Efficient and Privacy-Preserving Data Aggregation in Mobile Sensing

Qinghua Li, Guohong Cao
Department of Computer Science and Engineering
The Pennsylvania State University, University Park
Email: {qxl118, gcao}@cse.psu.edu

***Abstract*—The proliferation and ever-increasing capabilities of mobile devices such as smart phones give rise to a variety of mobile sensing applications. This paper studies how an untrusted aggregator in mobile sensing can periodically obtain desired statistics over the data contributed by multiple mobile users, without compromising the privacy of each user. Although there are some existing works in this area, they either require bidirectional communications between the aggregator and mobile users in every aggregation period, or has high computation overhead and cannot support large plaintext spaces. Also, they do not consider the Min aggregate which is quite useful in mobile sensing. To address these problems, we propose an efficient protocol to obtain the Sum aggregate, which employs an additive homomorphic encryption and a novel key management technique to support large plaintext space. We also extend the sum aggregation protocol to obtain the Min aggregate of time-series data. Evaluations show that our protocols are orders of magnitude faster than existing solutions.**

I. Introduction

Mobile devices such as smart phones are gaining an ever-increasing popularity. Most smart phones are equipped with a rich set of embedded sensors such as camera, microphone, GPS, accelerometer, ambient light sensor, gyroscope, etc. The data generated by these sensors provides opportunities to make sophisticated inferences about not only people (e.g., human activity, health, location, social event) but also their surrounding (e.g., pollution, noise, weather, oxygen level), and thus can help improve people's health as well as life. This enables various *mobile sensing* applications such as environmental monitoring [1], traffic monitoring [2], healthcare [3], etc.

In many scenarios, aggregation statistics need to be periodically computed from a stream of data contributed by mobile users [4], in order to identify some phenomena or track some important patterns. For example, the average amount of daily exercise (which can be measured by motion sensors [5]) that people do can be used to infer public health conditions. The average or maximum level of air pollution and pollen concentration in an area may be useful for people to plan their outdoor activities. Other statistics of interests include the lowest gasoline price in a city, the highest moving speed of road traffic during rush hour, etc.

Although aggregation statistics computed from time-series data is very useful, in many scenarios, the data from individual user may be privacy-sensitive, and users do not trust any single third-party aggregator to see their data in cleartext. For instance, to monitor the propagation of a new flu, the aggregator will count the number of users infected by this flu. However, a user may not want to directly provide her true status ("1" if being infected and "0" otherwise) if she is not sure whether the information will be abused by the aggregator. Accordingly, systems that collect users' true data values and compute aggregate statistics over them may not meet users' privacy requirement [4]. Thus, an important challenge is how to protect the users' privacy in mobile sensing, especially when the aggregator is untrusted.

978-1-4673-2447-2/12/$31.00 ©2012 IEEE

Most previous works on sensor data aggregation assume a trusted aggregator, and hence cannot protect user privacy against an untrusted aggregator in mobile sensing applications. Several recent works [6]–[9] consider the aggregation of time-series data in the presence of an untrusted aggregator. To protect user privacy, they design encryption schemes in which the aggregator can only decrypt the sum of all users' data but nothing else. Rastogi and Nath [6] use threshold Paillier cryptosystem [10] to build such an encryption scheme. To decrypt the sum, their scheme needs an extra round of interaction between the aggregator and all users in every aggregation period, which means high communication cost and long delay. Moreover, it requires all users to be online until decryption is completed, which may not be practical in many mobile sensing scenarios due to user mobility and the heterogeneity of user connectivity. Rieffel et al. [9] propose a construction that does not require bidirectional communications between the aggregator and the users, but it has high computation and storage cost to deal with collusions in a large system.

Shi et al. [7], [8] also propose a construction for sum aggregation which does not need the extra round of interaction. However, the decryption in their construction needs to traverse the possible plaintext space of the aggregated value, which is very expensive for a large system with large plaintext space. In mobile sensing, the plaintext space of some application can be large. For example, carbon dioxide levels can range from 350 ppm outdoors to over 10000 ppm in industrial workplaces [11]. Hence in applications which continuously monitor the carbon dioxide levels that people are exposed to in their daily life [12], [13], the plaintext space can reach 10^4. Under this plaintext space, for a large system with one million users, the construction in [7] requires 30 seconds to decrypt the sum on a modern 64-bit desktop PC. Its computation overhead is too high for an aggregator to run realtime monitoring applications with short aggregation intervals and to collect multiple aggregate statistics simultaneously. Moreover, none of these existing schemes considers the Min aggregate (i.e., the minimum value) of time-series data, which is also important in many mobile sensing applications.

In this paper, we propose a new protocol for mobile sensing to obtain the sum aggregate of time-series data in the presence of an untrusted aggregator. Our protocol employs an additive homomorphic encryption and a novel key management scheme based on efficient HMAC to ensure that the aggregator can only obtain the sum of all users' data, without knowing individual user's data or intermediate result. In our protocol, each user (the aggregator) only needs to compute a very small number of HMACs to encrypt her data (decrypt the sum). Hence, the computation cost is very low, and the protocol can scale to large systems with large plaintext spaces, resource-constrained devices and high aggregation loads. Another nice property of our protocol is that it only requires a single round of user-to-aggregator communication.

Based on the sum aggregation protocol, we also propose a protocol to obtain the Min aggregate. To our best knowledge, this is the first privacy-preserving solution to obtain the Min of time-series data in mobile sensing with just one round of user-to-aggregator communication. Our protocols for Sum and Min can be easily adapted to derive many other aggregate statistics such as Count, Average and Max.

The remainder of this paper is organized as follows. Section II discusses related work. Section III presents system models and assumptions. Section IV and Section V present our protocols for Sum and Min, respectively. Section VI evaluates the practical performance and cost of our solutions. The last two sections present discussions and conclusions.

II. Related Work

Many works have addressed various security and privacy issues in mobile sensing networks and systems (e.g., [14]–[18]), but they do not consider data aggregation. There are a lot of existing works (e.g., [19]–[22]) on security and privacy-preserving data aggregation, but most of them assume a trusted aggregator and cannot protect user privacy against untrusted aggregators. Yang et al. [23] proposed an encryption scheme that allows an untrusted aggregator to obtain the sum of multiple users's data without knowing any specific user's data. However, their scheme requires expensive re-keying operations to support multiple time steps, and thus may not work for time-series data.

Shi et al. [24] proposed a privacy-preserving data aggregation scheme based on data slicing and mixing techniques. However, their scheme relies on peer-to-peer communications among users, which is nontrivial in mobile sensing scenarios due to the high mobility of users. Also, their scheme does not work well for time-series data, since each user may need to select a new set of peers in each aggregation interval due to mobility. Besides, their scheme for non-additive aggregates (e.g., Max/Min) requires multiple rounds of bi-directional communications between the aggregator and mobile users which means long delays. In contrast, our scheme obtains those aggregates with just one round of uni-directional communication from users to the aggregator.

To achieve privacy-preserving sum aggregation of time-series data, Rastogi and Nath [6] designed an encryption scheme based on threshold Paillier cryptosystem [10], where the decryption key is divided into portions and distributed to the users. The aggregator collects the ciphertexts of users, multiplies them together and sends the aggregate ciphertext to all users. Each user decrypts a share of the sum aggregate. The aggregator collects all the shares and gets the final sum. However, their scheme requires an extra round of interaction between the aggregator and users in every aggregation period.

Based on an efficient additive homomorphic encryption scheme, Rieffel et al. [9] proposed a construction that does not require an extra round of interaction between the aggregator and the users. In their scheme, the computation and storage cost is roughly equal to the number of colluding users that the system can tolerate. Thus, their scheme has high overhead to achieve good resistance to collusion, especially when the system is large and a large number of users collude. In contrast, our scheme tolerates a high fraction of colluding users (e.g., 30%) with very small cost even when the system is large. Acs and Castelluccia [25] also proposed a scheme based on additive homomorphic encryption, but in their scheme each node shares a pairwise key with any other node.

Shi et al. [7] proposed a construction for sum aggregation based on the assumption that the Decisional Diffie-Hellman problem is hard over finite cyclic groups. In their construction, each user sends her ciphertext to the aggregator and no communication is needed from the aggregator to the users. To decrypt the sum, their construction needs to traverse the possible plaintext space of sum, and thus it is not efficient for a large system with large plaintext spaces. Chan et al. [8] extended the construction in [7] with a binary interval tree technique, but their scheme still has the limitation in plaintext spaces. Jawurek and Kerschbaum [26] proposed a scheme which provides differential privacy for sum. Our aggregation protocol for sum can be used as a building block of their scheme to improve the computational efficiency. Also, existing works do not consider the Min of time-series data.

III. Preliminaries

A. Models and Assumptions

Figure 1 shows our system model, which is similar to the model in [7]. An aggregator wishes to get the aggregate statistics of n mobile users *periodically*, e.g., in every hour. The time periods are numbered as 1, 2, 3, ..., etc. In every time period, each user i encrypts her data x_i with key k_i and sends the derived ciphertext to the aggregator. From the ciphertexts, the aggregator decrypts the needed aggregate statistics using her aggregator capability k_0. The value of each user's data is an integer within range $[0, \Delta]$. Two types of aggregate statistics are considered in this work, which are Sum and Min. Sum is defined as the sum of all users' data and Min is defined as the minimum value of the users' data. From Sum and Min,

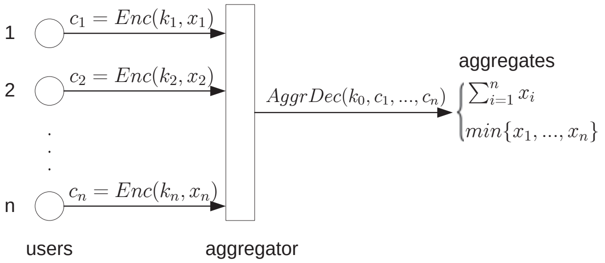


Fig. 1. Our system model of time-series data aggregation.

many other aggregate statistics can be easily derived, such as Count (i.e., the number of users that satisfy certain predicate), Average (which is derivable from Sum and Count), and Max (which can be obtained from the Min of $\Delta - x$).

In each time period, a mobile user sends her encrypted data to the aggregator via WiFi, 3G or other available access networks. No peer-to-peer communication is required among mobile users, since such communication is nontrivial in mobile sensing scenarios due to the high mobility of users and users may not be aware of each other for privacy reasons.

We consider an untrusted aggregator that is curious about each individual user's data. The aggregator may eavesdrop all the messages sent from/to every user. A number of users may collude with the aggregator, and reveal their data to the aggregator. A number of users may also collude to obtain the aggregate. Similar to [7], we assume that the fraction of users that collude with/against the aggregator is at most γ ($0 \leq \gamma < 1$), and the system has a priori estimate over the upper bound of γ which can be used in practice. In addition, the aggregator and users have limited computation capability.

We assume a trusted authority which issues proper keys to the users and the aggregator via a secure channel.

Our goal is to guarantee the privacy of each user's data against the untrusted aggregator, i.e., the aggregator obtains the aggregate statistics without knowing any individual user's data. Note that we aim to protect the privacy of data content not data source [27]. Also, we aim to guarantee that any party without an appropriate aggregator capability obtains nothing.

Malicious users may also perform data pollution attacks in which they provide false data values in order to sway the final aggregate statistics. Data pollution attacks are outside the scope of this paper, and their influence can be bounded if each user uses a non-interactive zero-knowledge proof to prove that her data lies in a valid range.

B. Underlying Encryption Scheme

One building block of our solution is the additive homomorphic encryption scheme proposed by Castelluccia et al. [21], [28]. This scheme works as follows.

Encryption:

1) Represent message m as integer $m \in [0, M-1]$ where M is a large integer.
2) Let k be a randomly generated key, $k \in \{0,1\}^{\lambda}$, where λ is a security parameter.
3) Output ciphertext $c = (m + h(f_k(r))) \mod M$, where f_k is a pseudorandom function (PRF) that uses k as a parameter, h is a length-matching hash function (see details below) and r is a nonce for this message.

Decryption:

1) Output plaintext $m = (c - h(f_k(r))) \mod M$.

TABLE I
NOTATIONS

n	The number of users in the system
γ	The maximum fraction of users that collude
Δ	The maximum value of any user's data
M	$M = 2^{\lceil \log_2 (n\Delta) \rceil}$
$\mathbb{F}_\lambda$	$\mathbb{F}_\lambda = \{f_s : \{0,1\}^\lambda \to \{0,1\}^\lambda\}_{s \in \{0,1\}^\lambda}$ is a family of pesudorandom functions indexed by key s
h	A length-matching hash function, $h : \{0,1\}^\lambda \to \{0,1\}^\alpha$, where $\alpha = \lceil \log_2 (n\Delta) \rceil$
k_0	The decryption key used by the aggregator
k_i	The encryption key used by user i
l	The required security level, e.g., $l = 80$
c	The number of secrets assigned to each user in our protocol
q	The number of secrets assigned to the aggregator in our protocol

The PRF f_k is a function of the PRF family $\mathbb{F}_\lambda = \{f_k : \{0,1\}^\lambda \to \{0,1\}^\lambda\}_{k \in \{0,1\}^\lambda}$ indexed by k. Since provably secure PRFs are usually computationally expensive, Castelluccia et al. [21] advocate using keyed hash functions (e.g., HMAC) as PRFs. HMAC is a PRF if the underlying compression function of the hash function in use is a PRF [29]. When HMAC is used, $f_k(r)$ is the HMAC of r with k as the key.

The purpose of h is to shorten a long bit string. It maps the output of f_k to a shorter bit string of length α, where α is the modulus size of M (i.e., $\alpha = |M|$). h is not required to be collision-consistent, but its output should be uniformly distributed over $\{0,1\}^\alpha$. An example construction for h is to truncate the output of f_k into shorter bit strings of length α, take exclusive-OR on all these strings and use it as the output of h. This scheme is proved to be semantically secure [21].

This scheme allows additive homomorphic encryption. Given two ciphertexts $c_1 = (m_1 + h(f_k(r))) \mod M$ and $c_2 = (m_2 + h(f_{k'}(r))) \mod M$, an individual that knows k and k' can compute the sum of m_1 and m_2 directly from the aggregate ciphertext $c = c_1 + c_2$:

$$m = m_1 + m_2 = (c - h(f_k(r)) - h(f_{k'}(r))) \mod M.$$

To correctly compute the sum of n messages m_1, m_2, ..., m_n, M must be larger than $\sum_{i=1}^{n} m_i$. In practice, M should be selected as $M = 2^{\lceil \log_2 (max(m_i) \cdot n) \rceil}$.

Table I shows the notations used in this paper.

IV. Aggregation Protocol for Sum

A. Protocol Overview

Setup: The trusted authority assigns a set of secret values (*secrets* for short) to each user and the aggregator.

Enc: In each time period, user i ($i \in [1, n]$) generates encryption key k_i using the secrets that it is assigned. It encrypts its data x_i by computing

$$c_i = (k_i + x_i) \mod M \tag{1}$$

where $M = 2^{\lceil \log_2 (n\Delta) \rceil}$. Then it sends the ciphertext c_i to the aggregator.

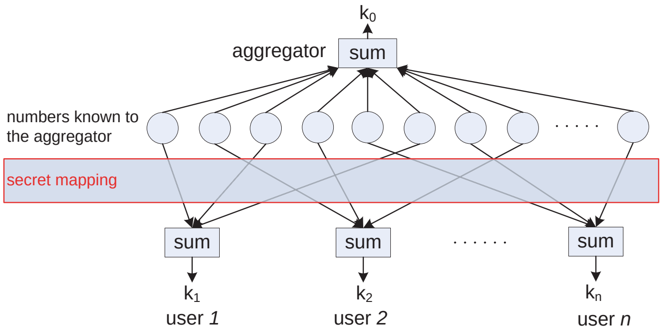


Fig. 2. The intuition behind the straw-man construction. The aggregator computes the sum of a set of numbers as the decryption key. These numbers are secretly allocated to the users, and each user computes the sum of its allocated numbers as the encryption key. The aggregator cannot know any user's encryption key since it does not know the mapping between the numbers and the users.

AggrDec: In each time period, the aggregator generates decryption key k_0 using the secrets that it is assigned, and decrypts the sum aggregate $S = \sum_{i=1}^{n} x_i$ by computing

$$S = (\sum_{i=1}^{n} c_i - k_0) \mod M. \tag{2}$$

The keys are generated using a PRF family and a length-matching hash function (see later). According to [28], the aggregator can get the correct sum so long as the following equation holds:

$$k_0 = (\sum_{i=1}^{n} k_i) \mod M. \tag{3}$$

In our protocol, the setup phase only runs once. After the setup phase, the trusted authority does not need to distribute secrets to the users and the aggregator again. In addition, the users and the aggregator do not have to synchronize their key generations with communications in every time period. These restrictions make it challenging for the users and the aggregator to generate their keys such that Equation 3 holds *in every time period* and the encryption (decryption) key used by each user (the aggregator) cannot be learned by any other party besides the trusted authority.

We propose a construction for key generations which preserves the privacy of each user and the Sum aggregate efficiently. Before presenting our construction, we first discuss a straw-man construction which is very efficient for the users but not efficient for the aggregator. Then we extend this straw-man scheme to derive our construction.

Both constructions include three processes, which are *secret setup*, *encryption key generation* and *decryption key generation*. They proceed in the Setup phase, Enc phase and AggrDec phase of the aggregation protocol, respectively.

B. A Straw-man Construction for Key Generation

1) Intuition: Figure 2 shows the intuition of the straw-man construction. Suppose there are nc random numbers. The aggregator has access to all the numbers, and it computes the sum of these numbers as the decryption key k_0. These numbers are divided into n random disjoint subsets, each of size c. These n subsets are assigned to the n users, where each user has access to one subset of numbers. User i computes the sum of the numbers assigned to it as the encryption key k_i.

TABLE II
SECURITY LEVELS OF THE STRAW-MAN CONSTRUCTION WHEN $\gamma = 0.1$.

$n = 10^2$	c	10	11	12	13	14
	p_b	$2^{-76.3}$	$2^{-84.1}$	$2^{-92.0}$	$2^{-99.9}$	$2^{-107.7}$
$n = 10^3$	c	6	7	8	9	10
	p_b	$2^{-64.9}$	$2^{-76.0}$	$2^{-87.2}$	$2^{-98.4}$	$2^{-109.6}$
$n = 10^4$	c	4	5	6	7	8
	p_b	$2^{-56.0}$	$2^{-70.4}$	$2^{-84.8}$	$2^{-99.3}$	$2^{-113.8}$
$n = 10^5$	c	3	4	5	6	7
	p_b	$2^{-51.5}$	$2^{-69.2}$	$2^{-87.0}$	$2^{-104.8}$	$2^{-122.6}$
$n = 10^6$	c	2	3	4	5	6
	p_b	$2^{-40.6}$	$2^{-61.5}$	$2^{-82.5}$	$2^{-103.6}$	$2^{-124.7}$

Clearly, Equation 3 holds. The aggregator cannot know any user's encryption key since it does not know the mapping between the numbers and the users. When c is large enough, it is infeasible for the aggregator to guess the numbers assigned to a particular user with a brute-force method. The aggregator's decryption key cannot be revealed by any user since no user knows all the numbers.

2) Construction: The construction is as follows:

Secret Setup: The trusted authority generates nc random and different secrets s_1, ..., s_{nc}. It divides these secrets into n random disjoint subsets, with c secrets in each subset. Let $\mathcal{S}$ denote the set of all secrets, and let $\mathcal{S}_i$ denote the i^{th} subset. Clearly, $\mathcal{S} = \bigcup_{i=1}^{n} \mathcal{S}_i$ and $\forall i \neq j, \mathcal{S}_i \bigcap \mathcal{S}_j = \phi$. The trusted authority sends the secrets in subset $\mathcal{S}_i$ to user i and sends all the secrets in $\mathcal{S}$ to the aggregator.

Encryption Key Generation: In time period $t \in \mathbb{N}$, user i generates its encryption key as follows:

$$k_i = (\sum_{s' \in \mathcal{S}_i} h(f_{s'}(t))) \mod M. \tag{4}$$

Decryption Key Generation: In time period $t \in \mathbb{N}$, the aggregator generates the decryption key as follows:

$$k_0 = (\sum_{s' \in \mathcal{S}} h(f_{s'}(t))) \mod M. \tag{5}$$

In Equation 4, since each $h(f_{s'}(t))$ is uniformly distributed over $\{0,1\}^{\alpha}$, k_i is also uniformly distributed over $\{0,1\}^{\alpha}$. Thus, the encryption keys satisfy the security requirement of the underlying cryptosystem. Equation 3 also holds since

$$\begin{aligned}\sum_{i=1}^{n} k_i &= (\sum_{i=1}^{n} \sum_{s' \in \mathcal{S}_i} h(f_{s'}(t))) \mod M \\ &= (\sum_{s' \in \mathcal{S}} h(f_{s'}(t))) \mod M \\ &= k_0 \mod M.\end{aligned}$$

3) Security level: If the aggregator knows the c secrets used by a user, it can obtain the encryption key of the user. We can derive the probability that the aggregator finds the c secrets used by a user. Let p_b denote the probability that in a single

TABLE III
THE MINIMUM VALUES OF c FOR 80-BIT SECURITY IN THE STRAW-MAN CONSTRUCTION.

n	10^2	10^3	10^4	10^5	10^6
$\gamma = 0, 0.1, 0.2, 0.3$	11	8	6	5	4

trial the aggregator can successfully guess the secrets assigned to the user. Recall that γ is the maximal fraction of users that collude with the aggregator. In the worst case, the aggregator knows the γnc secrets assigned to the colluding users, but it does not know how the remaining $(1-\gamma)nc$ secrets are assigned to other users. There are $\binom{(1-\gamma)nc}{c}$ possible secret assignments for each user. Hence we have:

$$p_b = \frac{1}{\binom{(1-\gamma)nc}{c}}. \qquad (6)$$

With a smaller p_b, better security can be achieved. Table II shows the values of p_b for varying parameters n and c. As c increases, the security level increases quickly.

Given the number of users n and an estimate of γ, we can derive the minimum value of c to achieve a certain required security level. (c is minimized to minimize the cost.) For l-bit security (e.g., $l = 80$), c should be selected as the minimum value that satisfies $p_b \leq 2^{-l}$. Table III shows the values of c for 80-bit security.

A fraction of users may collude against the aggregator to reveal the aggregate. To achieve this goal, they need to obtain all the secrets that the aggregator has. However, each participant only knows a subset of the secrets. So long as not all users collude, they cannot obtain all the secrets.

4) Cost: In each time period, each user computes c PRFs and the aggregator computes nc PRFs. Since c is small as shown in Table III, the computation cost at each user is very low. However, when the number of users n is very large, the computation cost at the aggregator is high.

C. Our Construction for Key Generation

Our construction extends the straw-man construction to reduce the computation overhead at the aggregator.

1) Intuition: Consider an equation:

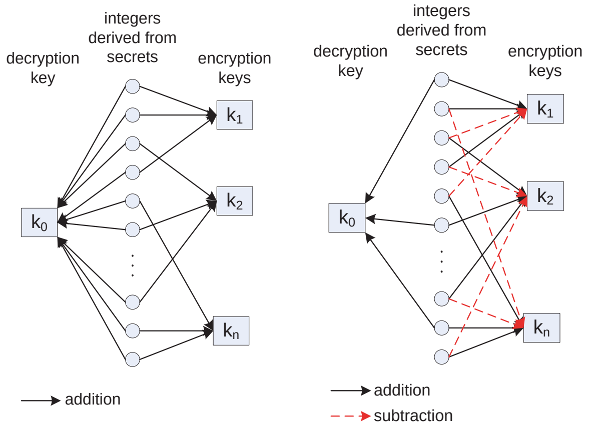
$$a_1 + a_2 + \cdots + a_{nc} = a_1 + a_2 + \cdots + a_{nc}. \qquad (7)$$

If we remove $nc-q$ summands from the right side and subtract them from the left side, the derived equation

$$a_1 + \cdots + a_{nc} + (-a_1) + \cdots + (-a_{nc-q}) = a_{nc-q+1} + \cdots + a_{nc} \qquad (8)$$

is equivalent to the original equation.

To meet the requirement of Equation 3, the straw-man construction essentially mimics Equation 7, i.e., the users collectively generate the summands on the left side and add them to the aggregate, while the aggregator alone generates the summands on the right side and subtracts them from the perturbed aggregate (see Figure 3(a)). Each summand is generated from a secret. Since Equation 7 and 8 are equivalent, we can remove some summands from the aggregator side and subtract them from the user side without violating Equation 3. Now the aggregator has less computation overhead since it needs to generate less summands. The reduced computation does not come for free, as it is amortized among the users such that each user generates more summands (see Figure 3(b)). A nice property is that it is now more difficult to guess the summands generated by each user and thus each user has better security.



(a) Straw-man construction (b) Our construction

Fig. 3. The intuition behind our construction in comparison with the straw-man construction.

2) Construction: The construction is as follows:

Secret Distribution: The trusted authority generates nc random and different secrets s_1, ..., s_{nc}. Let $\mathcal{S}$ denote the set composed of all the secrets. The trusted authority divides these secrets into n random disjoint subsets, with c secrets in each subset. For convenience, we call these subsets *additive subsets*. Let $\mathcal{S}_i$ denote the i^{th} additive subset. Clearly, $\mathcal{S} = \bigcup_{i=1}^{n} \mathcal{S}_i$.

Out of the nc secrets, the trusted authority randomly selects q secrets and assigns them to the aggregator. Let $\hat{\mathcal{S}}$ denote the set of secrets assigned to the aggregator. The trusted authority divides the remaining $nc - q$ secrets evenly into n random disjoint subsets. Among them, $(nc-q) - n\lfloor \frac{nc-q}{n} \rfloor$ subsets have $\lfloor \frac{nc-q}{n} \rfloor + 1$ secrets each, and the other $n(1 + \lfloor \frac{nc-q}{n} \rfloor) - nc + q$ subsets have $\lfloor \frac{nc-q}{n} \rfloor$ secrets each. For convenience, we call these subsets *subtractive subsets*. Let $\bar{\mathcal{S}}_i$ denote the i^{th} subtractive subset. Clearly, $\mathcal{S} = (\bigcup_{i=1}^{n} \bar{\mathcal{S}}_i) \bigcup \hat{\mathcal{S}}$. The trusted authority assigns the secrets in the additive subset $\mathcal{S}_i$ and subtractive subset $\bar{\mathcal{S}}_i$ to user i.

Encryption Key Generation: In time period $t \in \mathbb{N}$, user i generates its encryption key by computing

$$k_i = (\sum_{s' \in \mathcal{S}_i} h(f_{s'}(t)) - \sum_{s' \in \bar{\mathcal{S}}_i} h(f_{s'}(t))) \mod M. \qquad (9)$$

Decryption Key Generation: In time period $t \in \mathbb{N}$, the aggregator generates the decryption key by computing

$$k_0 = (\sum_{s' \in \hat{\mathcal{S}}} h(f_{s'}(t))) \mod M. \qquad (10)$$

TABLE IV
THE SECURITY LEVEL OF OUR CONSTRUCTION WHEN $\gamma = 0.1$.

$n = 10^2$	c	4	5	6	7	8
	p_b	$2^{-51.0}$	$2^{-66.5}$	$2^{-82.1}$	$2^{-97.7}$	$2^{-113.3}$
$n = 10^3$	c	3	4	5	6	7
	p_b	$2^{-52.2}$	$2^{-74.3}$	$2^{-96.4}$	$2^{-118.7}$	$2^{-140.9}$
$n = 10^4$	c	2	3	4	5	6
	p_b	$2^{-40.4}$	$2^{-68.8}$	$2^{-97.5}$	$2^{-126.3}$	$2^{-155.2}$
$n = 10^5$	c	1	2	3	4	5
	p_b	$2^{-16.5}$	$2^{-50.4}$	$2^{-85.5}$	$2^{-120.8}$	$2^{-156.2}$
$n = 10^6$	c	1	2	3	4	5
	p_b	$2^{-19.8}$	$2^{-60.3}$	$2^{-102.1}$	$2^{-144.0}$	$2^{-186.1}$

TABLE V
THE VALUES OF c FOR 80-BIT SECURITY IN OUR CONSTRUCTION.

n	10^2	10^3	10^4	10^5	10^6
$\gamma = 0, 0.1, 0.2$	6	5	4	3	3
$\gamma = 0.3$	7	5	4	3	3

TABLE VI
THE VALUES OF q FOR 80-BIT SECURITY IN OUR CONSTRUCTION.

n	10^2	10^3	10^4	10^5	10^6
$\gamma = 0$	12	8	6	5	4
$\gamma = 0.1$	13	8	6	5	4
$\gamma = 0.2$	13	8	6	5	4
$\gamma = 0.3$	13	9	7	5	5

The requirement in Equation 3 is satisfied since

$$\begin{aligned}
\sum_{i=1}^{n} k_i &= (\sum_{i=1}^{n}(\sum_{s' \in \mathcal{S}_i} h(f_{s'}(t)) - \sum_{s' \in \bar{\mathcal{S}}_i} h(f_{s'}(t)))) \mod M \\
&= (\sum_{s' \in \mathcal{S}} h(f_{s'}(t)) - \sum_{s' \in \bigcup_{i=1}^{n} \bar{\mathcal{S}}_i} h(f_{s'}(t))) \mod M \\
&= (\sum_{s' \in \hat{\mathcal{S}}} h(f_{s'}(t))) \mod M \\
&= k_0.
\end{aligned}$$

3) Security level: The aggregator cannot learn any user's encryption key since it does not know the additive secrets (i.e., secrets in the additive subset) and the subtractive secrets (i.e., secrets in the subtractive subset) assigned to this user. Each user has c additive secrets and at least $\lfloor \frac{nc-q}{n} \rfloor$ subtractive secrets. The aggregator may know the secrets assigned to itself and those to its γn colluders, but there are still $(1-\gamma)nc$ additive secrets and at least $(1-\gamma)n\lfloor \frac{nc-q}{n} \rfloor$ subtractive secrets that the aggregator does not know how they are assigned to the good users. There are at least $\binom{(1-\gamma)nc}{c} \cdot \binom{(1-\gamma)n\lfloor \frac{nc-q}{n} \rfloor}{\lfloor \frac{nc-q}{n} \rfloor}$ possible secret assignments for each good user. Thus,

$$p_b \le \frac{1}{\binom{(1-\gamma)nc}{c} \cdot \binom{(1-\gamma)n\lfloor \frac{nc-q}{n} \rfloor}{\lfloor \frac{nc-q}{n} \rfloor}}. \tag{11}$$

p_b decreases (i.e., the security for the users is better) when n and c increase, but p_b increases when γ increases.

Under the same total computation cost, the smaller q is, the more subtractive secrets the users are assigned and the better security the users have. However, if q is too small, the secrets (and hence the decryption key) used by the aggregator may be learned by a number of colluding users in the brute-force way. We can derive the minimum value of q to make it infeasible for γ fraction of users to collusively obtain the decryption key. These colluders know at most γnc subtractive secrets, but they do not know which q of the remaining $(1-\gamma)nc$ secrets the aggregator has. There are $\binom{(1-\gamma)nc}{q}$ possible secret assignments for the aggregator. Let p_c denote the probability that the q secrets assigned to the aggregator can be guessed in a single trial. We have

$$p_c \le \frac{1}{\binom{(1-\gamma)nc}{q}}. \tag{12}$$

When q increases, p_c decreases which means better security for the aggregator.

4) Practical considerations: To achieve l-bit security for each user and the aggregator, it is required that $p_b \le 2^{-l}$ and $p_c \le 2^{-l}$ respectively. Given parameters n and γ, large-enough values should be set for c and q to meet these requirements.

Since the values of c and q depend on each other, which can be seen from Equation 11 and 12, they can be set as follows. First, we assume $q \in (0, n]$. Under this assumption, Equation 11 can be rewritten as:

$$p_b \le \frac{1}{\binom{(1-\gamma)nc}{c} \cdot \binom{(1-\gamma)n(c-1)}{c-1}}. \tag{13}$$

We can derive the minimum value of c that makes the right-hand side of Equation 13 smaller than 2^{-l}. Then we apply the derived value of c to Equation 12, and obtain the minimum value of q that makes the right-hand side of Equation 12 smaller than 2^{-l}. If the obtained value of q falls into the assumed range $(0, n]$, the values of c and q are accepted. Otherwise, we can increase the value of c, until the minimum value of q that makes the right-hand side of Equation 12 smaller than 2^{-l} is not larger than n.

This method of setting c and q ensures that $q \le n$, and thus p_b is given in Equation 13. Table IV shows the values of p_b when n and c change. Table V and Table VI show the values of c and q respectively for 80-bit security. It can be seen that both c and q are very small.

5) Cost: Since the setup phase is run only once, we analyze the cost of our construction in each aggregation period. The computation cost is measured by the number of PRFs computed, since the length-matching hash function (which mainly consists of exclusive-OR operations) and arithmetic addition are much less expensive in computation.

In each time period, each user computes $2c - \frac{q}{n}$ PRFs on average, while the aggregator computes q PRFs. As for the storage cost, the trusted authority stores nc secrets as well as $2nc$ mappings between the secrets and the users/aggregator. Each user stores $2c - \frac{q}{n}$ secrets on average, while the aggregator stores q secrets. Besides sending the encrypted data to the aggregator, each user does not make any extra communications.

6) Comparisons with the Straw-man Construction: Table VII compares our construction to the straw-man construction in security and cost. When the total computation cost (for users

TABLE VII

THE SECURITY AND COST OF OUR CONSTRUCTION AND THE STRAW-MAN CONSTRUCTION. FOR COMPUTATION COST, THE VALUE IS THE COST PER TIME PERIOD.

	Straw-man	Ours
p_b	$\frac{1}{\binom{(1-\gamma)nc}{c}}$	$\frac{1}{\binom{(1-\gamma)nc}{c}\cdot\binom{(1-\gamma)n(c-1)}{c-1}}$
Comp. (total)	$2nc$	$2nc$
Comp. (user)	c	$2c - \frac{q}{n}$
Comp. (aggregator)	nc	$q(q < n)$
Storage (user)	c	$2c - q$
Storage (aggregator)	nc	q

TABLE VIII

THE COMPUTATION COST OF OUR CONSTRUCTION AND THE STRAW-MAN CONSTRUCTION FOR 80-BIT SECURITY WHEN $\gamma = 0.1$.

	n	10^2	10^3	10^4	10^5	10^6
User	Straw-man	11	8	6	5	4
	Ours	12	10	8	6	6
Aggregator	Straw-man	1100	8000	$6 \cdot 10^4$	$5 \cdot 10^5$	$4 \cdot 10^6$
	Ours	13	8	6	5	4

and the aggregator) is the same, our construction achieves better security. Also, it has smaller computation cost at the aggregator. Upon initial inspection, our construction may seem to double the computation cost at each user (i.e., from c to roughly $2c$). In practice, however, it can use a smaller c to achieve the same security level. Table VIII shows the computation cost of the two constructions at the same security level. For a wide range of n ($10^2 - 10^6$), the computation cost at each user is slightly higher (i.e., one or two PRFs) in our construction, but the computation cost at the aggregator is orders of magnitude smaller.

V. AGGREGATION PROTOCOL FOR MIN

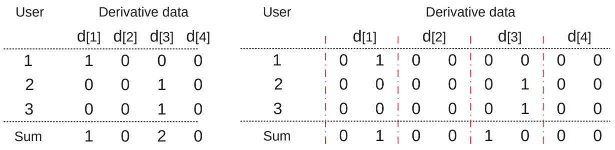
The Min aggregate is defined as the minimum value of the users' data. This section presents a protocol which employs the Sum aggregate to get Min.

A. The Basic Scheme

This scheme gets the Min aggregate of each time period using $\Delta + 1$ parallel Sum aggregates in the same time period. The sums used to obtain Min are based on a number of 1-bit *derivative data* (denoted by d) derived from the users' raw data x. Without loss of generality, we assume that Δ is a power of two.

The scheme works as follows. In each time period, each user generates $\Delta+1$ derivative data $d[0]$, $d[1]$, ..., $d[\Delta]$, where each derivative data corresponds to one possible data value in the plaintext space. For each $j \in [0, \Delta]$, the user assigns 1 to $d[j]$ if its raw data value is equal to j and assigns 0 otherwise. For each $j \in [0, \Delta]$, the aggregator can obtain the Sum aggregate of $d[j]$ using the sum aggregation protocol presented in Section IV. Then Min is the smallest j that returns a positive sum.

In each time period, each user involves in $\Delta + 1$ sum aggregates over $\Delta + 1$ derivative data. Note that in the sum aggregation protocol each user computes $2c$ PRFs to encrypt her data. It is inefficient to compute $2c$ PRFs for each derivative data. Since these data are independent, we use a more efficient technique that concatenates multiple data together and encrypts them as a whole.



User	Derivative data			
	d[1]	d[2]	d[3]	d[4]
1	1	0	0	0
2	0	0	1	0
3	0	0	1	0
Sum	1	0	2	0

(a) Original derivative data

User	Derivative data							
	d[1]		d[2]		d[3]		d[4]	
1	0	1	0	0	0	0	0	0
2	0	0	0	0	0	1	0	0
3	0	0	0	0	0	1	0	0
Sum	0	1	0	0	1	0	0	0

(b) Extended and concatenated data

Fig. 4. An example of sum based on extended and concatenated derivative data.

This technique extends each derivative data from one bit to $\lceil \log{(n+1)} \rceil$ bits by adding $\lceil \log{(n+1)} \rceil - 1$ 0's on the left, and then concatenates all extended derivative data into a single bit string. The sum of the concatenated string (interpreted as an integer) is obtained using the sum aggregation protocol. The obtained sum is considered as a bit string, and split into substrings of $\lceil \log{(n+1)} \rceil$ bits each. Each substring, when interpreted as an integer, represents the sum of one derivative data. Note that these substrings do not affect each other (i.e., no carries among them), since the sum of each derivative data does not exceed n. Figure 4 shows an example of this process.

Clearly, the concatenated data has $(\Delta + 1)\lceil \log{(n+1)} \rceil$ bits. The ciphertext generated by each user has $\alpha = (\Delta + 1)\lceil \log{(n+1)} \rceil$ bits. If α is larger than H, which is the size of the output generated by the PRF (i.e., an HMAC), we can divide the derivative data into $\frac{(\Delta+1)\lceil \log{(n+1)} \rceil}{H}$ groups and apply the above technique to each group. Thus, $\frac{(\Delta+1)\lceil \log{(n+1)} \rceil}{H}$ instances of the sum aggregation protocol are needed in each time period. For example, when $n = 1000$, $\Delta = 10000$ and SHA-512 is used as the hash function of HMAC, 196 instances of the sum aggregation protocol are needed.

Each user uses just one set of secrets for all instances of the sum aggregation protocol. For instance j, it uses $h(f_{s'}(j|t))$ to generate the encryption key instead of using $h(f_{s'}(t))$ in the original protocol (see Equation 9). Similarly, the aggregator also uses just one set of secrets.

Since the sum aggregation protocol does not leak the derivative data of any user, the aggregator cannot know the data value of any specific user.

B. Low-cost Min Aggregation

When the plaintext space is large, the cost of the basic scheme is high. In some application scenarios, it may not be necessary to get the exact Min, but an approximate answer is good enough. For such scenarios, the basic scheme can be extended to get an approximate Min with much smaller cost.

Specifically, we wish to obtain an approximate Min where the relative error (defined as $\frac{|\text{Exact Min}-\text{Approximate Min}|}{max\{\text{Exact Min},1\}}$) is required to be lower than $\frac{1}{2^{\epsilon}}$ ($\epsilon \geq 0$). To meet this requirement, the exact value of Min should be obtained if Min is smaller than or equal to 2^{ϵ}, and the ϵ-bit segment of Min (when Min is interpreted as a bit string) that starts from the first '1' bit should be obtained if Min is larger than 2^{ϵ}. For example, suppose Min is 42 (00101010) out of 8-bit data. To make the relative error smaller than $\frac{1}{2^3}$, it is sufficient to know that Min has the bit pattern $00101xxx$. Then we can set the bit that follows the known bits as 1 and set other bits as 0. The obtained approximate Min is 00101100, which is 44. The relative error is $\frac{1}{21}$, which is smaller than the required $\frac{1}{2^3}$.

To obtain the approximate Min, each user appends $\epsilon + 1$ padding bits to its raw data. If the data value is zero, the first padding bit is 1 and the others are 0; otherwise, all the padding bits are 0. The padded data has $\log \Delta + \epsilon + 2$ bits and at least one bit is 1. The first '1' bit of Min may appear in any of the first $\log \Delta + 2$ bits of the padded data. In the case it appears at the first padding bit, Min is zero.

Suppose in the binary representation of data value, the weight of bit decreases from the left to the right. Let δ ($\delta \in [1, \log \Delta + 2]$) denote the location (indexed from the left) of the first '1' bit of Min. Smaller δ means larger Min. Let σ denote the value of the $(\epsilon - 1)$-bit segment of Min that follows δ. When δ is the same, a larger σ means larger Min. Clearly, there are $2^{\epsilon-1}(\log \Delta + 2)$ possible combinations of $\langle \delta, \sigma \rangle$. We map these combinations to an *auxiliary* plaintext space $0, 1, ..., 2^{\epsilon-1}(\log \Delta + 2) - 1$, such that if one combination means smaller Min than another combination, it is mapped to a smaller value in the auxiliary plaintext space than that combination. Let $v[\delta, \sigma]$ denote the value that combination $\langle \delta, \sigma \rangle$ maps to.

In each time period, each user converts its padded raw data to a value x' in the auxiliary plaintext space as follows: if in its padded raw data the first '1' bit appears at δ' and the value of the $(\epsilon - 1)$-bit segment that follows the first '1' bit is σ', it sets $x' = v[\delta', \sigma']$. The aggregator can get the Min of x' using the basic scheme, and reversely map the Min of x' to a pair of $\langle \delta, \sigma \rangle$. It knows that the first '1' bit of the Min aggregate over padded raw data appears at location δ, and the $(\epsilon - 1)$-bit segment that follows the first '1' bit is σ. Then it sets the bit that follows the $(\epsilon - 1)$-bit segment as 1, and sets the remaining bits as 0. This derives the Min aggregate over padded raw data, which has the form $\{0\}^{\delta-1}\{1\}^1\{0,1\}^{\epsilon-1}\{1\}^1\{0\}^{\log \Delta + 2 - \delta}$. From this bit string, the last $\epsilon + 1$ padding bits are removed and then the approximate Min of users' raw data is obtained. Figure 5 shows a running example of this process.

In total, this scheme uses $2^{\epsilon-1}(\log \Delta + 2)$ Sum aggregates of 1-bit derivative data. Thus, $\frac{2^{\epsilon-1}(\log \Delta + 2)}{H}$ instances of the sum aggregation protocol are needed in each time period. For example, when $\Delta = 10^4$, SHA-512 is used as the hash function of HMAC and it is required to limit the relative error to 1% (i.e., $\epsilon = 7$), only 2 instances are needed.

Table IX summarizes the relative error and cost of the basic scheme and the low-cost scheme.

VI. EVALUATIONS

This section evaluates the cost of our aggregation protocols for Sum and Min. We compare our solution against two existing privacy-preserving aggregation protocols for time-series data: the protocol proposed in [7] (denoted by *EXP*) and CollaPSE [9].

TABLE IX
THE RELATIVE ERROR AND COST OF THE BASIC SCHEME AND THE LOW-COST SCHEME.

	Basic	Low-cost
Relative error	0	$\leq \frac{1}{2^\epsilon}$
Encryption (PRFs)	$\frac{2c(\Delta+1)\lceil \log (n+1) \rceil}{H}$	$\frac{2^\epsilon c(\log \Delta+2)}{H}$
Decryption (PRFs)	$\frac{q(\Delta+1)\lceil \log (n+1) \rceil}{H}$	$\frac{2^{\epsilon-1} q(\log \Delta+2)}{H}$
Ciphertext size (bit)	$(\Delta + 1)\lceil \log (n + 1) \rceil$	$2^{\epsilon-1}(\log \Delta + 2)$

H is the size of the output generated by the PRF.

TABLE X
COMPARISONS BETWEEN OUR SUM AGGREGATION PROTOCOL AND EXISTING PROTOCOLS IN COMPUTATION AND STORAGE COST.

	Encryption (user)	Decryption (aggregator)	Storage (user)	Storage (aggr.)
EXP	2 Mod. Exp.	$\sqrt{n\Delta}$ Mod. Exp.	1	1
CollaPSE	$(\gamma n + 1)$ PRFs	$(\gamma n + 1)$ PRFs	$\gamma n + 1$	$\gamma n + 1$
Ours	$2c$ PRFs	q PRFs	$2c$	q

Mod. Exp. stands for modular exponentiations.
In practice, HMAC can be used as a pseudorandom function (PRF).
Parameters of the three schemes are set to tolerate γn colluding users.
In most practical settings, $c \leq 7$ and $q \leq 13$ (see Table V and VI).

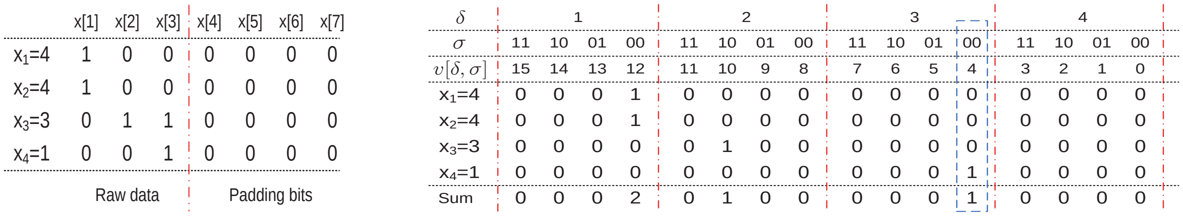
A. Analytical Comparisons

1) Cost of Sum aggregation: EXP is a Sum aggregation protocol based on the decisional Diffie-Hellman assumption (see [7] for details). In EXP, encryption mainly requires two modular exponentiations. The aggregator decrypts the sum via a brute-force trial of possible plaintext. It takes one modular exponentiation to try each possible plaintext and a total of $n\Delta$ modular exponentiations are needed. If Pollard's Rho method is used, the decryption cost can be reduced to about $\sqrt{n\Delta}$ modular exponentiations. Here, we use the reduced cost of EXP for comparison, and later we will show that our solution can further reduce the cost.

Similar to our protocol, CollaPSE also uses the homomorphic encryption scheme proposed in [21] to derive the Sum aggregate, but in a different way (see [9] for details). In CollaPSE, each user (the aggregator) computes $s + 1$ PRFs to encrypt her data (decrypt the sum). Here, s is a system parameter and it denotes the number of colluding users that the protocol can tolerate.

Table X shows the computation and storage cost of the three aggregation protocols for Sum, where the cost is derived under the same condition that they can tolerate γn colluding users. Compared with CollaPSE, our protocol has much smaller computation and storage cost at both the users and the aggregator, especially for a large system with possibly many colluding users. Compared with EXP, our protocol has slightly higher storage cost (i.e., around 10 secrets each with just tens of bytes), but our computation overhead is much lower since PRF (when implemented with HMAC) can run orders of magnitude faster than modular exponentiation in practice. We elaborate this point further in Section VI-B.

2) Cost of Min aggregation: The Min aggregation scheme presented in Section V-B derives Min from $2^{\epsilon-1}(\log \Delta + 2)$ parallel Sum aggregates of 1-bit data, where each sum is obtained using our Sum aggregation protocol. Here, each sum can also be obtained using EXP, and we refer to the Min



	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]
x_1=4	1	0	0	0	0	0	0
x_2=4	1	0	0	0	0	0	0
x_3=3	0	1	1	0	0	0	0
x_4=1	0	0	1	0	0	0	0
	Raw data			Padding bits			

(a) Padded raw data

δ	1				2				3				4			
σ	11	10	01	00	11	10	01	00	11	10	01	00	11	10	01	00
$v[\delta,\sigma]$	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
x_1=4	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
x_2=4	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
x_3=3	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
x_4=1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
Sum	0	0	0	2	0	1	0	0	0	0	0	1	0	0	0	0

(b) Derivative data

Fig. 5. An example of the process that obtains an approximate Min, where $\Delta = 4$ and $\epsilon = 3$. The four users' auxiliary data values are $x'_1 = 12$, $x'_2 = 12$, $x'_3 = 10$ and $x'_4 = 4$. The Min of the auxiliary data is 4, and it is reversely mapped to $\langle \delta, \sigma \rangle = \langle 3, 00 \rangle$. Thus, the approximate Min of padded raw data is 0010010. After the last four padding bits are removed, the output is 001.

aggregation scheme that uses EXP as a building block as *EXP-Min*. In EXP-Min, each user computes $2^{\epsilon}(\log \Delta + 2)$ modular exponentiations to encrypt her data, and the aggregator computes $2^{\epsilon-1}\sqrt{n}(\log \Delta + 2)$ modular exponentiations to decrypt the Min. Similarly, CollaPSE can also be used as a building block of Min aggregation, and the resulting scheme is referred to as *CollaPSE-Min*. In CollaPSE-Min, the computation cost is $\frac{2^{\epsilon-1}(\gamma n+1)(\log \Delta+2)}{H}$ PRFs for both encryption and decryption, considering that the concatenation technique in Section V-A also works for CollaPSE.

Compared with CollaPSE-Min, our Min aggregation scheme improves the computation cost of encryption and decryption by a factor of $\frac{\gamma n+1}{2c}$ and $\frac{\gamma n+1}{q}$ respectively (see Table IX for the cost of our scheme). Our scheme is also much more efficient than EXP-Min in computation, as shown in Section VI-B.

B. Practical Performances

In Table X, the computation costs of our Sum aggregation protocol and EXP are measured by different units, i.e., modular exponentiation and PRF invocation. Here, we elaborate the comparison between them with results in running time. Note that PRF can be implemented with HMAC in practice.

According to the bench-marking data reported by eBACS [30], it takes roughly 0.3 *ms* to compute a modular exponentiation using high-speed elliptic curves such as "curve25519" on a 64-bit desktop PC, and it takes roughly 0.26 μs[1] to compute an HMAC when SHA-512 is used as the hash function.

Based on these numbers, Table XI shows the running time of our Sum aggregation protocol and EXP. Our protocol is much faster than EXP in both encryption and decryption. Specifically, encryption is two orders of magnitude faster. When the plaintext space $\Delta \geq 10^3$, decryption is at least four orders of magnitudes faster. In our protocol, the computation cost decreases as the system scale increases, and it does not change with the plaintext space (so long as the size of plaintext data does not exceed the size of an HMAC output). Thus, our protocol can support large systems and large plaintext spaces.

Table XII shows the running time of our Min aggregation protocol and EXP-Min. Here, the plaintext space is set as $\Delta = 10^4$. The parameters of our protocol are set according to Table V and Table VI when $\gamma = 0.2$. In all the shown cases, our protocol is at least five (six) orders of magnitude faster than EXP-Min in encryption (decryption). Especially, as the system scale increases, the running time of decryption in EXP-Min increases quickly which shows the poor scalability of EXP-Min, but the running time of our protocol decreases and is always very low, which shows that our protocol is scalable.

TABLE XI
THE RUNNING TIME OF OUR PROTOCOL FOR SUM AND THE CONSTRUCTION PROPOSED IN [7] (DENOTED BY "EXP").

	n	10^2	10^3	10^4	10^5	10^6
Enc.	Ours	3.1μs	2.6μs	2.1μs	1.6μs	1.6μs
	EXP	600μs	600μs	600μs	600μs	600μs
Dec.	Ours	3.3μs	2.1μs	1.6μs	1.3μs	1μs
	EXP($\Delta = 10^2$)	30ms	95ms	300ms	950ms	3s
	EXP($\Delta = 10^3$)	95ms	300ms	950ms	3s	9.5s
	EXP($\Delta = 10^4$)	300ms	950ms	3s	9.5s	30s
	EXP($\Delta = 10^5$)	950ms	3s	9.5s	30s	95s

Δ is the plaintext space. Each user's data value is from $\{0, 1, ..., \Delta\}$.

VII. Discussions

More aggregate statistics. In the basic aggregation scheme for Min presented in Section V-A, the aggregator can actually get the number of times that each possible data value appears, and derive the accurate distribution of the users' data in the plaintext space $[0, \Delta]$. From the distribution, other aggregate statistics such as Median, Percentile and Histogram can be obtained. In this process, the aggregator knows nothing about each individual user's data.

Differential privacy for Sum. Differential privacy [31] provides strong and provable privacy guarantee for users such that a user's participation in the system only leaks negligible information about the user. Our protocol for Sum can be adapted to provide computational differential privacy [8]. To achieve this goal, an appropriate noise is added to each user's data (e.g., using the data perturbation algorithms proposed in [7] or [6]), and then the sum of noisy data is obtained using our protocol.

Dynamic joins/leaves and fault tolerance When a user joins or leaves, the trusted authority can issue a new set of secrets to every user and the aggregator. For a large system with high churn, the communication cost caused by dynamic joins or leaves may be high. To reduce the cost, the *binary* construction proposed in [8] can be applied on top of our Sum aggregation protocol, such that the expensive rekeying is only needed in a small number of joins or leaves. With data perturbation [7] and the binary construction incorporated, fault tolerance can also be achieved. When a number of users

[1] According to eBACS, it takes 0.13 μs to compute one SHA-512. Since one HMAC mainly computes two hashes, we simply double this time to get the running time of HMAC.

TABLE XII

THE RUNNING TIME OF OUR MIN AGGREGATION PROTOCOL AND EXP-MIN WHICH USES EXP AS A BUILDING BLOCK.

		Encryption				Decryption			
		$n = 10^3$	$n = 10^4$	$n = 10^5$	$n = 10^6$	$n = 10^3$	$n = 10^4$	$n = 10^5$	$n = 10^6$
Relative error $< 1\%$	EXP-Min	587ms	587ms	587ms	587ms	9.3s	29s	93s	294s
	Ours	5μs	4μs	3μs	3μs	4μs	3μs	2.5μs	2μs
Relative error $< 0.1\%$	EXP-Min	4.7s	4.7s	4.7s	4.7s	74s	235s	743s	2348s
	Ours	40μs	32μs	24μs	24μs	32μs	24μs	20μs	16μs

fail (e.g., due to loss of power or network connection), the aggregator can still get the aggregate statistics of the remaining users.

VIII. CONCLUSIONS

To facilitate the collection of useful aggregate statistics in mobile sensing without leaking mobile users' privacy, we proposed a new privacy-preserving protocol to obtain the Sum aggregate of time-series data. The protocol utilizes additive homomorphic encryption and a novel, HMAC-based key management technique to perform extremely efficient aggregation. Comparisons based on bench-marking measurement data show that operations at user and aggregator in our protocol are orders of magnitude faster than existing work. Thus, our protocol can be applied to a wide range of mobile sensing systems with various scales, plaintext spaces, aggregation loads and resource constraints.

Based on the Sum aggregation protocol, we also proposed two schemes to derive the Min aggregate of time-series data. One scheme can obtain the accurate Min while the other one can obtain an approximate Min with provable error guarantee at much lower cost.

REFERENCES

[1] M. Mun, S. Reddy, K. Shilton, N. Yau, J. Burke, D. Estrin, M. Hansen, E. Howard, R. West, and P. Boda, "Peir, the personal environmental impact report, as a platform for participatory sensing systems research," in *Proceedings of the 7th international conference on Mobile systems, applications, and services*, ser. MobiSys '09, 2009, pp. 55–68.

[2] A. Thiagarajan, L. Ravindranath, K. LaCurts, S. Madden, H. Balakrishnan, S. Toledo, and J. Eriksson, "Vtrack: accurate, energy-aware road traffic delay estimation using mobile phones," in *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys '09, 2009, pp. 85–98.

[3] S. Consolvo, D. W. McDonald, T. Toscos, M. Y. Chen, J. Froehlich, B. Harrison, P. Klasnja, A. LaMarca, L. LeGrand, R. Libby, I. Smith, and J. A. Landay, "Activity sensing in the wild: a field trial of ubifit garden," in *Proceedings of the twenty-sixth annual SIGCHI conference on Human factors in computing systems (CHI)*, 2008, pp. 1797–1806.

[4] J. Hicks, N. Ramanathan, D. Kim, M. Monibi, J. Selsky, M. Hansen, and D. Estrin, "Andwellness: an open mobile system for activity and experience sampling," in *Proc. Wireless Health*, 2010, pp. 34–43.

[5] N. D. Lane, M. Mohammod, M. Lin, X. Yang, H. Lu, S. Ali, A. Doryab, E. Berke, T. Choudhury, and A. Campbell, "Bewell: A smartphone application to monitor, model and promote wellbeing," in *5th International ICST Conference on Pervasive Computing Technologies for Healthcare*, 2011.

[6] V. Rastogi and S. Nath, "Differentially private aggregation of distributed time-series with transformation and encryption," *ACM SIGMOD*, 2010.

[7] E. Shi, T.-H. H. Chan, E. Rieffel, R. Chow, and D. Song, "Privacy-preserving aggregation of time-series data," *Network and Distributed System Security Symposium (NDSS)*, 2011.

[8] T.-H. H. Chan, E. Shi, and D. Song, "Privacy-preserving stream aggregation with fault tolerance," *Financial Cryptography and Data Security (FC)*, 2012.

[9] E. G. Rieffel, J. Biehl, W. van Melle, and A. J. Lee, "Secured histories: computing group statistics on encrypted data while preserving individual privacy." *In submission*, 2010.

[10] P.-A. Fouque, G. Poupard, and J. Stern, "Sharing decryption in the context of voting or lotteries," in *Proceedings of the 4th International Conference on Financial Cryptography*, ser. FC '00, 2000, pp. 90–104.

[11] MNDOLI, "Mnosha permissible exposure limits," available at http://www.dli.mn.gov/OSHA/PDF/pels.pdf.

[12] S. B. Eisenman, E. Miluzzo, N. D. Lane, R. A. Peterson, G.-S. Ahn, and A. T. Campbell, "The bikenet mobile sensing system for cyclist experience mapping," in *Proceedings of the 5th international conference on Embedded networked sensor systems (SenSys)*, 2007, pp. 87–101.

[13] M. G. Apte, W. J. Fisk, and J. M. Daisey, "Indoor carbon dioxide concentrations and sbs in office workers," in *Proceedings of Healthy Buildings*, 2000, pp. 133–138.

[14] Z. Zhu and G. Cao, "Applaus: A privacy-preserving location proof updating system for location-based services," in *Proc. IEEE INFOCOM*, 2011.

[15] Q. Li and G. Cao, "Mitigating routing misbehavior in disruption tolerant networks," *IEEE Transactions on Information Forensics and Security*, vol. 7, no. 2, pp. 664–675, April 2012.

[16] E. D. Cristofaro and C. Soriente, "Short paper: Pepsi—privacy-enhanced participatory sensing infrastructure," in *Proceedings of the fourth ACM conference on Wireless network security (WiSec)*, 2011, pp. 23–28.

[17] Q. Li, S. Zhu, and G. Cao, "Routing in socially selfish delay tolerant networks," in *Proc. IEEE INFOCOM*, 2010, pp. 1–9.

[18] Q. Li, W. Gao, S. Zhu, and G. Cao, "A routing protocol for socially selfish delay tolerant networks," *Ad Hoc Networks*, vol. 10, no. 8, pp. 664–675, 2012.

[19] D. Bonet, E.-J. Goh, and K. Nissim, "Evaluating 2-dnf formulas on ciphertexts," *TCC*, 2005.

[20] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *ACM symposium on Theory of computing (STOC)*, 2009, pp. 169–178.

[21] C. Castelluccia, A. C.-F. Chan, E. Mykletun, and G. Tsudik, "Efficient and provably secure aggregation of encrypted data in wireless sensor networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 5, no. 3, pp. 20:1–20:36, 2009.

[22] Y. Yang, X. Wang, S. Zhu, and G. Cao, "Sdap: A secure hop-by-hop data aggregation protocol for sensor networks," *ACM Transactions on Information and System Security (TISSEC)*, vol. 11, no. 4, 2008.

[23] Z. Yang, S. Zhong, and R. N. Wright, "Privacy-preserving classification of customer data without loss of accuracy," in *SIAM SDM*, 2005, pp. 21–23.

[24] J. Shi, R. Zhang, Y. Liu, and Y. Zhang, "Prisense: privacy-preserving data aggregation in people-centric urban sensing systems," in *Proc. IEEE INFOCOM*, 2010, pp. 758–766.

[25] G. Ács and C. Castelluccia, "I have a dream!: differentially private smart metering," in *Proceedings of the 13th international conference on Information hiding*, ser. IH'11, 2011, pp. 118–132.

[26] M. Jawurek and F. Kerschbaum, "Fault-tolerant privacy-preserving statistics," in *The 12th Privacy Enhancing Technologies Symposium (PETS)*, 2012.

[27] M. Shao, Y. Yang, S. Zhu, and G. Cao, "Towards statistically strong source anonymity for sensor networks," in *Proc. IEEE INFOCOM*, 2008.

[28] C. Castelluccia, "Efficient aggregation of encrypted data in wireless sensor networks," in *In MobiQuitous*. IEEE Computer Society, 2005, pp. 109–117.

[29] M. Bellare, "New proofs for nmac and hmac: security without collision-resistance," in *Proceedings of the 26th annual international conference on Advances in Cryptology*, ser. CRYPTO'06. Springer-Verlag, 2006, pp. 602–619.

[30] D. J. Bernstein and T. L. (editors), "ebacs: Ecrypt benchmarking of cryptographic systems," *http://bench.cr.yp.to, accessed 11 Feb 2012.*

[31] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," *TCC*, 2006.