Private and Dynamic Time-Series Data Aggregation with Trust Relaxation

Iraklis Leontiadis, Kaoutar Elkhiyaoui, Refik Molva

EURECOM, Sophia Antipolis, France
{leontiad,elkhiyao,molva}@eurecom.fr

Abstract. With the advent of networking applications collecting user data on a massive scale, the privacy of individual users appears to be a major concern. The main challenge is the design of a solution that allows the data analyzer to compute global statistics over the set of individual inputs that are protected by some confidentiality mechanism. Joye *et al.* [7] recently suggested a solution that allows a centralized party to compute the sum of encrypted inputs collected through a smart metering network. The main shortcomings of this solution are its reliance on a trusted dealer for key distribution and the need for frequent key updates. In this paper we introduce a secure protocol for aggregation of time-series data that is based on the Joye *et al.* [7] scheme and in which the main shortcomings of the latter, namely, the requirement for key updates and for the trusted dealer are eliminated. As such, during the protocol execution none of the parties apart from the users themselves are aware of the secret keys. Moreover our scheme supports a dynamic group management, whereby as opposed to Joye *et al.* [7] leave and join operations do not trigger a key update at the users.

Keywords: data aggregation, privacy, time-series data

1 Introduction

Progress in statistical data processing enables data analyzers to infer extremely useful information from the massive amount of data collected through networks and distributed applications. Such data analysis has tremendous benefits in a wide range of applications. In an e-health scenario, statistics derived from the collected data sets would greatly help field studies about diseases and the effect of a specific medicine. Another scenario entails a different environment whereby data is produced by a set of users that hold smart meters. Smart meters can report accurately at specific time intervals energy, gas or water consumption. Considering electricity consumption for instance, a data analyzer with cooperation of an energy provider can compute useful statistics such as average electricity consumption over large population of users along a specific time period. These statistics can then help the energy provider perform various operations such as load balancing and forecasting for potential acquirement.

Despite its merits, statistical data processing is challenged with privacy issues such as the confidentiality of private data. Frequent smart-readings with inappropriate analysis by companies may leak private information such as the number of people that live

in a place, the time period in which the house is empty and personal habits that can be a valuable asset to marketing retailers [10]. Serious privacy breaches are possible without any doubt in the medical scenario as well, in which the disclosure of personal data to untrusted data analyzers jeopardizes user personal information in various ways: It affects insurance coverage since records are exposed to firms and insurance companies. Moreover, a social discrimination is possible owing to the exposure of medical treatments.

While encryption of data would protect data privacy, data analysis by an untrusted aggregator would become challenging. Existing privacy preserving solutions for data analysis benefit from cryptographic algorithms [13] that either introduce a high computation cost at the aggregator or restrict the possible range of values that users can submit due to the need of discrete logarithm computation. To mitigate this drawback, Joye *et al.* [7] proposed a nifty construction which calls for a fully trusted dealer. The reliance on trusted key dealer however can be deemed unrealistic for real world applications. In addition, this solution builds upon a static key management scheme whereby user joins and leaves induce a significant overhead in terms of communication.

In this paper, we improve the design of the privacy preserving aggregation protocol suggested by Joye *et al.* [7] by eliminating the need for key redistribution following a user join or leave and the need for fully trusted key dealer. The features of the enhanced protocol can be summarized as follows:

- *No key dealer*. Unlike most previous privacy preserving aggregation protocols, there is no trusted key dealer in our scheme. In contrast, we introduce a *semi-trusted* party called *collector* which gathers partial key information from users through a *secure* channel.
- *Support for dynamic user populations*. No coordination is required to manage changes in the population of users. This is possible due to a self-generated key mechanism by which no key agreement between users is required.
- *Privacy*. With respect to privacy, the scheme assures *aggregator obliviousness* as introduced by Elaine Shi *et al.* [13]. That is, the untrusted aggregator only learns the sum and the average over users' private data at the end of the protocol execution. Moreover, we show that the collector does not derive any information about the users' private data.
- *Efficiency*. Like Joye *et al.* [7] our scheme enables the computation of the sum and the average over a large number of users without restrictions on the range of users' values. It is also scalable in the sense that decryptions performed by the aggregator do not depend on the number of users.

Outline of the paper. The sequel of the paper is organized as follows: Section 2 presents the problem statement and the privacy definitions. Section 3 depicts the idea of our solution while Section 4 provides the full protocol description. In Section 5, we analyze the privacy of our protocol, and in Section 6 we assess its performances. Before concluding the paper in Section 8, we give a quick overview of previous work in Section 7.

2 Problem Statement

We consider a scenario where an aggregator $\mathcal{A}$ would like to compute some aggregate sum of the private data of some users $\mathcal{U}_i$. Similarly to the work of [7] and [13], we restrict ourselves to time-series data which is a series of data point observations measured at equally spaced time intervals. A straightforward approach to compute the aggregate sum would be encrypting $\mathcal{U}_i$'s individual data using the public key of $\mathcal{A}$. This solution however relies on a *trusted* aggregator which first decrypts the users' individual data using its secret key then computes the sum. To tackle this issue, [7] and [13] employ a combination of secret sharing techniques and additively homomorphic encryption to enable aggregator $\mathcal{A}$ to compute the sum of users' data without compromising users' privacy. The idea is to have a *trusted third party* called *key dealer* that provides each user $\mathcal{U}_i$ with a secret share sk_i while supplying the aggregator $\mathcal{A}$ with the secret key sk_A defined as $-\sum \mathsf{sk}_i$. Each user $\mathcal{U}_i$ encrypts its private data using its secret share sk_i and forwards the resulting ciphertext to the aggregator, which in turn combines the received ciphertexts so as to obtain an encryption of the sum of the users' data that can be decrypted using the aggregator's secret key sk_A.

Although such solutions prevent the aggregator from learning users' confidential data, they suffer from two main limitations which we aim to address in this paper. The first limitation is that they build upon the assumption that the key dealer is *trusted* and does not have any interest in undermining user privacy. Whereas the second shortcoming –which is generally overlooked– is that these solutions only support static groups of users and as a result they are fault intolerant. Namely, in the case of user failures, aggregator $\mathcal{A}$ cannot compute the aggregate sum. Along these lines, we propose a solution for privacy preserving data aggregation of time-series data that draws upon the work of [7] and which in addition to supporting *dynamic group management* and arbitrary *user failures* does not depend on trusted key dealers. The idea is that instead of having a key dealer that distributes the secret shares to users $\mathcal{U}_i$ and aggregator $\mathcal{A}$, we rely on an *untrusted* entity that we call *collector* which acts as an *intermediary* between users $\mathcal{U}_i$ and aggregator $\mathcal{A}$, and helps accordingly $\mathcal{A}$ compute the aggregate sum of users' individual data without any prior secret key distribution by a trusted key dealer. Now before presenting our solution, we give an overview of the entities and the privacy definitions that we will refer to in the sequel of this paper.

2.1 Entities

A scheme for dynamic and privacy preserving data aggregation for time-series involves the following entities:

- Users $\mathcal{U}_i$: At each specific time interval t, each user $\mathcal{U}_i$ produces a data point $x_{i,t}$ that it wants to send to an aggregator. Each data point contains private sensitive information pertaining to user $\mathcal{U}_i$. To protect the confidentiality of the value of $x_{i,t}$ against the aggregator and eavesdroppers, user $\mathcal{U}_i$ encrypts $x_{i,t}$ using some secret input sk_i and forwards the resulting ciphertext $c_{i,t}$ to the aggregator. It also sends to the collector some auxiliary information $\mathsf{aux}_{i,t}$ that will be used later to compute the aggregate sum of individual data. Without loss of generality, we denote $\mathbb{U}$ the set of users $\mathcal{U}_i$ in the system.

- Collector $\mathcal{C}$: It is an *untrusted party* which upon receiving the auxiliary information $\mathsf{aux}_{i,t}$ sent by users $\mathcal{U}_i \in \mathbb{U}$ at time interval t computes a function g of $\mathsf{aux}_{i,t}$. Hereafter, we denote aux_t the output of function g at time interval t.
- Aggregator $\mathcal{A}$: It is an *untrusted entity* which upon receipt of ciphertexts $c_{i,t}$ and the auxiliary information aux_t at time interval t computes the sum $\sum_{\mathcal{U}_i \in \mathbb{U}} x_{i,t}$ over the data points $x_{i,t}$ underlying ciphertexts $c_{i,t}$.

2.2 Privacy Preserving and Dynamic Time-Series Data Aggregation

A privacy preserving and dynamic time-series data aggregation protocol consists of the following algorithms:

- $\mathsf{Setup}(1^\tau) \rightarrow (\mathcal{P}, \mathsf{sk}_A, \mathsf{sk}_C, \{\mathsf{sk}_i\}_{\mathcal{U}_i \in \mathbb{U}})$: It is a randomized algorithm which on input of a security parameter τ, outputs the public parameters $\mathcal{P}$ that will be used by subsequent algorithms, the secret key sk_A of aggregator $\mathcal{A}$, the secret key sk_C of collector $\mathcal{C}$ and the secret keys $\{\mathsf{sk}_i\}_{\mathcal{U}_i \in \mathbb{U}}$ of users $\mathcal{U}_i$.
- $\mathsf{Encrypt}(t, \mathsf{sk}_i, x_{i,t}) \rightarrow c_{i,t}$: It is a deterministic algorithm which on input of time interval t, secret key sk_i of user $\mathcal{U}_i$ and data point $x_{i,t}$, encrypts $x_{i,t}$ and outputs the resulting ciphertext $c_{i,t}$.
- $\mathsf{Collect}((\mathsf{aux}_{i,t})_{\mathcal{U}_i \in \mathbb{U}}, \mathsf{sk}_C) \rightarrow \mathsf{aux}_t$: It is a deterministic algorithm executed by collector $\mathcal{C}$ which on input of the auxiliary information $(\mathsf{aux}_{i,t})_{\mathcal{U}_i \in \mathbb{U}}$ provided by individual users $\mathcal{U}_i$ and collector $\mathcal{C}$'s secret key sk_C computes a function g over $\mathsf{aux}_{i,t}$ and outputs the result aux_t.
- $\mathsf{Aggregate}(\{c_{i,t}\}_{\mathcal{U}_i \in \mathbb{U}}, \mathsf{aux}_t, \mathsf{sk}_A) \rightarrow \sum x_{i,t}$: It is a deterministic algorithm run by aggregator $\mathcal{A}$. It takes as inputs ciphertexts $\{c_{i,t}\}_{\mathcal{U}_i \in \mathbb{U}}$, auxiliary information aux_t supplied by collector $\mathcal{C}$ and aggregator $\mathcal{A}$'s secret key sk_A, and outputs the sum $\sum x_{i,t}$, where $x_{i,t}$ is the plaintext underlying ciphertext $c_{i,t}$.

2.3 Privacy Definitions

In accordance with the work of [7,13], we assume in this paper an honest-but-curious model. This means that while the participants in the protocol are interested in learning the individual data of users, they still comply with the aggregation protocol. Namely, users are always presumed to submit a correct input to the aggregation protocol. Actually, data pollution attacks where users submit bogus values to the aggregator is orthogonal to the problem of privacy preserving data aggregation. We also assume that while users $\mathcal{U}_i$ may collude with either aggregator $\mathcal{A}$ or collector $\mathcal{C}$ by disclosing their private individual data, aggregator $\mathcal{A}$ and collector $\mathcal{C}$ never collude.

In this section, we present two formalizations: the first one defines privacy against aggregator $\mathcal{A}$ which we call in compliance with previous work *aggregator obliviousness*, whereas the second formalization defines privacy against collector $\mathcal{C}$ which we refer to as *collector obliviousness*.

Aggregator Obliviousness Aggregator Obliviousness (AO) ensures that for each time interval t, the aggregator learns nothing other than the value of $\sum_{\mathcal{U}_i \in \mathbb{U}} x_{i,t}$ from ciphertexts $c_{i,t}$ and the auxiliary information aux_t that it receives from users $\mathcal{U}_i \in \mathbb{U}$ and collector $\mathcal{C}$ respectively. It ensures also that even if aggregator $\mathcal{A}$ colludes with an arbitrary set of users $\mathbb{K} \subset \mathbb{U}$, it will only be able to learn the value of the aggregate sum of honest users (i.e. $\sum_{\mathcal{U}_i \in \mathbb{U} \setminus \mathbb{K}} x_{i,t}$) and nothing else.

To formally capture the capabilities of an aggregator $\mathcal{A}$ against the privacy of aggregation protocols, we assume that $\mathcal{A}$ is given access to the following oracles:

- $\mathcal{O}_{\mathsf{setup},\mathcal{A}}$: When called, this oracle provides aggregator $\mathcal{A}$ with the public parameters denoted $\mathcal{P}$ of the aggregation protocol and any secret information sk_A that may be needed by aggregator $\mathcal{A}$ to perform the aggregation.
- $\mathcal{O}_{\mathsf{encrypt}}$: When queried with time t, identifier uid_i of some user $\mathcal{U}_i$ and a data point $x_{i,t}$, oracle $\mathcal{O}_{\mathsf{encrypt}}$ outputs the encryption $c_{i,t}$ of $x_{i,t}$ in time interval t using $\mathcal{U}_i$'s secret input sk_i.
- $\mathcal{O}_{\mathsf{corrupt}}$: When queried with the identifier uid_i of some user $\mathcal{U}_i$, the oracle $\mathcal{O}_{\mathsf{corrupt}}$ returns the secret key sk_i of user $\mathcal{U}_i$.
- $\mathcal{O}_{\mathsf{collect},\mathcal{A}}$: When called with time t, this oracle returns the auxiliary information aux_t that collector $\mathcal{C}$ computed during time interval t. We note that in schemes such as [13,7] where a collector is not needed, the aggregator will not call this oracle.
- $\mathcal{O}_{\mathsf{AO}}$: When called with a subset of users $\mathbb{S} \subset \mathbb{U}$ and with two time-series $(\mathcal{U}_i, t, x^0_{i,t})_{\mathcal{U}_i \in \mathbb{S}}$ and $(\mathcal{U}_i, t, x^1_{i,t})_{\mathcal{U}_i \in \mathbb{S}}$ such that $\sum x^0_{i,t} = \sum x^1_{i,t}$, this oracle flips a random coin $b \in \{0,1\}$ and returns an encryption of the time-serie $(\mathcal{U}_i, t, x^b_{i,t})_{\mathcal{U}_i \in \mathbb{S}}$ (that is the tuple of ciphertexts $(c^b_{i,t})_{\mathcal{U}_i \in \mathbb{S}}$) and the corresponding auxiliary information aux^b_t that aggregator $\mathcal{A}$ should receive from the collector in time interval t.

Aggregator $\mathcal{A}$ has access to the above oracles in two phases: a learning and a challenge phase. In the learning phase (cf. Algorithm 1), aggregator $\mathcal{A}$ first calls the oracle $\mathcal{O}_{\mathsf{setup},\mathcal{A}}$ that provides $\mathcal{A}$ with the set of public parameters $\mathcal{P}$ associated with the aggregation protocol together with any secret information sk_A that aggregator $\mathcal{A}$ may need to execute the aggregation correctly. Next, $\mathcal{A}$ compromises users $\mathcal{U}_i$ by calling the oracle $\mathcal{O}_{\mathsf{corrupt}}$ which returns the secret keys of compromised users. Then, $\mathcal{A}$ picks a time interval t and issues encryption queries $(t, \mathsf{uid}_i, x_{i,t})$ to the oracle $\mathcal{O}_{\mathsf{encrypt}}$ which outputs the corresponding ciphertexts $c_{i,t}$. Finally, $\mathcal{A}$ calls the oracle $\mathcal{O}_{\mathsf{collect}}$ to get the auxiliary information aux_t computed by collector $\mathcal{C}$ in time interval t.

In the challenge phase (see Algorithm 2), aggregator $\mathcal{A}$ chooses a subset $\mathbb{S}^*$ of users that were not compromised and a challenge time interval t^* for which it did not make an encryption query during the learning phase. $\mathcal{A}$ then submits two time-series $\mathcal{X}^0_{t^*} = (\mathcal{U}_i, t^*, x^0_{i,t^*})_{\mathcal{U}_i \in \mathbb{S}^*}$ and $\mathcal{X}^1_{t^*} = (\mathcal{U}_i, t^*, x^1_{i,t^*})_{\mathcal{U}_i \in \mathbb{S}^*}$ to the oracle $\mathcal{O}_{\mathsf{AO}}$, such that $\sum x^0_{i,t} = \sum x^1_{i,t}$. Oracle $\mathcal{O}_{\mathsf{AO}}$ accordingly flips a coin $b \in \{0,1\}$ and returns the encryption $(c^b_{i,t^*})_{\mathcal{U}_i \in \mathbb{S}^*}$ of the time-serie $\mathcal{X}^b_{t^*}$ and the auxiliary information $\mathsf{aux}^b_{t^*}$ computed by collector $\mathcal{C}$ for time interval t^*. At the end of the challenge phase, aggregator $\mathcal{A}$ outputs a guess b^* for the bit b.

We say that aggregator $\mathcal{A}$ succeeds in the aggregator obliviousness game, if its guess $b^* = b$.

Algorithm 1: Learning phase of the aggregator obliviousness game

$(\mathcal{P}, \mathsf{sk}_A) \leftarrow \mathcal{O}_{\mathsf{setup},\mathcal{A}}$;
// $\mathcal{A}$ executes the following a polynomial number of times
$\mathsf{sk}_i \leftarrow \mathcal{O}_{\mathsf{corrupt}}(\mathsf{uid}_i)$;
$\mathcal{A} \rightarrow t$;
// $\mathcal{A}$ is allowed to call $\mathcal{O}_{\mathsf{encrypt}}$ for all users $\mathcal{U}_i$
$c_{i,t} \leftarrow \mathcal{O}_{\mathsf{encrypt}}(t, \mathsf{uid}_i, x_{i,t})$;
$\mathsf{aux}_t \leftarrow \mathcal{O}_{\mathsf{collect},,\mathcal{A}}(t)$;

Algorithm 2: Challenge phase of the aggregator obliviousness game

$\mathcal{A} \rightarrow t^*, \mathbb{S}^*$;
$\mathcal{A} \rightarrow \mathcal{X}^0_{t^*}, \mathcal{X}^1_{t^*}$;
$\langle (c^b_{i,t^*})_{\mathcal{U}_i \in \mathbb{S}^*}, \mathsf{aux}^b_{t^*} \rangle \leftarrow \mathcal{O}_{\mathsf{AO}}(\mathcal{X}^0_{t^*}, \mathcal{X}^1_{t^*})$;
$\mathcal{A} \rightarrow b^*$;

Definition 1 (Aggregator Obliviousness). *An aggregation protocol is said to ensure aggregator obliviousness if for any aggregator $\mathcal{A}$, the probability $\Pr(b = b^*) \leqslant \frac{1}{2} + \epsilon$, where ϵ is a negligible function.*

Collector Obliviousness Collector Obliviousness (CO) guarantees that collector $\mathcal{C}$ cannot infer any information about the private input of individual users $\mathcal{U}_i$ either from the messages it receives directly from the users or the protocol exchange between the users and the aggregator. It also entails that even in the case where collector $\mathcal{C}$ colludes with a set of users $\mathbb{K}$, it does not gain any additional information about the individual values of honest users $\mathcal{U}_i$ in $\mathbb{U} \setminus \mathbb{K}$.

To formally reflect the adversarial capabilities of collector $\mathcal{C}$ against aggregation protocols, we assume that in addition to the oracles $\mathcal{O}_{\mathsf{encrypt}}$ and $\mathcal{O}_{\mathsf{corrupt}}$, collector $\mathcal{C}$ is given access to the following oracles:

- $\mathcal{O}_{\mathsf{setup},\mathcal{C}}$: When queried, this oracle supplies collector $\mathcal{C}$ with the public parameters denoted $\mathcal{P}$ of the aggregation protocol and any secret information sk_C that collector $\mathcal{C}$ may need during the aggregation protocol.
- $\mathcal{O}_{\mathsf{collect},\mathcal{C}}$: When invoked with time t, identifier uid_i of some user $\mathcal{U}_i$ and ciphertext $c_{i,t}$, this oracle returns the auxiliary information $\mathsf{aux}_{i,t}$ that corresponds to ciphertext $c_{i,t}$ that user $\mathcal{U}_i$ computed during time interval t.
- $\mathcal{O}_{\mathsf{CO}}$: When called with a subset of users $\mathbb{S} \subset \mathbb{U}$ and with two time-series $(\mathcal{U}_i, t, x^0_{i,t})_{\mathcal{U}_i \in \mathbb{S}}$ and $(\mathcal{U}_i, t, x^1_{i,t})_{\mathcal{U}_i \in \mathbb{S}}$, this oracle flips a random coin $b \in \{0, 1\}$ and returns to collector $\mathcal{C}$ an encryption of the time-serie $(\mathcal{U}_i, t, x^b_{i,t})_{\mathcal{U}_i \in \mathbb{S}}$ (i.e. the ciphertexts $(c^b_{i,t})_{\mathcal{U}_i \in \mathbb{S}}$) and the corresponding auxiliary information computed by users $\mathcal{U}_i \in \mathbb{S}$ (i.e. $(\mathsf{aux}^b_{i,t})_{\mathcal{U}_i \in \mathbb{S}}$).

Collector $\mathcal{C}$ accesses the aforementioned oracles in a learning and a challenge phase. In the learning phase (cf. Algorithm 3), collector $\mathcal{C}$ first queries the oracle $\mathcal{O}_{\mathsf{setup},\mathcal{C}}$

Algorithm 3: Learning phase of the collector obliviousness game

$(\mathcal{P}, \mathsf{sk}_C) \leftarrow \mathcal{O}_{\mathsf{setup},\mathcal{C}}$;
// $\mathcal{C}$ executes the following a polynomial number of times
$\mathsf{sk}_i \leftarrow \mathcal{O}_{\mathsf{corrupt}}(\mathsf{uid}_i)$;
$\mathcal{C} \rightarrow t$;
// $\mathcal{C}$ is allowed to call $\mathcal{O}_{\mathsf{encrypt}}$ and $\mathcal{O}_{\mathsf{collect},\mathcal{C}}$ for all users $\mathcal{U}_i$
$c_{i,t} \leftarrow \mathcal{O}_{\mathsf{encrypt}}(t, \mathsf{uid}_i, x_{i,t})$;
$\mathsf{aux}_{i,t} \leftarrow \mathcal{O}_{\mathsf{collect},\mathcal{C}}(t, \mathsf{uid}_i, c_{i,t})$;

Algorithm 4: Challenge phase of the collector obliviousness game

$\mathcal{C} \rightarrow t^*, \mathbb{S}^*$;
$\mathcal{C} \rightarrow \mathcal{X}^0_{t^*}, \mathcal{X}^1_{t^*}$;
$(\langle c^b_{i,t^*}, \mathsf{aux}^b_{i,t^*} \rangle)_{\mathcal{U}_i \in \mathbb{S}^*} \leftarrow \mathcal{O}_{\mathsf{CO}}(\mathcal{X}^0_{t^*}, \mathcal{X}^1_{t^*})$;
$\mathcal{C} \rightarrow b^*$;

which supplies $\mathcal{C}$ with the set of public parameters $\mathcal{P}$ of the aggregation protocol and the secret information sk_C that collector $\mathcal{C}$ should have to execute the aggregation properly. Then, $\mathcal{C}$ calls the oracle $\mathcal{O}_{\mathsf{corrupt}}$ to compromise users in the system. Next, it selects a time interval t and submits encryption queries $(t, \mathsf{uid}_i, x_{i,t})$ to the oracle $\mathcal{O}_{\mathsf{encrypt}}$ which outputs the corresponding ciphertexts $c_{i,t}$. Finally, it issues queries $(t, \mathsf{uid}_i, c_{i,t})$ to the oracle $\mathcal{O}_{\mathsf{collect},\mathcal{C}}$ to get the auxiliary information $\mathsf{aux}_{i,t}$ generated by users $\mathcal{U}_i$ for time interval t and ciphertext $c_{i,t}$.

In the challenge phase (see Algorithm 4), collector $\mathcal{C}$ selects a subset $\mathbb{S}^*$ of honest users and a challenge time interval t^* for which it did not make an encryption query in the learning phase. Then, $\mathcal{C}$ queries the oracle $\mathcal{O}_{\mathsf{CO}}$ with two time-series $\mathcal{X}^0_{t^*} = (\mathcal{U}_i, t^*, x^0_{i,t^*})_{\mathcal{U}_i \in \mathbb{S}^*}$ and $\mathcal{X}^1_{t^*} = (\mathcal{U}_i, t^*, x^1_{i,t^*})_{\mathcal{U}_i \in \mathbb{S}^*}$. $\mathcal{O}_{\mathsf{CO}}$ then picks randomly a bit $b \in \{0, 1\}$ and returns the tuple $(\langle c^b_{i,t^*}, \mathsf{aux}^b_{i,t^*} \rangle)_{\mathcal{U}_i \in \mathbb{S}^*}$ for the time-serie $\mathcal{X}^b_{t^*}$. At the end of the challenge phase, collector $\mathcal{C}$ outputs a guess b^* for the bit b.

We say that collector $\mathcal{C}$ succeeds in the collector obliviousness game, if its guess $b^* = b$.

Definition 2 (Collector Obliviousness). *An aggregation protocol is said to ensure collector obliviousness if for any collector $\mathcal{C}$, the probability* $\Pr(b = b^*) \leqslant \frac{1}{2} + \epsilon$, *where* ϵ *is a negligible function.*

3 Idea of Solution

The homomorphic scheme suggested by Joye and Libert [7] allows an untrusted aggregator to evaluate the sum or the average without any access to individual data. However to support this functionality, a fully trusted dealer has to distribute secret keys to each $\mathcal{U}_i$ and as a result, it will be able to decrypt. Our scheme extends Joye and Libert scheme [7] through two major enhancements :

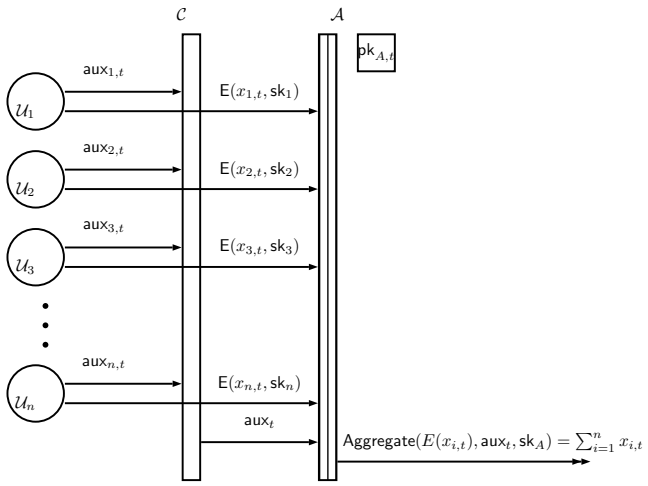

Fig. 1. Overview of our protocol for a single time interval t. $\mathsf{pk}_{A,t}$ is public known value.

- **No key dealer:** Our scheme does not require a trusted key dealer that might get individual private data samples.
- **Dynamic group management:** In the Joye and Libert scheme [7] each join or leave operation triggers a new key redistribution for all the users in the aggregation system, whereas in our protocol, join and leave operations are possible without any key update at the users. Hence, dynamic group management is assured with significantly lower communication and computation overhead. The proposed protocol is also resilient to user failures that may occur due to communication errors or hardware failures.

In order to eliminate the need for a fully trusted dealer and to support *dynamic group management* without inducing additional communication or computation overhead, we employ two techniques:

- *Responsibility splitting mechanism*: Each user $\mathcal{U}_i$ sends an encryption of its private data sample to aggregator $\mathcal{A}$ and an obfuscated version of its secret key sk_i to the semi trusted collector $\mathcal{C}$, in such a way that neither the aggregator nor the collector can violate the privacy of individual samples provided by users.
- *Self-generation of secret keys:* The secret keys used to encrypt individual data samples are generated independently by users with no coordination by a trusted key dealer.

An overview of our solution is depicted in figure 1. Each user $\mathcal{U}_i$ chooses independently its secret key sk_i whereas the untrusted aggregator generates a random key sk_A. For each time interval t, aggregator $\mathcal{A}$ publishes an obfuscated version $\mathsf{pk}_{A,t}$ of the secret key sk_A. Users $\mathcal{U}_i$ on the other hand encrypt their private data samples $x_{i,t}$ with their secret keys sk_i using the Joye-Libert cryptosystem, and send the corresponding ciphertexts $c_{i,t}$ to aggregator $\mathcal{A}$. They also obfuscate their secret keys sk_i using $\mathsf{pk}_{A,t}$ and

sends the resulting auxiliary information $\mathsf{aux}_{i,t}$ to collector $\mathcal{C}$ through a secure channel. Collector $\mathcal{C}$ computes a function $g(t)$ of the auxiliary information $\mathsf{aux}_{i,t}$ it has received and forwards the output aux_t to aggregator $\mathcal{A}$. Upon receiving the ciphertexts $c_{i,t}$ and the auxiliary information aux_t, $\mathcal{A}$ uses its secret key sk_A and learns the sum $\sum x_{i,t}$ for the time interval t.

In this manner, we eliminate the need of a trusted key dealer that knows users' private keys while ensuring that neither the aggregator nor the collector can infer information about users' individual data, and we achieve efficient dynamic group management that does not call for any key update mechanism.

4 Protocol Description

Without loss of generality, we assume in the remainder of this section that the aggregation system comprises n users denoted $\mathbb{U} = \{\mathcal{U}_1, ..., \mathcal{U}_n\}$.

Now before providing the description of our solution, we first give a brief overview of the Joye-Libert scheme[7].

4.1 Joye-Libert Scheme [7] (JL)

- $\mathsf{Setup}_{\mathsf{JL}}$: A trusted dealer $\mathcal{D}$ selects randomly two safe prime numbers p and q and sets $N = pq$. Then, it defines a cryptographic hash function $H : \mathbb{Z} \rightarrow \mathbb{Z}^*_{N^2}$ and outputs the public parameters $\mathcal{P}_{\mathsf{JL}} = (N, H)$. Finally, the dealer $\mathcal{D}$ distributes to each user $\mathcal{U}_i \in \mathbb{U}$ a secret key $\mathsf{sk}_i \in [0, N^2]$ and sends $\mathsf{sk}_A = -\sum_{i=1}^{n} \mathsf{sk}_i$ to the untrusted aggregator $\mathcal{A}$.
  We note that hereafter that all computations are performed $\mod N^2$ unless mentioned otherwise.
- $\mathsf{Encrypt}_{\mathsf{JL}}$: For each time interval t, each user $\mathcal{U}_i$ encrypts its private data $x_{i,t}$ using the secret key sk_i and outputs the ciphertext $c_{i,t} = (1 + x_{i,t}N)H(t)^{\mathsf{sk}_i}$. We note that ciphertexts $c_{i,t}$ fulfills the following property:

$$\prod_{i=1}^{n} c_{i,t} = \prod_{i=1}^{n} (1 + x_{i,t}N)H(t)^{\mathsf{sk}_i} = (1 + \sum_{i=1}^{n} x_{i,t}N)H(t)^{\sum_{i=1}^{n} \mathsf{sk}_i}$$
$$= (1 + \sum_{i=1}^{n} x_{i,t}N)H(t)^{-\mathsf{sk}_A}$$

- $\mathsf{Aggregate}_{\mathsf{JL}}$: Upon receiving $c_{i,t}$ the untrusted aggregator computes $P_t = \prod_{i=1}^{n} c_{i,t}$ $H(t)^{\mathsf{sk}_A} = 1 + \sum_{i=1}^{n} x_{i,t}N$ and recovers $\sum_{i=1}^{n} x_{i,t}$ by computing $\frac{P_t - 1}{N}$ in $\mathbb{Z}$. The value $\frac{P_t - 1}{N}$ is meaningful as long as $\sum_{i=1}^{n} x_{i,t} < N$.

We recall that the JL scheme is aggregator oblivious in the random oracle model under the decisional composite residuosity (DCR) assumption (cf. [7]).

4.2 Description

Our protocol runs in four phases:

- Setup: A trusted third party $\mathcal{TP}$ selects two safe primes p and q, sets $N = pq$, and picks a cryptographic hash function $H : \{0,1\}^* \rightarrow \mathbb{Z}^*_{N^2}$. $\mathcal{TP}$ then publishes the public parameters $\mathcal{P} = (N, H)$ and goes offline. Next, aggregator $\mathcal{A}$ generates a random secret key $\mathsf{sk}_A \in \mathbb{Z}^*_{N^2}$, and each user $\mathcal{U}_i \in \mathbb{U}$ independently chooses its random secret key $\mathsf{sk}_i \in [0, N^2]$ *without any coordination by a trusted key dealer*. It is important to note here that contrary to the JL scheme, the trusted third party $\mathcal{TP}$ does not know the individual secret keys of users $\mathcal{U}_i$, and once the public parameters $\mathcal{P}$ are published it can go offline.
- Encrypt: For each time interval t, each user $\mathcal{U}_i$ encrypts its private data $x_{i,t}$ using its secret key sk_i and the algorithm $\mathsf{Encrypt}_{\mathsf{JL}}$ as shown in subsection 4.1, and sends the resulting ciphertext $c_{i,t} = (1 + x_{i,t}N)H(t)^{\mathsf{sk}_i} \mod N^2$ to aggregator $\mathcal{A}$.
- Collect: For each time interval t, aggregator $\mathcal{A}$ publishes $\mathsf{pk}_{A,t} = H(t)^{\mathsf{sk}_A}$. Each user $\mathcal{U}_i$ then computes the auxiliary information $\mathsf{aux}_{i,t} = \mathsf{pk}_{A,t}^{\mathsf{sk}_i} = H(t)^{\mathsf{sk}_A \mathsf{sk}_i}$ using its secret key sk_i and sends $\mathsf{aux}_{i,t}$ to collector $\mathcal{C}$ through a *secure channel*.
  Upon receiving $\mathsf{aux}_{i,t}$ $(1 \leqslant i \leqslant n)$ from users $\mathcal{U}_i \in \mathbb{U}$, collector $\mathcal{C}$ computes
  $$\mathsf{aux}_t = \prod_{i=1}^{n} \mathsf{aux}_{i,t} = \prod_{i=1}^{n} H(t)^{\mathsf{sk}_A \mathsf{sk}_i} = H(t)^{\mathsf{sk}_A \sum_{i=1}^{n} \mathsf{sk}_i}$$
  and sends the result to aggregator $\mathcal{A}$.
  Notice here that $\mathcal{C}$ does not obtain the secret value $H(t)^{\mathsf{sk}_i}$ employed by users $\mathcal{U}_i$ during the encryption, rather it only learns an obfuscated encoding of it which is $\mathsf{aux}_{i,t} = H(t)^{\mathsf{sk}_A \mathsf{sk}_i}$.
- Aggregate: Upon receiving the ciphertexts $c_{i,t}$ $(1 \leqslant i \leqslant n)$ and the auxiliary information aux_t, aggregator $\mathcal{A}$ calculates:
  $$P_t = (\prod_{i=1}^{n} c_{i,t})^{\mathsf{sk}_A} = ((1 + \sum_{i=1}^{n} x_{i,t}N)H(t)^{\sum_{i=1}^{n} \mathsf{sk}_i})^{\mathsf{sk}_A}$$
  $$= (1 + \sum_{i=1}^{n} x_{i,t}N)^{\mathsf{sk}_A} H(t)^{\mathsf{sk}_A \sum_{i=1}^{n} \mathsf{sk}_i}$$
  Since the order of $(1 + \sum_{i=1}^{n} x_{i,t}N)$ is either N or divides N, we have:
  $$P_t = (1 + \sum_{i=1}^{n} x_{i,t}N)^{\mathsf{sk}'_A} H(t)^{\mathsf{sk}_A \sum_{i=1}^{n} \mathsf{sk}_i} = (1 + \mathsf{sk}'_A \sum_{i=1}^{n} x_{i,t}N)H(t)^{\mathsf{sk}_A \sum_{i=1}^{\mathsf{sk}_i}}$$
  where $\mathsf{sk}'_A = \mathsf{sk}_A \mod N$.
  Finally, aggregator $\mathcal{A}$ computes $I_t = \frac{\frac{P_t}{\mathsf{aux}_t} - 1}{N} = \mathsf{sk}'_A \sum_{i=1}^{n} x_{i,t}$ in $\mathbb{Z}$ and evaluates $R_t = {\mathsf{sk}'_A}^{-1} I_t \mod N = \sum_{i=1}^{n} x_{i,t} \mod N$ to obtain the sum of $x_{i,t}$. Notice that since $\mathsf{sk}_A \in \mathbb{Z}^*_{N^2}$, sk'_A is in $\mathbb{Z}^*_N$. Now to obtain the average of the data points $x_{i,t}$, aggregator $\mathcal{A}$ computes $\frac{R_t}{n}$ in $\mathbb{Z}$.
  As in the JL scheme, the result of the aggregation is meaningful as long as $\sum_{i=1}^{n} x_{i,t} < N$.

Fault Tolerance Suppose at time interval t a set of users $\mathbb{F}$ fail to participate in the protocol execution. This event does not affect the computation of the aggregate sum by the aggregator $\mathcal{A}$. Indeed, each user $\mathcal{U}_i \notin \mathbb{F}$ computes: $\mathsf{aux}_{i,t} = \mathsf{pk}_{A,t}^{\mathsf{sk}_i}$ and encrypts its data by computing $c_{i,t} = (1+x_{i,t}N)H(t)^{\mathsf{sk}_i}$. Upon receiving the auxiliary information $\mathsf{aux}_{i,t}$ from users $\mathcal{U}_i \notin \mathbb{F}$, collector $\mathcal{C}$ computes $\mathsf{aux}_t = \prod_{\mathcal{U}_i \notin \mathbb{F}} \mathsf{aux}_{i,t} = \prod_{\mathcal{U}_i \notin \mathbb{F}} H(t)^{\mathsf{sk}_A \mathsf{sk}_i}$. When aggregator $\mathcal{A}$ receives the ciphertexts $c_{i,t}$ from users $\mathcal{U}_i \notin \mathbb{F}$ and aux_t from collector $\mathcal{C}$, it first computes the product $\prod_{\mathcal{U}_i \notin \mathbb{F}} c_{i,t}$ and computes as depicted above the value of $\sum_{\mathcal{U}_i \notin \mathbb{F}} x_{i,t}$. Thus, our solution will still function correctly even when an arbitrary number of users fail to submit their contributions to the protocol as long as collector $\mathcal{C}$ operates properly.

5 Privacy Analysis

In the following, we state the main theorems of this paper.

5.1 Aggregator Obliviousness

Theorem 1. *The proposed solution ensures aggregator obliviousness under the decisional composite residuosity (DCR) assumption in $\mathbb{Z}_{N^2}^*$.*

Proof. Assume there is an aggregator $\mathcal{A}$ that breaks the aggregator obliviousness of our scheme with a non-negligible advantage ϵ. We show in what follows that there exists an aggregator $\mathcal{B}$ that uses $\mathcal{A}$ to break the aggregator obliviousness of the JL protocol (which is ensured under DCR) with a non-negligible advantage ϵ.

For ease of exposition, we denote $\mathcal{O}_{\mathsf{setup}}^{\mathsf{JL}}$, $\mathcal{O}_{\mathsf{corrupt}}^{\mathsf{JL}}$, $\mathcal{O}_{\mathsf{encrypt}}^{\mathsf{JL}}$ and $\mathcal{O}_{\mathsf{AO}}^{\mathsf{JL}}$ the oracles needed for the aggregator obliviousness game of the JL protocol. We also assume that the aggregation system of the JL scheme involves n users $\mathbb{U} = \{\mathcal{U}_1, ..., \mathcal{U}_n\}$, each is endowed with secret key sk_i.

Now to break the aggregator obliviousness of the JL scheme, aggregator $\mathcal{B}$ simulates the aggregator obliviousness game of our scheme for aggregator $\mathcal{A}$ as follows:

Learning phase.

- To simulate the oracle $\mathcal{O}_{\mathsf{setup}}$ for aggregator $\mathcal{A}$, $\mathcal{B}$ first invokes the oracle $\mathcal{O}_{\mathsf{setup}}^{\mathsf{JL}}$ which returns the public parameters $\mathcal{P} = \{N, H\}$ (where N is the product of two safe primes, and $H : \mathbb{Z} \to \mathbb{Z}_{N^2}^*$ is a cryptographic hash function) and the aggregator secret key sk_B. We recall that according to the description of the JL scheme $\mathsf{sk}_B = -\sum_{i=1}^{n} \mathsf{sk}_i$. Then, $\mathcal{B}$ supplies aggregator $\mathcal{A}$ in our scheme with the public parameters $\mathcal{P} = \{N, H\}$. After receiving $\mathcal{P}$, aggregator $\mathcal{A}$ selects a secret key $\mathsf{sk}_A \in \mathbb{Z}_{N^2}^*$ and for each time interval t it publishes $\mathsf{pk}_{A,t} = H(t)^{\mathsf{sk}_A}$.
- Whenever $\mathcal{A}$ submits a corruption query for some user $\mathcal{U}_i$ to the oracle $\mathcal{O}_{\mathsf{corrupt}}$, $\mathcal{B}$ relays this query to the corruption oracle $\mathcal{O}_{\mathsf{corrupt}}^{\mathsf{JL}}$ of the JL scheme which accordingly returns the secret key sk_i of user $\mathcal{U}_i$.

- Whenever $\mathcal{A}$ calls the encryption oracle $\mathcal{O}_{\mathsf{encrypt}}$ with an encryption query $(t, \mathsf{uid}_i, x_{i,t})$, $\mathcal{B}$ forwards this query to $\mathcal{O}^{\mathsf{JL}}_{\mathsf{encrypt}}$ which returns the matching ciphertext $c_{i,t} = (1 + x_{i,t}N)H(t)^{\mathsf{sk}_i}$ to $\mathcal{B}$. Next, $\mathcal{B}$ provides $\mathcal{A}$ with $c_{i,t}$.
- Whenever $\mathcal{A}$ queries the collection oracle $\mathcal{O}_{\mathsf{collect},\mathcal{A}}$ with time interval t, $\mathcal{B}$ computes $\mathsf{aux}_t = \mathsf{pk}_{A,t}^{-\mathsf{sk}_B}$ which it returns to $\mathcal{A}$. Note that $\mathsf{aux}_t = \mathsf{pk}_{A,t}^{-\mathsf{sk}_B} = H(t)^{-\mathsf{sk}_A\mathsf{sk}_B} = H(t)^{\mathsf{sk}_A \sum \mathsf{sk}_i}$ corresponds to the actual auxiliary information that a collector in our scheme could have computed.

Challenge phase. In the challenge phase, $\mathcal{A}$ chooses a subset $\mathbb{S}^*$ of users that were not compromised and a challenge time interval t^* for which it did not make an encryption query during the learning phase. $\mathcal{A}$ publishes $\mathsf{pk}_{A,t^*} = H(t^*)^{\mathsf{sk}_A}$. $\mathcal{A}$ then submits two time-series $\mathcal{X}^0_{t^*} = (\mathcal{U}_i, t^*, x^0_{i,t^*})_{\mathcal{U}_i \in \mathbb{S}^*}$ and $\mathcal{X}^1_{t^*} = (\mathcal{U}_i, t^*, x^1_{i,t^*})_{\mathcal{U}_i \in \mathbb{S}^*}$ such that $\sum x^0_{i,t^*} = \sum x^1_{i,t^*}$ to $\mathcal{B}$ which simulates oracle $\mathcal{O}_{\mathsf{AO}}$ as follows:

- It submits the time-series $\mathcal{X}^0_{t^*}$ and $\mathcal{X}^1_{t^*}$ to the oracle $\mathcal{O}^{\mathsf{JL}}_{\mathsf{AO}}$ which picks randomly $b \in \{0, 1\}$ and returns the encryption $(c^b_{i,t})_{\mathcal{U}_i \in \mathbb{S}^*}$ for the time-serie $\mathcal{X}^b_{t^*}$.
- Then it computes the auxiliary information $\mathsf{aux}^b_{t^*} = \mathsf{pk}_{A,t^*}^{-\mathsf{sk}_B} = H(t^*)^{-\mathsf{sk}_A\mathsf{sk}_B} = H(t^*)^{\mathsf{sk}_A \sum \mathsf{sk}_i}$ matching the time interval t^*.
- Finally, $\mathcal{B}$ returns $(c^b_{i,t^*})_{\mathcal{U}_i \in \mathbb{S}^*}$ and $\mathsf{aux}^b_{t^*}$ to $\mathcal{A}$.

It is important to notice here that aggregator $\mathcal{A}$ cannot tell whether it is interacting with the actual oracles or with aggregator $\mathcal{B}$ during this simulated game. As a matter of fact, the messages that $\mathcal{A}$ receives during this simulation are correctly computed.

Now at the end of the challenge phase, $\mathcal{A}$ outputs a guess b^* for the bit b. Note that if $\mathcal{A}$ has a non-negligible advantage ϵ in breaking the aggregator obliviousness of our scheme, then this entails that it outputs a correct guess b^* for the bit b with a non-negligible advantage ϵ. Notably, if $\mathcal{A}$ outputs $b^* = 1$, then $(c^b_{i,t^*})_{\mathcal{U}_i \in \mathbb{S}}$ is an encryption of time-serie $\mathcal{X}^1_{t^*}$; otherwise it is an encryption of time-serie $\mathcal{X}^0_{t^*}$. Now to break the aggregator obliviousness of the JL scheme, it suffices that $\mathcal{B}$ outputs the bit b^*.

To conclude, if there is an aggregator $\mathcal{A}$ which breaks the aggregator obliviousness of our solution, then there exists an aggregator $\mathcal{B}$ which breaks the aggregator obliviousness of the JL scheme with the same non-negligible advantage ϵ. This leads to a contradiction under the decisional composite residuosity assumption in $\mathbb{Z}^*_{N^2}$. □

5.2 Collector Obliviousness

Theorem 2. *The proposed scheme assures collector obliviousness in the random oracle model under the decisional composite residuosity (DCR) assumption in $\mathbb{Z}^*_{N^2}$, the quadratic residuosity (QR) assumption in $\mathbb{Z}^*_N$ and the decisional Diffie-Hellman (DDH) assumption in the subgroup of quadratic residues in $\mathbb{Z}^*_N$.*

Proof. Assume there is a collector $\mathcal{C}$ that breaks the collector obliviousness of our scheme with a non-negligible advantage ϵ. We show in what follows that there exists an aggregator $\mathcal{B}$ that uses $\mathcal{C}$ to break the aggregator obliviousness of the JL protocol (which is ensured under DCR) with a non-negligible advantage ϵ'.

To break the aggregator obliviousness of the JL scheme, aggregator $\mathcal{B}$ simulates the collector obliviousness game of our scheme to collector $\mathcal{C}$ as follows:

Learning phase.

- To simulate the oracle $\mathcal{O}_{\mathsf{setup},\mathcal{C}}$ for collector $\mathcal{C}$, $\mathcal{B}$ first queries the oracle $\mathcal{O}^{\mathsf{JL}}_{\mathsf{setup}}$ which returns the aggregator's secret key sk_B and the public parameters $\mathcal{P} = \{N, H\}$ (where N is the product of two safe primes and $H : \mathbb{Z} \rightarrow \mathbb{Z}^*_{N^2}$ is a cryptographic hash function). Then, $\mathcal{B}$ supplies collector $\mathcal{C}$ with the public parameters $\mathcal{P} = \{N, H\}$. Finally, aggregator $\mathcal{B}$ picks randomly $\mathsf{sk}_A \in \mathbb{Z}^*_{N^2}$ and for each time interval t, $\mathcal{B}$ simulates aggregator $\mathcal{A}$ by publishing $\mathsf{pk}_{A,t} = H(t)^{\mathsf{sk}_A}$.
- Whenever $\mathcal{C}$ queries the oracle $\mathcal{O}_{\mathsf{corrupt}}$ for some user $\mathcal{U}_i$, $\mathcal{B}$ forwards the query to the corruption oracle of the JL scheme $\mathcal{O}^{\mathsf{JL}}_{\mathsf{corrupt}}$ which outputs the secret key sk_i of user $\mathcal{U}_i$.
- Whenever $\mathcal{C}$ submits an encryption query $(t, \mathsf{uid}_i, x_{i,t})$ to oracle $\mathcal{O}_{\mathsf{encrypt}}$, $\mathcal{B}$ sends this query to $\mathcal{O}^{\mathsf{JL}}_{\mathsf{encrypt}}$ which returns the matching ciphertext $c_{i,t} = (1+x_{i,t}N)H(t)^{\mathsf{sk}_i}$ to $\mathcal{B}$. $\mathcal{B}$ then provides $\mathcal{C}$ with ciphertext $c_{i,t}$.
- Whenever $\mathcal{C}$ queries the collection oracle $\mathcal{O}_{\mathsf{collect},\mathcal{C}}$ with time interval t, user identifier uid and ciphertext $c_{i,t}$, $\mathcal{B}$ simulates $\mathcal{O}_{\mathsf{collect}}$ as follows:
  - It submits the encryption query $(t, \mathsf{uid}_i, 0)$ to $\mathcal{O}^{\mathsf{JL}}_{\mathsf{encrypt}}$ which returns accordingly $(1 + 0 \cdot N)H(t)^{\mathsf{sk}_i} = H(t)^{\mathsf{sk}_i}$.
  - Then using sk_A it computes $\mathsf{aux}_{i,t} = H(t)^{\mathsf{sk}_i \mathsf{sk}_A}$.

It is noteworthy that the messages that $\mathcal{C}$ received so far are correctly computed. This entails that $\mathcal{C}$ cannot detect during the learning phase that it is interacting with aggregator $\mathcal{B}$.

Challenge phase. In the challenge phase, $\mathcal{C}$ chooses a subset $\mathbb{S}^*$ of users that were not compromised and a challenge time interval t^* for which it did not make an encryption query during the learning phase. Next, $\mathcal{C}$ submits two time-series $\mathcal{X}^0_{t^*} = (\mathcal{U}_i, t^*, x^0_{i,t^*})_{\mathcal{U}_i \in \mathbb{S}^*}$ and $\mathcal{X}^1_{t^*} = (\mathcal{U}_i, t^*, x^1_{i,t^*})_{\mathcal{U}_i \in \mathbb{S}^*}$ to $\mathcal{B}$ which simulates oracle $\mathcal{O}_{\mathsf{CO}}$ as follows:

- It picks time-serie $\mathcal{X}^0_{t^*}$ and generates a new time serie $\mathcal{X}'^1_{t^*} = (\mathcal{U}_i, t^*, x'^1_{i,t^*})_{\mathcal{U}_i \in \mathbb{S}^*}$ such that $\sum x^0_{i,t} = \sum x'^1_{i,t}$ and provides oracle $\mathcal{O}^{\mathsf{JL}}_{\mathsf{AO}}$ with the time series $\mathcal{X}^0_{t^*}$ and $\mathcal{X}'^1_{t^*}$. $\mathcal{O}^{\mathsf{JL}}_{\mathsf{AO}}$ consequently flips a coin $b \in \{0, 1\}$ and returns the tuple of ciphertexts $(c^b_{i,t^*})_{\mathcal{U}_i \in \mathbb{S}^*}$ such that $(c^b_{i,t^*})_{\mathcal{U}_i \in \mathbb{S}^*}$ is an encryption of the time-serie $\mathcal{X}^0_{t^*}$ if $b = 0$; otherwise, it is an encryption of the time-serie $\mathcal{X}'^1_{t^*}$.
- Upon receipt of $(c^b_{i,t^*})_{\mathcal{U}_i \in \mathbb{S}^*}$, $\mathcal{B}$ selects randomly $\mathsf{pk}_{A,t^*} \in \mathbb{Z}^*_{N^2}$, and computes aux^b_{i,t^*} of each user $\mathcal{U}_i \in \mathbb{S}$ by picking a random number $r^b_{i,t^*} \in \mathbb{Z}^*_{N^2}$ and setting $\mathsf{aux}^b_{i,t^*} = r^b_{i,t^*}$.
- Finally, $\mathcal{B}$ gives $(\langle c^b_{i,t^*}, \mathsf{aux}^b_{i,t^*} \rangle)_{\mathcal{U}_i \in \mathbb{S}^*}$ to collector $\mathcal{C}$. It is important to indicate here that under the DDH assumption and the random oracle model, $\mathcal{C}$ cannot detect that pk_{A,t^*} and aux^b_{i,t^*} are generated randomly, instead of being computed as $\mathsf{pk}_{A,t^*} = H(t^*)^{\mathsf{sk}_A}$ and $\mathsf{aux}_{i,t^*} = H(t^*)^{\mathsf{sk}_i \mathsf{sk}_A}$ (cf. Lemma 1).

Now notice that if $b = 0$ and if $\mathcal{C}$ does not detect that $\langle (\mathsf{aux}_{i,t^*})_{\mathcal{U}_i \in \mathbb{S}^*}, \mathsf{pk}_{A,t^*} \rangle$ are generated randomly, then from the point of view of collector $\mathcal{C}$ $(\langle c^b_{i,t^*}, \mathsf{aux}^b_{i,t^*} \rangle)_{\mathcal{U}_i \in \mathbb{S}^*}$ corresponds to a well formed tuple for the time-serie $\mathcal{X}^0_{t^*}$, and as a result, $\mathcal{C}$ will have a

non-negligible advantage ϵ in breaking collector obliviousness of our scheme. Notably, $\mathcal{C}$ will output the correct guess $b^* = 0$ for the bit b with a non-negligible advantage ϵ. In this case, if $\mathcal{B}$ outputs the bit $b^* = 0$ then it will break the aggregator obliviousness of the JL scheme with a non-negligible advantage ϵ.

If $b = 1$, then the tuple $(\langle c^b_{i,t^*}, \mathsf{aux}^b_{i,t^*}\rangle)_{\mathcal{U}_i \in \mathbb{S}^*}$ is independent of the time-series $\mathcal{X}^0_{t^*}$ and $\mathcal{X}^1_{t^*}$ submitted by $\mathcal{C}$. Consequently, $\mathcal{C}$ will return with probability $1/2$ either the bit $b^* = 1$ or the bit $b^* = 0$.

Therefore, to break the aggregator obliviousness of the JL scheme, all $\mathcal{B}$ needs to do is output b^*. □

Lemma 1. *In the random oracle model, collector $\mathcal{C}$ cannot detect that pk_{A,t^*} and $(\mathsf{aux}_{i,t^*})_{\mathcal{U}_i \in \mathbb{S}^*}$ are generated randomly under the decisional composite residuosity (DCR) assumption in $\mathbb{Z}^*_{N^2}$, the quadratic residuosity (QR) assumption in $\mathbb{Z}^*_N$ and the decisional Diffie-Hellman (DDH) assumption in the subgroup of quadratic residues in $\mathbb{Z}^*_N$.*

Due to space limitation, the proof of Lemma 1 can be found in Appendix A.

6 Performance Analysis

6.1 Theoretical Evaluation

Table 1 depicts the theoretical computation and communication cost of our protocol. In each time interval t, aggregator $\mathcal{A}$ first publishes $\mathsf{pk}_{A,t} = H(t)^{\mathsf{sk}_A}$, whereas each user $\mathcal{U}_i$ computes the ciphertext $c_{i,t} = (1 + x_{i,t}N)H(t)^{\mathsf{sk}_i}$, which consists of one exponentiation, one multiplication, one addition and one hash evaluation in $\mathbb{Z}^*_{N^2}$. Users $\mathcal{U}_i$ also perform an additional exponentiation to compute the auxiliary information $\mathsf{aux}_{i,t} = H(t)^{\mathsf{sk}_A \mathsf{sk}_i} \in \mathbb{Z}^*_{N^2}$. Then, the collector receives the auxiliary information $\mathsf{aux}_{i,t}$ $(1 \leqslant i \leqslant n)$ and computes the product $\mathsf{aux}_t = \prod_{i=1}^n \mathsf{aux}_{i,t}$ which calls for $n-1$ multiplications in $\mathbb{Z}^*_{N^2}$. Finally, the aggregator computes the sum $\sum_{i=1}^n x_{i,t}$ by performing $n-1$ multiplications, one exponentiation, one division in $\mathbb{Z}^*_{N^2}$ and one division in $\mathbb{Z}$.

Moreover, if l is the size in bits of N, then each user $\mathcal{U}_i$ sends $2l$ bits for ciphertext $c_{i,t}$ to aggregator $\mathcal{A}$ and $2l$ bits for $\mathsf{aux}_{i,t}$ to collector $\mathcal{C}$. As such, the overall communication cost per user is $4l$ per time interval.

6.2 Implementation

We implemented our scheme in Charm [2,1]. Charm is a programming framework that provides cryptographic abstraction in order to build security protocols. We extended the

Algorithm	Computation	Communication
User	$\mathbf{2}$ EXP $+\mathbf{1}$ MULT $+\mathbf{1}$ ADD $+\mathbf{1}$HASH	$\mathbf{4 \cdot l}$
Aggregator	$\mathbf{2}$ EXP $+\mathbf{2}$ DIV $+(\mathbf{n-1})$ MULT $+\mathbf{1}$HASH	$\mathbf{2 \cdot l}$
Collector	$(\mathbf{n-1})$ MULT	$\mathbf{2 \cdot l}$

Table 1. Performance analysis

N \ Values	[1-10]	[1-100]	[1-1000]
1024	**110.13** μs	**112.23** μs	**114.57** μs
2048	**116.50** μs	**117.15** μs	**118.34** μs
3072	**116.99** μs	**118.23** μs	**120.83** μs

Table 2. Computation overhead of encryption with different security levels and possible plaintext range values. The benchmarks were executed 1000000 times in order to eliminate time inconsistencies due to concurrent CPU usage. Security level parameters were chosen according to ENISA report [4] on recommended sizes for the composite modulus N.

N \ Users	350	700	1000	2500
1024	**0.26** s	**2.40** s	**9.65** s	**49.92** s
2048	**0.65** s	**5.82** s	**24.16** s	**123.19** s
3072	**1.01** s	**9.37** s	**39.34** s	**198.12** s

Table 3. Aggregation time as a function of the size of modulus N and the number of users n.

Charm framework with an implementation of the JL encryption using Python 3.2.3. All of our benchmarks are executed on Intel Core i5 CPU M 560 @ 2.67GHz $\times$ 4 with 8GB of memory, running Ubuntu 12.04 32bit.

To evaluate our scheme empirically, we generated a synthetic dataset with numbers ranging between 1 and 1000 and we varied the size of the modulus N. Table 2 shows the encryption time for different data ranges and different modulus sizes. As expected, a slight increase in the encryption time (which is in the magnitude of microseconds) occurs as we increase the size of N.

We also assessed the computation cost at the aggregator $\mathcal{A}$. More specifically, we measured the time needed to compute the product $P_t = (\prod_{i=1}^{n} c_{i,t})^{\mathsf{sk}_A}$. In table 3 the benchmarks results are shown. The multiplication time was measured in seconds and experiments were conducted for different values of N and the number of users n.

7 Related Work

Önen and Molva [11] introduced a scheme to compute aggregate statistics over wireless sensor networks with multilayer encryption by transforming a block cipher into a symmetrically homomorphic encryption. Even if the proposed solution provides generic confidentiality, the sink-aggregator is fully trusted and shares keys with the sensors. In [5], the authors proposed a protocol for secure aggregation of data using a modified version of Paillier homomorphic encryption. The aggregator which is interested in learning the aggregate sum of data is able to decrypt without knowing the decryption key. The idea behind the scheme is a secret sharing mechanism executed between users such that the aggregation of encrypted data reveals the sum if and only if all users' data is aggregated. However, this scheme suffers from an increased communication cost due to secret share exchange between users. A solution that blends multiparty computation

with homomorphic encryption is also presented in [8], but contrary to our scheme it does not address the issue of dynamic group management.

The authors in [12,3,6] studied privacy preserving data collection protocols with differential privacy. The combination of differential privacy with non conventional encryption schemes can provide an acceptable trade-off between privacy and utility. In [12], a secret sharing mechanism and additively homomorphic encryption are employed together with the addition of appropriate noise to data by the users. Upon receiving the encrypted values a second round of communication is required between users and aggregator to allow for partial decryption and noise cancellation. At the end of the protocol, the aggregator learns the differential private sum. Jawurek and Kerschbaum [6] eliminate this extra communication round between the users and the aggregator by introducing a key manager which unfortunately can decrypt users' individual data.

Chan *et al.* [3] devised a privacy preserving aggregation scheme that computes the sum of users' data, and handles user joins and leaves of smart meters and arbitrary user failures. The decrypted sum is perturbed with geometric noise which ensures differential privacy. Nonetheless, this solution calls for a fully trusted dealer that is able to decrypt users' individual data. The authors in [9] presented a solution to tackle the issue of key redistribution after a user joins or leaves. The propounded solution is based on a ring based grouping technique in which users are clustered into disjoint groups, and consequently, whenever a user joins or leaves only a fraction of the users is affected.

The existing work that resembles the most ours is the work of [13,7]. Actually, Song *et al.* [13] employs an additively homomorphic encryption scheme with differential noise to ensure aggregator obliviousness. The proposed solution is based on a linear correlation between the keys which is known to the untrusted aggregator. However the decrypted sum is encoded as an exponent, thus forcing a small plaintext space. Whereas Joye *et al.* [7] designed a solution that addresses the efficiency issues of [13]. Notably, Joye *et al.* [7] introduced a nifty solution to compute discrete logarithms in composite order groups in which the decision composite residuosity problem is intractable. Still, the scheme in [7] depends on a fully trusted key dealer which renders the scheme impractical for a real world application. Moreover, both schemes do not tackle either the issue of dynamic group management or user failures.

8 Concluding Remarks

In this paper, we presented a privacy preserving solution for time-series data aggregation which contrary to existing work supports arbitrary user failures and does not depend on trusted key dealers. The idea is to rely on a semi-trusted collector which plays the role of an intermediary between the users and the aggregator, and which enables the aggregator to compute the aggregate sum of users' private data without undermining users' privacy. An interesting feature of the proposed scheme is that users' joins and leaves do not incur any additional computation or communication cost at either the users or the aggregator. Furthermore, the scheme is provably privacy preserving against honest-but-curious aggregators and collectors. Finally, initial evaluation results using the Charm cryptographic framework demonstrate the practicality of the propounded solution.

References

1. J. A. Akinyele, M. Green, and A. D. Rubin. Charm: A tool for rapid cryptographic prototyping. http://www.charm-crypto.com/Main.html. 14
2. J. A. Akinyele, M. Green, and A. D. Rubin. Charm: A framework for rapidly prototyping cryptosystems. *IACR Cryptology ePrint Archive*, 2011:617, 2011. http://eprint.iacr.org/2011/617.pdf. 14
3. T.-H. H. Chan, E. Shi, and D. Song. Privacy-preserving stream aggregation with fault tolerance. In *Financial Cryptography*, pages 200–214, 2012. 16
4. ENISA. Algorithms, key sizes and parameters report. https://www.enisa.europa.eu/activities/identity-and-trust/library/deliverables/algorithms-key-sizes-and-parameters-report/at_download/fullReport, Oct 29, 2013. 15
5. Z. Erkin and G. Tsudik. Private computation of spatial and temporal power consumption with smart meters. In *ACNS*, pages 561–577, 2012. 15
6. M. Jawurek and F. Kerschbaum. Fault-tolerant privacy-preserving statistics. In *Privacy Enhancing Technologies*, pages 221–238, 2012. 16
7. M. Joye and B. Libert. A scalable scheme for privacy-preserving aggregation of time-series data. In *Financial Cryptography*, 2013. 1, 2, 3, 4, 5, 7, 8, 9, 16
8. K. Kursawe, G. Danezis, and M. Kohlweiss. Privacy-friendly aggregation for the smart-grid. In *PETS*, pages 175–191, 2011. 16
9. Q. Li and G. Cao. Efficient privacy-preserving stream aggregation in mobile sensing with low aggregation error. In *PETS*, pages 60–81, 2013. 16
10. M. Lisovich, D. Mulligan, and S. Wicker. Inferring personal information from demand-response systems. *Security Privacy, IEEE*, 8(1):11–20, Jan.-Feb. 2
11. M. Önen and R. Molva. Secure data aggregation with multiple encryption. In *Proceedings of the 4th European Conference on Wireless Sensor Networks*, EWSN'07, pages 117–132, Berlin, Heidelberg, 2007. Springer-Verlag. 15
12. V. Rastogi and S. Nath. Differentially private aggregation of distributed time-series with transformation and encryption. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, SIGMOD '10, pages 735–746, New York, NY, USA, 2010. ACM. 16
13. E. Shi, T.-H. H. Chan, E. G. Rieffel, R. Chow, and D. Song. Privacy-preserving aggregation of time-series data. In *NDSS*, 2011. 2, 3, 4, 5, 16

A Proof of Lemma 1

Lemma 1 *In the random oracle model, collector $\mathcal{C}$ cannot detect that* pk_{A,t^*} *and* $(\mathsf{aux}_{i,t^*})_{\mathcal{U}_i \in \mathbb{S}^*}$ *are generated randomly under the decisional composite residuosity (DCR) assumption in $\mathbb{Z}^*_{N^2}$, the quadratic residuosity (QR) assumption in $\mathbb{Z}^*_N$ and the decisional Diffie-Hellman (DDH) assumption in the subgroup of quadratic residues in $\mathbb{Z}^*_N$.*

Let $\mathcal{O}_{\mathsf{DDH}}$ be an oracle which upon a DDH query, first selects randomly g in the subgroup of quadratic residues in $\mathbb{Z}^*_N$ and the pair $(a, b) \in \mathbb{Z}^*_{\phi(N)/4}$ ($\phi(N)$ is the Euler totient of N), then flips a random coin $b \in \{0, 1\}$. If $b = 0$, then $\mathcal{O}_{\mathsf{DDH}}$ selects c randomly from $\mathbb{Z}^*_{\phi(N)/4}$; otherwise, it sets $c = ab$. Finally, $\mathcal{O}_{\mathsf{DDH}}$ returns the tuple (g, g^a, g^b, g^c).

We say that an adversary $\mathcal{B}$ breaks the DDH assumption in the subgroup of quadratic residues, if it can tell whether $g^c = g^{ab}$ or not.

Proof (sketch). Assume there is a collector $\mathcal{C}$ that detects pk_{A,t^*} and $(\mathsf{aux}_{i,t^*})_{\mathcal{U}_i \in \mathbb{S}^*}$ are generated randomly. We show in the random oracle model that there exists an adversary $\mathcal{B}$ that uses collector $\mathcal{C}$ to break DDH in the subgroup of quadratic residues in $\mathbb{Z}_N^*$ under DCR in $\mathbb{Z}_{N^2}^*$ and QR in $\mathbb{Z}_N^*$.

- Let (g, g^a, g^b, g^c) be the DDH tuple provided by $\mathcal{O}_{\mathsf{DDH}}$ to adversary $\mathcal{B}$.
- Let $\mathcal{R}_N$ denote the subgroup of $\mathbb{Z}_{N^2}^*$ defined as $\mathcal{R}_N = \{h^N, h \in \mathbb{Z}_{N^2}^*\}$. We recall that $\mathcal{R}_N$ is of order $\phi(N) = (p-1)(q-1)$ and thus there exists an isomorphism $\rho : \mathbb{Z}_N^* \to \mathcal{R}_N$. Notably, ρ could be defined as: $\forall\, g \in \mathbb{Z}_N^*$, $\rho(g) = g^N \mod N^2$.
- Let $\mathcal{QR}_N$ denote the subgroup of $\mathcal{R}_N$ defined as $\mathcal{QR}_N = \{\tilde{h}^2, \tilde{h} \in \mathcal{R}_N\}$.

Game 0. This game is the collector obliviousness game: Adversary $\mathcal{B}$ executes the setup algorithm by generating the users' secret keys sk_i and the aggregator $\mathcal{A}$'s secret key sk_A and by publishing the public parameters $\mathcal{P} = (N, H)$, where H is a cryptographic hash function $H : \mathbb{Z} \to \mathbb{Z}_{N^2}^*$. By having the user's secret keys sk_i and $\mathcal{A}$'s secret key sk_A, adversary $\mathcal{B}$ can simulate successfully the collector obliviousness game to collector $\mathcal{C}$.

Game 1. This game is identical to the above game except for the following:

- For each time interval t, $\mathcal{B}$ publishes $\mathsf{pk}_{A,t} = H(t)^{\mathsf{sk}_A N} \in \mathcal{R}_N$ instead of $\mathsf{pk}_{A,t} = H(t)^{\mathsf{sk}_A} \in \mathbb{Z}_{N^2}^*$ (i.e., the aggregator's secret key is actually $\mathsf{sk}_A N$ instead of sk_A).
- For each time interval t, $\mathcal{B}$ computes $\mathsf{aux}_{i,t} = (H(t)^{\mathsf{sk}_i})^{\mathsf{sk}_A N} \in \mathcal{R}_N$ instead of $\mathsf{aux}_{i,t} = (H(t)^{\mathsf{sk}_i})^{\mathsf{sk}_A} \in \mathbb{Z}_{N^2}^*$.

Note that under the DCR assumption, $\mathcal{C}$ cannot tell whether $\mathsf{pk}_{A,t}$ and $\mathsf{aux}_{i,t}$ are in $\mathcal{R}_N$ or not, and accordingly, Game 0 and Game 1 are computationally indistinguishable.

Game 2. In this game, we compute $\mathsf{pk}_{A,t}$ as $H(t)^{2\mathsf{sk}_A N} \mod N^2$ and $\mathsf{aux}_{i,t}$ as $(H(t)^{\mathsf{sk}_i})^{2\mathsf{sk}_A N} \mod N^2$. Note that under the quadratic residuosity assumption in $\mathbb{Z}_N^*$, Game 1 and Game 2 are computationally indistinguishable. Indeed, if there is a distinguisher $\mathcal{D}$ that is able to tell for instance whether $\mathsf{pk}_{A,t}$ is an element of $\mathcal{QR}_N$ or not, then $\mathcal{D}$ can be used to break the quadratic assumption in $\mathbb{Z}_N^*$ by employing the isomorphism $\rho : \mathbb{Z}_N^* \to \mathcal{R}_N$. Namely, given an element $g \in \mathbb{Z}_N^*$, one computes $\hat{h} = \rho(g)$ and submits $\hat{h}$ to $\mathcal{D}$. If $\mathcal{D}$ outputs that $\hat{h}$ is of the form $\tilde{h}^2$ ($\tilde{h} \in \mathcal{R}_N$), then one outputs that g is quadratic residue in $\mathbb{Z}_N^*$.

Game 3. This game is identical to Game 2 except that this time adversary $\mathcal{B}$ controls a random oracle $\mathcal{H}$ and instead of generating the secret key sk_A randomly in $\mathbb{Z}_{N^2}^*$, it sets $\mathsf{pk}_A = g^{aN} \mod N^2$ and uses the random oracle to simulate that it possesses the secret key $\mathsf{sk}_A = aN$. We recall that (g, g^a, g^b, g^c) is the DDH tuple that adversary $\mathcal{B}$ received from $\mathcal{O}_{\mathsf{DDH}}$.

Without loss of generality, we assume that collector $\mathcal{C}$ makes q hash queries to the random oracle $\mathcal{H}$.

Random Oracle Simulation. To answer the queries of the random oracle $\mathcal{H}$, adversary $\mathcal{B}$ keeps a table T_H of tuples $(t_i, r_i, \mathsf{coin}_i(t), H(t_i))$ as explained next. On a query $H(t)$ to $\mathcal{H}$, adversary $\mathcal{B}$ replies as follows:

- If there is a tuple $(t, r, \mathsf{coin}(t), H(t))$ that corresponds to t, then $\mathcal{B}$ returns $H(t)$.
- If t has never been queried before, then $\mathcal{B}$ picks a random number $r \in [0, N/4]$, and flips a random coin $\mathsf{coin}(t) \in \{0, 1\}$ such that: $\mathsf{coin}(t) = 1$ with probability p, and it is equal to 0 with probability $1 - p$. If $\mathsf{coin}(t) = 0$, then $\mathcal{B}$ answers with $H(t) = g^{rN} \mod N^2$. Otherwise, it answers with $H(t) = g^{rbN} \mod N^2$. Finally, adversary $\mathcal{B}$ stores the tuple $(t, r, \mathsf{coin}(t), H(t))$ in table T_H.

Suppose that $\mathsf{coin}(t^*) = 1$, then $H(t^*)$ is of the form $g^{r^* bN}$. Accordingly, $\mathcal{B}$ simulates the oracle $\mathcal{O}_{\mathsf{CO}}$, by computing $\mathsf{pk}_{A,t^*} = g^{r^* cN} \mod N^2$ and $\mathsf{aux}_{i,t^*} = g^{r^* \mathsf{sk}_i cN} \mod N^2$. Note that in the case where $c = ab$, then $\mathsf{pk}_{A,t^*} = H(t^*)^{\mathsf{sk}_A}$ and $\mathsf{aux}_{i,t^*} = H(t^*)^{\mathsf{sk}_i \mathsf{sk}_A}$, and as a result, collector $\mathcal{C}$ continues the collector obliviousness game. However, if $c \neq ab$ and if $\mathcal{C}$ has a non-negligible advantage ϵ in detecting that pk_{A,t^*} and aux_{i,t^*} are randomly generated, then $\mathcal{C}$ aborts the game with non-negligible advantage ϵ. Therefore, to break the DDH assumption, $\mathcal{B}$ outputs 1 when collector $\mathcal{C}$ continues the collector obliviousness game; and outputs 0 otherwise.

We remark here that the event $\mathsf{coin}(t^*) = 1$ occurs with probability $\Pi = p(1 - p)^{q-1}$ and that this probability is maximal when $p = 1/q$ and it equals to $\Pi_{\mathsf{max}} \simeq \frac{1}{eq}$. □