).

All sensors and the portal are loosely synchronized in time epochs. Each sensor independently and periodically reports its latest data to its aggregator, which stores the data. The reporting frequency can be multiple times per epoch, once per epoch, or once per multiple epochs. Without loss of generality, we assume that sensor readings are positive integers—other kinds of reading can always be encoded as positive integers. Some sensors may directly connect to the Internet while others (e.g., wireless embedded sensors) may connect to the Internet via some proxy (not shown in Figure 1). To simplify terminology, we simply consider the proxy as part of the sensor. We assume that sensors (or their proxies) and aggregators have the computational power to perform RSA encryptions. Thus, our algorithms do not target classic wireless sensor networks.

Attack model. We assume the portal is trusted, but aggregators are potentially malicious and are thus Byzantine. Examples of malicious actions by an aggregator include fabricating, replaying, duplicating, ignoring sensor readings,

[5]From security point of view, this requirement is fundamental; otherwise a malicious aggregator can introduce fake sensors. In fact, all previous secure aggregation approaches (including proof-sketch, SIA, SHIA, etc.) have such requirements.

computing the aggregate incorrectly, etc. Some sensors may be malicious (and thus Byzantine) as well, and we allow all malicious aggregators and malicious sensors to collude. We do not consider denial-of-service attacks where aggregators fail to or refuses to provide any acceptable result.

A malicious sensor can always report an arbitrary reading (on its own behalf), and this fundamentally cannot be prevented. Similar to most prior work on secure aggregation [5, 6, 11], we do not aim at preventing such attack. Fortunately, all aggregates that we consider in this paper (except Max) are robust [26] against (i.e., relatively insensitive to) those arbitrary readings for a small number of malicious sensors. We consider Top-k and Max mainly because they enable us to generalize to other (robust) aggregates such as Frequent Items.

Security goal. Our goal is to enable the portal to verify whether an aggregate reported by an aggregator is correct. Specifically, the portal should accept a reported aggregate if and only if it equals to the output of a correct execution of the aggregation function over all the readings reported in the most recent epoch by those sensors qualified by the query predicate, excluding sensors that are reported to have failed. We need to exclude failed sensors since physical sensor failures (e.g., due to dead battery, problematic network connection, etc.) are unavoidable in a large-scale system. The root aggregator will thus include a list of failed sensors together with the query result.

It is possible for an aggregator to include some properly functioning sensors in the list of failed sensors, in order to maliciously "hide" the readings from those sensors. However, one can show that without (occasional) out-of-band communication between the portal and sensors, this attack cannot be prevented. All previous secure aggregation algorithms (e.g., proof-sketch, SIA, SHIA, etc.) have the same vulnerability. One thus has to rely on various ad hoc techniques to overcome this. For example, a sensor coming back from its malfunctioning state can securely report to the portal so that the portal can verify aggregator's claim latter. Alternatively, if occasional communication from the portal to sensors are allowed, the portal can occasionally probe some randomly selected sensors in the list of failed sensors.[6] A probe does not need to be made at the current epoch. For example, a probe during epoch z can ask a sensor "did you report during epoch w, x, and y?" Such techniques are orthogonal to our work, since previous algorithms will need them as well. Thus, without loss of generality, we will describe all our protocols assuming that the list of failed sensors is empty. If the list is not empty, it is trivial to simply exclude those sensors in our protocol.

3.2 Cryptographic Essentials

Message Authentication Code (MAC). A message authentication code (MAC) is a short piece of information used to authenticate a message. A MAC algorithm accepts as input a (secret) symmetric key and an arbitrary-length message, and outputs a MAC. The MAC value protects both a message's data integrity as well as its authenticity, by allowing verifiers (knowing the secret symmetric key) to detect any changes to the message content.

[6]Notice that this is fundamentally different from the deflation attack. In deflation attack, the portal cannot efficiently probe the sensors since it does not know whose readings have been dropped. In contrast, here aggregators disclose the list of sensors whose readings are not included, which makes efficient probing possible.

One-way chain. A one-way chain is based on some (secret) seed s and some public one-way function F (with or without a trapdoor). The one-way function F is easy to compute but computationally infeasible in general to invert (without knowing the trapdoor). The chain has the sequence of values $F(s)$, $F(F(s))$, $F(F(F(s)))$, ... Throughout this paper, we use $F^x()$ to denote recursively applying the function F for x times. Thus the x'th value in the sequence is $F^x(s)$, and is called the *SEAL* (SElf-Authenticating vaLue) at position x. Given a SEAL at position x, one can easily apply F to obtain all SEALs at positions $y \geq x$. In other words, the chain can be *rolled forward*. On the other hand, it is computationally infeasible (without knowing the trapdoor of s) to roll the chain backward to obtain any SEAL at position $z < x$.

Any one-way function can be used as F in one-way chains, though previous research mostly used one-way hash functions such as MD5 and SHA-1. We will use RSA encryption [23] (where the encryption key is publicly known) as the one-way function F in our one-way chains. Since RSA encryption is one-way (assuming the decryption key is not disclosed), it obviously satisfies the requirement.

SEAL folding. Certain one-way functions allow folding[7] multiple SEALs into one SEAL for efficient communication and verification. For example, when RSA encryption is used as the one-way function, multiple SEALs based on the same encryption key (but potentially different seeds) can be folded together using modulo multiplication. The folding operation (i.e., modulo multiplications) can be done by anyone, as it does not require the decryption key. The folded SEAL has the same size as an individual SEAL and it can be used to verify all SEALs together. In addition, folded SEALs are secure—an adversary cannot produce a folded SEAL without knowing *all* the individual SEALs (by exactly the same argument that shows the security of condensed RSA signatures [14] and RSA batch verification [4]). As another example, one can also fold multiple MACs into one using XOR as the folding operation [6]. To unify terminology, we will use the symbol $\odot$ to denote the folding operator.

4. SECURE COMPUTATION OF MAX

Starting from this section, we present a family of protocols for computing various aggregates that satisfy the security goal in Section 3.1. To help understanding, Section 4.1 first explains our approach of securely computing Max where there is a single aggregator. Section 4.2 extends to hierarchical aggregators. Sections 5 to 7 generalize to a wide range of aggregates beyond Max, and also describe critical optimizations that dramatically reduce the protocols' overhead.

4.1 Max with a Single Aggregator

Protocol intuition. For now assume that each sensor reports exactly one reading per epoch; we will relax this assumption later. We start from two basic pieces of information that each sensor generates in our protocol. Let K_i denote the symmetric key shared between sensor i and the portal. Let v_i be the reading on sensor i at the current

[7]We use the term *folding*, instead of the more commonly used term *aggregation*, to distinguish aggregation of SEALs from aggregation of sensor data.

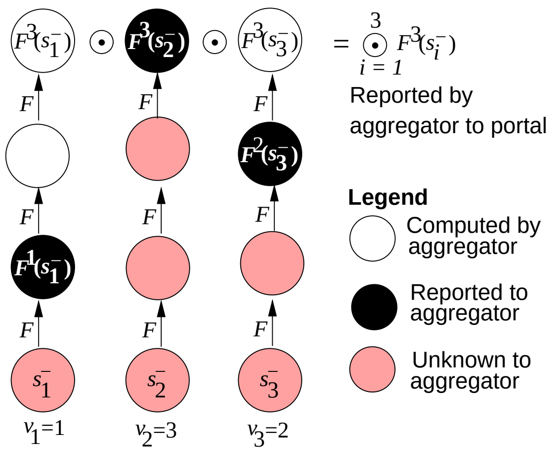


Figure 2: Secure Max with one-way chains.

recent epoch. Each sensor i generates:

1. $s_i^+ = \{i, \text{MAC}_{K_i}(v_i \| \text{epoch}\#)\}$, which is called sensor i's inflation-free proof. This is basically an authentication code generated by the sensor on its reading.

2. $s_i^- = \{\text{MAC}_{K_i}(\text{epoch}\#)\}$, which is called sensor i's deflation-free proof. To prevent attacks under a random oracle model, the MAC is generated with a full-domain hash function [4].

Including the epoch number is a standard technique to prevent replay attacks where aggregators use stale data. Given the properties of MAC, only sensor i can generate s_i^+ and s_i^-. Thus, for a given epoch, the portal can always verify the validity of s_i^+. The naive approach of securely computing Max would be for the aggregator to forward all the individual sensor readings and the s_i^+'s to the portal, which obviously defeats the purpose of outsourced aggregation.

To avoid this, we decompose the security requirements into two aspects: preventing inflation and preventing deflation (i.e., preventing reporting a value larger/smaller than the true Max). To prevent inflation, it is sufficient to require the aggregator to include the s_i^+ from the sensor i that the aggregator claims to have reported the Max value. Obviously, this would at least confirm that sensor i's true reading is indeed v_i, and in turn that the Max is *at least* v_i and there is no malicious inflation.

Preventing deflation is much harder. We need to verify that the aggregator did not throw away the true Max. Our key idea is to construct one-way chains whose seeds are all the s_i^-'s (Figure 2). Specifically, let F be some one-way function. Sensor i reports to the aggregator $F^{v_i}(s_i^-)$ (i.e., the SEAL at position v_i on the one-way chain rooted at s_i^-). In Figure 2, these SEALs correspond to the solid black circles. Let v_m be the maximum sensor reading seen by the aggregator. The aggregator then rolls the SEAL $F^{v_i}(s_i^-)$ forward by a certain number of steps to obtain the SEAL $F^{v_m}(s_i^-)$ at position v_m. This is represented by the top row of circles in Figure 2. After performing this operation for all the $F^{v_i}(s_i^-)$'s, the aggregator uses some proper folding function $\odot$ to fold all the SEALs into $\odot_i F^{v_m}(s_i^-)$. For example, if F is MD5, then the folding function can be simple XOR. This acts as a concise proof that no values greater than v_m were reported by any sensor. Intuitively, for the aggregator to compute the SEAL at position v_m on the one-way chain rooted at s_i^-, sensor i must have released some SEAL at position $\leq v_m$, implying that $v_i \leq v_m$.

General protocol skeleton. Before presenting the complete protocol for computing Max, we set up a general protocol skeleton that will be used throughout this paper. In the protocol skeleton, each sensor uses a *synopsis generation function* $SG()$ to generate a *verifiable synopsis (VS)*, based on its reading (and static attributes). A VS is a concise structure that contains information for both answering the query and proving the correctness of the answer. The sensor then sends the VS to its aggregator, which will store the VS. For each query, an aggregator uses a *synopsis aggregation function* $SA(,)$ to combine all VS's received from its children into a single VS, and sends it to the portal (or, to its parent aggregator, in case of hierarchical aggregation in later sections). Our synopsis aggregation functions are associative and commutative. Therefore, without loss of generality, we discuss $SA(,)$ for aggregating only two VS's—more than two VS's can be aggregated by applying $SA()$ multiple times. Finally, the portal uses the *synopsis evaluation function* $SE()$ to evaluate the VS received from the aggregator to obtain the query answer. $SE()$ also creates a *reference synopsis (RS)*, based on static metadata of the sensors involved in the query. $SE()$ then compares the RS against certain information in the VS. The answer is accepted if and only if they match. With the above protocol skeleton, when we present our protocols, we only need to describe the three functions $SG()$, $SA(,)$, and $SE()$.

Secure Max protocol. We now present the protocol for secure Max computation.

- Synopsis generation $SG()$: Sensor i with a reading v_i generates the VS:

$$(v_i, s_i^+, F^{v_i}(s_i^-))$$

- Synopsis aggregation $SA()$: Consider two VS's $\alpha_i = (v_i, s_i^+, F^{v_i}(s_i^-))$ and $\alpha_j = (v_j, s_j^+, F^{v_j}(s_j^-))$. Let v_m be the globally maximum value, which is known by the aggregator (since we have only a single aggregator here). $SA()$ aggregates them into

$$(v_m, s_m^+, \odot_i F^{v_m}(s_i^-))$$

- Synopsis evaluation $SE()$: To verify the correctness of a VS $\sigma = (v_m, s_m^+, f)$, $SE()$ first verify the validity of s_m^+. Then it computes all individual s_i^-'s and folds them together to create the RS of $\odot_i F^{v_m}(s_i^-)$. The answer v_m is accepted if and only if this RS equals the f from σ.

THEOREM 4.1. *If the portal accepts a result in the above protocol, the accepted result must be the correct maximum of values reported by all sensors.*

Proof (sketch). Let the true maximum of the reported values be v_m. Let the result reported by the aggregator be v'_m. Prove by contradiction and consider two cases. Case 1: $v'_m > v_m$. To pass verification, x must be $s_i^+ = \{v'_m, \text{MAC}_{K_i}(v'_m \| \text{epoch}\#)\}$ for some i. Since no sensor has a reading of v'_m, the adversary must have successfully forged the MAC, which is impossible. Case 2: $v'_m < v_m$. Let the sensor m be the sensor reporting the true maximum v_m.

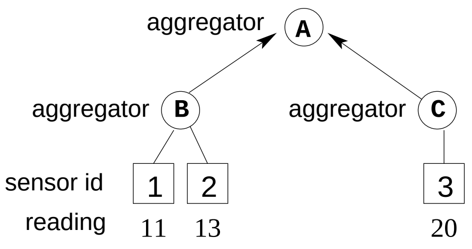


Figure 3: Example for hierarchical aggregators.

Since F is a good one way hash function that maps every input value to a random point on the range, SEALs of (uniformly) randomly chosen inputs are not correlated (i.e., they are also uniformly randomly distributed). Suppose the aggregator was able to guess $\odot_i F^{v'_m}(s_i)$ with non-negligible probability. Since SEALs are uncorrelated, this implies some non-trivial knowledge of the distribution of the possible values of $F^{v'_m}(s_m)$. However, since the aggregator knows only $F^{v_m}(s_m)$ which is a subsequent image of $F^{v'_m}(s_m)$, this implies the one-way function leaks information about preimages (i.e., it is not pre-image resistant), which violates the one-way property. □

Note that our protocol is *optimally secure* (according to the definition in [6]) since the aggregator, by tampering with the aggregation process, can not induce the portal to accept an incorrect aggregation result.

Heterogeneous data reporting frequency. We have been assuming that each sensor reports one reading per epoch. It is trivial to extend to cases where a sensor reports only one reading per multiple epochs. Namely, as long as the portal is aware of the reporting schedule of each sensor, it can always use the proper epoch numbers for different sensors when verifying the answer. For sensors reporting multiple readings in one epoch, a malicious aggregator will be able to bias towards any one of the readings. But we still allow sensors to submit multiple readings in our protocol, so that a benign aggregator can still use the latest reading.

4.2 Max with Hierarchical Aggregators

Our previous protocol cannot be directly used in a hierarchical setting. As an example, consider Figure 3, where a non-leaf aggregator A has two child aggregators B and C. Suppose that B receives readings from sensors 1 and 2 with value of 11 and 13, respectively. Suppose that C receives reading from sensor 3 with reading 20. A will then receive, from B and C respectively, two VS's $(13, s_2^+, F^{13}(s_1^-) \odot F^{13}(s_2^-))$ and $(20, s_3^+, F^{20}(s_3^-))$. According to the synopsis aggregation function, A should now aggregate the two VS's into $(20, s_3^+, F^{20}(s_1^-) \odot F^{20}(s_2^-) \odot F^{20}(s_3^-))$. The problem is that A cannot easily produce $F^{20}(s_1^-) \odot F^{20}(s_2^-) \odot F^{20}(s_3^-)$— It only knows $F^{13}(s_1^-) \odot F^{13}(s_2^-)$ (from B) and $F^{20}(s_3^-)$ (from C). In particular, $F^{13}(s_1^-)$ and $F^{13}(s_2^-)$ have already been folded together, so A cannot compute $F^{20}(s_1^-)$ or $F^{20}(s_2^-)$ to produce $F^{20}(s_1^-) \odot F^{20}(s_2^-)$. On the other hand, B has to fold $F^{13}(s_1^-)$ and $F^{13}(s_1^-)$ together, otherwise it defeats the purpose of hierarchical aggregation.

Leveraging homomorphic functions. Thus, hierarchical aggregation requires that one-way chains can be rolled forward even after they have been folded together with an operation like $\odot$. More precisely, we require a one-way function F and a folding operator $\odot$ such that, the following two processes yield identical results:

1. Individually roll a set of one-way chains forward with F for $x + y$ steps. Then fold the results together using $\odot$.

2. Individually roll a set of one-way chains forward with F, x steps. Then fold the results together using $\odot$. Finally, roll this folded result forward with F further by y steps.

One can easily show that the sufficient and necessary condition for the above property is for F to be *homomorphic* with respect to the operator $\odot$ both in the domain and the range, such that $F(x_1 \odot x_2) = F(x_1) \odot F(x_2)$. Interestingly, RSA encryption is homomorphic with respect to the folding operator modulo multiplication (hash functions such as SHA1 and MD5 are not homomorphic). RSA encryption is performed by exponentiating the plaintext by e, modulo some number m. Hence, $E(x_1 \cdot x_2) = (x_1 \cdot x_2)^e \bmod m = (x_1^e \bmod m) \cdot (x_2^e \bmod m) = E(x_1) \cdot E(x_2)$. Thus, two SEALs satisfy our desired property as long as they are generated using the same RSA encryption key (even if the seeds are generated with different MAC keys).

Most existing works leveraging such property of RSA use the homomorphic property of RSA signatures (instead of encryptions). They rely on the property that the adversary (not knowing the secret key) cannot generate an RSA signature. In contrast, we use the homomorphic property of RSA encryption and rely on the property that an encrypted message cannot be decrypted without knowing the secret key. In fact, our protocols (in Sections 4 through 6) do not use expensive RSA decryption or signature generation, which makes our protocols computationally efficient. RSA encryption is significantly cheaper than decryption and signature generation. For example, with 1024-bit keys, an Intel 2.5GHz desktop can perform around $20,000$ encryptions of a small seed (as in our protocol), which is $\approx 30\times$ faster than decryption or signature generation [1].

From now on, we will always use RSA encryption as F and modulo m multiplication as the folding function $\odot$. While individual RSA operations are more expensive than hashing, leveraging the homomorphic property of RSA, Section 5 will develop optimizations that can make the overhead of using RSA-based chains even smaller than using hashing-based chains in our protocols.

5. COUNT AND RELATED QUERIES

This and the next two sections will generalize the secure Max protocol developed earlier to a wide range of aggregates (Table 1). We will always consider hierarchical aggregators and use RSA encryption as F.

5.1 Basic Protocols

Reducing Count to Max. The Count aggregate asks for the number of sensors whose readings satisfy certain predicate (e.g., number of sensors sensing fire). Our secure Count protocol reduces Count to Max, based on the approximate algorithm by Alon, Matias, and Szegedy (AMS) [2]. Every sensor i whose reading satisfies the predicate (independently) picks a random positive integer v_i, such that integer x is chosen with probability of 2^{-x}. Sensors whose readings

Basic aggregates	Trivial extensions
Max (§ 4)	Min
Count(§ 5)	Count Distinct, Sum, Mean Count-Min Sketch
Top-k Readings (§ 6)	Uniform Sample, k-th Moments Quantiles
Top-k Groups (§ 7)	Popular Items, Frequent Items Threshold Groups

Table 1: Aggregates that can be computed in our framework.

do not satisfy the predicate simply pick 0. The value v_i is usually called a *sketch* (inherited from streaming database terminology). One can use our secure Max protocol to compute v_m, the maximum of all v_i's, and can estimate the Count to be 2^{v_m}. To reduce estimation error, one can invoke multiple independent instances, take the average $\bar{v}$ of all v_m values, and estimate the count as $2^{\bar{v}}$. See [2] for rigorous arguments on the approximation error.

As mentioned before, Count is robust against a small fraction of malicious sensors. Namely, Count will not be impacted significantly when a small fraction of malicious sensors submit fake readings. On the other hand, Max is not robust even against a single malicious senor. In the above reduction from Count to Max, it is possible for a malicious sensor i to simply pick a large v_i, without following the required probability distribution. This will make Count no longer robust. We use existing techniques [5] to prevent such problem. Namely, we require that each sensor i picks its random value v_i using a pseudo-random number generator initialized with some known seed (e.g., the sensor's id). When the portal verifies the s_i^+ corresponding to the Max, it (knowing the seed) also verifies whether sensor i picked the random integer in the proper way. This will make Count robust. We will assume such technique for all remaining aggregates in this paper.

Instead of the AMS algorithm, we can also readily use several other approximate counting algorithms to reduce Count to Max. For example, we can map each sensor (that satisfies the predicate) to some uniformly random value within $[0, 1]$. The Count is then estimated to be $1/(1 - v_m)$ where v_m is the maximum value [3]. We can also use the algorithm by Flajolet and Martin (FM) [10], where each sensor (that satisfies the predicate) generates a bit vector with exactly one 1-bit. The 1-bit is at index x with probability of 2^{-x}. Let V be the bitwise OR of all bit vectors from all sensors. Note that bit-wise OR is exactly bit-wise Max. The Count is estimated to be $2^{y-1}/0.7735$ where y is the index of the lowest-order 0-bit in V [10].

Aggregates related to Count. Many aggregates are related to or can be reduced to Count. The Count Distinct aggregate, which asks for the number of unique sensor readings, can be easily computed with our Count protocol. The only modification needed is that when generating a random AMS sketch, a sensor needs to use its reading (e.g., the name of the animal seen by the sensor) as the seed of a deterministic pseudorandom number generator. The aggregate Sum can be naturally reduced to Count as well. Conceptually, a sensor with a reading of v_i simply pretends to be v_i different sensors, creates v_i AMS sketches, and merges them. In the actual protocol, existing optimizations [7, 17] can readily avoid the need of producing and merging v_i sketches.

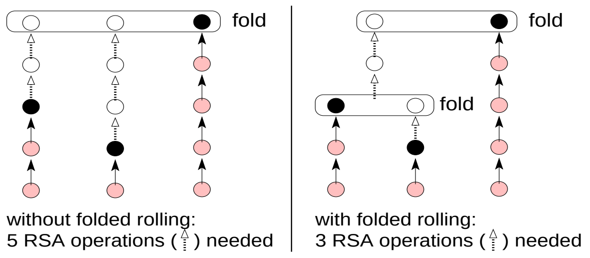


Figure 4: Folded rolling. Here we need to roll the two shorter chains forward to the 5th position to fold with the longest chain.

The aggregate Mean is simply Sum divided by Count. Finally, the Count-Min sketch [8], which consists of a matrix of Count elements, trivially reduces to Count. The Count-Min sketch can in turn be used [8] to answer kth statistical moments, median, quantiles, inner product queries, etc.

High overhead of naive reduction. Directly using the Max protocol as a blackbox in the above reduction from Count to Max will unfortunately, pose excessive computation load on the portal and the aggregator, due to the large number of the Max instances needed for each Count. Consider a single instance of the AMS algorithm (and thus our Max protocol). Let N be the total number of sensors and v_m be the maximum AMS sketch value, where v_m is usually on the order of $\log N$. When generating the reference synopsis (RS), the portal needs to generate all the s_i^-'s, roll them to position v_m, and then fold them together. This takes $N \cdot v_m$ rolling operations and N folding operations. Now with J instances of the AMS algorithm for each Count, it becomes $J \cdot N \cdot v_m$ rolling operations and $J \cdot N$ folding operations. To have reasonable estimation error, J usually needs to be large (e.g., a few hundreds). With $J = 300$, $N = 100,000$, and $v_m \approx \log N \approx 17$, the portal needs to perform 510 million rolling operations and 30 million folding operations. For RSA encryption based one-way chains, this will take more than 7 hours (assuming 20,000 RSA encryptions/sec [1])! Even if we use cheaper hash functions (e.g., MD5 or SHA-1) to construct one-way chains, 510 million rolling operations will take more than 510 seconds (assuming 1 million hash operations/sec [1]). Such overhead is obviously not acceptable and will offset the advantage of outsource aggregation itself. In addition to the creation of RS, the protocol has similar overheads elsewhere.

Even though RSA encryptions are substantially more expensive than hashing, interestingly, its homomorphic property enables us to develop a number of aggressive optimizations. These optimizations together will help to reduce the overhead by up to 7 orders of magnitude and make the overhead quite acceptable. With our optimizations, verifying a Count query answer requires only up to $2 \log N$ encryptions and $2 \log N$ multiplications, which takes less than 2 milliseconds for $N = 100,000$. We believe that this is an interesting finding by itself, as using RSA enables us to achieve an overhead smaller than using hashing. We next describe our two key optimizations. These optimizations also apply to Max, though they are obviously more important for Count due to the J factor.

5.2 Reducing Rolling Cost via Folded Rolling

A key source of overhead in our protocol is that in several places (e.g., when generating the RS or when merging VS), the protocol needs to roll a large number of SEALs forward to some position x, and then fold the SEALs at position x together. In *folded rolling*, we aggressively fold SEALs whenever possible so that we only need to roll forward the (single) folded SEAL instead of multiple ones. To maximize the benefit, we will roll the SEALs in the order of their initial positions (from smallest to largest). For example in Figure 4, we need to fold together three SEALs, at positions 3, 2, and 5 respectively. Doing so naïvely would require five RSA encryptions. With folded rolling, we start from the SEAL at the lowest position (i.e., the SEAL on the second chain at position 2), and roll it forward to position 3. We then immediately fold it with the SEAL on the first chain (since they are at the same position). Finally, we roll the folded SEAL to position 5 and then fold with the last SEAL. As shown, this requires only 3 RSA encryptions. Folded rolling relies on the homomorphic property of RSA encryptions and thus is not applicable to MD5 or SHA-1.

One can trivially prove that with folded rolling, regardless of the total number of SEALs, the total number of RSA encryptions will be no larger than the difference of the highest and the smallest positions among all SEALs. A naïve implementation of folded rolling requires sorting all the SEALs (e.g., 1000 SEALs) according to their positions, which can add non-trivial overhead. However, notice that for Count, the highest position of the SEALs is only on the order of $\log N$. Thus we can always first fold all SEALs at the same positions together, and then sort the resulting $\log N$ SEALs and apply folded rolling.

Folded rolling is effective in reducing overheads in several places in our Count protocol. First, for each of the J instances of the Max protocol, an aggregator needs to aggregate all the VS's received from its children. Doing so naively would incur up to $J \cdot C \cdot \log N$ RSA encryptions, where C is the number of children. Folded rolling reduces this to $J \cdot \log N$, which is independent of C.

Second, when the portal receives the J VS's (one for each instance) from the root aggregator, the portal needs to verify each of them. To enable folded rolling, all J instances use the same RSA encryption function (i.e., with the same encryption key) to construct their one way chains. For the jth instance, a sensor i will generate its deflation-free proof as $s^-_{i,j} = \{MAC_{K_i}(\text{epoch}\#\|j)\}$. This $s^-_{i,j}$ is then used as the seed of the one-way chain for sensor i in instance j. Let the folded SEAL in the VS submitted to the portal for the jth instance be x_j. Namely, x_j should be $\odot_i F^{v_m}(s^-_{i,j})$ (when v_m is the maximum for this instance) if the aggregators are all honest. For the jth instance, with folded rolling, the portal can create the RS by first folding all $s^-_{i,j}$'s (for the given j) together, and then rolling forward to v_m. Doing so reduces the total number of RSA encryptions from $J \cdot N \cdot \log N$ to $J \cdot \log N$.

A more careful look reveals that one can even apply folded rolling across the J instances. Let w be the maximum of all J sketches, which is roughly $\log N$. Instead of verifying x_j's individually, the portal first conceptually rolls all x_j's to position w and then folds them together. Obviously, we can apply folded rolling here so that the total number of RSA encryptions needed is no larger than $\log N$. Next, for the RS, the portal conceptually roll all the $s^-_{i,j}$'s forward to w and fold them. Again, folded rolling enables us to do so with no more than $\log N$ RSA encryptions. Finally, the portal compares the two folded SEALs to see if they match. This aggressive optimization helps to reduce the number of RSA encryptions on the portal to $2 \log N$ per verification, which is completely independent of J.

Finally, as a beneficial side effect, the root aggregator can actually fold all those x_j's at identical positions (on the one-way chains) together before sending to the portal. Since there can be at most $\log N$ distinct positions for all x_j's, the total number of SEALs sent from the root aggregator to the portal will be no more than $\log N$. For $N = 100,000$ and 1024-bit SEALs, this will be less than 2KB. The root aggregator can also fold J inflation-free proof $s^+_{i,j}$'s together (using XOR) before sending to the portal.

5.3 Reducing Folding Costs via B-Tree

Folded rolling only helps to reduce the number of rolling operations. For each query, the portal still needs to generate an RS which takes up to $J \cdot N$ folding operations (modulo multiplication) to fold all the $s^-_{i,j}$ together. To better understand our optimization, let us first focus on queries with range predicates that cover sensors with continuous id ranges. For these queries, the portal can create a binary tree (or a B+-tree or an R-Tree) at the beginning of each epoch. The leaves of the tree are all the $s^-_{i,j}$'s, ordered by i from left to right. Each internal node of the tree is the modulo multiplication of its children. Constructing such a tree still takes $J \cdot N$ folding operations, but we only need to do this once per epoch.

One can easily show that regardless of the id range required by the query, we can always find no more than $2 \log N$ disjoint subtrees, such that the set of all the leaves of these subtrees is exactly the set of sensors falling within that range. Roots of these subtrees can be found with a preorder traversal of the tree [13]. Multiplying all the root of these subtrees then yields the required folded result. Doing so thus reduces the number of multiplications from $J \cdot N$ to $2 \log N$ per query. It is trivial to generalize such technique to dealing with range predicates based on other static attributes of the sensors. One can simply build one tree for each static attribute.

6. TOP-k READINGS AND RELATED QUERIES

We move on to Top-k Readings queries and other aggregates related to Top-k Readings. A Top-k Readings query asks for the top-k values among the sensors' readings. Without loss of generality, we assume that sensor readings are all distinct (e.g., by appending the sensor id).

6.1 Protocol Intuition

One could trivially answer a Top-k Readings query by invoking our Max protocol (as a black box) *sequentially* for k times. Namely, the first invocation will return the largest value, together with which sensor i_1 generated this Max. The second invocation of the protocol will simply exclude sensor i_1, and then compute the Max again. To exclude sensor i_1, the root aggregator (after the first invocation and after knowing i_1) needs to potentially broadcast i_1 to all the aggregators, so that they can exclude sensor i_1. The second invocation will then produce the second largest value, as well as which sensor i_2 generated that value. More precisely, for the second largest value, the VS submitted to the portal will

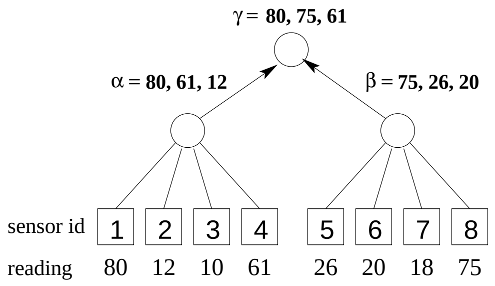


Figure 5: Example for Top-k Readings query.

be

$$(v_{i_2}, s^+_{i_2}, \odot_{i \neq i_1} F^{v_{i_2}}(s^-_i))$$

Invoking the Max protocol in this way sequentially for k times can then answer the Top-k Readings query.

Invoking Max sequentially, however, can incur unnecessary delay. Thus we aim to mimic the above process via k parallel invocations of the Max protocol. It is not immediately obvious how to do so: For the ith largest value, we need to produce folded SEALs of all sensors excluding those generating the top $i-1$ values. The challenge is that each aggregator, when performing the aggregation, does not yet know which sensors generated the *globally* top $i-1$ values. Consider the example in Figure 5 where $k=3$ and we want to aggregate VS α and VS β. Suppose that each VS contains the top-k values observed so far. The VS α covers sensors 1 through 4, and the Top-3 values observed so far are 80, 61, and 12. It is easy to include similar verification information in α as before in the Max protocol. For example, to prove that 61 is the maximum reading seen so far excluding the reading from sensor 1, α can include $F^{61}(s^-_2) \odot F^{61}(s^-_3) \odot F^{61}(s^-_4)$. β covers sensors 5 through 8 and the Top-3 values observed so far are 75, 26, and 20. It is obvious that when merging two synopses, we can simply take the top-k values in their union. Thus the new Top-3 values after aggregation should be 80, 75, and 61.

In the new VS, the verification information for 61 needs to be the folded SEALs of sensor 2 through 7, since 61 is the 3rd largest value and we need to exclude sensor 1 and 8 who generated the top 2 values. Notice that for the value 61, α only contains the folded SEALs of sensors 2, 3, and 4 (i.e., $F^{61}(s^-_2) \odot F^{61}(s^-_3) \odot F^{61}(s^-_4)$). We do not know $F^{61}(s^-_5)$, $F^{61}(s^-_6)$, or $F^{61}(s^-_7)$. Fortunately, the verification information for 26 in β should contain the folded SEALs from sensors 5, 6, and 7 (i.e., $F^{26}(s^-_5) \odot F^{26}(s^-_6) \odot F^{26}(s^-_7)$). Since $61 > 26$, all we need to do is to roll this folded SEAL forward, and then fold it with the SEAL from α.

Notice that the above is not entirely obvious: For example, if β only contained $F^{20}(s^-_6) \odot F^{20}(s^-_7)$ and $F^{75}(s^-_5) \odot F^{75}(s^-_6) \odot F^{75}(s^-_7) \odot F^{75}(s^-_8)$, we would not be able to generate the verification information for 61. However, we will prove later that we can always generate the verification information for 61 by finding the largest value in β that is smaller than 61, and use its corresponding verification information. This means that the previous problematic scenario will never occur.

6.2 Protocol Description

For any given aggregator in the aggregation tree, its *coverset* is defined as the set of all sensors in its subtree. For Top-k Readings query, the VS sent by an aggregator with coverset U has the form of:

$$\{VS_{i_1}, VS_{i_2}, ..., VS_{i_{k'}}\}, \text{ where}$$
$$VS_{i_j} = (v_{i_j}, s^+_{i_j}, \odot_{i \in U \setminus \{i_1, i_2, ..., i_{j-1}\}} F^{v_{i_j}}(s^-_i)), \text{ and}$$
$$v_{i_1} > v_{i_2} > ... > v_{i_{k'}}$$

The value of k' is within $[1, k]$. Roughly speaking, VS_{i_1} through $VS_{i_{k'}}$ contains the top k' values observed so far, and these values are from sensors i_1 through $i_{k'}$. The value k' can be smaller than k initially when we observe less than k values.

For $SG()$, a sensor i with reading v_i generates:

$$\{(v_i, s^+_i, F^{v_i}(s^-_i))\}$$

Consider two VS's $\alpha = \{\alpha_{i_1}, \alpha_{i_2}, ...\}$ and $\beta = \{\beta_{j_1}, \beta_{j_2}, ...\}$. The $SA()$ function will aggregate them into $\gamma = \{\gamma_{t_1}, \gamma_{t_2}, ...\}$. Here γ_{t_1}, γ_{t_2}, ..., simply correspond to the top-k values among α_{i_1}, α_{i_2}, ..., and β_{j_1}, β_{j_2}, ... Without loss of generality, consider some $\alpha_x = (v_x, s^+_x, f_x)$ in α that is one of the new top-k values, where we need to construct a corresponding γ_x in γ. Let v_y be the largest value in β (i.e., among v_{j_1}, v_{j_2}, ...) such that $v_x > v_y$.

If v_y does exist, let the element $\beta_y = (v_y, s^+_y, f_y)$. We construct γ_x as $(v_x, s^+_x, f_x \odot F^{(v_x - v_y)}(f_y))$. To understand why doing so is correct, suppose α covers U_α and β covers U_β. Then f_x covers all sensors in U_α except those sensors with readings larger than v_x. Similarly, f_y covers all sensors in U_β except those sensors with readings larger than v_y. But because $v_x > v_y$ and U_β does not contain any sensor with a reading between v_x and v_y, f_y actually covers all sensors in U_β except those sensors with readings larger than v_x. As a result, $f_x \odot F^{(v_x - v_y)}(f_y)$ exactly covers all sensors in $U_\alpha \cup U_\beta$ except those sensors with readings larger than v_x.

If all values in β are larger than v_x, then v_y does not exist and we set $\gamma_x = \alpha_x$. In such case, β must have less than k values, since otherwise v_x can never be among the new top-k. Let U_β be the set of sensors covered by β. Then all sensors in U_β have readings larger than v_x. As a result, we do not need to fold additional SEALs from any sensor in U_β.

Synopsis evaluation is trivial. Namely, the portal verifies the largest value in the same way as for Max, and then verifies the second largest value as the Max excluding the largest value, and so on.

6.3 Correctness and Optimizations

The correctness of the above protocol naturally follows from Theorem 4.1. Namely, Theorem 4.1 already tells us that the largest value reported must be correct. When verifying the second largest value, the portal excludes the sensor i_1 that generated the maximum value. Applying Theorem 4.1 then tells us that the value must be the largest value if we exclude sensor i_1, which means that it must be the second largest value. Continuing this argument can prove the correctness of all the top-k values reported.

To further reduce the overhead of verification, the portal should verify the VS_{i_k}, ..., VS_{i_2}, VS_{i_1} according to that order. After constructing the RS of

$$\rho_k = \odot_{i \notin \{i_1, i_2, ..., i_{k-1}\}} F^{v_{i_k}}(s^-_i)$$

for VS_{i_k}, the portal can generate the RS for $VS_{i_{k-1}}$ as $\rho_{k-1} = F^{v_{i_{k-1}} - v_{i_k}}(\rho_k \odot s^-_{i_{k-1}})$. Doing so will make the number of RSA encryption operations independent of k.

6.4 Aggregates Reducible to Top-k Readings

A Uniform Samples query can be readily reduced [17] to a Top-k Readings query, by having each sensor generate a uniformly random number in $(0, 1)$ and by returning the sensors with the top-k numbers. It is known [17] that k-th statistical moments, quantiles, and median can all be computed from uniform samples, and can thus be securely answered via Top-k Readings query.

7. TOP-K GROUPS AND RELATED QUERIES

We move on to more complex queries which we call *Top-k Groups* queries. We continue assuming that sensor readings are all distinct. For a Top-k Group query, each sensor is first mapped to some *group*, where the mapping can be either static or dynamic. For example, a group may correspond to some geographic region. An immobile sensor will thus belong to some fixed group, while a mobile sensor may belong to different groups at different times, making the mapping dynamic. The mapping can also be dynamic if the groups are defined based on sensor readings (e.g., when we later consider frequent items and popular items query). We assume that each sensor knows which group it currently belongs to, while the portal may not have such information (since the groups can be dynamic). Each group has a local max, called the *value* of the group, which is the maximum sensor reading within that group. A Top-k Groups query asks for k groups with the k largest values. Thus, if some group has many sensors with large readings, the group will only "occupy" one position in the final answer, and the query can still discover other $k-1$ groups. In comparison, a Top-k Readings query may return k readings from sensors in that single group.

Top-k Groups queries are interesting mainly because some other common queries, such as Most Popular Items and Frequent Items, can be reduced to Top-k Groups queries. Top-k Groups queries can also have interesting real world applications themselves when Top-k Readings queries are less appropriate. For example, consider a pressure sensor network in a ski resort that detects the high pressure caused by snow mass in avalanche and uses such information to warn visitors about dangerous regions. A Top-k Readings query may return readings from sensors in one single region with a large avalanche. In contrast, a Top-k Groups query will identify k most dangerous regions.

Challenge. While Top-k Groups query appears similar as Top-k Readings, it is substantially harder. The reason is that in Top-k Readings, to verify the correctness of the second largest value, we only need to fold the SEALs of all sensors excluding the sensor reporting the largest value. For Top-k Groups query, to prove the correctness of the second largest value, we need to exclude all sensors in the top-1 group. This is difficult since the portal does not know which sensors belong to the top-1 group. Directly sending the group membership to the portal would incur linear communication complexity (in terms of the number of sensors), which can be prohibitive.

7.1 Protocol Intuition

Define g_i to be the id of the group containing sensor i, which is known to sensor i. Let sensor i_1 (with value v_{i_1}) be the sensor with the globally maximum reading, which also implies that g_{i_1} is the group with the largest value. Let sensor i_2 (with value v_{i_2}) be the sensor with the maximum reading except all those sensors in group g_{i_1}. The group g_{i_2} is thus the group with the second largest value. Define i_3 (g_{i_3}) through i_k (g_{i_k}) similarly. We further define a special group with id g_0 that includes all sensors not in any of the groups g_{i_1}, g_{i_2}, ..., g_{i_k}. We explained earlier that without group membership information, the portal cannot verify the folded SEALs from the sensors in individual groups.

Notice that each sensor must be either in g_0 or in exactly one of the k group. To see how this can help, assume for now that each sensor knows whether it belongs to one of the top-k groups. For each group g_i, we define an independent one-way function F_{g_i}. We similarly define F_0 for the special group g_0. We further assume (only for this protocol) that the portal (and only the portal) knows the trap-door of these functions so that the portal can roll the one-way chain backward. For example, our implementation uses independent RSA encryption functions as F_0, F_{g_1}, F_{g_2}, ..., F_{g_k}, where the portal knows all the corresponding decryption keys. By "independent", we mean that each such function uses an independent RSA key pair under an independent modulus. Independent modulus will later help us to prevent the Common Modulus Attack [24] on RSA.

Imagine now that we invoke k parallel instances of the max protocol for groups g_{i_1}, g_{i_2}, ..., g_{i_k}. We also invoke one additional parallel instance for the special group g_0 (i.e., for all the sensors not in one of the top-k groups). The portal will eventually receive $k + 1$ folded SEALs $F_{g_{i_1}}^{v_{i_1}}(x_1)$ (for group g_{i_1}), ..., $F_{g_{i_k}}^{v_{i_k}}(x_k)$ (for group g_{i_k}), and $F_0^{v_0}(x_0)$ (for group g_0). Here v_0 is the largest reading from the sensors in group g_0. The portal can then roll these SEALs backward to obtain x_1, x_2, ..., x_k, and x_0. Each x_j here is the folded SEALs of those sensors in group g_j. Though the portal still cannot verify the individual x_j's, it can verify that $x_1 \odot ... \odot x_k \odot x_0 = \odot_i(s_i^-)$. The security of this protocol comes from the fact that each s_i^- must be folded into one and exactly one of the x_j's.

In reality, a sensor does not know whether it belongs to one of the top-k groups. A sensor i will thus generate both $F_{g_i}^{v_i}(s_i^-)$ and $F_0^{v_i}(s_i^-)$, and submit to some aggregator. When using RSA encryption as the one-way function, notice that the moduli of the two encryption functions (i.e., F_{g_i} and F_0) are different, which prevents the Common Modulus Attack [24]. Submitting both $F_{g_i}^{v_i}(s_i^-)$ and $F_0^{v_i}(s_i^-)$ enables the aggregator to perform proper folding regardless of whether group g_i is among top-k or not. On the other hand, providing both $F_{g_i}^{v_i}(s_i^-)$ and $F_0^{v_i}(s_i^-)$ also gives a malicious aggregator the freedom to choose whether to classify sensor i into group g_i or group g_0. Consider an example where sensor i_1 generated the globally maximum 15, together with $F_{g_{i_1}}^{15}(s_{i_1}^-)$ and $F_0^{15}(s_{i_1}^-)$. A malicious aggregator could potentially fold $F_0^{15}(s_{i_1}^-)$ into the folded SEAL for group g_0. Doing so can "hide" this maximum and thus deflate the reported value for group g_{i_1}. A closer look, however, will reveal that if this indeed happens, then v_0 will reach 15. It is then easy for the portal to detect that the reported v_0 is larger than the values of some of the top-k groups.

One can generalize the above argument to make it more precise (see Figure 6). Remember that v_{i_k} denotes the value of the group with the kth largest value (e.g., $v_{i_k} = 10$ in Figure 6). For any sensor i with $v_i \geq v_{i_k}$, if the aggregator

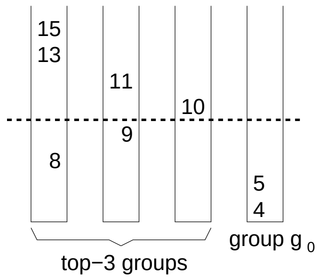


Figure 6: An example with 8 sensors. The numbers are sensor readings. For all sensors with readings below 10, regardless of which group they are classified into, the final result (i.e., 15, 11, and 10) will not be affected.

maliciously uses $F_0^{v_i}(s_i^-)$ instead of $F_{g_i}^{v_i}(s_i^-)$, it will necessarily cause the reported v_0 to be larger than the reported values of some of the top-k groups. On the other hand, for any sensor i with $v_i < v_{i_k}$, even if the adversary maliciously classifies it into the wrong group (e.g., by using $F_0^{v_i}(s_i^-)$ instead of $F_{g_i}^{v_i}(s_i^-)$), it will never affect the correctness of the query answer (see Figure 6).

7.2 Protocol Description

We first re-define s_i^+ to include g_i:

$$s_i^+ = \{i, MAC_{K_i}(v_i \| g_i \| \text{epoch}\#)\}$$

For any sensor set U, define U_i to be *any* non-empty set such that $U_i \subseteq (U \cap \text{group } g_i)$. For Top-$k$ Groups query, the VS sent by an aggregator with coverset U has the form of:

$$\begin{aligned} VS &= \{VS_{i_1}, VS_{i_2}, ..., VS_{i_{k'}}, VS_0\}, \text{ where} \\ VS_x &= (v_x, g_x, s_x^+, \odot_{i \in U_x} F_{g_x}^{v_x}(s_i^-), \odot_{i \in U_x} F_0^{v_x}(s_i^-)) \\ &\quad \text{for } x = i_1, i_2, ..., i_{k'}, \\ VS_0 &= (v_0, \odot_{i \in U'} F_0^{v_0}(s_i^-)) \\ &\quad \text{for } U' = U \setminus (U_{i_1} \cup U_{i_2} \cup ... \cup U_{i_{k'}}), \\ &\quad v_{i_1} > v_{i_2} > ... > v_{i_{k'}} > v_0 \end{aligned}$$

The value of k' is within $[1, k]$. Roughly speaking, VS_{i_1} through $VS_{i_{k'}}$ contains the top k' local max from k' different groups (from sensors i_1 through $i_{k'}$) observed so far. VS_0 corresponds to the maximum value (v_0) observed excluding these k' groups. A VS_x contains the folded SEALs based on both F_{g_x} and F_0 because the intermediate aggregators do not know whether group g_x is among top-k or not.

For synopsis generation, a sensor i with reading v_i generates:

$$\{VS_{i_1} = (v_i, g_i, s_i^+, F_{g_i}^{v_i}(s_i^-), F_0^{v_i}(s_i^-)), VS_0 = (null)\}$$

Let $\alpha = \{\alpha_{i_1}, \alpha_{i_2}, ..., \alpha_0\}$ and $\beta = \{\beta_{j_1}, \beta_{j_2}, ..., \beta_0\}$. The synopsis aggregation function will aggregate the two synopses into $\gamma = \{\gamma_{t_1}, \gamma_{t_2}, ..., \gamma_0\}$. Here γ_{t_1}, γ_{t_2}, ..., simply correspond to the top-k groups among α_{i_1}, α_{i_2}, ..., and β_{j_1}, β_{j_2}, ... Notice that it is possible for some α_x and β_y to correspond to the same group, in which case they will be merged (i.e., by keeping the larger group value only).

Without loss of generality, consider some $\alpha_x = (v_x, g_x, s_x^+, f_x, f'_x)$ where v_x is the value of one of the new top-k groups. We need to construct a corresponding γ_x for γ. If there does not exist any $\beta_y = (v_y, g_y, s_y^+, f_y, f'_y)$ such that $g_x = g_y$, we simply set $\gamma_x = \alpha_x$. Otherwise it means that there exists a β_y such that α_x and β_y correspond to the same group. In such case, we must have $v_x > v_y$, since otherwise v_x will not be the value of one of the new top-k groups (i.e., it will be v_y instead). We construct γ_x to be

$$(v_x, g_x, s_x^+, f_x \odot F_{g_x}^{(v_y - v_x)}(f_y), f'_x \odot F_0^{(v_y - v_x)}(f'_y))$$

We also say that α_x and β_y have been *incorporated* into γ. Finally, we combine all the α_x's and γ_y's (including α_0 and β_0) that have not been incorporate into γ to construct $\gamma_0 = (v_0, \odot F_0^{v_0}(s_i^-))$. Here v_0 is the maximum value among all these α_x's and γ_y's. The folding operation is done after properly rolling forward all the SEALs based on F_0 (in α_x's and γ_y's) until the v_0th position.

For synopsis evaluation, let $\sigma = \{\sigma_{i_1}, ..., \sigma_{i_k}, \sigma_0\}$ be the VS reported to the portal. The portal first verifies i) all the s_i^+'s are valid, ii) v_{i_1} (from σ_{i_1}) > ... > v_{i_k} (from σ_{i_k}) > v_0 (from σ_0), and iii) g_{i_1}, g_{i_2}, ..., g_{i_k} are all distinct. Next, for each σ_x where $x = i_1$ through i_k, the portal decrypts $\odot F_{g_x}^{v_x}(s_i^-)$ for v_x times using the decryption key corresponding to g_x. This will recover $\odot s_i^-$. For σ_0, the portal similarly decrypts $\odot F_0^{v_0}(s_i^-)$ to get $\odot s_i^-$. The portal then folds all these $\odot s_i^-$'s together. Finally, the portal creates all the s_i^-'s itself and folds them together to see if the two results match.

THEOREM 7.1. *If the portal accepts a result in the above protocol, the accepted result must be correct.*

Proof (sketch). We prove the following statement for all integer $x \in [1, k]$: If the portal accepts a result in the above protocol, then conditioned upon the top-$(x-1)$ group values being correct, v_{i_x} must be the value of the group with the xth largest value. We define "top-0 group value being correct" to be an event that always happens. Obviously, the theorem can be trivially proved via an induction on x in the previous statement (if it indeed holds).

Let the group with the xth largest value be group g_m and let its value be v_m. Let g'_m and v'_m be the corresponding answers returned by the aggregator. We prove by contradiction and consider two cases. First consider $v'_m > v_m$. The portal must successfully verified some inflation-free proof s_i^+ for v'_m. Since the top-$(x-1)$ group values are all correct, we know that the sensors in the remaining groups all have values no larger than v_m and none of them will generate an s_i^+ for the value v'_m. It is possible that some sensor within the top-$(x-1)$ groups generated an s_i^+ for the value v'_m. However, the s_i^+ will also be only for the group id that corresponds to one of those top-$(x-1)$ groups. Thus the aggregator must have forged an s_i^+ corresponding to v'_m and g'_m, which is impossible.

Consider the second case of $v'_m < v_m$. Let the values of the top-k groups returned by the aggregator be v_{i_1} through v_{i_k} (where $v_{i_x} = v'_m$). Let the value of group 0 as returned by the aggregator be v_0. Since the portal accepts the answer, we have $v_{i_1} > ... > v_{i_k} > v_0$. Since the top-$(x-1)$ groups are all correct, we know that group g_m is not one of the top-$(x-1)$ groups returned by the aggregator. Now consider all the decryption operations done by the portal during the verification process. We claim that the total number of decryptions using the description key corresponding to g_m can be at most v'_m. The reason is that g_m does not correspond to the first $(x-1)$ values in $v_{i_1} > ... > v_{i_k} > v_0$, while all later values are no larger than v'_m. Similarly, the total number of decryption operations using the description key corresponding to g_0 can be at most v'_m as well. Now consider the sensor m with reading v_m in group g_m. It has only generated $F_{g_m}^{v_m}(s_m^-)$ and $F_0^{v_m}(s_m^-)$. To pass verification, the SEAL s_m^- must be present in the folded result after all the decryption operations. Since folding is secure, this implies that the aggregator must have obtained either $F_{g_m}^{v'_m}(s_m^-)$ or $F_0^{v'_m}(s_m^-)$, which is impossible given $v'_m < v_m$. □

7.3 Popular Items and Frequent Items

For Popular Items and Frequent Items queries, each sensor's reading is considered as an "item". A Popular Items query asks for the top-k items in terms of their occurrence counts. It is known [18] that a Popular Items query can be reduced to a Top-k Groups query as follows. Each sensor uses the value of its reading as its group id. Thus, sensors with the same reading (i.e., "item") are considered to be in the same group. Next, each sensor generates an AMS sketch, and the Top-k Groups query is executed on the values of AMS sketches of all sensors. In other words, the value of each group is given by the maximum of all AMS sketches in that group. Intuitively, the maximum AMS sketch within each group represents the occurrence count of the corresponding item. The approximation error can be control by using multiple AMS sketches.

A Frequent Items query asks for all items whose occurrence count is above certain threshold. This query can be reduced [17] to the *Threshold Groups* query, which is the same as a Top-k Groups query except that it returns all the groups whose values exceed a given threshold. A Threshold Groups query can be answered in almost exactly the same way as the Top-k Groups query—the only difference is that for synopsis aggregation, we need to keep all groups whose values exceed the threshold, instead of only the top k groups. To reduce a Frequent Items query to a Threshold Groups query, again it suffices to consider sensors with the same reading (i.e., "item") as in the same group, and then use the AMS sketch as the reading for the Threshold Groups query.

8. EVALUATION

We have implemented our secure aggregation protocols for sensor, aggregator, and portal, in C++, using the Crypto++ 5.5.2 library. This section evaluates our prototype using 2.5 GHz PC with Intel Core 2 Duo CPU and 2GB RAM. (We assume that the proxies connecting sensors to the Internet are desktop-class computers, so we use the same hardware for sensors/proxies as well.)

8.1 Basic Performance of SECOA

We first use a real dataset to demonstrate the advantages of outsourced aggregation. We use 16,106 stream gauge sensors provided by the U.S. Geological Survey (USGS) all over the U.S.A. The sensors report different real-time data such as water discharge, gauge height, water temperature, etc. We use a two-level aggregation tree with a single root aggregator and 17 leaf aggregators, and report the load of the most loaded aggregator. We consider Max, Count, Top-10 Readings, and Top-10 Groups queries. The range of sensor readings is $[0, 200]$. For Count, we use 300 AMS sketches to bound the relative approximation error within 10% with a

Query	Sensor (ms/reading)	Aggregator (ms/query)	Portal (ms/query)
Max	0.84	11.97	1.05
Count	35.97	158.9	1.11
Top-10 Readings	1.09	10.9	1.12
Top-10 Groups	0.78	8.2	80.9

Table 2: Average computation time (milliseconds)

probability of 0.9 [3]. We assume that all queries ask for the aggregate of readings from all sensors, which demonstrates the worst-case overhead.

Each SEAL in our implementation is 128 bytes while each MAC is 20 bytes. With all the optimizations in Section 5, for each Count query, the portal in our prototype needs to receive only less than 3KB data from the root aggregator (300 bytes for 300 AMS sketches, 2048 bytes for 16 SEALs, and the 20 bytes for folded s_i^+'s). The communication overhead at the portal for Max, Top-10 Readings, and Top-10 Groups is less than 0.5KB, 1.5KB, and 1.6KB, respectively. In contrast to such overhead, if the aggregators simply forward the raw data to the portal, then 16,106 signed readings from the sensors would need to be delivered to the portal, which will be at least 320KB (assuming signature is done via 20-byte MAC). This demonstrates the obvious benefit of in-network aggregation.

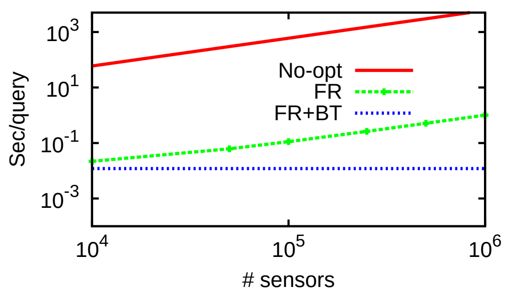
Table 2 further gives the computational overhead on the sensors, the aggregators, and the portal, when using our protocol. These overheads are rather small and acceptable in all cases. In particularly, the overhead on the portal is only about one millisecond per query except for Top-10 Groups query (which still only takes around 80 milliseconds). Notice that this is despite of the use of RSA encryption-based one-way chains in SECOA. The overhead on the aggregators is usually larger than the overhead on the portal. This is expected, since outsourced aggregation precisely intends to offload the load from the portal to the aggregators. Nevertheless, the absolute overhead on the aggregators remain quite acceptable (i.e., around 0.1 to 0.01 second per query).

8.2 Computational Overhead: A Deeper Look

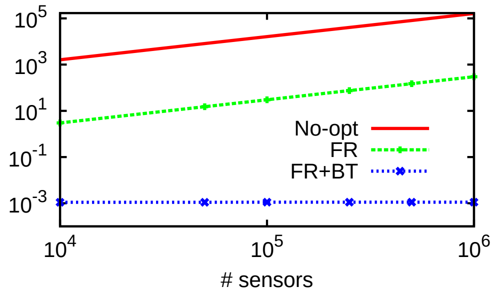
To have a deeper understanding of the computational overhead and in particular the utility of our optimizations, we run our prototype with different optimizations incrementally enabled. We will use Max and Count as examples. Figures 7(a) and (b) plot the overhead on the portal to verify the answer for Count and Max, respectively, as the total number of sensors in the system change. The results show that in all cases, our optimizations are critical and they help to reduce the computational overhead by up to 7 orders of magnitude. Figures 7(c) and (d) further plot the computational overhead on an aggregator, as a function of the number of children it has. There our optimizations help to reduce the overhead by roughly 2 to 3 orders of magnitude.

8.3 Benefits of No False Positives or Negatives

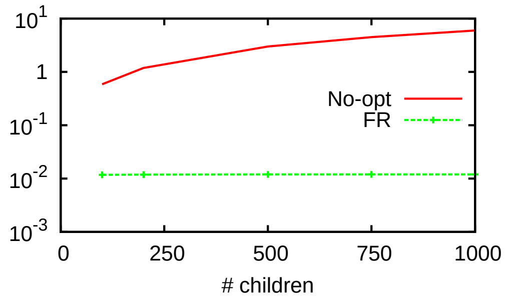
In addition to supporting more aggregates, a key advantage of SECOA over proof-sketch [11] is that SECOA does not have false positives and false negatives. To demonstrate such advantage, we implemented proof-sketch for Count and then compare against SECOA. Because proof-sketch uses the FM algorithm [10] for reducing Count to Max, we also



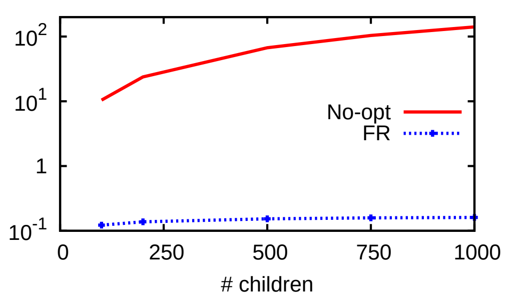
(a) Portal (Max)



(b) Portal (Count)

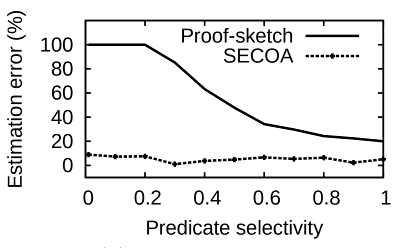


(c) Aggregator (Max)

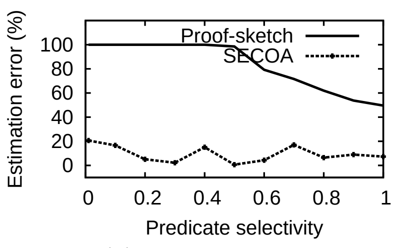


(d) Aggregator (Count)

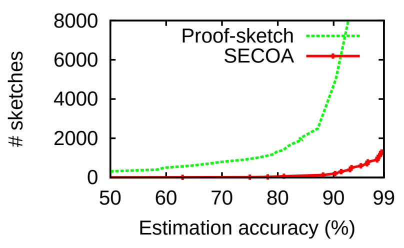
Figure 7: Per-query computational load on portal and aggregator. "No-opt" means no optimizations, "FR" means folded rolling, and "BT" means B-Tree.



(a) 333 FM sketches



(b) 54 FM sketches



(c) # FM sketches needed for target accuracy

Figure 8: Comparing SECOA with proof-sketches. (c) uses a selectivity of 1.

use the same reduction in SECOA for a fair comparison. We consider an adversary that aims to maximize the error in the query answer, without being detected. Since SECOA does not have false positives or false negatives, to avoid being detected, the adversary can only behave honestly in SECOA. For proof-sketch, the adversary can use the safe cheating strategy mentioned in [11]. Due to the existence of such safe cheating strategy, the inaccuracy in proof sketch (which comes from both approximation error and adversarial manipulation) will be larger than the inaccuracy in SECOA (which is purely approximation error). We quantify the inaccuracy or error as $|\hat{C} - C|/C$, where C is the accurate count while $\hat{C}$ is the answer returned. In the following experiments, we consider a system with 100,000 sensors.

Figure 8(a) and (b) plot the errors in SECOA and proof-sketch under different predicate *selectivity* x. A selectivity of x means that x fraction of the sensors satisfy the predicate in the Count query. The figures show that the inaccuracy in SECOA is always substantially lower than than in proof-sketch. In fact, the error in proof sketch is often larger than 50%. Such high relative error can easily limit the utility of the final answer. Under relatively small selectivity values (e.g., below 0.2), the aggregator can always safely report 0 as the Count answer in proof-sketch, without being detected. This translates to an error of 100%.

Since using a larger number of FM sketches can always reduce the error, one can interpret the above results from another perspective. Namely, to achieve the same accuracy, proof-sketch needs to use a larger number of FM sketches than SECOA. Figure 8(c) plots the number of FM sketches needed by the two schemes to achieve some given target accuracy. The figure shows that to achieve the same accuracy, proof-sketch can easily require 20 to 100 times more FM sketches than SECOA. This will in turn, increase the communication/computational overhead for dealing with the FM sketches by over an order of magnitude.

9. CONCLUSION

We have presented SECOA, a framework with a family of novel and optimally-secure protocols for secure outsourced aggregation. Our framework is based on a unified use of one-way chains. It supports a large and diverse set of aggregate functions, can have multiple hierarchically organized aggregators, can deterministically detect any malicious aggregation behavior without communication with sensors, and incurs small communication and computational overheads. Our evaluation results demonstrate feasibility and significant benefits of SECOA.

Acknowledgements

We thank Josh Benaloh and Melissa Chase for many useful discussions. This work is partly supported by NUS Young Investigator Award R-252-000-334-123.

10. REFERENCES

[1] Speed comparison of popular crypto algorithms. `http://www.cryptopp.com/benchmarks.html`, 2008.

[2] Alon, N., Matias, Y., and Szegedy, M. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci. 58*, 1 (1999), 137–147.

[3] Bar-Yossef, Z., Jayram, T., Kumar, R., Sivakumar, D., and Trevisan, L. Counting distinct elements in a data stream. In *RANDOM* (2002).

[4] Bellare, M., Garay, J., and Rabin, T. Fast batch verification for modular exponentiation and digital signatures. In *Eurocrypt* (1998).

[5] Chan, H., Perrig, A., Przydatek, B., and Song, D. SIA: Secure information aggregation in sensor networks. *Journal of Computer Security 15*, 1 (2007), 69–102.

[6] Chan, H., Perrig, A., and Song, D. Secure hierarchical in-network aggregation in sensor networks. In *13th ACM conference on Computer and communications security (CCS)* (2006).

[7] Considine, J., Li, F., Kollios, G., and Byers, J. Approximate aggregation techniques for sensor databases. In *ICDE* (2004).

[8] Cormode, G., and Muthukrishnan, S. An improved data stream summary: The count-min sketch and its applications. *LATIN 2004: Theoretical Informatics* (2004), 29–38.

[9] Deshpande, A., Nath, S., Gibbons, P. B., and Seshan, S. Cache-and-query for wide area sensor databases. In *SIGMOD* (2003).

[10] Flajolet, P., and Martin, G. N. Probabilistic counting algorithms for data base applications. *J. Comput. Syst. Sci. 31*, 2 (1985), 182–209.

[11] Garofalakis, M., Hellerstein, J., and Maniatis, P. Proof sketches: Verifiable in-network aggregation. In *ICDE* (2007).

[12] Kansal, A., Nath, S., Liu, J., and Zhao, F. Senseweb: An infrastructure for shared sensing. *IEEE Multimedia 14*, 4 (2007).

[13] Li, F., Hadjieleftheriou, M., Kollios, G., and Reyzin, L. Dynamic authenticated index structures for outsourced databases. In *ACM SIGMOD* (2006).

[14] Mykletun, E., Narasimha, M., and Tsudik, G. Signature bouquets: Immutability for aggregated/condensed signatures. In *ESORICS* (2004).

[15] Mykletun, E., Narasimha, M., and Tsudik, G. Authentication and integrity in outsourced databases. *ACM Trans. Storage 2*, 2 (2006), 107–138.

[16] Narasimha, M., and Tsudik, G. DSAC: Integrity of outsourced databases with signature aggregation and chaining. In *CIKM* (2005).

[17] Nath, S., Gibbons, P. B., Seshan, S., and Anderson, Z. Synopsis diffusion for robust aggregation in sensor networks. *ACM Transactions on Sensor Networks (TOSN) 4*, 2 (2008).

[18] Nath, S., Gibbons, P. B., Seshan, S., and Anderson, Z. R. Synopsis diffusion for robust aggregation in sensor networks. In *ACM SenSys* (2004).

[19] Nuckolls, G. Verified query results from hybrid authentication trees. In *Data and Applications Security (DBSec)* (2005).

[20] Pang, H., Jain, A., Ramamritham, K., and Tan, K.-L. Verifying completeness of relational query results in data publishing. In *ACM SIGMOD* (2005).

[21] Pang, H., and Tan, K. Verifying completeness of relational query answers from online servers. *ACM Transactions on Information and System Security (TISSEC) 11*, 2 (2007).

[22] Pang, H., and Tan, K.-L. Authenticating query results in edge computing. In *ICDE* (2004).

[23] Rivest, R., Shamir, A., and Adleman, L. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM 21*, 2 (1978), 120–126.

[24] Simmons, G. J. A weak privacy protocol using the RSA crypto algorithm. *Cryptologia 7*, 2 (1983), 180–182.

[25] Sion, R. Query execution assurance for outsourced databases. In *VLDB* (2005).

[26] Wagner, D. Resilient aggregation in sensor networks. In *2nd ACM workshop on security of ad hoc and sensor networks* (2004).

[27] Yu, H. Secure and highly-available aggregation queries in large-scale sensor networks via set sampling. In *ACM/IEEE IPSN* (2009).