Communication-Efficient and Privacy-Preserving Data Aggregation without Trusted Authority

Xuhui Gong*, Qiang-Sheng Hua*†, Lixiang Qian*, Dongxiao Yu*† and Hai Jin*

* Services Computing Technology and System Lab/Big Data Technology and System Lab/Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, P.R. China.

Abstract—**Privacy-preserving data aggregation has been extensively studied in the past decades. However, most of these works target at specific aggregation functions such as additive or multiplicative aggregation functions. Meanwhile, they assume there exists a trusted authority which facilitates the keys and other information distribution. In this paper, we aim to devise a communication efficient and privacy-preserving protocol that can exactly compute arbitrary data aggregation functions without trusted authority. In our model, there exist one untrusted aggregator and n participants. We assume that all communication channels are insecure and are subject to eavesdropping attacks. Our protocol is designed under the semi-honest model, and it can also tolerate k $(k \leq n-2)$ collusive adversaries. Our protocol achieves $(n-k)$-source anonymity. That is, for the source of each collected data aparting from the colluded participants, what the aggregator learns is only from one of the $(n-k)$ non-colluded ones. Compared with recent work [1] that computes arbitrary aggregation functions by collecting all the participants' data using the trusted authority, our protocol increases merely by at most a factor of $O((\frac{\log n}{\log \log n})^2)$ in terms of computation time and communication cost. The key of our protocol is that we have designed algorithms that can efficiently assign unique sequence numbers to each participant without the trusted authority.**

I. Introduction

Data aggregation has gained a lot of attentions in the past decades, and it has been used in many real life applications such as mobile cloud computing and smart grid [2]. In these applications, customers provide sensitive data (e.g., electricity usage information by the smart meter) to the control center. Then the control center calculates some functions such as the average or maximum power consumption of customers. In the problem of data aggregation, the participant acts like customers and the aggregator just likes control center.

Generally, participants are distributed in a network. They communicate with the aggregator in each time interval. These communications occupy the network bandwidth and consume energy, which will be a big problem if the communication is heavy, especially in today's data center. Thus, how to design a communication-efficient protocol is an important problem. In addition, if the aggregator is untrusted, it might abuse these data from the participants. What's worse, hackers or eavesdroppers might tamper the data which participants send to the aggregator. This causes the aggregator to produce a wrong result. Therefore, privacy protection is another key problem that should be considered.

†The corresponding authors are Qiang-Sheng Hua and Dongxiao Yu {qshua,dxyu}@hust.edu.cn

Most of previous works on privacy-preserving data aggregation focus on specific functions such as Sum, Min and Max functions [3] [4] [5] [6] [7]. The limitation is that we have to customize the algorithm according to different functions, which makes the system hard to extend. To let the system more scalable, we prefer to devise a protocol for handling arbitrary functions. One straightforward idea is that the aggregator collects all the data of participants, and then it computes arbitrary functions. Based on this idea, the method of secure sum computation can realize the goal [6] [4] [5]. Using these protocols, we can obtain all the participants' data by traversing the entire data space. However, this method will lead to high communication cost and computation time when the data space is large.

Very recently, Zhang et al. [1] proposed a new privacy-preserving protocol that can exactly compute arbitrary aggregation functions. They assign a unique sequence number to each participant. Then based on these unique sequence numbers, their approach can effectively reduce the communication cost and computation time compared with the straightforward method. However, their approach needs a trusted authority, which is used to send unique sequence numbers and keys to every participant. Note that a totally trusted authority is hard to find in practice [8].

There are some techniques which can achieve efficient key distribution and management without the trusted authority, such as public key schemes. Xiao et al. [9] provided a comprehensive survey for these key management techniques. But all of these techniques cannot fulfill the task of assigning unique sequence numbers to all participants, which is a key ingredient in designing an algorithm for handling arbitrary aggregation functions. Thus, how to generate unique sequence numbers without the trusted authority is a big challenge for designing the data aggregation protocol.

In this paper, we propose a randomized privacy-preserving protocol which can process arbitrary aggregation functions. Our protocol mainly consists of three phases. In the first two phases, a unique sequence number for each participant is generated. According to the unique sequence numbers, our protocol can collect all the participants' data and process arbitrary aggregation functions in the final phase. Different from [1], no trusted authority is required in our protocol.

TABLE I: Comparison with the Previous Work

Computation time	Communication cost (bits)	TA††	Method
$O(kn^2)^*$	$O(n^2 \log M)^\dagger$	**Yes**	[1]
$O(kn^2(\frac{\log n}{\log\log n})^2)$	$O(n^2(\frac{\log n}{\log\log n})^2 \log M)$	**No**	Our

*Although, the computation time of each participant is $O(kn)$ in [1], the total computation time of both the aggregator and the participants is $O(kn^2)$. ††TA denotes trusted authority. $^\dagger M$ denotes a security parameter.

Our contributions are briefly summarized as below and the comparison of our work with [1] is listed in Table I.

- We present a randomized privacy-preserving protocol that can exactly compute arbitrary aggregation functions. Our protocol does not rely on trusted authority. Compared with the work in [1] which employs trusted authority, our protocol increases merely by at most a factor of $O((\frac{\log n}{\log\log n})^2)$ in terms of computation time and communication cost.
- Our protocol can protect the privacy of participants' data against the untrusted aggregator, even if the aggregator colludes with k $(k \leq n-2)$ participants.

This paper is organized as follows: Section II briefly introduces the related work. Section III presents the problem definition, models and our protocol goals. In Section IV, we show how to generate secure unique sequence numbers without the trusted authority and give the protocol of exactly computing arbitrary aggregation functions. The detailed analyses for the correctness, security and complexities are given in Section V. Section VI shows the practical performance evaluation. Finally, we conclude our paper in section VII.

II. Related Work

There are three main techniques which are used to compute the data aggregation: secure multi-party computation, homomorphic encryption, and anonymous broadcast.

Yao [10] firstly proposed the notion of secure two-party computation in order to solve millionaires' problem in 1982. Then it was extended to secure multi-party computation (SMC). The millionaires' problem can be converted into secure multi-party sort problem. The secure multi-party sort can be used as our subroutine of assigning unique sequence numbers. Liu et al. [11] presented two protocols solving secure multi-party sort problem by vectorization method and Paillier encryption scheme. Compared with Yao's works, these works have low computation overhead. Most proposed protocols for SMC including those given in [10] and [11] have high communication and computation cost in our model. In addition, most of the previous protocols solving millionaires' problem cannot handle our problem. These works may expose the information of the rank in an ordered sequence. For example, if participant i knows that the data of participant j $(j \neq i)$ is less than its data and the rank of participant i is the second in the ordered sequence, then participant i can deduce the rank of participant j in the ordered sequence. Thus, the information of j is exposed.

During the past decades, many privacy-preserving data aggregation protocols were designed with an untrusted aggregator. In some ways, most of these protocols can be attributed to homomorphic encryption. Shi et al. [5] presented a protocol of verifiable privacy-preserving data aggregation based on the data slicing and mixing technique. The protocol supports additive and non-additive aggregation functions. Li et al. [6] proposed a protocol for Sum aggregation function by homomorphic encryption, which could handle the case that the aggregator is untrusted. In addition, it can be extended to solve the Min aggregation function. He et al. [12] proposed two novel privacy-preserving data aggregation protocols in wireless sensor networks. Groat et al. [3] presented a privacy-preserving protocol for Min and Max aggregation functions using k-indistinguishability. In mobile sensing systems, Zhang et al. [7] designed an efficient and privacy-preserving scheme by XOR homomorphic encryption and probabilistic coding technique. Jung et al. [13] [14] presented a collusion-tolerable and privacy-preserving protocol based on the hardness of CDH (Computational Diffie-Hellman) problem without secure channel. The majority of these works only deal with the specific aggregation functions such as Sum, Min and Product aggregation functions. Zhang et al. [15] designed a privacy-preserving protocol that can handle additive and non-additive aggregation functions approximately. Very recently, Zhang et al. [1] proposed a new privacy-preserving protocol that can solve arbitrary aggregation functions exactly. However, it needs a trusted authority.

In addition, some works had been done with respect to protecting source privacy. Conti et al. [16] presented a complete survey on protecting source privacy. In this comprehensive survey, some protocols that support anonymous broadcast are included [17] [18] [19] [20] [21]. The majority of these works require a multi-hop path in their model, which cannot be applied in our model. In addition, DC-Net [19] will suffer from collision problems if applied to solve our problem. Crowds [20] cannot defend against the global eavesdropper. Thus, anonymous broadcast protocol is not applicable for our problem.

III. System Model

A. System Model and Problem Definition

We use the One Aggregator model, which consists of one untrusted aggregator and n participants. Let p_i $(i \in [n] = \{1, ..., n\})$ represent the i-th participant. The aggregator is denoted as $\mathcal{A}$. Each participant p_i generates one input data $D_i \in \{0,1\}^d$ in a time interval. Meanwhile every participant p_i can communicate with the aggregator via a bi-directional communication channel. The aggregator is mainly responsible for calculating and publishing the final result of aggregation functions.

We aim to solve the problem that the aggregator computes arbitrary aggregation functions, while preserving the participants' privacy. Here, arbitrary aggregation functions include two types of functions. One is additive aggregation functions such as mean and variance, and the other is non-additive aggregation functions such as Max/Min, Percentile

TABLE II: Frequently Used Notations

Notation	Description
n	The number of participants
$\mathcal{A}$	The aggregator
p_i	The i-th participant
D_i	The input data where the participant p_i generates
$\overset{c}{\equiv}$	Computational indistinguishability
$[n]$	The set {1, 2,..., n}
$\mathbb{F}$	$\mathbb{F} = \{f_c : \{0,1\}^d \rightarrow \{0,1\}^d\}_{c \in \{0,1\}^d}$ is a pseudo-random function family
f_e	A pseudo-random function indexed by e from $\mathbb{F}_c$
φ	The nonce information $\{1, 2, ...\}$
M	A security parameter larger than $2^{\lceil \log(\max(D_i, n) \cdot n) \rceil}$
$\mathcal{I}$	The interval $[1, n^{\alpha+2}]$
$\mathcal{Q}_i$	The i-th subinterval $[(i-1) \cdot n^{\alpha+1} + 1, i \cdot n^{\alpha+1}]$
s_i	The sample value of participant p_i in the interval $\mathcal{I}$
$Seq(i)$	The unique sequence number of participant p_i
$\lvert Q_i \rvert$	The number of sample values where $\mathcal{Q}_i$ contains
β	The maximum number of sample values where $\mathcal{Q}_i$ contains for all $i \in [n]$
Φ_i, θ_i	The two shared key sets for participant p_i $(i \in [n])$

and Histogram. We summarize the notations used in the paper in Table II.

B. Threat Model

This paper focuses on the so-called "semi-honest model" [22], which means that the adversary is honest-but-curious. The adversary has three features: (1) She (He) faithfully executes a protocol and does not intentionally terminate the protocol at any time; (2) She (He) does not tamper the results of the computations; (3) She (He) is curious about the private content of others (e.g., other participants or the aggregator). For any participant, we assume it does not trust others, including the aggregator. The aggregator could collude with some participants. They can communicate with each other hoping to compute other participants' data. We assume that there are at most $n-2$ participants conspiring with the aggregator. Otherwise, they can deduce the private data of the other participants in our model.

C. Security Model

We adapt a simulation paradigm [22] to justify the security of our protocol as follows.

Definition 1 (*Privacy preserving*)**.** *In the one aggregator model, there are n participants and one aggregator $\mathcal{A}$. For a deterministic function $f = \{f_1, ..., f_n\}$, suppose that there are k $(k \leq n-2)$ participants colluding with the aggregator. Let $I = \{p_1, ..., p_k\}$ denote the set of participants colluding with $\mathcal{A}$. $X = \{x_1, ..., x_n\}$ represents the input. Let $f_0 = \{f_1(X), ..., f_k(X), f_{\mathcal{A}}(X)\}$, where $f_i(X)$ $(i = 1, ..., k)$ and $f_{\mathcal{A}}(X)$ denote the output of p_i and $\mathcal{A}$, respectively. Any probabilistic polynomial time protocol π privately computes function f, if there exists a polynomial time simulator S, on input $\{x_1, ..., x_k\}$, such that*

$$S(I \cup \{\mathcal{A}\}, \{x_1, ..., x_k\}, f_0) \overset{c}{\equiv} view_0^{\pi}(X) \tag{1}$$

where $\overset{c}{\equiv}$ denotes computational indistinguishability and $view_0^{\pi}(X) = \{I \cup \{\mathcal{A}\}, view_1^{\pi}(X), ..., view_k^{\pi}(X), view_{\mathcal{A}}^{\pi}(X)\}$. $view_i^{\pi}$ is the view of participant p_i that executes the protocol π on the input X. $view_i^{\pi}$ contains $\{x_i, r^i, m_1^i, m_2^i, ..., m_t^i\}$, where r^i is the outcome of her (his) internal coin tosses, and m_j^i $(j \in \{1, 2, ...\})$ represents the i-th message she (he) obtained during the implementation of the protocol π and the view of the aggregator is denoted as $view_{\mathcal{A}}^{\pi}(X)$ similarly.

Definition 2 ($(n-k)$*-source anonymity*)**.** *In the one aggregator model, there are n participants and one aggregator, where participant p_j $(j \in [n])$ has one input data $D_j \in \{0,1\}^d$ and let $I = \{p_1, ..., p_k\}$ denote the set of the k participants which collude with the aggregator. For a protocol π, it achieves $(n-k)$-source anonymity, if for any pair of participants p_i and p_j $(p_i, p_j \notin I)$ such that*

$$view_{\mathcal{A}}^{\pi}(Y) \overset{c}{\equiv} view_{\mathcal{A}}^{\pi}(Z) \tag{2}$$

where $Y = \{p_i(\mathcal{B}_i) : i \in [n]\}$, $Z = \{p_l(\mathcal{B}_l) : l \in [n], l \neq i, l \neq j\} \cup \{p_i(\mathcal{B}_j), p_j(\mathcal{B}_i)\}$. $\mathcal{B}_i$ represents the data of the participant p_i containing D_i and her (his) unique sequence number, and $view_{\mathcal{A}}^{\pi}(\cdot)$ denotes the view of the aggregator.

We assume that the Decision Diffie-Hellman (DDH) assumption holds for any probabilistic polynomial time adversary. For the DDH assumption, it is described as follows.

Definition 3 (*DDH assumption*)**.** *Let $\mathbb{G}$ denote a cyclic group with the prime order q, and g be a generator of the group $\mathbb{G}$. Given only the elements $g, g^x, g^y \in \mathbb{G}$, no probabilistic polynomial time adversaries can distinguish between the Diffie-Hellman tuples (g^x, g^y, g^{xy}) and the random tuples (g^x, g^y, g^z), where x, y, z are sampled from $\mathbf{Z}_q$ at random.*

D. Complexity Measure

We mainly consider two measures of efficiency of the protocol, i.e., computation time and communication cost. They are defined as follows:

Definition 4 (*Computation Time*)**.** *The computation time of a protocol is the total time that the aggregator and participants perform a computational process from the beginning of the execution to the end.*

Definition 5 (*Communication Cost*)**.** *The communication cost of a protocol is the total number of bits which are sent by the aggregator and all participants from the beginning of the execution to the end.*

E. Our Design Goals

In this paper, we propose a protocol to compute arbitrary aggregation functions on the aggregator. We focus on three aspects about designing a protocol as follows.

1) Result Accuracy: exactly computing arbitrary aggregation functions.
2) Data Privacy: achieving $(n-k)$-source anonymity.
3) Communication Efficiency: realizing low communication cost.

IV. Our Protocol

A. Protocol Overview

In this section, we will describe our protocol for computing arbitrary aggregation functions. There are three phases in our protocol: the initialization phase, creating the unique sequence numbers phase and computing the aggregation functions phase. In the first two phases, each participant produces a unique sequence number and does not know the unique sequence numbers of other participants. In the last phase, the aggregator can collect all participant's data based on the unique sequence numbers. Our protocol can defend against an untrusted aggregator that even colludes with $n-2$ participants.

We next explain our protocol in detail as follows.

B. The Initialization Phase

First, we need to distribute the system parameters to each participant so that they can establish shared keys with each other. We adopt the Diffie-Hellman key exchange algorithm [23] to establish the shared keys. Using their techniques, we will produce a q-order cyclic (multiplicative) group $\mathbf{G}$. It is constructed as follows:

Two large primes p, q are picked such that $q|(p-1)$. Then, a number $h \in Z_p$ is randomly selected. Finally, the generator g of the group $\mathbf{G}$ is computed:

$$g = h^{\frac{p-1}{q}} \mod p,$$

where $g \neq 1 \mod p$.

Then, the aggregator sends the system parameters $(p, q, \overline{G}, g$, n and a constant $\alpha)$ to all participants, where $\overline{G}$ denotes a description of the group $\mathbf{G}$.

Second, using the techniques as shown in [23], participants p_i and p_j $(i \neq j)$ can obtain shared keys.

The detailed operations are described as follows. Specifically, participant p_i samples the $2(n-1)$ secret numbers from $\mathbf{Z}_q$ uniformly at random. The numbers are divided into the first set $\{r_{i,j} : j \in [n], j \neq i\}$ and the second set $\{t_{i,j} : j \in [n], j \neq i\}$. Using the generator g, participant p_i generates ciphertext $b_{i,j} = g^{r_{i,j}} \mod p$ $(j \neq i)$ via modular exponentiation operation and sends the ciphertext $b_{i,j}$ to the aggregator. When the aggregator receives the ciphertext, it forwards the ciphertext $b_{i,j}$ to the participant p_j. When the participant p_j receives the ciphertext, it computes $c_{i,j} = (b_{i,j})^{t_{j,i}} \mod p = g^{r_{i,j} t_{j,i}} \mod p$ by using the secret number $t_{j,i}$ and sends the ciphertext $d_{j,i} = g^{t_{j,i}} \mod p$ to p_i by the aggregator. The participant p_i obtains $c_{i,j}$ by calculating $(d_{j,i})^{r_{i,j}} \mod p$. Thus, participants p_i and p_j obtain a shared key $c_{i,j} = g^{r_{i,j} t_{j,i}} \mod p$. Similarly, the above process can be applied to other $r_{i,j}$ for participant p_i. Then, p_i can get one key set $\Phi_i = \{c_{i,\nu} : \nu \in [n], \nu \neq i\}$, where $c_{i,\nu} = g^{r_{i,\nu} t_{\nu,i}} \mod p$.

For other participants p_j $(j \neq i, j \in [n])$, they follow a similar process for the first set $\{r_{j,l} : l \in [n], l \neq j\}$. Then, participants p_i can also obtain another key set $\theta_i = \{c_{l,i} : l \in [n], l \neq i\}$, where $c_{l,i} = g^{r_{l,i} t_{i,l}} \mod p$. Thus, each participant p_i $(i \in [n])$ gets two shared key sets Φ_i and θ_i.

Next, each participant p_i $(i \in [n])$ uses two key sets to pick the corresponding pseudo-random functions $\{f_{c_{l,i}} : l \in [n], l \neq i\}$ and $\{f_{c_{i,\nu}} : \nu \in [n], \nu \neq i\}$ from the PRF (Pseudo-Random Function) family $\mathbb{F} = \{f_c : \{0,1\}^d \rightarrow \{0,1\}^d\}_{c \in \{0,1\}^d}$. Then, it constructs the function $H_i(\varphi)$ as follows:

$$H_i(\varphi) = \big(\sum_{j \in \theta_i} f_j(\varphi) - \sum_{v \in \Phi_i} f_v(\varphi)\big) \mod M, \quad (3)$$

where φ represents the nonce information and $M \geq 2^{\lceil \log(\max(D_i, n)) \cdot n \rceil}$ is a security parameter $(\log M \leq d)$.

Our idea for designing function $H_i(\varphi)$ originates from additive homomorphic encryption scheme which is proposed by [24].

C. Creating the Unique Sequence Numbers Phase

In this phase, we will create a unique sequence number for each participant. We use random sampling method and partitioning technique to establish unique sequence numbers. There are four steps in this phase. The detailed operations are presented as follows.

The first step: each participant p_i $(i \in [n])$ samples an integer value s_i independently and uniformly from $[1, n^{\alpha+2}]$ at random.

We have the following Observation 1.

Observation 1. *Let $s_1, s_2, ..., s_n$ be n numbers independently and uniformly sampled from $[1, n^{\alpha+2}]$ at random, then we have $s_i \neq s_j$ for $\forall i \neq j$ $(i, j \in [n])$ with probability at least $1 - 1/n^{\alpha}$.*

Due to the lack of space, we put the proof in the full version [25].

According to the sample value s_i, participant p_i divides interval $\mathcal{I} = [1, n^{\alpha+2}]$ into n disjoint subintervals, that is, $\{\mathcal{Q}_i = [(i-1)n^{\alpha+1} + 1, i \cdot n^{\alpha+1}] : i \in [n]\}$. Since the sample value s_i is independently and uniformly sampled from $[1, n^{\alpha+2}]$ at random. Thus, the probability that s_i belongs to subinterval $\mathcal{Q}_j$ is $\frac{1}{n}$ for any $i, j \in [n]$. we can obtain that the number of sample values in each subinterval $\{\mathcal{Q}_i : i \in [n]\}$ is at most $O(\frac{\ln n}{\ln \ln n})$ with high probability*, according to the following Lemma 1.

Lemma 1. *Let $\beta = 2(\lambda+1)\frac{\ln n}{\ln \ln n}$ $(\lambda \geq 1)$ and χ_i $(i \in [n])$ be the number of sample values located in $\mathcal{Q}_i$. Then, we have $max_{i \in [n]} \chi_i \leq \beta$ with probability at least $1 - \frac{1}{n^{\lambda}}$.*

Due to the lack of space, we put the proof in the full version [25].

The second step: participant p_i takes the following operations. First, it produces an n-dimensional vector on the basis of which subintervals the sample value locates. Without loss of generality (W.l.o.g.), we assume that sample value s_i belongs to subinterval $\mathcal{Q}_j$. Then, p_i produces an n-dimensional vector

*A high probability means a probability $1 - 1/n^{\lambda}$ for some constant $\lambda \geq 1$

$\mathbf{V}_i$ where the value of the j-th coordinate sets 1 and the others 0, that is, it can be expressed in the following form:

$$\mathbf{V}_i = (\underbrace{0, ..., 0}_{j-1}, 1, 0, ..., 0).$$

Next, participant p_i encrypts $\mathbf{V}_i = (0, ..., 0, 1, ..., 0)$ by using the $H_i(\varphi)$ and produces a new n-dimensional vector V_i^1 as follows:

$$V_i^1 = (v_i^1, ..., v_i^l, ..., v_i^n), \tag{4}$$

where

$$\begin{cases} v_i^l = (H_i(l) + 0) \mod M & (l \in [n], l \neq j), \\ v_i^l = (H_i(l) + 1) \mod M & (l = j). \end{cases}$$

Finally, each participant p_i sends V_i^1 to the aggregator. When the aggregator receives n vectors $\{V_j^1 : j \in [n]\}$, it counts up the vectors by vector addition and obtains a new vector $V_{agg}^1 = (v_a^1, ..., v_a^l, ..., v_a^n)$, where

$$v_a^l = \sum_{i=1}^{n} v_i^l \mod M, l \in [n]. \tag{5}$$

Note that the value of the i-th coordinate of vector V_{agg}^1 denotes the number of sample values in the subinterval $\mathcal{Q}_i$, i.e., the following Lemma,

Lemma 2. *In our first and second steps, the aggregator accurately obtains the number of the sample values s_j $(j \in [n])$ which belongs to corresponding subinterval $\mathcal{Q}_i$ $(i \in [n])$.*

Proof: W.l.o.g., we assume that there are s among all participants whose sample values locate in $\mathcal{Q}_1$, and let $p_1, .., p_s$ denote the participants. The participant p_i computes $v_i^1 = (H_i(1)+1) \mod M$ for any $i \in \{1, ..., s\}$ and the remaining participants p_i $(i \in \{s+1, ..., n\})$ compute $v_i^1 = (H_i(1) + 0) \mod M$. The aggregator receives v_i^1 from all participants p_i $(i \in [n])$. Since we have $\sum_{i=1}^{n} H_i(1) = 0$, the aggregator computes

$$\begin{aligned} V_a^1 &= \sum_{i=1}^{n} v_i^1 \mod M \\ &= \Big(\sum_{i=1}^{s}\big(H_i(1)+1\big) + \sum_{i=s+1}^{n}\big(H_i(1)+0\big)\Big) + \mod M \\ &= \big(\sum_{i=1}^{n} H_i(1)\big) + s \mod M = s. \end{aligned} \tag{6}$$

■

The third step: we will handle subintervals $\mathcal{Q}_i$ $(i \in [n])$ which contains more than one sample value s_j $(j \in [n])$. For the subinterval $\mathcal{Q}_i$, it will be handled by using a partitioning technique, which is presented in Algorithms 1 and 2. The Algorithm 1 aims to partition subintervals and the Algorithm 2 counts the number of sample values in the subintervals with privacy preservation. The aggregator starts partitioning operations based on V_{agg}^1. The steps will be introduced as follows.

Algorithm 1 PartitionIntervals($\mathcal{Q}_i$, num)

```
Input: $\mathcal{Q}_i$, $num$
Output: $T$
1: $L_0 = \lceil \frac{\log n}{\log\log n} \rceil, L_0' = \lceil \frac{\log n}{\log\log n} \rceil, L_0'' = \lceil \log n \rceil^{3/2}$;
2: if $num == 0$ then
3:     return;
4: else if $num == 1$ then
5:     return $T = T \bigcup \mathcal{Q}_i$;
6: else if $2 \leq num \leq L_0$ then
7:     Aggregator sets $\xi_1 = (L_0')^2$;
8:     Aggregator partitions $\mathcal{Q}_i$ into $\xi_1$ disjoint subintervals $\mathcal{Q}_{i,\tau}$ $(\tau = 1, 2, ..., \xi_1)$ uniformly;
9:     for $\tau = 1 : \xi_1$ do
10:        PartitionIntervals ($\mathcal{Q}_{i,\tau}$, CountNumber($\mathcal{Q}_{i,\tau}$));
11:    end for
12: else
13:    Aggregator sets $\xi_2 = (L_0'')^2$;
14:    Aggregator partitions $\mathcal{Q}_i$ into $\xi_2$ disjoint subintervals $\mathcal{Q}_{i,\tau}$ $(\tau = 1, 2, ..., \xi_2)$ uniformly ;
15:    for $\tau = 1 : \xi_2$ do
16:        PartitionIntervals($\mathcal{Q}_{i,\tau}$, CountNumber($\mathcal{Q}_{i,\tau}$) );
17:    end for
18: end if
```

Algorithm 2 CountNumbers($\mathcal{Q}_{i,\tau}$)

```
Input: $\mathcal{Q}_{i,\tau}$
Output: $VS_a^\tau$
1: Aggregator sends $\mathcal{Q}_{i,\tau}$ to all participants;
2: Each participant $p_j$ $(j \in [n])$ computes $z = z + 1$;
3: Each participant $p_j$ $(j \in [n])$ computes function $H_j(z)$;
4: for $j = 1 : n$ do
5:     if $s_j \in \mathcal{Q}_{i,\tau}$ then
6:         Participant $p_j$ computes $VC_j^\tau = \big(H_j(z) + 1\big) \mod M$ and sends to aggregator;
7:     else
8:         Participant $p_j$ computes $VC_j^\tau = \big(H_j(z) + 0\big) \mod M$ and sends to aggregator;
9:     end if
10: end for
11: Aggregator computes $VS_a^\tau = \sum_{i=1}^{n} VC_i^\tau \mod M$;
```

(1) When n is sufficiently large ($\rho^\dagger \geq 1/2$).

The aggregator sets $L_0 = \lceil \frac{\log n}{\log\log n} \rceil, L_0' = \lceil \frac{\log n}{\log\log n} \rceil$ and $L_0'' = \lceil \log n \rceil^{3/2}$, and generates an empty set T which is used to place subintervals in Algorithm 1. Each participant sets $z = n$ in Algorithm 2.

Then, the aggregator handles the subinterval $\mathcal{Q}_i$ for all i $(i \in [n])$. By Lemma 1, we have $\max_{i \in [n]} v_a^i \leq \beta$ with probability at least $1 - \frac{1}{n^\lambda}$. In the light of v_a^i $(i \in [n])$, we perform the following operation:

$^\dagger \rho = \frac{\log(\log n - e \log\log n)}{\log\log n} + \frac{\log(\log n - \log\log n)}{2 \log n}$

If $v_a^i = 0$, i.e., $|\mathcal{Q}_i| = 0$, in the Lines $2-3$ of Algorithm 1, the aggregator returns.

If $v_a^i = 1$, i.e., $|\mathcal{Q}_i| = 1$, in the Lines $4-5$ of Algorithm 1, the aggregator adds $\mathcal{Q}_i$ to the set T and returns.

If $2 \le v_a^i \le L_0$, i.e., $2 \le |\mathcal{Q}_i| \le L_0$, in the Lines $6-11$ of Algorithm 1, the aggregator sets $\xi_1 = (L_0')^2$ and divides $\mathcal{Q}_i$ into ξ_1 uniform subintervals $\mathcal{Q}_{i,\tau}$ ($\tau \in [\xi_1] = \{1, 2, ..., \xi_1\}$). Then, for each $\mathcal{Q}_{i,\tau}$ ($\tau \in [\xi_1]$), the aggregator sends it to all participants. Participant p_i ($i \in [n]$) uses Algorithm 2 to produce corresponding ciphertexts and sends them to the aggregator. So the aggregator obtains $VS_a = (VS_a^1, ..., VS_a^{\xi_1})$. For $VS_a^j \le 1$ ($j \in [\xi_1]$), the processing methods of the aggregator is the same as that in Lines $2-4$ of Algorithm 1 and Lines $5-7$ of Algorithm 1. For $2 \le VS_a^j \le L_0$, it reuses the Algorithm 1 and the Algorithm 2 to divide the subintervals. The aggregator continues to divide subintervals until $VS_a^i \le 1$ for any $i \in [\xi_1]$. For subinterval $\mathcal{Q}_{i,\tau}$, the partitioning process is completed when $VS_a^\tau \le 1$.

In Lines $13-17$ of Algorithm 1, if $v_a^i \ge L_0 + 1$, i.e., $|\mathcal{Q}_i| \ge L_0 + 1$, the aggregator sets $\xi_2 = (L_0'')^2$. The processing methods used by the aggregator is similar to the above steps .

(2) When n is relatively small ($\rho \le 1/2$). We set $L_0' = \lceil \lceil \log n \rceil^{2/3} \rceil$ and the rest remains the same in Algorithms 1 and 2.

The final step: After the aggregator and participants have performed the above algorithms for all $\mathcal{Q}_i$ ($i \in [n]$), the aggregator gets a set T. The set T consists of the n subintervals, where each subinterval contains only one sample value. The aggregator sends the set T to every participant, and then each participant obtains the rank of their sample values among all sample values in an ascending order. It is the unique sequence number $Seq(i)$ ($i \in [n]$) for any participant p_i ($i \in [n]$).

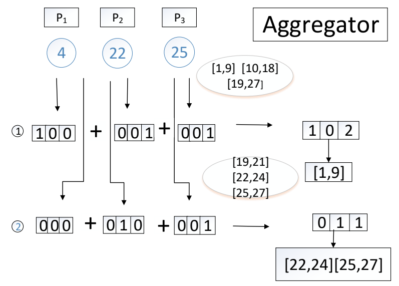


Fig. 1: An example of our Algorithms 1 and 2

An example to illustrate our four steps is displayed in Figure 1 while ignoring the privacy aspects of data. We consider that there are three participants and one aggregator. The three participants sample 4, 22 and 25 from $[1, 3^3]$($\alpha = 1$) uniformly at random, respectively. Each participant divides $[1, 3^3]$ into $[1, 9], [10, 18], [19, 27]$. Based on the subintervals, each participant generates a vector as shown in the ① and sends it to the aggregator. The aggregator gets $(1, 0, 2)$ by vector addition. Thus, it adds subinterval $[1, 9]$ to the set T. Furthermore, since $\lceil \lceil \log 3 \rceil^{\frac{4}{3}} \rceil = 3$, it sets $\xi_1 = 3$. Then, it divides $[19, 27]$ into $[19, 21], [22, 24], [25, 27]$ and sends them to all participants. Each participant generates a vector as shown in the ② and sends it to the aggregator. The aggregator gets $(0, 1, 1)$ by vector addition. It adds the subintervals $[22, 24], [25, 27]$ to the set T. Finally, the aggregator obtains the set T which contains $[1, 9], [22, 24], [25, 27]$ and sends it to every participant. Thus, participant p_i knows her own $Seq(i)$. The unique sequence numbers $Seq(i)$ of participant p_i ($i = 1, 2, 3$) are 1, 2 and 3, respectively.

D. Computing the Aggregation Functions Phase

Algorithm 3 Computing The Aggregation Functions Phase

```
Input: $Seq(i), D_i$
Output: $VT^a$
1: Run the first two phases, each participant $p_i$ gets an unique sequence number $Seq(i)$ for any $i$ ($i \in [n]$);
2: for each participant $p_i$ do
3:     for $j = 1 : n$ do
4:         if $Seq(i) == j$ then
5:             Participant $p_i$ computes $\varphi' = \nu + j$, $VT_i^j = (H_i(\varphi') + D_i) \mod M$ and sends to the aggregator.
6:         else
7:             Participant $p_i$ computes $\varphi' = \nu + j$, $VT_i^j = (H_i(\varphi') + 0) \mod M$ and sends to the aggregator.
8:         end if
9:     end for
10: end for
11: for $j = 1 : n$ do
12:     Aggregator computes $VT_j^a = \sum_{i=1}^n VT_i^j \mod M$
13: end for
14: Aggregator outputs $VT^a = (VT_1^a, ..., VT_n^a)$
```

In the above section, each participant has secretly obtained a unique sequence number $Seq(i) \in [n]$. In this section, we design a privacy-preserving scheme which can collect data of all participants based on $Seq(i)$. This process is illustrated in Algorithm 3.

Specifically, each participant p_i generates a vector $\mathbf{VG}_i^1$. The construction of $\mathbf{VG}_i^1$ is shown as follows:

$$\mathbf{VG}_i^1 = (VT_i^1, ..., VT_i^j, ..., VT_i^n), \tag{7}$$

where

$$\begin{cases} VT_i^j = (H_i(\nu + j) + 0) \mod M & (j \in [n], j \ne Seq(i)), \\ VT_i^j = (H_i(\nu + j) + D_i) \mod M & (j = Seq(i)), \end{cases}$$

ν denotes the total number of subintervals which were produced in the first two phases of our protocol.

Next, participant p_i ($i \in [n]$) sends $\mathbf{VG}_i^1$ to the aggregator. So the aggregator receives $\mathbf{VG}_1^1$, $\mathbf{VG}_2^1$,...,$\mathbf{VG}_n^1$ and computes $VT^a = (VT_1^a, ..., VT_n^a)$, where

$$VT_j^a = \sum_{i=1}^n VT_i^j \mod M, j \in [n]. \tag{8}$$

VT^a is a vector which consists of all participants' data. The aggregator can compute arbitrary aggregation functions using the vector VT^a.

V. Correctness, Security And Complexity Analysis

In this section, we analyze correctness, security and complexity of our protocol.

A. Correctness

First, we show the correctness of our protocol as follows.

Lemma 3. *In our Algorithm 2, the aggregator accurately obtains the number of the sample values s_j ($j \in [n]$) which belongs to corresponding subinterval.*

Lemma 4. *In our Algorithm 3, the aggregator accurately obtains all participants' data.*

The proof of Lemmas 3 and 4 are similar to Lemma 2, which we omit here.

B. Security

Next, we discuss the security of our protocol, according to Definitions 1 and 2. We show that our protocol is secure and does not leak any information for non-collusion participants, except what can be deduced from the output of our protocol.

Theorem 1. *Algorithms 1 and 2 are secure in the semi-honest model, even if the aggregator colludes with k ($k \leq n-2$) participants.*

Proof: Suppose that there are k ($k \leq n-2$) participants which colludes with the aggregator $\mathcal{A}$. Let $I = \{p_1, ..., p_k\}$ denote the set of conspirators with colluding with $\mathcal{A}$ and $\{p_{k+1}, ..., p_n\}$ denote the other participants. Let $C = \{1, ..., k\}$ and $P = \{k+1, ..., n\}$. The participants in set A can send all information to $\mathcal{A}$ so that $\mathcal{A}$ deduces information of participants p_i ($i \in P$), which implies that $view_{\mathcal{A}}^{\pi}(\cdot) \supseteq \{view_i^{\pi}(\cdot) : i \in C\}$. Thus, we only consider the view of $\mathcal{A}$. Without loss of generality, we consider the first coordinate of vectors V_{agg}^1, $\mathbf{V}_i$ and V_j^1, denoted by $V_{agg}^1(1)$, $\mathbf{V}_i(1)$ and $V_j^1(1)$, respectively.

According to Definition 1, we need to construct a simulator S_0 as follows:

$\mathcal{A}$ receives the set $\Theta = \{s_i, \theta_i, \Phi_i, b_{q,l} = g^{r_{q,l}} \mod p, d_{l,q} = g^{t_{l,q}} \mod p, V_j^1(1), \mathbf{V}_i(1), V_{agg}^1(1) : i \in C, j = [n], l, q \in P\}$ and uses them as its input.

According to $\mathbf{V}_i(1)$ ($i \in C$) and $V_{agg}^1(1)$, $\mathcal{A}$ computes $SV = V_{agg}^1(1) - \sum_{i \in C} \mathbf{V}_i(1)$, that is, the number of sample values in non-collusion participants located in the subinterval $\mathcal{Q}_1$. Then it randomly constructs $\{\mathbf{V}_j^{'}(1)$: $j \in P\}$ such that $\sum_{j \in P} \mathbf{V}_j^{'}(1) = SV$.

For any $l \in P$, $\mathcal{A}$ knows all the $b_{q,l} = g^{r_{q,l}} \mod p$, $d_{l,q} = g^{t_{l,q}} \mod p$ ($l, q \in P$). Then it uniformly picks $\hat{e}_{q,l}, \hat{e}_{l,q}$ from $\mathbf{G}$ ($q \neq l, q, l \in P$) at random. Based on the DDH assumption, we have $c_{q,l} \overset{c}{\equiv} \hat{e}_{q,l}$ and $c_{l,q} \overset{c}{\equiv} \hat{e}_{l,q}$. Next, S_0 calculates $\widehat{V_j^1(1)}$ ($j \in P$) as follows:

$$\{\widehat{V_j^1(1)} = (G_j(1) + \mathbf{V}_j^{'}(1)) \mod M : j \in P\}, \tag{9}$$

where

$$\begin{aligned} G_j(1) =(&\sum_{c_{i,j} \in \theta_j \cap i \in C} f_{c_{i,j}}(1) - \sum_{c_{j,i} \in \Phi_j \cap i \in C} f_{c_{j,i}}(1) \\ &+ \sum_{q \in P-\{j\}} \hat{f}_{q,j} - \sum_{q \in P-\{j\}} \hat{f}_{j,q}) \mod M \end{aligned} \tag{10}$$

and $\{\hat{f}_{q,j}, \hat{f}_{j,q} : j, q \in P\}$ are sampled from $\{0,1\}^d$ uniformly at random so that

$$\sum_{j \in P} G_j(1) = \sum_{j \in P} H_j(1) \mod M. \tag{11}$$

We have

$$S_0(I \cup \{\mathcal{A}\}, \Theta) = \{V_1^1(1), ... V_k^1(1), \widehat{V_{k+1}^1(1)}, ..., \widehat{V_n^1(1)}\}.$$

Meanwhile, we know

$$view_0^{\pi}(\cdot) = \{V_1^1(1), ..., V_k^1(1), V_{k+1}^1(1), ..., V_n^1(1)\}.$$

According to Definition 1, we need to prove

$$S_0(I \cup \{\mathcal{A}\}, \Theta) \overset{c}{\equiv} view_0^{\pi}(\cdot).$$

Since s_i ($i \in P$) is sampled independently and uniformly at random from $[1, n^{\alpha+2}]$, the vector $\mathbf{V}_i$ can be considered as a vector which is sampled independently and uniformly at random from $\{\mathbf{1}_i : i \in [n]\}$, where

$$\mathbf{1}_i = (\underbrace{0, ..., 0}_{i-1}, 1, \underbrace{0, ..., 0}_{n-i}).$$

Thus, we obtain $\mathbf{V}_j(1) \overset{c}{\equiv} \mathbf{V}_j^{'}(1)$ for any $j \in P$. Furthermore, since $\{f_{c_{i,j}}\}_{i,j \in [n]}$ are pseudo-random functions from $\mathbb{F}$, we have

$$f_{c_{i,j}} \overset{c}{\equiv} U_{i,j}, \tag{12}$$

where $U_{i,j}$ ($i, j \in [n]$) are random variables and obey uniform distribution over $\{0,1\}^d$. Note that

$$\begin{aligned} H_j(\varphi) = &\sum_{c_{i,j} \in \theta_j \cap i \in C} f_{c_{i,j}}(\varphi) + \sum_{c_{i,j} \in \theta_j \cap i \in P} f_{c_{i,j}}(\varphi) \\ &- \sum_{c_{j,i} \in \Phi_j \cap i \in C} f_{c_{j,i}}(\varphi) - \sum_{c_{j,i} \in \Phi_j \cap i \in P} f_{c_{j,i}}(\varphi) \end{aligned} \tag{13}$$

and

$$V_j^1(1) = H_j(1) + \mathbf{V}_j(1). \tag{14}$$

Applying Equations (9), (10), (11), (12), (13) and (14), we have $\{V_j^1(1) : j \in P\} \overset{c}{\equiv} \{\widehat{V_j^1(1)} : j \in P\}$, where $\varphi = 1$.

Thus it holds that

$$S_0(I \cup \{\mathcal{A}\}, \Theta) \overset{c}{\equiv} view_0^{\pi}(\cdot).$$

■

Theorem 2. *Our protocol is $(n-k)$-source anonymity in the semi-honest model, even if the aggregator colludes with k ($k \leq n-2$) participants.*

Proof: (Sketch) We use the same notations as those in the proof of Theorem 1. Suppose that there are k ($k \leq n-2$) participants colluding with the aggregator $\mathcal{A}$. $\mathcal{A}$ wants to confirm which participants p_i $(i \in P)$ generates D_i. The view of $\mathcal{A}$ is a set of random variables as follows:

$$view_{\mathcal{A}}^{\pi} = \{VT_i^j : i, j \in [n]\}. \tag{15}$$

where VT_i^j represents the message which is the value of the j-th coordinate of $\mathbf{VG}_i^1$ in our protocol.

For any pair of participants p_i and p_j $(i < j, i, j \in P)$, we assume $Seq(i) < Seq(j)$. Participants p_i and p_j exchange the data $\{D_i, Seq(i)\}$ and $\{D_j, Seq(j)\}$. Then, the view of $\mathcal{A}$ is as follows:

$$\widehat{view_{\mathcal{A}}^{\pi}} = \{\widehat{VT_i^j} : i, j \in [n]\}. \tag{16}$$

According to Definition 2, we need to prove $view_{\mathcal{A}}^{\pi} \overset{c}{\equiv} \widehat{view_{\mathcal{A}}^{\pi}}$. We construct a simulator $\mathbf{S}$ as follows. For participants p_l $(l \in P)$, the simulator $\mathbf{S}$ first randomly generates a random unique sequence numbers set $\{r_l : l \in P\}$ from the set [n] except the unique sequence number of $p_{l'}$ $(l' \in A)$. Noting that the n sample values are sampled from $\mathcal{I}$ uniformly at random, $Seq(1), ..., Seq(n)$ are a random permutation in $[n]$. So there are $(n-k)!$ kinds of permutations for the participants in P. The simulator $\mathbf{S}$ randomly constructs $\overline{f}_{q,r}, \overline{f}_{r,q}$ $(r \neq q, r, q \in P)$ by sampling from $\{0,1\}^d$, and computes $\overline{VT_q^b} = (G_q'(\nu + b) + D_{\{a:Seq(a)=r_q\}}) \mod M$ for $b = r_q$ $(a, q \in P, b \in [n])$ and $\overline{VT_q^b} = (G_q'(\nu + b) + 0) \mod M$ for $b \neq r_q$ $(a, q \in P, b \in [n])$, respectively. Let $\overline{view_{\mathcal{A}}^{\pi}}$ denote the view of $\mathcal{A}$ in the process. Similar to the proof of Theorem 1, we have $view_{\mathcal{A}}^{\pi} \overset{c}{\equiv} \overline{view_{\mathcal{A}}^{\pi}}$. Similarly, we can obtain $\overline{view_{\mathcal{A}}^{\pi}} \overset{c}{\equiv} \widehat{view_{\mathcal{A}}^{\pi}}$.

Thus, we obtain $view_0^{\pi} \overset{c}{\equiv} \widehat{view_0^{\pi}}$ and our protocol achieves $(n-k)$-source anonymity. ∎

Similarly, semi-honest participants cannot learn any information except the output of our protocol.

C. Complexity

Finally, we discuss communication cost and computation time of our protocol in this section.

Theorem 3. *With probability at least $1 - \frac{1}{n^2}$, the number of subintervals where the aggregator sends to any participant p_i is at most $O(n(\frac{\log n}{\log\log n})^2)$ for a sufficiently large n $(\rho \geq 1/2)$.*

Due to the lack of space, we put the proof in the full version [25].

In our protocol, the aggregator sends the $O(n(\frac{\log n}{\log\log n})^2)$ subintervals to each participant and vice verse. Thus, we have the following corollary.

Corollary 1. *With probability at least $1 - \frac{1}{n^2}$, the communication cost of our protocol is at most $O(n^2(\frac{\log n}{\log\log n})^2 \log M)$ for a sufficiently large n $(\rho \geq 1/2)$.*

According to Theorem 3, we have the following theorem about the computation time.

Theorem 4. *With probability at least $1 - \frac{1}{n^2}$, each participant needs $O(n^2(\frac{\log n}{\log\log n})^2)$ d-bit hashing operations, $O(n^2(\frac{\log n}{\log\log n})^2)$ $\log M$-bit modular operations and $O(n)$ $\log p$-bit modular exponentiation operations, and the aggregator needs $O(n^2(\frac{\log n}{\log\log n})^2)$ $\log M$-bit modular operations and $O(n(\frac{\log n}{\log\log n})^2)$ partitioning operations.*

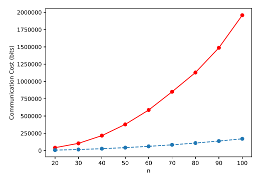
Noting that if we know that there are k participants colluding with the aggregator in parctice, each participant p_i only samples the $2(k+1)$ secret numbers: the first set $\{r_{i,j} : j = (q \mod n) + 1, q = i, ..., i+k\}$ and the second set $\{t_{j,i} : j = (q \mod n) + 1, q = i, ..., i+k\}$ in the initialization phase. The others are similar to the three phases of our protocol. Then we obtain a more efficient protocol and have the following corollary.

Corollary 2. *With probability at least $1 - \frac{1}{n^2}$, each participant needs $O(kn(\frac{\log n}{\log\log n})^2)$ d-bit hashing operations, $O(kn(\frac{\log n}{\log\log n})^2)$ $\log M$-bit modular operations and $O(k)$ $\log p$-bit modular exponentiation operations, and the aggregator needs $O(n^2(\frac{\log n}{\log\log n})^2)$ $\log M$-bit modular operations and $O(n(\frac{\log n}{\log\log n})^2)$ partitioning operations.*

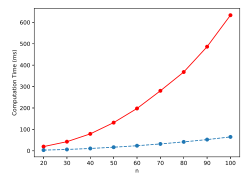
VI. Performance Evalution

In this section, we perform experiments to analyze the performance of our algorithms.

We perform the algorithms on a desktop running the 64-bits Debian GNU/Linux 9 operating system with Intel Core, i5-3230M G530 CPU and 12GB memory. The experiment is based on the mutilprocesses and network programming. We assume that the aggregator (participant) does not collude with any participant. We perform a process to simulate aggregator and a process to simulate n participants. To construct hash functions, we utilize Crypto++ library and use HMAC<SHA256> as the pseudo-random function family. The length of security parameter are 64-bits in the experiment. The sample values s_i and D_i of participants are sampled from $[1, n^5]$ uniformly at random. The number of participants is taken from 20 to 100. Noting that n is relatively small in this case $(\rho \leq 1/2)$, we set $L_0' = \lceil\lceil\log n\rceil^{2/3}\rceil$ and the rest remains unchanged in Algorithms 1 and 2. Similar to the above theorem, the number of subintervals is $O(n^2(\log n)^{4/3})$ with high probability. The communication cost and computation time are averaged by 500 runs. We show the results in Figures 2a and 2b. In Figures 2a and 2b, the experiment results show that the trend of the graph is a quadratic curve, and there is a gap of the constant factor between experimental results and theoretical analysis. The experiment also shows that the communication cost and the computation time of our algorithm are $O((\log n)^{4/3})$ times of those in [1], which matches our theoretical analysis. Furthermore, we compare the performance of our protocol with a baseline scheme. The baseline scheme can obtain all participants' data by using the secure data aggregation protocol [6], but it needs to traverse the space of participants' data. The baseline scheme will be run until the aggregator obtains all participants' data. In this experiment, the input data D_i of participants are sampled from $[1, n^2]$ uniformly at random and other parameters are the same as the first experiment. It is not hard to see that expected

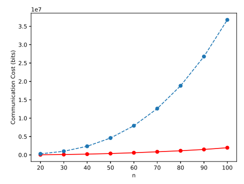


(a) The figure shows the relationship between the number of participants and the communication cost.

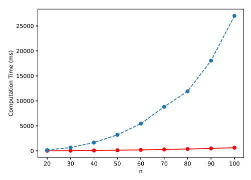


(b) The figure shows the relationship between the number of participants and the computation time.

Fig. 2: Experiments of our algorithm and the algorithm in [1]. The red solid line is the performance of our algorithm, the blue dashed line is the performance of the algorithm in [1].



(a) The figure shows the relationship between the number of participants and the communication cost.



(b) The figure shows the relationship between the number of participants and the computation time.

Fig. 3: Experiments of our algorithm and the baseline scheme by using the secure data aggregation protocol [6]. The red solid line is the performance of our algorithm, the blue dashed line is the performance of the baseline scheme.

communication cost and computation time of the baseline scheme are $O(n^3 \log M)$ and $O(n^3)$, respectively. In Figures 3a and 3b, the experiment shows that the communication cost and the computation time of the baseline scheme are almost $O(n)$ times of our algorithm, which matches our theoretical analysis.

VII. Conclusion

In this paper, we propose a privacy preserving protocol that can compute arbitrary aggregation functions without any trusted authority. Our protocol is designed under the semi-honest model, and it can tolerate k $(k \leq n-2)$ adversarial participants and achieve $(n-k)$-source anonymity. The key of our protocol is we have devised an efficient randomized algorithm, which can securely assign unique sequence numbers to the participants without trusted authority. Compared with a recent work [1] using a trusted authority, our protocol increases merely by at most a factor of $O((\frac{\log n}{\log\log n})^2)$ in terms of computation time and communication cost. How to deploy it in practice [26] and how to design privacy preserving protocols that can compute arbitrary aggregation functions in the malicious model will be interesting in future work.

VIII. Acknowledgement

This work is supported in part by National Key Research and Development Program of China under grant No.2017YFC0803700, the National Natural Science Foundation of China Grants 61572216, 61602195, 61433019 and U1435217.

References

[1] Y. Zhang, Q. Chen, and S. Zhong, "Privacy-preserving data aggregation in mobile phone sensing," *IEEE Trans. Inf. Forensics Security*, 11(5): 980-992, 2016.

[2] Y. Yan, Y. Qian, H. Sharif, and D. Tipper, "A survey on cyber security for smart grid communications," *IEEE Commun. Surveys Tuts.*, 14(4): 998-1010, 2012.

[3] M. M. Groat, W. Hey, and S. Forrest, "Kipda: k-indistinguishable privacy-preserving data aggregation in wireless sensor networks," in *INFOCOM*, 2011.

[4] E. Shi, H. Chan, E. Rieffel, R. Chow, and D. Song, "Privacy-preserving aggregation of time-series data," in *NDSS*, 2011.

[5] J. Shi, R. Zhang, Y. Liu, and Y. Zhang, "Prisense: privacy-preserving data aggregation in people-centric urban sensing systems," in *INFOCOM*, 2010.

[6] Q. Li and G. Cao, "Efficient and privacy-preserving data aggregation in mobile sensing," in *ICNP*, 2012.

[7] Y. Zhang, Q. Chen, and S. Zhong, "Efficient and privacy-preserving min and k th min computations in mobile sensing systems," *IEEE Trans. Dependable Secure Comput.*, 14(1):9-12, 2017.

[8] J. Ni, K. Zhang, X. Lin, and X. S. Shen, "Edat: Efficient data aggregation without ttp for privacy-assured smart metering," in *ICC*, 2016.

[9] Y. Xiao, V. K. Rayi, B. Sun, X. Du, F. Hu, and M. Galloway, "A survey of key management schemes in wireless sensor networks," *Computer communications*, 30(11): 2314–2341, 2007.

[10] A. C. Yao, "Protocols for secure computations," in *FOCS*, 1982.

[11] X. Liu, S. Li, X. Chen, G. Xu, X. Zhang, and Y. Zhou, "Efficient solutions to two-party and multiparty millionaires problem," *Secur. Commun. Netw.*, 2017.

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

[13] T. Jung, X.-Y. Li, and M. Wan, "Collusion-tolerable privacy-preserving sum and product calculation without secure channel," *IEEE Trans.Dependable Secure Comput.*, 12(1): 45-57, 2015.

[14] T. Jung, X. Mao, X.-Y. Li, S.-J. Tang, W. Gong, and L. Zhang, "Privacy-preserving data aggregation without secure channel: Multivariate polynomial evaluation," in *INFOCOM*, 2013.

[15] W. Zhang, C. Wang, and T. Feng, "Gpˆ 2s: Generic privacy-preservation solutions for approximate aggregation of sensor data (concise contribution)," in *PERCOM*, 2008.

[16] M. Conti, J. Willemsen, and B. Crispo, "Providing source location privacy in wireless sensor networks: a survey," *IEEE Commun. Surveys Tuts.*, 15(3): 1238-1280, 2013.

[17] D. L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Communications of the ACM*, 24(2): 84-90, 1981.

[18] C. Cornelius, A. Kapadia, D. Kotz, D. Peebles, M. Shin, and N. Triandopoulos, "Anonysense: privacy-aware people-centric sensing," in *MOBISYS*, 2008.

[19] D. Chaum, "The dining cryptographers problem: Unconditional sender and recipient untraceability," *Journal of cryptology*, 1(2): 65-75, 1988.

[20] M. K. Reiter and A. D. Rubin, "Crowds: Anonymity for web transactions," *ACM Trans. Inf. Syst. Secur.*, 1(1): 66–92, 1998.

[21] E. G. Sirer, M. Polte, M. Robson, E. Gün, S. Milo, and P. M. Robson, "Cliquenet: A self-organizing, scalable, peer-to-peer anonymous communication substrate," [Online]. Available: http://www.cs.cornell.edu/People/egs/papers/cliquenet-iptp.pdf,2001.

[22] G. Oded, "Foundations of cryptography. basic applications, vol. 2," 2004.

[23] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Trans. Inf. Theory*, 22(6):644–654, 1976.

[24] C. Castelluccia, A. C. Chan, E. Mykletun, and G. Tsudik, "Efficient and provably secure aggregation of encrypted data in wireless sensor networks," *ACM Trans. Sensor Netw.*, 5(3): 20:1-20:36, 2009.

[25] http://grid.hust.edu.cn/qshua/infocom18full.pdf.

[26] Y. Liu, A. Liu, Y. Li, Z. Li, Y. J. Choi, H. Sekiya, and J. Li, "Apmd: A fast data transmission protocol with reliability guarantee for pervasive sensing data communication," *Perv Mob Comput*, 413-435, 2017.