Efficient Privacy-Preserving Stream Aggregation in Mobile Sensing with Low Aggregation Error

Qinghua Li and Guohong Cao

Department of Computer Science and Engineering
The Pennsylvania State University
{qxl118, gcao}@cse.psu.edu

Abstract. Aggregate statistics computed from time-series data contributed by individual mobile nodes can be very useful for many mobile sensing applications. Since the data from individual node may be privacy-sensitive, the aggregator should only learn the desired statistics without compromising the privacy of each node. To provide strong privacy guarantee, existing approaches add noise to each node's data and allow the aggregator to get a noisy sum aggregate. However, these approaches either have high computation cost, high communication overhead when nodes join and leave, or accumulate a large noise in the sum aggregate which means high aggregation error. In this paper, we propose a scheme for privacy-preserving aggregation of time-series data in presence of untrusted aggregator, which provides differential privacy for the sum aggregate. It leverages a novel ring-based interleaved grouping technique to efficiently deal with dynamic joins and leaves and achieve low aggregation error. Specifically, when a node joins or leaves, only a small number of nodes need to update their cryptographic keys. Also, the nodes only collectively add a small noise to the sum to ensure differential privacy, which is $O(1)$ with respect to the number of nodes. Based on symmetric-key cryptography, our scheme is very efficient in computation.

1 Introduction

Mobile devices such as smart phones are ubiquitous today with an ever-increasing popularity. These devices are equipped with various sensors such as camera, accelerometer, GPS, etc. Mobile sensing exploits the data contributed by mobile users (via the mobile devices they carry) to infer rich information about people (e.g., health, activity, and social event) and their surrounding (e.g., pollution and weather). Applications of mobile sensing include traffic monitoring [1], environmental monitoring [2], healthcare [3], etc.

In many scenarios, stream data from the mobile users over time can be collected, aggregated, and mined for obtaining or identifying useful patterns or statistics over a population [4]. In applications such as CarTel [5] and N-SMARTS [6], participants generate time-series data such as their location, speed, the pollution density and noise level in their surroundings, etc. These data can be aggregated to obtain the traffic pattern and pollution map. For another example, the average amount of exercise (which can be measured by motion sensors on smartphones [3]) that people do in every day can be used to infer public health conditions.

The above examples suggest that aggregate statistics computed from time-series data contributed by individual users can be very useful in mobile sensing. However, in

many cases, the data from individual users may be privacy-sensitive, and the users do not trust any single third party to see their data in cleartext. A user's data, if directly collected, can be used to infer the user's daily activities and health conditions. For instance, to monitor the propagation of a new flu, the aggregator will collect information on the number of patients infected by this flu. However, a patient may not want to directly tell the aggregator that she is infected. Thus, a challenge is to allow an untrusted aggregator to obtain the useful aggregate while preserving individual users' privacy.

Table 1. Comparison between existing schemes and our scheme.

Scheme	Total comm. per interval	Comm. model	Aggregation error	Dynamic join & leave	Comm. per join & leave	Cryptography
[7]	$O(n)$	N $\leftrightarrow$ A	$O(1)$	No	-	Public-key
Basic [8]	$O(n)$	N $\rightarrow$ A	$O(1)$	No	$O(n)$	Public-key
Binary [9]	$O(n \log n)$	N $\rightarrow$ A	$\tilde{O}((\log n)^{\frac{3}{2}})$	Yes	$O(1)$	Public-key
[10]	$O(n)$	N $\rightarrow$ A	$O(1)$	Yes	$O(1)$	Public-key
Our scheme	$O(n)$	N $\rightarrow$ A	$O(1)$	Yes	$O(d)$	Symmetric-key

$N \rightarrow A$: node-to-aggregator uni-directional. $N \leftrightarrow A$: interactive between node and aggregator. n: number of nodes. d: a parameter of our scheme (smaller than 100 in most practical settings).

The problem of privacy-preserving aggregation of time-series data in presence of untrusted aggregator has been studied in [7] and [8]. They combine differential privacy [11, 12] and cryptography techniques to provide distributed differential privacy, such that only negligible information about the node can be leaked even if the aggregator has arbitrary auxiliary information. In these schemes, each node independently adds appropriate noise to her data before aggregation, and the aggregator gets a noisy sum instead of the accurate sum. A large enough noise is accumulated in the aggregate to achieve differential privacy. These schemes also rely on a special encryption technique where each node encrypts its noisy data with a key, sends the encrypted data to the aggregator which can decrypt the sum of the nodes' noisy data without learning anything else.

In these schemes, all nodes *collectively* add a sufficient amount of noise (required for differential privacy) to the sum aggregate, and there is only $O(1)$ aggregation error (i.e., the difference between the noisy sum and the accurate sum). However, these schemes cannot efficiently support dynamic joins and leaves. For example, in [8], when a node joins or leaves, the encryption keys of all nodes are updated, which means high communication overhead in a large system. Thus, these schemes are not suitable for applications with many nodes and high churn rate. Chan *et al.* [9] propose a binary interval tree technique which can reduce the communication cost for joins and leaves, but their scheme has high aggregation error which means poor utility of the aggregate. A recent scheme [10] can efficiently support dynamic joins and leaves, but it has high computation overhead due to the use of public-key cryptography, and thus may not be appropriate for mobile sensing scenarios with resource-constrained devices, short aggregation periods, and many aggregate statistics collected simultaneously.

In this paper, we propose a new scheme which can efficiently deal with dynamic joins and leaves and achieve low aggregation error (see Table 1). Specifically, when a node joins or leaves, only a small number of nodes need to update their encryption keys, which is in the order of tens in most practical settings irrespective of the total number of nodes. Also, the nodes collectively add $O(1)$ noise to the aggregate for differential

privacy. Our main contribution is a new technique – interleaved grouping. We propose a novel ring-based interleaved grouping construction, which divides nodes into groups of smaller size such that (1) at most three (four) groups of nodes need to be updated for each join (leave) and (2) the aggregator can only learn the sum of all nodes' data but nothing else. The scheme is very efficient in computation since it is based on HMAC. Implementation-based measurements show that our scheme is two orders of magnitude faster than existing schemes in encryption and/or decryption.

The remainder of this paper is organized as follows. Section 2 discusses related work. Section 3 presents the system overview and models. Section 4 and 5 describe the basic idea of interleaved grouping and a ring-based construction. Section 6 presents an aggregation protocol based on ring-based interleaved grouping. Section 7 presents evaluation results. Section 8 discusses extensions. The last section concludes this paper.

2 Related Work

Security and privacy in mobile sensing systems have been addressed by many works (e.g., [13–16]), but they do not consider aggregation of data. There are many existing works (e.g., [17]) on privacy-preserving data aggregation, but most of them assume a trusted aggregator. Shi et al. [18] proposed an aggregation scheme for mobile sensing, but the scheme does not consider time-series data. The two constructions [19, 20] based on additive homomorphic encryption [17] do not guarantee differential privacy.

Recent works [7–9] address differentially private aggregation of time-series data. Rastogi and Nath [7] designed an encryption scheme based on threshold Paillier cryptosystem, but their construction requires an extra round of interaction between the aggregator and the nodes in every aggregation period. Shi et al. [8] proposed a Diffie-Hellman-based encryption scheme, where no communication is required from the aggregator to the nodes. However, their scheme redistributes encryption keys to all nodes when a node joins or leaves, inducing high communication cost. To deal with dynamic joins and leaves and provide fault tolerance, Chan et al. [9] extend the construction in [8] with a binary interval tree technique which reduces expensive rekeying operations. However, in their scheme, since each node's data is aggregated into multiple sums, a large noise is added to each node's data to provide differential privacy, which leads to high aggregation error. Recent designs [21] employ an honest-but-curious proxy server to tolerate the churn of a small fraction of nodes, but it is unknown how they work when many nodes leave. Jawurek et al. [10] proposed a fault-tolerant aggregation scheme, but it employs Paillier cryptosystem which is expensive in computation.

Grouping has been recently used for differentially-private publication of graph topologies [22, 23]. These solutions divide a dataset into *disjoint* groups, which is different from our *interleaved* grouping technique.

3 Overview

3.1 Problem Definition

Our system model is shown in Figure 1. An aggregator wants to get the sum aggregate of n mobile nodes periodically. Let $x_i^{(t)}$ ($x_i^{(t)} \in \{0, 1, ..., \Delta\}$) denote the data of node i in aggregation period t ($t = 1, 2, 3, ...$). Then the sum aggregate for time period t is

$\sum_{i=1}^{n} x_i^{(t)}$. Since the accurate sum may leak node privacy in presence of side information [11, 12], the aggregator is only allowed to obtain a noisy sum (i.e., the accurate sum plus some noise). In each time period t, each node i adds noise $r_i^{(t)}$ to her data $x_i^{(t)}$, encrypts the noisy data $\hat{x}_i^{(t)} = x_i^{(t)} + r_i^{(t)}$ with her key $k_i^{(t)}$ and sends the ciphertext to the aggregator. The aggregator uses the capability $k_0^{(t)}$ to decrypt the noisy sum $\sum_{i=1}^{n}(x_i^{(t)} + r_i^{(t)})$. Here, $k_i^{(t)}$ and $k_0^{(t)}$ change in every time period. In the following, when we describe our solution, we usually focus on the aggregation scheme in one time period. For simplicity, we omit the superscript t and write x_i, r_i, k_i and k_0 instead.

Each mobile node communicates with the aggregator via 3G, WiFi, or other available access networks. Source anonymity [24] is not necessary since data content is protected. Peer-to-peer communication among the nodes is not required, because nodes may not know each other for privacy reasons and such communication is nontrivial due to the mobility of nodes in mobile sensing. We assume that time is synchronized among nodes. In mobile sensing, mobile devices (e.g., smartphones) usually have embedded GPS receivers, which can easily synchronize time without communications among them.

There are three requirements regarding privacy. First, the aggregator only learns the noisy sum but nothing else (e.g., intermediate results). Second, a party without the aggregator capability learns nothing. This is *aggregator obliviousness* [8]. The third requirement is *differential privacy*. Intuitively, the sum obtained by the aggregator is roughly the same no matter if a specific node is in the system or not.

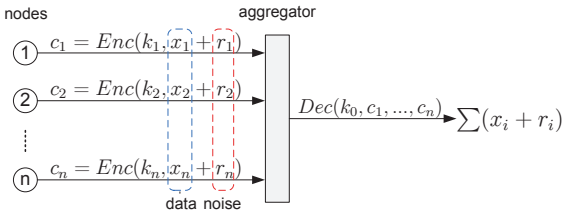


Fig. 1. An overview of our system.

3.2 Trust Model

The aggregator is untrusted. A number of nodes may collude with the aggregator and reveal their data and noise values. We refer to these nodes as *compromised nodes* and refer to others as *good nodes*. We assume that the fraction of compromised nodes that collude is at most γ, and nodes are equally likely to collude. Similar to [8], we assume that the system has an a priori estimate over the upper bound of γ, and uses it in our protocol. The aggregator may eavesdrop all messages sent to/from every node. Also, all entities are computationally bounded.

We also assume a key dealer which issues keys to the nodes and the aggregator via a secure channel. For now, we assume that the key dealer is trusted, and we relax this assumption in Section 8. Malicious nodes may perform data pollution attacks in which they lie about their data values in order to change the aggregate. Data pollution attacks are outside the scope of this paper, and their influence can be bounded if each node uses a non-interactive zero-knowledge proof to prove that its data is in a valid range.

3.3 Basic Scheme

Let us first look at a simple basic scheme, which is a variant of the scheme proposed in [8]. Basically, it uses the same data perturbation algorithm as in [8]. However, since the encryption method of [8] is not efficient in computation (especially for the aggregator to get the sum), it replaces the encryption method with the construction in [20] which also achieves aggregator obliviousness but has much less computation overhead.

Encryption method *Setup*: The key dealer generates a set $\mathcal{S}$ of nc random secrets s_1, ..., s_{nc}. It divides them into n random disjoint subsets $\mathcal{S}_1$, ..., $\mathcal{S}_n$, with c secrets in each subset. Clearly, $\mathcal{S} = \bigcup_{i=1}^{n} \mathcal{S}_i$. The key dealer randomly selects a subset $\hat{\mathcal{S}}$ of q secrets and assigns them to the aggregator. It then evenly divides $\mathcal{S} - \hat{\mathcal{S}}$ into n random disjoint subsets $\bar{\mathcal{S}}_1$, ..., $\bar{\mathcal{S}}_n$. Clearly, $\mathcal{S} = (\bigcup_{i=1}^{n} \bar{\mathcal{S}}_i) \bigcup \hat{\mathcal{S}}$. It assigns $\mathcal{S}_i$ and $\bar{\mathcal{S}}_i$ to node i.
Encryption: In time period $t \in \mathbb{N}$, node i generates key $k_i = (\sum_{s' \in \mathcal{S}_i} h(f_{s'}(t)) - \sum_{s' \in \bar{\mathcal{S}}_i} h(f_{s'}(t))) \mod M$, where $M = 2^{\lceil \log_2 (n\Delta) \rceil}$. Here, $f_{s'}$ is a member of the pseudorandom function (PRF) family $\mathbb{F}_\lambda = \{f_{s'} : \{0,1\}^\lambda \rightarrow \{0,1\}^\lambda\}_{s' \in \{0,1\}^\lambda}$ indexed by s'. As suggested in [17, 20], it can be implemented with HMAC, where $f_{s'}(t)$ is the HMAC of t with s' as the key. Function h maps the output of $f_{s'}$ to a uniform random value in $[0, M-1]$. A simple construction for h is to truncate the output of $f_{s'}$ into shorter bit strings of length $\log_2 M$ and use the exclusive-OR of all these strings as the output. Node i encrypts its noisy data $\hat{x}_i$ by computing $c_i = (k_i + \hat{x}_i) \mod M$.
Decryption: In time period $t \in \mathbb{N}$, the aggregator generates key $k_0 = (\sum_{s' \in \hat{\mathcal{S}}} h(f_{s'}(t))) \mod M$ and decrypts the noisy sum $\hat{S} = \sum_{i=1}^{n} \hat{x}_i$ by computing $\hat{S} = (\sum_{i=1}^{n} c_i - k_0) \mod M$. It is easy to verify that $k_0 = (\sum_{i=1}^{n} k_i) \mod M$. Hence, the aggregator can get the correct noisy sum. Also, since only the trusted key dealer and the aggregator know the secrets used by the aggregator, no other party can learn the sum.

By assigning a large-enough number of secrets to each node and the aggregator (i.e., c and q are large enough), this method ensures that it is computationally infeasible for the adversary to guess the secrets assigned to a particular node or the aggregator. More formally, for l-bit security, the probability of a successful guess is smaller than 2^{-l}. Table 2 shows the values of c and q for 80-bit security when $\gamma = 0.2$.

Table 2. The values of c and q for 80-bit security [20]

n	10^3	10^4	10^5	10^6
c	5	4	3	3
q	8	6	5	4

As a common practice of security, the secret used to calculate HMAC cannot be repeatedly used forever and should be updated after being used for a certain length of time (e.g., one year). The proper length of time can be determined following the guidelines discussed in [25]. After this period of time (which is usually long), the key dealer needs to rerun the setup phase and update the secrets.

The basic scheme is very efficient in computation due to the use of HMAC. We note that our interleaved grouping technique (see later) can also be applied upon other aggregator oblivious encryption schemes (e.g., [8, 19]) with different tradeoffs.

Data Perturbation Shi et al [8] show that differential privacy can be achieved by adding a noise that follows diluted geometric distribution to each node's data.

Definition 1. *Geometric Distribution. Let* $\alpha > 1$. *Geom*(α) *denotes the symmetric geometric distribution with parameter* α. *Its probability mass function at* k *(*$k = 0, \pm 1, \pm 2, ...$*) is* $\frac{\alpha-1}{\alpha+1} \cdot \alpha^{-|k|}$.

Definition 2. *Diluted Geometric Distribution. Let* $\alpha > 1$ *and* $0 < \beta \leq 1$. *A random variable follows* β*-diluted Geometric distribution Geom*$^{\beta}(\alpha)$ *if it is sampled from Geom*(α) *with probability* β*, and is set to 0 with probability* $1 - \beta$.

In time period t, node i generates a noise r_i from Geom$^{\beta}(\alpha)$ and computes $\hat{x}_i = x_i + r_i$. Parameters α and β are set as $\alpha = e^{\frac{\epsilon}{\Delta}}$ and $\beta = \min(\frac{1}{(1-\gamma)n} \ln \frac{1}{\delta}, 1)$, where ϵ and δ are privacy parameters. Given that the encryption method is aggregator oblivious (i.e., the aggregator only learns the noisy sum but nothing else), the data perturbation procedure achieves (ϵ,δ)-differential privacy [8]:

Theorem 1. *Let* $0 < \delta < 1$, $\epsilon > 0$, $\alpha = e^{\frac{\epsilon}{\Delta}}$ *and* $\beta = min(\frac{1}{(1-\gamma)n} \ln \frac{1}{\delta}, 1)$, *where* γ *is the maximum fraction of nodes compromised. If each node adds noise Geom*$^{\beta}(\alpha)$, *the above perturbation procedure achieves (*ϵ,δ*)-distributed differential privacy*[1].

With this data perturbation method, roughly one copy of geometric noise Geom(α) is added to the sum, which is required to ensure ϵ-differential privacy [26]. Strictly speaking, the basic scheme achieves differential privacy against polynomial-time adversaries, since the encryption method is secure against polynomial-time adversaries.

Problem with the basic scheme. In the basic encryption method [20], when a node joins or leaves, the key dealer must issue a new set of secrets to every node and the aggregator to ensure security. This induces high communication overhead and makes it impractical for large-scale mobile sensing applications with high churn rate. Thus, efficient techniques should be proposed to deal with dynamic joins and leaves.

3.4 Naive Grouping

Intuitively, we can apply grouping on top of the basic scheme to reduce the communication cost of dynamic joins and leaves.

Naive Grouping. The n nodes are divided into g *disjoint groups* of equal size, and the basic scheme is applied to each group independently. The aggregator has the capability to decrypt the sum of each group. To provide differential privacy, one copy of geometric noise is added to the sum of each group. As some positive and negative noises cancel out, the accumulated noise in the final aggregate is $O(\sqrt{g})$ with high probability. When a node joins or leaves a group, only the $\frac{n}{g}$ nodes in this group are redistributed secrets.

The problem with Naive Grouping is that it cannot achieve both low communication cost for dynamic joins and leaves and low aggregation error, since these two goals require the parameter g to be tuned in reverse directions. Thus, it is nontrivial to use grouping to achieve both churn resilience and low aggregation error.

Table 3 summarizes the notations used in this paper.

4 Interleaved Grouping

Although grouping can be used to reduce the communication overhead of dynamic joins and leaves, the naive grouping scheme has high aggregation error since it adds

[1] If each node is compromised independently with probability γ, the data perturbation procedure is proved to achieve (ϵ,δ)-computational differential privacy [9].

Table 3. Notations

n	Num. of nodes	g	Num. of groups in Naive Grouping/our scheme
γ	Max. fraction of compromised nodes	α, β	Paras. used to generate noise
x_i	Data of node i	d, x	Parameters of our proposed scheme
r_i	The noise that node i adds to her data	k_0	The capability used by the aggregator to decrypt noisy sum
$\hat{x}_i$	Noisy data of node i, $\hat{x}_i = x_i + r_i$		
Δ	Each node's data is from $\{0, 1, ..., \Delta\}$	k_i	The encryption key used by node i
ϵ, δ	Parameters of differential privacy	l	The required security level is l-bit, e.g., $l = 80$

one independent copy of geometric noise to each group. To address this problem, we propose to use a new technique called *interleaved grouping*.

4.1 Basic Idea

The basic idea is to divide the nodes into *interleaved groups*, where each group shares some nodes with other groups. In the setup phase, an independent set of secrets are assigned to each group of nodes and the aggregator similarly as that in the basic scheme. For each group, the aggregator does not know the secrets assigned to each member of the group. A node that belongs to multiple groups will receive secrets from each group it belongs to. Each node (the aggregator) uses the union of the secrets that it receives to derive its encryption key (decryption capability). The encryption (decryption) process is the same as the basic scheme. Clearly the aggregator can decrypt the sum.

Interleaved grouping guarantees that *the aggregator cannot learn the sum of any individual group or the sum of any subset of (not all) nodes* (see the example below), and thus the nodes can collectively add just one copy of geometric noise to the aggregate, minimizing the aggregation error. When a node joins or leaves a group, the key dealer runs the setup phase again for this group only. Thus the communication cost is low.

We use the example in Figure 2 to show how interleaved grouping provides that guarantee. Seven nodes are divided into three interleaved groups, where group $\mathcal{G}_1$ shares node A with $\mathcal{G}_2$ and shares C with $\mathcal{G}_3$. The secrets assigned to each node and the aggregator are shown in the figure. Suppose the aggregator tries to get the sum of group $\mathcal{G}_2$, i.e., $\hat{x}_A + \hat{x}_D + \hat{x}_E$. From A's, D's and E's ciphertexts $\hat{x}_A + h(f_{s_1}(t)) + h(f_{s_4}(t))$, $\hat{x}_D + h(f_{s_5}(t)) - h(f_{s_4}(t))$ and $\hat{x}_E + h(f_{s_6}(t))$, it sums them and gets $(\hat{x}_A + \hat{x}_D + \hat{x}_E) + h(f_{s_1}(t)) + h(f_{s_5}(t)) + h(f_{s_6}(t))$. Although it knows $\hat{\mathcal{S}}_{Agg} = \{s_5, s_6\}$ and can get $h(f_{s_5}(t)) + h(f_{s_6}(t))$, it does not know s_1 (i.e., the secrets assigned to A in group $\mathcal{G}_1$) and hence cannot get the sum of $\mathcal{G}_2$. Similarly, it cannot get the sum of $\mathcal{G}_1$, $\mathcal{G}_3$ or any other strict and non-empty subset of nodes.

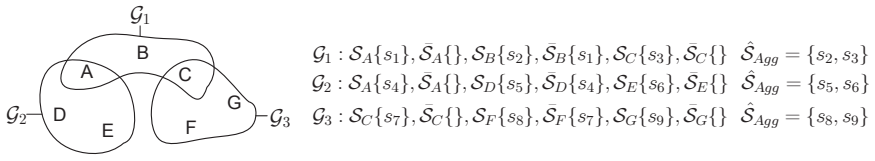


Fig. 2. The basic idea of interleaved grouping. In this example, A is assigned secrets from both $\mathcal{G}_1$ and $\mathcal{G}_2$. A sets $k_A = h(f_{s_1}(t)) + h(f_{s_4}(t))$. B only receives secrets from group $\mathcal{G}_1$, and it sets $k_B = h(f_{s_2}(t)) - h(f_{s_1}(t))$. Other nodes set their keys similarly. The aggregator sets $k_0 = \sum_{i=\{2,3,5,6,8,9\}} h(f_{s_i}(t))$. The aggregator can only get the sum of *all nodes*.

4.2 Security Condition

When nodes may be compromised, the following security condition should be satisfied.

Condition 1: *Let $\mathbb{S}$ denote an arbitrary strict and non-empty subset of groups. There exists a good node N, group $\mathcal{G} \in \mathbb{S}$ and group $\mathcal{G}' \notin \mathbb{S}$, such that $N \in \mathcal{G}$ and $N \in \mathcal{G}'$.*

The following theorem (see proof in the technical report [27]) explains why Condition 1 is needed.

Theorem 2. *In interleaved grouping, the aggregator cannot obtain the sum of any strict and nonempty subset of good nodes if and only if Condition 1 is satisfied.*

According to Theorem 2, any specific interleaved grouping scheme needs to and only needs to satisfy Condition 1. Thus, Condition 1 can be used to guide the construction of interleaved grouping schemes. In the next section, we present such a scheme based on a ring structure.

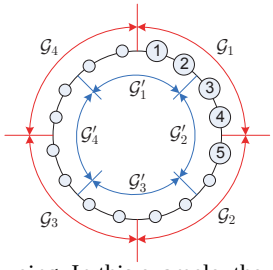


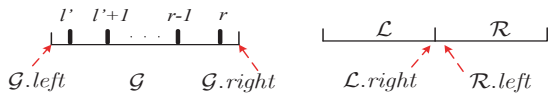
Fig. 3. Ring-based interleaved grouping. In this example, the nodes form eight groups, four disjoint groups in the outer ring ($\mathcal{G}_1$–$\mathcal{G}_4$) and four disjoint groups in the inner ring ($\mathcal{G}'_1$–$\mathcal{G}'_4$). Groups on different rings may overlap. Group $\mathcal{G}_1$ and $\mathcal{G}'_1$ overlap, and they share node 1 and 2.

5 Ring-based Interleaved Grouping

For interleaved grouping, it is nontrivial to satisfy Condition 1. With a total of g groups, there are about 2^g possible subsets of groups. Unless g is very small, it is infeasible to adjust each possible subset to satisfy the condition. In this section, we present a novel interleaved grouping construction which addresses this challenge with a ring structure.

5.1 Ring-based Group Structure

As shown in Figure 3, the nodes are arranged into a *node ring*, which is mirrored to two virtual rings, the *outer ring* and *inner ring*. Each virtual ring is partitioned into segments, and the nodes in a segment form a group. Each node belongs to two groups, one in each virtual ring. Groups in the same virtual ring are disjoint, but groups in different virtual rings may overlap (i.e., they share at least one node). A group in the inner (outer) ring will overlap with one or more groups in the outer (inner) ring.



(a) Group with nodes l', ..., r (b) Two neighbor groups

Fig. 4. Segment representation of groups.

The Basic Structure Since the node ring and the two virtual rings are identical, we use "the ring" to refer to the structure when the context is clear. Suppose there are n ($n \geq 2d$) nodes in the ring indexed from 0 to $n-1$ in the clockwise order. For convenience, we use a directed segment to represent a group, say $\mathcal{G}$ (see Fig. 4(a)). The direction from left to right corresponds to the clockwise order in the ring. Let l' and r denote the indexes of the leftmost and rightmost node in $\mathcal{G}$. Then the left and right boundary of $\mathcal{G}$ are defined as $\mathcal{G}.left = l' - 0.5$ and $\mathcal{G}.right = r + 0.5$, respectively. Let $|\mathcal{G}|$ denote the number of nodes in $\mathcal{G}$. We have $|\mathcal{G}| = (\mathcal{G}.right - \mathcal{G}.left) \mod n$.

Suppose $\mathcal{L}$ and $\mathcal{R}$ are two neighbor groups in the same virtual ring, and $\mathcal{L}.right = \mathcal{R}.left$ (see Fig. 4(b)). $\mathcal{L}.right$ (or $\mathcal{R}.left$) is the border between $\mathcal{L}$ and $\mathcal{R}$. Term "move $\mathcal{L}.right$ to the right by y" means that $\mathcal{L}.right$ and $\mathcal{R}.left$ are increased by $y \mod n$, i.e., the leftmost y nodes of $\mathcal{R}$ are moved to $\mathcal{L}$. Term "move $\mathcal{R}.left$ to the left by y" is interpreted similarly.

Overlap patterns Two groups in different virtual rings can overlap in two patterns (see Figure 5). In *Pattern I*, one group is a strict subset of the other group. In *Pattern II*, the two groups have nodes in common, but each group also has some nodes that the other group does not have.

5.2 Properties

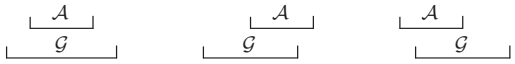
To satisfy Condition 1 and reduce the communication cost of dealing with dynamic joins and leaves, the ring-based group structure has the following properties:

Overlap Property: Any two groups that overlap share at least x nodes. x is large enough to make it infeasible (i.e., with probability smaller than 2^{-l}) that none of the shared nodes is good (see later).

Interleave Property: If two neighboring nodes in the ring belong to two neighboring groups in one virtual ring, they belong to the same group in the other virtual ring.

Group Size Property: Each group has d ($d > 2x$) to $2d-1$ nodes.

An example of the interleave property is shown in Figure 3. Two neighboring nodes 4 and 5 belong to neighboring groups $\mathcal{G}_1$ and $\mathcal{G}_2$ in the outer ring, and they are in the same group $\mathcal{G}'_2$ in the inner ring. The group size property is required by our group adjustment algorithms.



(a) Pattern I (b) Pattern II (c) Pattern II

Fig. 5. The two patterns that two groups $\mathcal{G}$ and $\mathcal{A}$ may overlap where $|\mathcal{G}| \geq |\mathcal{A}|$. $\mathcal{G}$ and $\mathcal{A}$ are in different virtual rings. In (b) and (c), $\mathcal{G}$ overlaps with the left and right part of $\mathcal{A}$, respectively.

Theorem 3. *In ring-based interleaved grouping, the overlap and interleave property ensure that Condition 1 is satisfied.*

Proof. See the technical report [27].

Comments: Due to Theorem 2 and 3, *the ring-based construction guarantees that the aggregator cannot obtain the sum of any strict and nonempty subset of good nodes*. For convenience, we say the ring-based grouping is *secure* if the overlap, interleave and group size property hold.

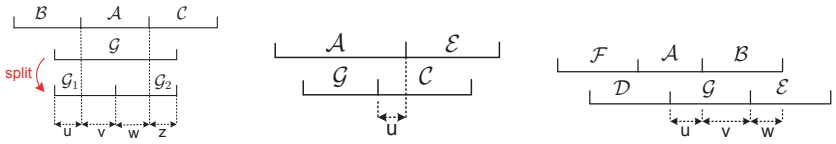
Practical Considerations In practice, the value of x should be sufficiently large such that for any two groups that overlap, with a high probability (denoted by p_s) at least one of their shared nodes is good. Since each node can be compromised with the same probability, with standard combinatorial techniques, we can derive that $p_s = 1 - \binom{\gamma n}{x} / \binom{n}{x}$. When n is large, $p_s \gtrsim 1 - \gamma^x$. Parameter x can be set as the minimum value that satisfies $p_s > 1 - 2^{-l}$:

$$x = \lceil -l \log_\gamma 2 \rceil. \tag{1}$$

To minimize the communication cost of dynamic joins and leaves (see later), parameter d can be set as the minimum value that satisfies $d > 2x$, i.e., $d = 2x + 1$. Table 4 shows the values of x and d for 80-bit security.

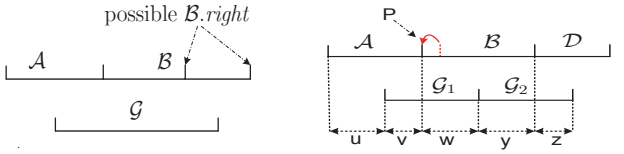
Table 4. The values of x and d for 80-bit security.

γ	0	0.01	0.05	0.1	0.15	0.2
x	1	13	19	25	30	35
d	3	27	39	51	61	71



(a) Regrouping in Alg. 2 (b) Regrouping in Alg. 4 (c) Regrouping in Alg. 5

Fig. 6. The regrouping in Alg. 2, 4 and 5.



(a) Groups before join

(b) Before join, $\mathcal{B}.right < \mathcal{G}.right$.

(c) Before join, $\mathcal{B}.right > \mathcal{G}.right$. After splitting $\mathcal{G}$ and moving $\mathcal{A}.right$, $|\mathcal{B}| < 2d$.

(d) Before join, $\mathcal{B}.right > \mathcal{G}.right$. After splitting $\mathcal{G}$ and moving $\mathcal{A}.right$, $|\mathcal{B}| = 2d$.

Fig. 7. The regrouping (in Alg. 3) when a node joins $\mathcal{G}$ and $\mathcal{A}$ which overlap in Pattern II.

5.3 Group Management

Suppose initially there are n nodes. Algorithm 1 initializes them into $\frac{2n}{d}$ groups. Figure 3 shows an instance of initialized groups for $n = 16$ and $d = 4$. Clearly, each group overlaps with two other groups and shares $\frac{d}{2}$ nodes with each overlapping group. It is easy to verify that the grouping is secure.

Algorithm 1 $\mathbf{initGroup}()$: Group initialization.

Require: Nodes 0,...,$n-1$. Without loss of generality, n is divisible by d.
Ensure: $\frac{2n}{d}$ groups $\mathcal{G}_1$, ..., $\mathcal{G}_{\frac{n}{d}}$, $\mathcal{G}'_1$, ..., $\mathcal{G}'_{\frac{n}{d}}$, each of size d.

1: **for** i from 1 **to** $\frac{n}{d}$ **do**
2: Add nodes $(i-1)\frac{n}{d}$, $(i-1)\frac{n}{d}+1$, ..., $(i-1)\frac{n}{d}+d-1$ to group $\mathcal{G}_i$
3: Add nodes $(i-1)\frac{n}{d}+\frac{d}{2} \mod n$, $(i-1)\frac{n}{d}+\frac{d}{2}+1 \mod n$, ..., $(i-1)\frac{n}{d}+\frac{d}{2}+d-1 \mod n$ to group $\mathcal{G}'_i$

When a new node joins, it is inserted to a random location in the ring, and added to the two groups that cover the location of insertion. The two changed groups may violate the group size property (i.e., having more than $2d-1$ nodes). Similarly, when a node leaves, it is removed from the ring and from the two groups it belonged to, which may violate the group size property and overlap property. In these cases, the nodes should be regrouped so that the three properties still hold.

Algorithm 2-5 show the group adjustment algorithms. Suppose the grouping is secure before the node joins or leaves. Lemma 1-4 show that these algorithms make the grouping secure. Their proofs can be found in the technical report [27].

Lemma 1(2): *Suppose a node joins two groups which overlap in Pattern I (II). After Algorithm 2 (3) is run, the grouping is secure.*

Lemma 3(4): *Suppose a node leaves two groups which overlap in Pattern I (II). After Algorithm 4 (5) is run, the grouping is secure.*

Algorithm 2 $\mathbf{adjust}(\text{JOIN}, \text{I})$: Re-grouping after a node joins two groups which overlap in pattern I (see Fig. 6(a)).

Require: $\mathcal{G}$, $\mathcal{A}$: The two groups that the node joins, $|\mathcal{G}| > |\mathcal{A}|$.

1: **if** $|\mathcal{G}| < 2d$ **then return**;
2: **else** Split $\mathcal{G}$ in the middle into two groups, each of size d;

Algorithm 3 $\mathbf{adjust}(\text{JOIN}, \text{II})$: Re-grouping after a node joins two groups which overlap in pattern II (see Fig. 7(a)).

Require: $\mathcal{G}$, $\mathcal{A}$: the two groups that the node joins, $|\mathcal{G}| \geq |\mathcal{A}|$.
Require: $\mathcal{B}$: $\mathcal{A}$'s neighbor group which also overlaps with $\mathcal{G}$.
Require: Without loss of generality, $\mathcal{G}$ overlaps with the right part of $\mathcal{A}$, i.e., $\mathcal{G}.left > \mathcal{A}.left$ and $\mathcal{G}.right > \mathcal{A}.right$. In this case, $\mathcal{B}$ is the right neighbor of $\mathcal{A}$. See Figure 7(a).

1: **if** $|\mathcal{G}| < 2d$ **then**
2: **return**;
3: **else**
4: Split $\mathcal{G}$ in the middle into two groups, each of size d;
5: Move $\mathcal{A}.right$ to the position $P = max\{\mathcal{G}.left + x, \mathcal{A}.left + d\}$;
6: **if** $|\mathcal{B}| < 2d$ **then**
7: **return**;
8: **else**
9: Create a group $\mathcal{C}$, with $\mathcal{C}.left = \mathcal{A}.right$ and $\mathcal{C}.right = \mathcal{C}.left + d$;
10: $\mathcal{B}.left \leftarrow \mathcal{C}.right$;

Algorithm 4 $\mathbf{adjust}(\text{LEAVE}, \text{I})$: Re-grouping after a node leaves two groups which overlap in pattern I (see Fig. 6(b)).

Require: $\mathcal{G}$, $\mathcal{A}$: the two groups that the node leaves, with $|\mathcal{G}| < |\mathcal{A}|$.
Require: $\mathcal{C}$, $\mathcal{E}$: the right neighbor of $\mathcal{G}$ and $\mathcal{A}$, respectively. See Figure 6(b).

if $|\mathcal{G}| \geq d$ **then**
  return;
else if $|\mathcal{C}| = d$ **then**
  Merge $\mathcal{G}$ and $\mathcal{C}$;
else
  $u \leftarrow |\mathcal{C} \bigcap \mathcal{A}|$;
  if $u \geq x + 1$ **then** Move $\mathcal{G}$.*right* to the right by 1;
  else if $u = x$ **and** $|\mathcal{C}| \geq d + 2x$ **then** Move $\mathcal{G}$.*right* to the right by $2x$;
  else if $u = x$ **and** $|\mathcal{C}| < d + 2x$ **then** Move both $\mathcal{G}$.*right* and $\mathcal{A}$.*right* to the right by 1;

Algorithm 5 $\mathbf{adjust}(\text{LEAVE}, \text{II})$: Re-grouping after a node leaves two groups which overlap in pattern II (see Fig. 6(c)).

Require: $\mathcal{G}$, $\mathcal{A}$: the two groups that the node leaves.
Require: $\mathcal{D}$, $\mathcal{E}$: $\mathcal{G}$'s neighbors. $\mathcal{D}$ overlaps with $\mathcal{A}$; $\mathcal{E}$ does not.
Require: $\mathcal{B}$, $\mathcal{F}$: $\mathcal{A}$'s neighbors. $\mathcal{B}$ overlaps with $\mathcal{G}$; $\mathcal{F}$ does not.
Require: Without loss of generality, suppose $\mathcal{G}$.*left* $> \mathcal{A}$.*left* and $\mathcal{G}$.*right* $> \mathcal{A}$.*right*.

if $|\mathcal{G} \bigcap \mathcal{A}| \geq x$ **then**
  if $|\mathcal{G}| \geq d$ **and** $|\mathcal{A}| \geq d$ **then**
    return;
  if $|\mathcal{G}| = d - 1$ **then**
    if $|\mathcal{E}| = d$ **then** Merge $\mathcal{G}$ and $\mathcal{E}$;
    else Move $\mathcal{G}$.*right* to the right by 1;
  if $|\mathcal{A}| = d - 1$ **then**
    if $|\mathcal{F}| = d$ **then** Merge $\mathcal{A}$ and $\mathcal{F}$;
    else Move $\mathcal{A}$.*left* to the left by 1;
else
  if $|\mathcal{G}| \geq d$ **and** $|\mathcal{A}| \geq d$ **then**
    if $|\mathcal{B}| \geq d + 1$ **then** Move $\mathcal{A}$.*right* to the right by 1;
    else if $|\mathcal{D}| \geq d + 1$ **then** Move $\mathcal{G}$.*left* to the left by 1;
    else Move $\mathcal{D}$.*right* to the right by $2x - 1$;
  if $|\mathcal{G}| = d - 1$ **then**
    if $|\mathcal{D}| = d$ **then** Merge $\mathcal{G}$ and $\mathcal{D}$;
    else Move $\mathcal{G}$.*left* to the left by 1;
  if $|\mathcal{A}| = d - 1$ **then**
    if $|\mathcal{B}| = d$ **then** Merge $\mathcal{A}$ and $\mathcal{B}$;
    else Move $\mathcal{A}$.*right* to the right by 1;

5.4 Communication Cost Analysis

We measure the communication cost of dynamic joins and leaves by the number of nodes that the key dealer should reissue secrets to (*number of updated nodes* for short). The communication cost of ring-based interleaved grouping is given in the following theorem (see proof in the technical report [27]).

Theorem 4. *In ring-based interleaved grouping, when a node joins (leaves), at most three (four) existing groups are updated and the number of updated nodes has an upper bound* $4d$ *(*$6d$*).*

This theorem indicates that the communication cost of ring-based interleaved grouping depends on parameter d, which in turn depends on parameter γ (see Table 4), the maximum fraction of compromised nodes. According to Table 4, the value of d is not large when γ is not too high. Thus, the communication cost is low. Note that the cost does not change with the number of nodes in the system.

6 Applying Ring-based Interleaved Grouping to Data Aggregation

In this section, we propose a scheme to demonstrate how to apply the ring-based interleaved grouping technique to privacy-preserving data aggregation to achieve both churn resilience and low aggregation error.

6.1 Encryption Scheme

Setup: The key dealer divides the nodes into groups using Alg. 1, and independently assigns secrets for each group as done in the basic scheme. Each node i gets secrets from the two groups that it is in. Let $\mathcal{S}_{i1}$ and $\bar{\mathcal{S}}_{i1}$ denote the sets of secrets received from one group, and $\mathcal{S}_{i2}$ and $\bar{\mathcal{S}}_{i2}$ denote the sets of secrets received from the other group. The node merges the secrets as follows: $\mathcal{S}_i = \mathcal{S}_{i1} \bigcup \mathcal{S}_{i2}$ and $\bar{\mathcal{S}}_i = \bar{\mathcal{S}}_{i1} \bigcup \bar{\mathcal{S}}_{i2}$. Let g denote the number of groups generated by Alg. 1. The aggregator obtains $\hat{\mathcal{S}}_1$, ..., $\hat{\mathcal{S}}_g$, where each $\hat{\mathcal{S}}_j$ is the decryption capability of one group. It sets the overall decryption capability as $\hat{\mathcal{S}} = \bigcup_{i=j}^{g} \hat{\mathcal{S}}_j$.
Encryption: The same as in the basic scheme.
Decryption: The same as in the basic scheme.

Due to the guarantee provided by ring-based interleaved grouping, the aggregator can only decrypt the noisy sum but nothing else.

6.2 Data Perturbation

The perturbation algorithm is the same as that in the basic scheme, except that parameter β is set differently (see below).

6.3 Dealing with Dynamic Joins and Leaves

When a node joins or leaves, the key dealer runs Algorithm 2-5 to adjust grouping. For each adjusted group, it reruns the setup phase to distribute a new set of secrets to the nodes of the group and to the aggregator. In this process, it only communicates with the nodes in the adjusted groups, which constitute only a small portion of all nodes.

Algorithm 6 Procedures run by the key dealer to manage the values of u for nodes.

Require: n: the real number of nodes
Require: u_i: the number of nodes that node i uses to set parameter β
Initialization:
if n is even **then** $u_1, u_2, ..., u_n \leftarrow \lfloor \frac{n}{2} \rfloor + 1, \lfloor \frac{n}{2} \rfloor + 1, \lfloor \frac{n}{2} \rfloor + 2, \lfloor \frac{n}{2} \rfloor + 2, ..., n, n$;
else $u_1, u_2, ..., u_n \leftarrow \lfloor \frac{n}{2} \rfloor + 1, \lfloor \frac{n}{2} \rfloor + 2, \lfloor \frac{n}{2} \rfloor + 2, ..., n, n$;

Join:
if Node i joins **then**
    $n \leftarrow n + 1$;
    $u_i \leftarrow n$;
    Find a node j with $u_j = \min\{u_1, u_2, ..., u_n\}$;
    $u_j \leftarrow n$;

Leave:
if Node i leaves **then**
    $n \leftarrow n - 1$;
    Find a node j with $u_j = \max\{u_1, u_2, ..., u_n\}$;
    if There exists another node m with $u_m = u_j$ **then** $u_m \leftarrow u_i$;
    $u_j \leftarrow \lfloor \frac{n}{2} \rfloor + 1$;

In the data perturbation part of the basic scheme, each node uses the number of nodes n to set parameter β (i.e., $\beta = \min(\frac{1}{(1-\gamma)n} \ln \frac{1}{\delta}, 1)$), such that all nodes collectively add just one copy of geometric noise to the aggregate. However, it requires communications with all nodes upon every join and leave to send the exact value of n to them. Obviously, it nullifies the benefits of grouping in reducing the communication cost. To address this issue, we relax the accuracy requirement on the value of n such that n does not have to be updated to every node upon every join and leave. As a tradeoff, a little more noise is added compared to the basic scheme.

Specifically, each node records the value of n according to its knowledge. Let u denote the value recorded by the node. u may not always reflect n, which is the real number of nodes. Each node uses u to set parameter β for data perturbation, i.e., $\beta = \min(\frac{1}{(1-\gamma)u} \ln \frac{1}{\delta}, 1)$. The smaller u is, the more noise is added. u should not be larger than n to ensure that at least one copy of geometric noise is added to provide differential privacy, but u cannot be too small to avoid too much aggregation error. Thus, the values of u at the nodes should be updated appropriately upon dynamic joins or leaves. The challenge is how to update u without incurring too much communication cost.

We propose Alg. 6 to address this challenge. Table 5 shows a running example of it.

Table 5. An example of Algorithm 6

step	operation	u_1	u_2	u_3	u_4	u_5	u_6	n
0	initialize	3	3	4	4	-	-	4
1	Node 5 joins	3	5	4	4	5	-	5
2	Node 6 joins	6	5	4	4	5	6	6
3	Node 2 leaves	5	-	4	4	5	3	5
4	Node 1 leaves	-	-	4	4	3	3	4

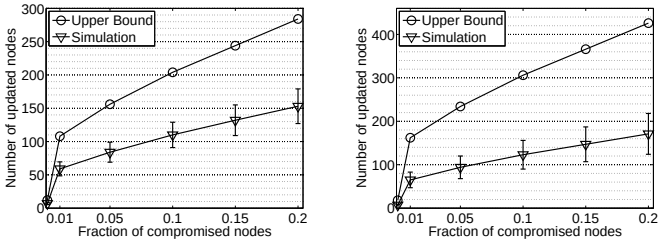
Aggregation Error We have a theorem (see proof in the technical report [27]).

Theorem 5. *Algorithm 6 guarantees that* $\forall i$, $u_i \in (\frac{n}{2}, n]$.

Comments: Since $u \leq n$, at least one copy of geometric noise is added to the sum aggregate to provide differential privacy; since $u > \frac{n}{2}$, at most one more copy of geometric noise is added. Thus, *the average aggregation error is roughly within twice of the geometric noise required for differential privacy.*

Communication Cost Alg. 6 only incurs very small communication cost. When a node joins, the joining node and another node with the minimum u are updated; when a node leaves, (at most) two remaining nodes with the maximum u are updated. Thus, the u of at most two nodes are updated for each join or leave. Considering this property and Theorem 4, the overall communication cost is as follows.

Corollary 1. *When a node joins (leaves), our scheme communicates with at most* $4d+2$ *(*$6d+2$*) nodes to update their secret keys and their values of* u*.*



(a) Join (b) Leave

Fig. 8. The communication cost of ring-based interleaved grouping.

7 Evaluations

This section evaluates the communication cost (measured by the number of updated nodes per join or leave) and the aggregation error of our solution through simulations, and evaluates the computation cost via implementation-based measurements. We compare our solution against four other schemes: the scheme proposed in [8] (denoted by *SCRCS*), the *Binary* scheme [9], the scheme proposed in [10] (denoted by *JK*), and the *Naive Grouping* scheme presented in Section 3.4.

7.1 Communication Cost of Ring-based Interleaved Grouping

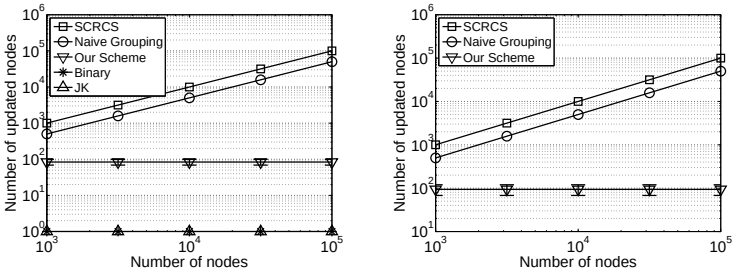
The communication cost of ring-based interleaved grouping is affected by parameter γ (i.e., the maximum fraction of nodes compromised). We first evaluate the effect of γ. We set x and d according to Table 4. To measure the communication cost of join (leave), we first generate a random initial grouping structure that includes 2000 (102,000) nodes, and then simulate 10^5 joins (leaves) over it, resulting in 102,000 (2000) nodes in the end. We compute the mean and standard deviation over the 10^5 measurements.

Figure 8 shows the simulation results as well as the analytical upper bound (see Theorem 4). As γ increases, the communication cost increases. This is because x is larger (see Eq. 1) and the group size is larger. However, only 170 nodes are updated even when 20% of nodes are compromised. Also, we found that the communication cost of our scheme is roughly half of the analytical upper bound.

7.2 Comparisons of Communication Cost

We compare our solution against the other four schemes in communication cost. In Naive Grouping, parameter g is set such that our scheme and Naive Grouping have the same average aggregation error. In our scheme, $\gamma = 0.05$. We vary the number of nodes n. For each value of n, we compute the mean and standard deviation over 10000 runs.

Figure 9 shows the results. Clearly, the communication cost of our scheme is much smaller than SCRCS and Naive Grouping, and is close to Binary and JK. This is because the ring-based interleaved grouping can effectively reduce the number of nodes that should be updated for dynamic joins and leaves. Also, the communication cost of our scheme does not change with parameter n, and hence it can scale to large systems. Interestingly, our scheme has much smaller communication cost than Naive Grouping with the same aggregation error, which demonstrates that interleaved grouping achieves a better balance between communication cost and aggregation error.



(a) Join (b) Leave

Fig. 9. Comparisons between our scheme and other schemes in the communication cost of dynamic joins and leaves (log-log scale). For dynamic leaves, the communication cost of Binary and JK is zero and thus not shown in the log-log plot.

7.3 Comparisons of Aggregation Errors

We compare our scheme with the other four schemes on aggregation error. In Naive Grouping, the parameter g is set such that Naive Grouping and our scheme have the same average communication cost for leave. In the simulation, we assume each node's data is either 0 or 1 (i.e., $\Delta = 1$). We vary parameters n, ϵ and δ. For each parameter, we measure the mean and standard deviation of aggregation error over 10000 runs.

Table 6 shows the mean and standard deviation of aggregation error. As the number of node increases, the aggregation error of Binary and Naive Grouping also increases quickly. However, the aggregation error of Our scheme, SCRCS and JK does not change too much, because on average they add a constant number of copies of geometric noise to the aggregate. As privacy parameter ϵ (δ, resp.) decreases, which means a stricter requirement for differential privacy, all schemes have a higher aggregation error, since each node needs to add more noise to meet the stronger privacy requirement. In nearly all cases, the aggregation error in our scheme is one order of magnitude smaller than Binary and Naive Grouping, and it is within 1.5 (2.5) times of the aggregation error SCRCS (JK). Thus, our scheme has low aggregation error.

7.4 Implementation and Comparisons in Running Time

We implemented a prototype system in Java. The prototype has two components which are used as the mobile node and the aggregator respectively. The mobile node compo-

Table 6. The mean and standard deviation of aggregation error

	$n = 10^3$	3162	10^4	31623	10^5
Binary	247/211	300/260	360/308	386/333	439/381
Naive Grp.	63/48	112/85	199/152	358/269	631/479
Our Sch.	26/23	27/22	26/23	26/22	26/22
SCRCS	18/17	18/17	18/17	18/17	18/17
JK	9.9/9.8	10.2/10.1	10/9.9	10.1/10	10/9.9
	$\epsilon = 0.05$	0.1	0.2	0.3	0.4
Binary	704/609	363/307	178/153	121/103	88/75
Naive Grp.	398/301	199/152	100/75	66/51	50/38
Our Sch.	52/44	26/22	13/11	9/7	6/5
SCRCS	36/34	18/17	8.7/8.5	5.8/5.6	4.6/4.3
JK	19.9/19.8	10/9.9	5.1/5	3.3/3.2	2.5/2.4
	$\delta = 0.01$	0.05	0.1	0.15	
Binary	409/341	363/307	328/285	315/281	
Naive Grp.	248/189	199/152	174/132	159/120	
Our Sch.	33/27	26/22	23/20	20/19	
SCRCS	23/20	18/17	16/15	15/13	
JK	10.2/10.1	10/9.9	10/9.8	9.9/9.9	

The number on the left (right) of "/" is mean (standard deviation).
By default, $n = 10000$, $\gamma = 0.05$, $\epsilon = 0.1$ and $\delta = 0.05$.

nent is implemented on a Android Nexus S Phone, which has 1GHz CPU and 512MB RAM, and runs Android 4.0.4 OS. The aggregator component is implemented on a Windows Laptop with 2.6GHz CPU, 4GB RAM and 64-bit Windows 7 OS. By running experiments over the prototype, we measured the time needed for the phone to encrypt a data value and the time needed for the laptop to decrypt the sum aggregate. For comparison, we also implemented SCRCS, JK and Binary and measured their running time. As to the JK scheme, the data consumer and the key manager (see [10]) are implemented together as the aggregator.

Table 7. The analytical computation cost of different schemes

	Mobile Node	Aggregator
Our Scheme	$4c$ PRFs	$\frac{2nq}{d}$ PRFs
JK	2 Paillier encryptions & 1 signature generation	2 Paillier decryptions & $2n$ mod. multiplications & n sig. verifications
SCRCS	2 mod. exp.	$\sqrt{n\Delta}$ mod. exp.
Binary	$2(\lfloor \log_2 n \rfloor + 1)$ mod. exp.	$\sqrt{n\Delta}$ mod. exp.

Table 7 shows the computation overhead of these schemes (see [8–10, 20] for details). In the implementation, HMAC-SHA256 is used as the PRF of our scheme. For JK, the Paillier cryptosystem uses a 1024-bit modulus, and RSA (1024-bit) is used as the digital signature algorithm. For SCRCS and Binary, the high-speed elliptic curve "curve25519" is used for modular exponentiation.

Table 8 shows the running time of these schemes. Compared with JK, our scheme is two orders of magnitude faster in both encryption and decryption. This is because JK uses the Paillier cryptosystem which is very expensive in computation, but our scheme is based on HMAC which is very efficient. Compared with SCRCS (Binary), our scheme is one (two) order(s) of magnitude faster in encryption, and two or three orders of magnitude faster in decryption when the plaintext space is 1000 or larger. SCRCS and Bi-

nary are very slow in decryption because they need to traverse the possible plaintext space to decrypt sum, computing one modular exponentiation for each possible value. Thus, our scheme is the most efficient in computation.

Comments. Our scheme's high efficiency in computation makes it a better choice for large-scale mobile sensing applications with large plaintext spaces, high aggregation loads (e.g., many parallel aggregation tasks per node or per aggregator, short aggregation period), resource-constraint mobile devices (e.g., personal healthcare devices besides smartphones) and rich statistics.

Table 8. The running time of different schemes

	Enc.	Decryption			
	-	$n = 10^3$	10^4	10^5	10^6
Our Scheme	3.4ms	1.2ms	12ms	0.12s	1.2s
JK	236ms	175ms	1.5s	15s	150s
SCRCS ($\Delta = 10^3$)	45ms	5.6s	18s	56s	177s
SCRCS ($\Delta = 10^4$)	45ms	18s	56s	177s	560s
SCRCS ($\Delta = 10^5$)	45ms	56s	177s	560s	1770s
Binary ($\Delta = 10^3$)	0.5∼0.9s	5.6s	18s	56s	177s
Binary ($\Delta = 10^4$)	0.5∼0.9s	18s	56s	177s	560s
Binary ($\Delta = 10^5$)	0.5∼0.9s	56s	177s	560s	1770s

The encryption time of Binary depends on n, and range [0.5, 0.9] is obtained for $n \in [10^3, 10^6]$. In our scheme, $\gamma = 0.2$.

8 Extensions and Discussions

Relaxing the trusted key dealer assumption The assumption of trusted key dealer can be relaxed. Instead, we can assume an honest-but-curious key dealer that does not collude with the aggregator. It follows our protocol as specified, but may attempt to infer the data value of nodes from the the protocol transcript and from eavesdropping all communications. Under this semi-honest model, the only adaptation that should be made to our protocol is adding one more encryption/decryption to the data that each node submits to the aggregator. Specifically, each node first encrypts its noisy data as previously specified with the secrets received from the key dealer, deriving an intermediate result c, and then encrypts c using a pre-shared key shared with the aggregator. It sends the final ciphertext to the aggregator. The aggregator first decrypts each node's intermediate result c using the pre-shared key, and then decrypts the noisy sum as previously specified. So long as the key dealer does not collude with the aggregator, it cannot get any node's intermediate result c, and thus cannot get the node's data value. In future work, we will explore how to completely remove the key dealer.

Dealing with node failures Fault tolerance is not the major focus of this paper, but our solution can be adapted in the following way such that, when some nodes fail, the aggregator can still obtain the aggregate over the remaining nodes.

Since the key dealer knows the secrets assigned to every node, if some nodes fail to submit data, the aggregator asks the dealer to submit data on behalf of those failed nodes. The dealer sets the data value as zero, adds a large-enough (see below) noise r to it, encrypts the noisy data with the secrets of all those failed nodes, and submits the obtained ciphertext to the aggregator. The aggregator can decrypt the sum over the functioning nodes' data. This method incurs a round trip communication between the key dealer and the aggregator. The overall communication cost is still $O(n)$.

Even if all nodes are functioning, the aggregator may dishonestly claim the failure of a subset $\tilde{\mathbb{S}}$ of nodes. (It can only claim one subset per aggregation period.) Then it can obtain two independent noisy sums, $S = \sum_i (x_i + r_i)$ and $S' = \sum_{i \notin \tilde{\mathbb{S}}} (x_i + r_i) + r$. It is easy to see that each node is included in at most two sums. Roughly speaking, to provide differential privacy, it suffices to add two copies of geometric noise to each sum. Thus, all nodes collectively add two copies of geometric noise to S and the key dealer itself adds two copies of geometric noise to S'. Considering that Alg. 6 may at most double the noise added to S, the final aggregation error is less than six copies of geometric noise when there is node failure, and less than four copies without node failure. Obviously, this method maintains $O(1)$ aggregation error.

If a node has failed for a long time, it can be removed from the system as a "leave". The recovery of a failed node can be processed as a "join".

In this method, the key dealer needs to be online, but it only makes online communications when there is node failure. In future work, we will study other solutions for fault tolerance, e.g., by exploring the direction pointed out in [9].

9 Conclusions

This paper proposed a novel ring-based interleaved grouping technique and applied it to privacy-preserving aggregation of time-series data in mobile sensing applications. Our solution achieves O(1) aggregation error irrespective of the number of nodes in the system. More importantly, it has very low communication overhead for dynamic joins and leaves. Simulation results show that only less than 170 nodes need to be updated for each join or leave when on average 20% of nodes are compromised, irrespective of the system scale. Our solution is very efficient in computation.

Acknowledgment

We would like to thank Elaine Shi for her insightful comments and suggestions. We would also like to thank the anonymous reviewers for their helpful suggestions.

References

1. 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 *Proc. SenSys*, 2009, pp. 85–98.
2. 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 *Proc. ACM MobiSys*, 2009, pp. 55–68.
3. 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 *Intl. ICST Conf. on Pervasive Computing Technologies for Healthcare*, 2011.
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. B. Hull, V. Bychkovsky, Y. Zhang, K. Chen, M. Goraczko, A. Miu, E. Shih, H. Balakrishnan, and S. Madden, "Cartel: a distributed mobile sensor computing system," in *SenSys*, 2006.
6. R. Honicky, E. A. Brewer, E. Paulos, and R. White, "N-smarts: networked suite of mobile atmospheric real-time sensors," in *NSDR*, 2008.

7. V. Rastogi and S. Nath, "Differentially private aggregation of distributed time-series with transformation and encryption," *ACM SIGMOD*, 2010.
8. 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.
9. T.-H. H. Chan, E. Shi, and D. Song, "Privacy-preserving stream aggregation with fault tolerance," *Financial Cryptography and Data Security (FC)*, 2012.
10. M. Jawurek and F. Kerschbaum, "Fault-tolerant privacy-preserving statistics," in *The 12th Privacy Enhancing Technologies Symposium (PETS)*, 2012.
11. C. Dwork, "Differential privacy," *Invited talk at ICALP*, 2006.
12. C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," *TCC*, 2006.
13. Q. Li and G. Cao, "Providing privacy-aware incentives for mobile sensing," in *Proc. IEEE PerCom*, 2013.
14. Z. Zhu and G. Cao, "Applaus: A privacy-preserving location proof updating system for location-based services," in *Proc. IEEE INFOCOM*, 2011.
15. E. D. Cristofaro and C. Soriente, "Short paper: Pepsi—privacy-enhanced participatory sensing infrastructure," in *Proc. ACM WiSec*, 2011, pp. 23–28.
16. 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.
17. 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.
18. 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.
19. 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.
20. Q. Li and G. Cao, "Efficient and privacy-preserving data aggregation in mobile sensing," in *Proc. IEEE ICNP*, 2012.
21. R. Chen, A. Reznichenko, P. Francis, and J. Gehrke, "Towards statistical queries over distributed private user data," in *Proc. of NSDI*, 2012.
22. D. Proserpio, S. Goldberg, and F. McSherry, "A workflow for differentially-private graph synthesis," in *Proc. ACM workshop on online social networks (WOSN)*, 2012, pp. 13–18.
23. A. Sala, X. Zhao, C. Wilson, H. Zheng, and B. Y. Zhao, "Sharing graphs using differentially private graph models," in *Proc. ACM IMC*, 2011, pp. 81–98.
24. M. Shao, Y. Yang, S. Zhu, and G. Cao, "Towards statistically strong source anonymity for sensor networks," in *Proc. IEEE INFOCOM*, 2008.
25. A. K. Lenstra and E. R. Verheul, "Selecting cryptographic key sizes," in *Proc. PKC*, 2000.
26. A. Ghosh, T. Roughgarden, and M. Sundararajan, "Universally utility-maximizing privacy mechanisms," in *ACM symposium on Theory of computing (STOC)*, 2009, pp. 351–360.
27. Q. Li and G. Cao, "Efficient privacy-preserving stream aggregation in mobile sensing with low aggregation error," *Technical Report, The Pennsylvania State University*, April 2013, http://www.cse.psu.edu/˜qxl118/papers/li2013tr.pdf.