NIKE-based Fast Privacy-preserving High-dimensional Data Aggregation for Mobile Devices

Kalikinkar Mandal
University of Waterloo
Waterloo, Ontario, Canada
kmandal@uwaterloo.ca

Guang Gong
University of Waterloo
Waterloo, Ontario, Canada
ggong@uwaterloo.ca

Chuyi Liu
University of Waterloo
Waterloo, Ontario, Canada
c436liu@uwaterloo.ca

ABSTRACT

Mobile apps are widely used for personalization and improvement of user experiences. These apps have access to a large amount of sensitive data on which learning algorithms such as machine learning (ML) algorithms are applied locally. Moreover, apps share user data and locally-computed models. Aggregate sum is a building block for many ML algorithms such as deep learning, least-squares linear fit, and linear regression. In this paper, we present a system for secure aggregate sum computation in which the server wishes to periodically compute the aggregate sum of a set of mobile users' private inputs without compromising the individual inputs.

As the aggregate sum operation is performed often, we devise the aggregation scheme to be communication and computation efficient. First, we present a non-interactive pairwise key generation scheme for mobile users where the non-interactivity among users is achieved by outsourcing the keying material generation task to two non-colluding cryptographic secret providers. Second, we design an efficient aggregate sum scheme that has low communication and computation overheads and the failure-robust property, which improves over Bonawitz et al.'s scheme (CCS 2017). The security and performance of the scheme are analyzed, followed by comparing it with existing ones. We implement our scheme on a smartphone as well as on a desktop to measure its efficiency. We conduct experiments on high-dimensional inputs, and our experimental results demonstrate about 1.5x to 3x improvement over existing schemes. Energy consumption results of our scheme on smartphone are presented.

KEYWORDS

Privacy-preserving computation, Data aggregation, ML Security, NIKE

1 INTRODUCTION

In the last few years, the development and use of smartphones, wearables, and tablets applications received growing attentions, and these markets are extensively increasing. Mobile applications use a large amount of sensitive data such as users' (contact) info and locations, healthcare data, banking payment information, and bioinformatics. For instance, there are more than 40,000 healthcare apps available for download from the Apple app store [17]. Cellphone users and navigation maps share their location data to avoid traffic jam in a city. Many apps also run on the cloud server, which is potentially untrusted, to minimize the storage and resource consumption of mobile devices. Sending such sensitive information without protection to the server is a potential security threat to the users. The server collects an enormous amount of useful data generated by multiple mobile devices and applies learning algorithms such as machine learning and artificial intelligence on them to analyze and make predictions for improving mobile users' experiences. The server may use Machine Learning as a Service (MLaaS) from other service providers and apply it on the collected data.

Modern mobile devices have access to a large data subtable for learning models. In [7], the authors discussed a model where the sensitive data remains at a mobile device which processes some ML or deep learning algorithms locally (i.e., the training data distributed in mobile devices), and learns a shared model by aggregating locally-computed updates. In particular, the aggregate sum is a useful primitive that is used in machine learning algorithms (e.g., gradient descent and logistic regression) and modeling, statistical analysis, and aggregation queries. The prediction accuracy of machine learning algorithms depends on the availability of adequate training data which is collected from various sources such as mobile devices. Applications such as time-series data and streaming data may need to perform the aggregate-sum computation frequently. For example, the recommender system is an important tool to enhance services for mobile applications, which employs machine learning algorithms such as logistic regression, SVM, deep learning, least-squares linear fit [41, 48]. Although data and model sharing improves the user experiences, the privacy of individual users' data and model can compromise sensitive information. One key challenge in these applications is to maintain the privacy of the user's data while making the data usable to an untrusted server.

Secure data aggregation problem has been studied in the literature from different applications such as wireless sensor network (WSN) data aggregation [11, 42, 43], smart meter data collection [1, 32], mobile users data collection [7, 21], to name a few. For smart meter data collection, the aggregate sum prevents leaking information about individual smart meter data usage [1]. For the WSN data collection, data aggregation techniques exploit the network structure to reduce resources such as power consumption. Due to various constraints from different applications, the techniques for aggregation of WSN data are different from the ones for smart meter applications. In this work, we consider secure aggregation of data from mobile devices for the use in machine learning. We consider a scenario where there is a set of mobile users and a (cloud) server that wishes to obtain an aggregate-sum of inputs of users present in the network. The mobility nature of users introduces a new challenge, called *robustness* or *failure-robust*, to the aggregate-sum computation of present users' inputs, which is also considered in [7]. Since mobile users join and drop out from the system, the communication among themselves is a limiting factor.

An ideal solution to the secure aggregation is to have a trusted aggregator that collects inputs from users and computes the aggregate sum and sends to the server. In the real world, it is hard to exist such trusted entity. Encryption-based solutions provide stronger privacy guarantees to the inputs, but sending an encrypted input directly to the server is not secure as the server needs to decrypt it to perform the aggregate sum. The solutions where users encrypt inputs using a threshold additive homomorphic encryption scheme and send the ciphertexts to the server are challenging for high-dimensional data. Particularly, the server multiplies the ciphertexts and obtains an encryption of the aggregate sum of user inputs. Then the server sends the ciphertext for a distributed decryption. There two main drawbacks with this approach: (1) the server may cheat here by adding some random number to an encryption of one user's input before sending for decryption; (2) the distributed decryption among the users and/or the server involves at least one round of communication. Considering the user drop out scenario, it is not a robust solution. In designing cryptographic protocols, often the tradeoff between the computation workload and the communication overhead arises.

Existing solutions to the aggregate sum protocols, e.g., in [1, 7], rely on the Diffie-Hellman (DH) key exchange protocol for establishing pairwise keys to mask user inputs. The inputs are masked in such a way that the server can compute the aggregate sum without learning anything about user inputs. The DH key establishment process involves a quadratic communication complexity in the number of users. That means, each aggregate-sum computation needs a quadratic communication in the network. This is not efficient for applications such as time series data and streaming data where ML algorithms need to perform multiple aggregate-sum computations. Another instance, the model coefficients of least-squares linear fits can be computed by a server on a set of user inputs using multiple aggregate-sums. A major drawback with the former solutions is that the pairwise key establishment phase is communication heavy. In other words, all the users need to be online simultaneously, and periodically establishing pairwise keys, which may be infeasible for a large-scale application. In [7], a user input is doubly masked by a user chosen key and the sum of pairwise keys with all other users present in the network where the user-chosen key and the DH key for the pairwise keys are secretly shared among all other users using a threshold secret sharing scheme to handle the failure recovery case. For a large-scale network, each execution of a threshold secret sharing operation is cost effective and the computation of additive noises from pairwise keys for high-dimensional data is an expensive operation.

Our goal in this paper is to design a private data aggregation protocol that is communication and computation efficient and suitable for frequent aggregate-sum computation applications. A communication efficient solution to this problem is to deploy a non-interactive key exchange (NIKE) protocol. Our solution for the pairwise key exchange phase is to use a NIKE scheme that is outsourced to a pair of non-colluding cryptographic secret service providers, instead of running the protocol by themselves in a distributed manner, meaning the users do not need to interact with each other, they communicate with the secret service providers only once when they join the system in offline, and obtain two pairs of key computation materials or when keying materials need to be refreshed. This makes the communication costs among users zero. Our construction of encoding the user inputs is based on the compilation of masking/encryption ideas in [1, 7]. We devise the construction for encoding the high-dimensional inputs to be more computation efficient as well as robust towards the failure-recovery scenario. We make this operation, called *mask-then-encrypt*, lighter by introducing an ℓ-regular (logical) communication network maintained by the server and by secretly sharing one-time pairwise keys among three entities, namely two holders of the pairwise key and the server. We also reduce the computational complexity of the server for the choice of the secret sharing scheme for one-time pairwise keys. Depending upon the machine learning applications, the aggregation protocols with high-dimensional inputs are of practical significance, for instance, many times, low-dimensional inputs can be translated to high-dimensional for fast and parallel execution of multiple data like SIMD-style. For details, see Appendix D.

Contributions. While keeping in mind the goal of developing a secure aggregate sum scheme that is both communication and computation efficient, we make the following contributions in this paper.

- **Non-interactive key establishment protocol.** We propose a non-interactive pairwise key establishment scheme for mobile users where the non-interactivity among users is achieved by outsourcing the key material generation task to two non-colluding cryptographic secret service providers (SSPs). For this, we develop a two-party protocol for securely generating a pair of secret shared polynomials over $\mathbb{Z}_N[x]$ for each user where N is a RSA modulus and each secret shared polynomial $f(x)$ is a triplet $\big(f_1(x), f_2(x), f(x)\big) \in (\mathbb{Z}_N[x])^3$ such that $f(x) = f_1(x)f_2(x)$, which is shared between two SSPs. This results in a zero communication cost for generating pairwise keys, which saves $O(m^2)$ communication overhead over network, but increases $O(\log^2(m))$ computation overhead per user where m is the number of users. We formalize the security of the protocol by considering three different security models and discuss its efficiency.
- **NIKE-based secure aggregation protocol.** We design a secure and efficient (aggregate-sum) aggregation protocol that has low communication and computation overheads, and has the failure-resistance property suitable for mobile applications with unstable network. In our scheme, privacy of the users' inputs is provided by a *mask-then-encrypt* operation, which is constructed as follows: A user's private input is first masked by adding *coordinated noises* derived from one-time pairwise keys with its neighboring users chosen based on an ℓ-regular network, and then the masked input is encrypted by a freshly chosen key. To achieve the failure-recovery property, one-time pairwise keys are secret-shared among three entities, namely the two holders of the pairwise key and the server using a 2-out-of-3 threshold secret sharing scheme ($(2, 3)$-tss), and the decryption key is secret-shared with all other users using a (t, m)-tss, which improves the state-of-art scheme. Security of our scheme under three different threat models is proved in the simulation paradigm. We present the performance of our scheme and compare with existing most related schemes.

- **Experimental evaluation and comparisons.** We implement our scheme on a smartphone in Java as well as on a desktop in C using OpenSSL, GMP and FLINT libraries to demonstrate its efficiency. We benchmark and compare the performance of the user's mask-then-encrypt operation and the server's decrypt-then-unmask operation on high-dimension inputs. Our experimental results show that our scheme is about 1.5x to 3x faster than existing ones for three different threat models. Energy consumption of cryptographic operations performed by a mobile user is presented.

2 PRELIMINARIES

In this section, we provide a brief background on the cryptographic primitives that we will use in the proposed secure aggregate sum protocol.

2.1 PRG and KDF

A pseudorandom generator (PRG), PRG : $\{0,1\}^n \rightarrow \{0,1\}^l$, is a deterministic function which accepts an n-bit binary string as input and outputs an l-bit binary string where l is much larger than n. The input to a PRG is chosen uniformly at random. The security of a cryptographically strong PRG is defined as its output bit stream is indistinguishable from a truly random one. A key derivation function (KDF) accepts an initial keying material and outputs one or more cryptographically strong keys. Many applications and protocols need KDFs to derive keying materials. Examples of the KDF constructions are HMAC-based KDF and extract-then-expand KDF [30]. We use a KDF in our scheme to derive keying materials.

2.2 Secret Sharing

Let $\mathbb{F}_q$ be a finite field where q is a prime. A t-out-of-m threshold secret sharing scheme ((t,m)-tss) over $\mathbb{F}_q$ is a tuple of algorithms $(\mathbf{Share}^t_m, \mathbf{Rec}^t_m)$ where the secret sharing algorithm $\mathbf{Share}^t_m$ is a randomized algorithm that takes an input message s over $\mathbb{F}_q$ and outputs m shares of s i.e., $(s_1, s_2, \ldots, s_m) \leftarrow \mathbf{Share}^t_m(s)$ and the reconstruction algorithm $\mathbf{Rec}^t_m$ accepts any t shares $(s_{i_1}, s_{i_2}, \cdots, s_{i_t})$ as input and outputs the original s. The correctness of the scheme is defined as for any s, $\Pr_{(s_1,\cdots,s_m)\leftarrow\mathbf{Share}^t_m(s)}[\mathbf{Rec}^t_m(s_{i_1}, s_{i_2}, \cdots, s_{i_t}) = s] = 1$. The security of the scheme is defined as any set of less than t users can learn nothing about the secret s. For any two secrets s and s' and $C \subset [m]$ with $|C| < t$, the following distributions are computationally indistinguishable $\{(s_i : i \in C) : (s_1, \cdots, s_m) \leftarrow \mathbf{Share}^t_m(s)\} \approx_c \{(s'_i : i \in C) : (s'_1, \cdots, s'_m) \leftarrow \mathbf{Share}^t_m(s')\}$. We use Shamir's threshold secret sharing scheme [44] and a 2-out-of-3 secret sharing scheme $(\mathbf{Share}^2_3, \mathbf{Rec}^2_3$) in our protocol. A verifiable secret sharing (VSS) of a (t,m)-tss has two additional algorithms, namely $\mathbf{Comm}^t$ that commits the random polynomial used in $\mathbf{Share}^t_m$, and $\mathbf{SVerify}^t$ that verifies the validity of a share produced by $\mathbf{Share}^t_m$. We use Feldman's VSS scheme [16] with EC group in our protocol and its security is based on the discrete logarithm problem.

2.3 Authenticated Encryption

An authenticated encryption (AE) scheme ($\mathcal{AE}$) consists of three algorithms $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ where $\mathcal{K}$ is a key generation algorithm that outputs a symmetric key, $\mathcal{E}$ is a symmetric-key encryption algorithm that accepts a key and a message as input and outputs a ciphertext and a tag, and $\mathcal{D}$ is a decryption algorithm that takes a key and a ciphertext and outputs the original message or a special symbol. We use an AE scheme $\mathcal{AE}$ that offers the IND-CPA and INT-CTXT securities.

2.4 One-way Function

A one-way function $F : \{0,1\}^k \rightarrow \{0,1\}^n$ is a function that is easy to compute and hard to invert. The security of the one-way function F is the amount of work needed to invert F on an input x, which is defined as given $F(x)$, there exists a PPT adversary $\mathcal{A}$ such that

$$\Pr[x \in \{0,1\}^k; x' \leftarrow \mathcal{A}(1^k, F(x)|F(x) = F(x'))] = \mathsf{negl}(k)$$

where $\mathsf{negl}(\cdot)$ is a negligible function. We need an additional property of the one-way function that the output distribution of F to indistinguishable from a random one. We use a cryptographic hash function to realize a one-way function in our scheme.

2.5 PKI Infrastructure

In our system, each user, each SSP and the server have a pair of secret and public keys and a certificate of the public key. Each entity will register to the system when they join first time. Users and SSPs have preshared or pre-established symmetric keys, and the users and the server also have pre-shared or pre-established symmetric keys that are used only for establishing a secure channel between a pair of entities. Users and the SSPs or users and the server are also allowed to establish a shared key using the DH protocol for establishing a secure channel. We emphasize that these keys are not used to mask users' private inputs.

Al the entities in the system receive a pair of signing key and verification key uniquely binded to their unique identity from a trusted third party. Denote by $(\mathsf{sk}_{U_i}, \mathsf{pk}_{U_i})$, $(\mathsf{sk}_S, \mathsf{pk}_S)$, and $(\mathsf{sk}_{SSP_i}, \mathsf{pk}_{SSP_i})$ the signing and verification keys of the user U_i, the server, and the SSP_i, respectively.

3 THE MODEL AND PROBLEM STATEMENT

In this section, we describe the system model, which consists of a set of mobile users, two secret service providers (SSPs) and a (cloud) server where the server collects data from users for computing the aggregate sum. Due to mobility nature, the mobile users have one of two statuses: present (alive) in or drop out from the network.

3.1 The System Model

In our system, mobile users have sensitive data and wish to share their data with the server in from of an aggregate sum with other users' data. Assume that there are m users (mobile devices) in the system, which are denoted by $\mathcal{U} = \{U_1, U_2, \cdots, U_m\}$ and each user has a unique identity in $[m] = \{1, \cdots, m\}$. Each user holds an r-dimensional data, denoted by $\mathbf{x}_i = (x_{i,1}, x_{i,2}, \cdots, x_{i,r}) \in \mathbb{Z}^r_{2^n}$. The goal of the system is to allow the server to learn only the aggregate-sum $\sum_i \mathbf{x}_i$ of present users' inputs, nothing else. Figure 1 depicts an overview of the system.

3.1.1 Distributed Secret Service Provider. Cryptographic secret service providers (SSPs) supply secret credentials to the users for deriving pairwise keys. Distributing private data, secret parameters of cryptographic algorithms, passwords, and secret keying materials

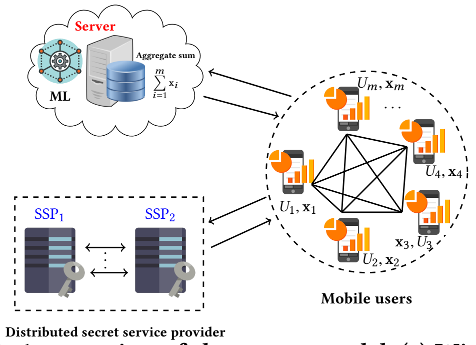


Figure 1: An overview of the system model. (a) Within the circle, the mobile users communicate each other to share keying materials among themselves. (b) Cryptographic secret service providers perform a joint computation to derive keying materials that are used by the mobile users. (c) Server computes an aggregate sum, which is used in an ML algorithm.

is a widely used assumption in many systems [12, 34, 37]. The SSPs perform a joint computation to generate a pair of secret shared polynomials for each user. This can be viewed as a distributed SSP. The difference between our distributed SSP and the distributed key distribution center in [6, 35] is that in our case all SSPs need to participate to compute all users' shares, but for the key distribution centers used in [6, 35], the security of the system is not affected if not all SSPs participate. Each user with a unique identity is allowed to register only once and the SSPs maintain a common identity table. When a user joins the system, each user sends a request to both SSPs, and in response, each SSP sends a secret shared polynomial to the user. The user can then derive pairwise keys using received secret shared polynomials. To refresh secret shared polynomials or in the event of losing, the SSPs generate fresh secret shared polynomials for all users. We assume that the users have high-bandwidth in the offline phase and in that phase they receive secret shared polynomials from the SSPs. We emphasize that as the SSPs are offline entities, they do not participate in the aggregate-sum protocol.

3.1.2 Communication Model. We consider the communication network model defined in Definition 3.1 where each node (user) has its neighboring nodes' information, which is a logical network maintained by the server. Note that given ℓ and m, there exists an ℓ-regular graph with m vertices if ℓ or m is even, which follows from the degree sum formula.

Definition 3.1 (ℓ-regular communication network). We define a graph for the communication network that will be used in our aggregate-sum scheme.

- $\mathcal{U} = \{U_1, U_2, \cdots, U_m\}$ is the set of users (nodes) with a unique identifier. We denote $\mathrm{id}(U_i) = i \in [m]$ by the identifier of the user U_i.
- Each user U_i has a set of neighbors, denoted by $\mathcal{N}(U_i)$, to which the node U_i can communicate. Moreover, it can communicate with any other nodes in the network. There exists an undirected edge between user U_i and each user in $\mathcal{N}(U_i)$.
- Each user has exactly ℓ neighbors, i.e., $|\mathcal{N}(U_i)| = \ell$ for all $i \in [m]$.

If $\mathcal{L}$ is the set of current users in the system, we denote by $\mathcal{N}_{\mathcal{L}}(U_i)$ the set of all neighboring users of U_i currently present in the system.

3.2 Problem Statement

Assume that there are m mobile users in the system. Let $\mathcal{D}$ be the set of all drop out users and $\mathcal{L}$ be the set of all alive users at the present time where $\mathcal{D} \cap \mathcal{L} = \phi$ and $\mathcal{D} \cup \mathcal{L} = [m]$.

Aggregate-sum Functionality with Robustness: As we considered an unstable network scenario for mobile users, the goal of the server is to learn the aggregate sum of all present user's inputs, i.e., $\mathsf{SUM}_{\mathcal{L}} = \sum_{U_i \in \mathcal{L}} \mathbf{x}_i$ for $\mathcal{L} \subseteq [m]$. The mobile users encode their inputs before sending to the server and we denote the encoded input by $\boldsymbol{\alpha}_i$ and the encoding is done in such a way that the server can learn only the aggregate sum. When all the users present in the system and the server receives $\boldsymbol{\alpha}_i$ from all U_i, it can compute the sum $\sum_{i=1}^{m} \boldsymbol{\alpha}_i$ and learn $\sum_{i=1}^{m} \mathbf{x}_i = \sum_{i=1}^{m} \boldsymbol{\alpha}_i$. For a set of drop out users $\mathcal{D} \subset [m]$ from the network, the robustness is defined as the ability of computing $\sum_{i \in \mathcal{L}} \mathbf{x}_i$ by the server from $\{\boldsymbol{\alpha}_i\}_{i \in \mathcal{L}}$ for alive users, without user U_j's input $j \in \mathcal{D}$ when $|\mathcal{D}|$ is large.

Security: The privacy of user inputs $\mathbf{x}_i$ should be protected against the server and other users. If an adversary controls the server and some number of users C, but neither of SSPs, then no matter how the adversary behaves, it learns nothing about any individual input $\mathbf{x}_i$, except the aggregate sum $\sum_{i \in \mathcal{L} \setminus C} \mathbf{x}_i$.

Communication: There are two types of communications the users do with other users and the server for computing the aggregate sum in the online phase. Assume that in the online phase the users have low-bandwidth to other users and the server. One is a communication between any two users. The other one is a communication between the server and a user. The communications among the users should be minimal, even in the event of dropping out of users.

3.3 Adversarial Model

We assume that the SSPs are two non-colluding entities and the users trust them. The adversary can control the server and a set of alive users in the system to behave them on its wishes. The adversary can control one of the secret service providers, but not both of them. If the adversary can control both SSPs, the system is not secure. We consider the *static* adversaries who choose the users at the beginning of the execution of the protocol. We consider *honest-but-curious (HBC or semi-honest)* who follow and observe the execution of the protocol. A semi-honest adversary never provides or injects false information during the execution of the protocol. It only collects information from honest users and try to learn private inputs $\mathbf{x}_i$'s of honest users from collected information. We also consider *active adversaries* who follow and observe the execution of the protocol, can abort anytime, and may inject false messages. Our system has three security goals: (1) the adversary can control only a set of users and try to learn about other users' inputs, we call it *the user-only adversary*; (2) the adversary can control a set of users and the server and may try to learn other users' inputs, we call it *the user-server adversary*; and (3) the adversary can control only the server and may try to learn an honest user's input from the protocol, we call it *the server-only adversary*. There are certain actions for active adversaries, specifically, refuse to participate the protocol, may change private inputs, and may abort the protocol prematurely cannot be prevented.

4 OUR NON-INTERACTIVE PAIRWISE KEY ESTABLISHMENT SCHEME

Establishing pairwise keys between each pair of users is an expensive task, especially for mobile devices, in terms of both communication and computation complexities. Our idea is to reduce mobile devices' task by outsourcing the pairwise key generation task of the users to a cryptographic service provider, which is realized by two non-colluding secret service providers. As a result, the users do not need to communicate each other for establishing pairwise keys. In this section, we prepose a new two-server aided NIKE scheme and formalize its security.

Basic Idea. Our idea is to use a bivariate polynomial to derive pairwise keys among users in a non-interactive way where a univariate polynomial is secret shared between two SSPs, namely SSP_1 and SSP_2. We use a special bivariate polynomial that product of a univariate polynomial with two independent variables, which is also used in [24]. The reason for employing a distributed SSP is to prevent a single point security failure and improve the security by distributing the secrets for deriving the keys. Our technique for generating a common pairwise key $MK_{i,j}$ between U_i and $U_j, j \in [m], j \neq i$ is that the user U_i uses a secret shared polynomial $g_i(x)$ to derive all pairwise keys with others. The polynomial-based key generation technique is efficient, compared to pairing-based approaches [18, 40] for mobile users.

Our non-interactive pairwise key computation (NPKC) scheme consists of a tuple of three algorithms:

- $(sp, pp) \longleftarrow$NPKC.DistRSAParam(1^v): On security parameter v, the randomized algorithm involving two SSPs outputs some private parameters (sp) and public parameters (pp) for the RSA scheme where $pp = (pp_1, pp_2)$ and $sp = (sp_1, sp_2)$.
- $\left\{(s_{1,j}(x), s_{2,j}(x)) : j \in [m]\right\} \longleftarrow$NPKC.DistSP$(sp, pp, k, m)$: It is a randomized algorithm that outputs m pairs of random secret share polynomials of degree $(k-1)$ for m users $U_1, \cdots, U_m$. Two SSPs jointly construct m pairs of secret shared polynomials over $\mathbb{Z}_N$ using their private and public parameters. Each SSP_i sends to each user U_j a secret share polynomial $s_{i,j}(x)$, $i = 1, 2$ and $j \in [m]$.
- $MK_{i,j} \longleftarrow$NPKC.CompKey$(s_{1,i}(x), s_{2,i}(x), j, pp)$: Each user U_i computes a pairwise-key ($MK_{i,j}$) with U_j by multiplying and evaluating the received pair of secret shared polynomials at j.

The correctness of the scheme is defined as the pairwise key computation for any pair of users should be the same key, and the privacy is defined as any set of less than k colluding users should not learn any other pair of users key, which are formalized below.

Definition 4.1 (Correctness). An NPKC scheme is *correct* if for any pairwise users $(U_i, U_j), i \neq j$

$$\Pr\Big[\text{NPKC.KeyComp}(s_{1,i}(x), s_{2,i}(x), j, pp) = \text{NPKC.KeyComp}(s_{1,j}(x), s_{2,j}(x), i, pp)\Big] = 1.$$

Definition 4.2 (Security against Users). An NPKC scheme is *k-private* if for all $C \subset \mathcal{U}$ with $|C| < k$ and $u, v \notin C$, there exists a probabilistic polynomial time (PPT) algorithm $\mathcal{A}$ such that

$$\left\{\mathcal{A}\Big((s_{1,i}(x),\ s_{2,i}(x))_{i \in C},\ C,\ pp,\ m,\ \{u,\ v\}\Big)\right\} \approx \left\{(s_{1,b}(x),\ s_{2,b}(x))_{b \in \{u,v\}}\right\}$$

where $(sp, pp) \longleftarrow$NPKC.DistRSAParam(1^v), $\{(s_{1,i}(x), s_{2,i}(x)) : i \in [m]\} \longleftarrow$NPKC.DistSP$(sp, pp, k, m)$ and $\approx$ denotes the distributions are identical.

4.1 Construction of NPKC

Construction in a nutshell. Since the pairwise keys will be generated using a secret shared polynomial $g_i(x)$, a single SSP should not generate the pairwise keys. For simplicity, we assume that the degree of $g_i(x)$ is $2(k-1)$ with $k < m$.

(1) **NPKC.DistRSAParam:** Let $N = PQ$ be a strong RSA modulus where $P = 2P' + 1$ and $Q = 2Q' + 1$ are safe primes, and P' and Q' are primes. The SSPs jointly generate the safe primes P and Q where each SSP holds a share of P and Q. Denote $e = m - 1$ if m is even and $e = m$ if m is odd. Note that $\phi(N) = 4P'Q'$ and $\gcd(m-1, \phi(N)) = 1$ when m is even and $\gcd(m, \phi(N)) = 1$ when m is odd, and m is less than P' and Q'. Then the public and private parameters are $pp_1 = pp_2 = (N, e)$ and $sp_i = (P_i, Q_i, \phi_i)$, $i = 1, 2$ where P_i and Q_i are chosen uniformly at random such that $P = P_1 + P_2$, $Q = Q_1 + Q_2$, and $\phi(N) = \phi_1 + \phi_2$.

(2) **NPKC.DistSP:** The SSPs jointly compute $d = \frac{1}{e} \bmod \phi(N)$ where SSP_i holds d_i, $i = 1, 2$ and $d = d_1 + d_2$.
- Let SSP_i generate the polynomial $f_i(x)$ over $\mathbb{Z}_N[x]$, $i = 1, 2$ where the degree of $f_i(x)$ equals $(k-1)$ and the coefficients of $f_i(x)$ are non-one and generated independently with random distribution.
- For each user U_j, SSP_i jointly computes $s_{i,j}(x) = f_i(j)^d f_i(x)$, $j \in [m]$ and $i = 1, 2$.

(3) **NPKC.CompKey:** After receiving $(s_{1,i}(x), s_{2,i}(x))$ from SSP_1 and SSP_2, respectively, U_i computes $g_i(x) = s_{1,i}(x)s_{2,i}(x) = f_1(i)^d f_1(x) f_2(i)^d f_2(x) = f(i)^d f(x)$ where $f(x) = f_1(x)f_2(x)$. It computes a key $MK_{i,j} = \text{KDF}(g_i(j) g_i(0)^{e-1}) \in \mathbb{F}_q$.

Note that $f(x)$ is multiplicatively shared between two SSPs and its degree is $2(k-1)$ as each $f_i(x)$ is of degree $(k-1)$. The secret shared polynomials $\big(g_i(x), i \in [m]\big)$ are derived by multiplicatively masking the coefficients of a random polynomial $f(x)$ with a RSA-like ciphertext whose encryption key is shared between two SSPs. Notice that **NPKC.DistSP** is run only once when SSPs do the system setup in the offline phase.

Proposition 4.3 (Correctness). *For the above construction of $g_i(x)$, the pairwise keys derived by users U_i and U_j are equal, i.e., $MK_{i,j} = MK_{j,i}$ where $MK_{i,j} = KDF\big(g_i(j)\ g_i(0)^{e-1}\big)$ and $MK_{j,i} = KDF\big(g_j(i)\ g_j(0)^{e-1}\big)$.*

Proof. The pairwise key between U_i and U_j (resp. U_j and U_i) is derived as $MK_{i,j} = \text{KDF}\big(g_i(j) g_i(0)^{e-1}\big)$ (resp. $MK_{j,i}$), which can be written as

$$\begin{aligned} MK_{i,j} &= \text{KDF}\big(g_i(j)g_i(0)^{e-1}\big) & MK_{j,i} &= \text{KDF}\big(g_j(i)g_j(0)^{e-1}\big) \\ &= \text{KDF}\big(f(i)^d f(j)(f(i)^d f(0))^{e-1}\big) & &= \text{KDF}\big(f(j)^d f(i)(f(j)^d f(0))^{e-1}\big) \\ &= \text{KDF}\big(f(i)^{de} f(j) f(0)^{e-1}\big) & &= \text{KDF}\big(f(j)^{de} f(i) f(0)^{e-1}\big) \\ &= \text{KDF}\big(f(i) f(j) f(0)^{e-1}\big). & &= \text{KDF}\big(f(j) f(i) f(0)^{e-1}\big). \end{aligned}$$

as $de = 1$. Hence the result is established. □

Proposition 4.4 (Randomness). *Let $KDF(\cdot)$ be a secure key derivation function with salt parameter set to null and the source key material and output length parameters given. Each pairwise key generated as $MK_{i,j} = KDF\big(g_i(j)\ g_i(0)^{e-1}\big)$ is indistinguishable from a random one.*

Proof. User U_i has the secret shared polynomial $g_i(x) = f(i)^d f(x)$. According to Proposition 4.3, $g_i(j)g_i(0)^{e-1} = f(i)f(0)^{e-1}f(j)$. When $f(x)$ is chosen uniformly randomly, $(f(1), \cdots, f(j), \cdots, f(m))$ is uniformly distributed and hence, $g_i(j)g_i(0)^{e-1}, j \in [m]$, which is used as a source keying material in the KDF. The security properties of the KDF ensure that $MK_{i,j} = \mathrm{KDF}\big(g_i(j)\ g_i(0)^{e-1}\big)$ is indistinguishable from a random looking bit string. □

Instantiating NPKC.DistRSAParam and NPKC.DistSP Algorithms. The distributed parameter generation for the RSA modulus has been studied extensively, e.g., [3, 8, 26] and its security is investigated against both active and passive adversaries. Let π_{RSA} be the distributed RSA parameter generation protocol where, at the end of the execution of the protocol, each SSP holds an additive share of primes P and Q and it outputs $N = PQ$. We use the modular inverse computation protocol by Catalano *et al.* [10] in two-party settings and denote the protocol by π_{MIC}. We present a new two-round protocol π_{EXP} for computing the exponentiation of a number modulo N for our setting in Figure 9 in Appendix A. The correctness and security of π_{EXP} are also provided in Appendix A. Figure 2 presents a two-party protocol for generating m pairs of secret shared polynomials. In the protocol, **Round 0** is run only once and secret parameters are stored safely, and for generating m pairs of secret shared polynomials, only the SSPs execute only **Round 1**.

4.2 Security of the NPKC Protocol

The distributed secret shared polynomial generation of the NPKC scheme is a secure two-party computation protocol. We consider the security of the NPKC protocol against honest-but-curious adversaries: a) security against other SSP, meaning one honest-but-curious SSP may try to learn information about other SSP's secret polynomials; and b) security against multiple colluding users, meaning a set of users may try to learn about both SSPs' secret polynomials $f_1(x)$ and $f_2(x)$ or learn a pair of honest users' pairwise key.

Theorem 4.5 (Security against Other SSP). *The distributed secret shared polynomial generation protocol π_{SSPG} is secure against honest-but-curious adversaries.*

Proof. The protocol π_{SSPG} is based on the sub-protocols π_{RSA}, π_{MIC} and π_{EXP}. In [26] and [10], it is proven that π_{RSA} and π_{MIC} are secure against honest-but-curious adversaries, respectively. We now show that π_{SSPG} is secure. Since SSP_1 chooses the polynomial $f_1(x)$ uniformly at random over $\mathbb{Z}_N$, the sequence of values $\{f_1(j) : 1 \le j \le m\}$ is uniformly distributed and so does their encryptions under the key e. As the security of π_{EXP} is based on the factorization problem, SSP_2 learns nothing about $\{f_1(j) : 1 \le j \le m\}$ as well as d_1. Similar argument applies to SSP_1 about $\{f_2(j) : 1 \le j \le m\}$ and d_2 for SSP_2. The security of the protocol π_{SSPG} follows from the composition theorem as the protocols π_{RSA}, π_{MIC} and π_{EXP} are secure against honest-but-curious adversaries. □

Protocol 1: Distributed Secret Shared Polynomial Generation (π_{SSPG})

Private inputs: Each $SSP_i, i \in [1, 2]$ has a secret polynomial $f_i(x) \in \mathbb{Z}_N[x]$ of degree $(k-1)$ with coefficients non-one.
Public input: Public integer m, e, N.
Output: SSP_i receives m secret polynomials $\{s_{i,j}(x), j \in [m], i = 1, 2\}$. Each user U_i receives a pair of secret shared polynomials $(s_{1,i}(x), s_{2,i}(x))$ from SSP_1 and SSP_2.

Round 0: Distributed RSA Parameter Generation

1. SSP_1 and SSP_2 jointly run the protocol π_{RSA} to generate a strong RSA modulus $N = PQ$.
2. SSP_i holds shares P_i and Q_i of primes P and Q, $i = 1, 2$ where $P = P_1 + P_2$ and $Q = Q_1 + Q_2$.
3. SSP_1 computes $\phi_1 = N + 1 - (P_1 + Q_1) \bmod N$.
4. SSP_2 computes $\phi_2 = -(P_2 + Q_2) \bmod N$.

Round 1: Distributed Secret Shared Polynomial $s_{1,j}(x)$, $s_{2,j}(x)$ Generation (NPKC.DistSP)

1. SSP_1 and SSP_2 compute modular inverse of e.
   - SSP_1 and SSP_2 run the protocol π_{MIC} to compute two shares of $d = \frac{1}{e} \bmod \phi(N)$.
   - SSP_1 receives d_1 and SSP_2 receives d_2 where $d = d_1 + d_2$.
2. For $j \in [m]$, SSP_1 and SSP_2 run the following steps:
   - SSP_1 computes $f_1(j)$.
   - SSP_1 and SSP_2 jointly compute $f_1(j)^d$ by running π_{EXP} with SSP_1's input $f_1(j)$ and d_1 and SSP_2's input d_2, and SSP_1 receives the output $f_1(j)^d$.
   - SSP_1 computes $s_{1,j}(x) = f_1(j)^d f_1(x)$.
3. For $j \in [m]$, SSP_1 and SSP_2 run the following steps:
   - SSP_2 computes $f_2(j)$.
   - SSP_1 and SSP_2 jointly compute $f_2(j)^d$ by running π_{EXP} with SSP_1's input d_1, and SSP_2's input $f_2(j)$ and d_2, and SSP_2 receives the output $f_2(j)^d$.
   - SSP_2 computes $s_{2,j}(x) = f_2(j)^d f_2(x)$.
4. For $j \in [m]$, SSP_1 computes $\mathcal{E}_{K_{j,SSP_1}}(s_{1,j}(x))$ and sends to U_j. SSP_2 computes $\mathcal{E}_{K_{j,SSP_2}}(s_{2,j}(x))$ and sends to U_j, $j \in [m]$.

Figure 2: The distributed secret shared polynomial generation protocol and distributing shares to users.

In the next theorem, we prove that the pairwise key computation scheme is k-secure i.e., any collusion of less than k users learn nothing about other users' pairwise keys.

Theorem 4.6 (Security against Colluding Users). *Assume that $C \subseteq [m]$ with $|C| < k$ is the set of colluding users who hold the set of secret shared polynomials $\{(s_{1,i}(x), s_{2,i}(x)) : i \in C\}$, giving $\{g_i(x) = s_{1,i}(x)s_{2,i}(x) : i \in C\}$ where $g_i(x) = f(i)^d f(x)$. For any $U_u, U_v \notin C$, there exists a PPT $\mathcal{A}$ such that*

$$\Big\{\mathcal{A}\Big((s_{1,i}(x),\ s_{2,i}(x))_{i\in C},\ C,\ pp,\ m,\ \{u,\ v\}\Big)\Big\} \approx \Big\{(s_{1,b}(x),\ s_{2,b}(x))_{b\in\{u,v\}}\Big\}.$$

Proof. Suppose that the adversary $\mathcal{A}$ captures the set of users in C. Given the set of inputs $(s_{1,i}(x), s_{2,i}(x)), i \in C$, $\{u, v\}$, pp and

m to $\mathcal{A}$, it performs the following for computing U_u's secret shared polynomial, which results in computing the pairwise key with U_v:

1. $\mathcal{A}$ evaluates $s_{1,i}(x)$ and $s_{2,i}(x)$ at the point u and then computes $z^1_{i,u} = s_{1,i}(u)(s_{1,i}(0))^{e-1}$ and $z^2_{i,u} = s_{2,i}(u)(s_{2,i}(0))^{e-1}$ for all $i \in C$.
2. $\mathcal{A}$ reconstructs $h^j_u(x)$ by applying Lagrange's interpolation on $|C|$ points $\{(i, y^j_{i,u} s_{1,j}(i)) : i \in C\}$ where $y^j_{i,u} = s_{j,i}(0)^{e-1}$, along with all possible uniformly distributed random secrets, $j = 1, 2$.
3. $\mathcal{A}$ computes $MK_{u,v}$ by evaluating the polynomial $h_u(x) = h^1_u(x)h^2_u(x)$ at v and then computing $MK_{u,v} = \mathsf{KDF}(h_u(v)) = \mathsf{KDF}(f(u)f(0)^{e-1}f(v))$.

$\mathcal{A}$ can similarly construct $h_v(x)$. The probability that correctly reconstructing the secret shared polynomial for u or v in Step 2 is given by $\Pr[h_u(v) = f(u)f(0)^{e-1}f(v)] = \frac{1}{N^{2k}}, \forall v \in [m]\backslash C$.

The coefficients of $s_{1,i}(x)$ and $s_{2,i}(x)$ are uniformly distributed as they were constructed from random polynomials $f_1(x)$ and $f_2(x)$ whose coefficients are uniformly distributed. Thus, the probability for randomly constructing the correct polynomial $h_u(x)$ or $h_v(x)$ is $\frac{1}{N^{2k}}$. Therefore, for any set C of colluding users with $|C| < k$, the distributions $\left\{\mathcal{A}\Big((s_{1,i}(x), s_{2,i}(x))_{i\in C}, C, pp, m, \{u,v\}\Big)\right\}$ and $\left\{(s_{1,b}(x), s_{2,b}(x))_{b\in\{u,v\}}\right\}$ are identical. □

Next, we consider the security for the case that one SSP is colluding with a set of users C of size less than k and prove the set of users C and one SSP learn nothing about other honest users pairwise keys. We guarantee no security when a set of users, even with size smaller than k, colluding with both SSPs.

Theorem 4.7 (Security against Colluding SSP-Users). *Assume that SSP_j is colluding with the set of users $C \subseteq [m]$ with $|C| < k$ who hold the set of secret shared polynomials $\{(s_{1,i}(x), s_{2,i}(x)) : i \in C\}$, giving $\{g_i(x) = s_{1,i}(x)s_{2,i}(x) : i \in C\}$ where $g_i(x) = f(i)^d f(x)$ and $f(x) = f_1(x)f_2(x)$. For any $U_u, U_v \notin C$, there exists a PPT $\mathcal{A}$ such that*

$$\left\{\mathcal{A}\Big(\{s_{3-j,i}(x)\}_{i\in C}, f_j(x),\ C,\ pp,\ m,\ \{u,v\}\Big)\right\} \approx \left\{\{s_{3-j,b}(x)\}_{b\in\{u,v\}}\right\}.$$

Proof. Suppose that the adversary $\mathcal{A}$ controls the set of users C and SSP_j, $j = 1$ or 2. The set of inputs $\{(s_{1,i}(x), s_{2,i}(x)), i \in C\}$, $\{u,v\}$, $f_j(x)$, pp and m to $\mathcal{A}$ are known to $\mathcal{A}$. Since $f_j(x)$ is known to $\mathcal{A}$, so does $s_{j,u}(x)$ for any honest user U_u. To compute a pairwise key for a pair of honest users U_u and U_v, it must learn the polynomial $s_{3-j,u}(x)$ for U_u or $s_{3-j,v}(x)$ for U_v. $\mathcal{A}$ performs the following steps for computing $s_{3-j,u}(x)$:

1. $\mathcal{A}$ evaluates $s_{3-j,i}(x)$ at the point u and then computes $z_{i,u} = s_{3-j,i}(u)(s_{3-j,i}(0))^{e-1}$ for all $i \in C$.
2. $\mathcal{A}$ reconstructs $h_u(x)$ by applying Lagrange's interpolation on $|C|$ points $\{(i, y_{i,u}s_{3-j,j}(i)) = (i, z_{i,u}) : i \in C\}$ where $y_{i,u} = s_{3-j,i}(0)^{e-1}$, along with all possible uniformly distributed random secrets.
3. $\mathcal{A}$ computes $MK_{u,v}$ by first evaluating the polynomial $H_u(x) = s_{j,u}(x)h_u(x)$ at v and then performing $MK_{u,v} = \mathsf{KDF}(H_u(v)) = \mathsf{KDF}(f(u)f(0)^{e-1}f(v))$.

Similarly, $\mathcal{A}$ can also construct $H_v(x)$ and then evaluate $H_v(x)$ at u to obtain $MK_{v,u}$. Since $|C| < k$, the probability for correctly reconstructing $H_u(x)$ (resp. $H_v(x)$) for U_u (resp. U_v) is given by $\Pr[H_u(v) = f(u)f(0)^{e-1}f(v)] = \frac{1}{N^k}, \forall v \in [m]\backslash C$.

Since the coefficients of $s_{3-j,i}(x)$ are uniformly distributed as it was constructed from $f_{3-j}(x)$ whose coefficients are uniformly distributed, the probability for randomly constructing the correct polynomial $h_u(x)$ is $\frac{1}{N^k}$. Therefore, for any set C with $|C| < k$, the distributions $\left\{\mathcal{A}\Big((s_{1,i}(x), s_{2,i}(x))_{i\in C}, f_j(x), C, pp, m, \{u,v\}\Big)\right\}$ and $\left\{(s_{1,b}(x), s_{2,b}(x))_{b\in\{u,v\}}\right\}$ are identical. □

4.3 Performance for SSPs

The most expensive task for the SSPs is the generation of safe primes in a distributed way, which is performed only once in offline when the system setup is done. For the details about the complexity analysis, the reader is referred to [26]. The basic protocol for computing the modular inverse due to Catalano *et al.* [10] needs two rounds of interactions between two SSPs. For computing m pairs of secret shared polynomials $\big(s_{1,j}(x), s_{2,j}(x)\big)$ in **Round 1**, the SSPs need to interact with each other $2m$ times for computing $f_1(j)^d$ and $f_2(j)^d$, $1 \le j \le m$, and each SSP performs m polynomial evaluations and m constant-polynomial multiplications. The computational complexity for this is $O(m\log^2(k) + mk)$ where $(k-1)$ is the degree of $f_i(x)$. The storage required by each SSP for m pairs of secret shared polynomials is $2mk$ units over $\mathbb{Z}_N$. Table 1 summarizes the number of unit operations performed by each SSP and "# Bits Trans" denotes the amount of transmitted bits over the network.

Table 1: Number of operations for computing m pairs of secret shared polynomials $\left\{\big(s_{1,j}(x), s_{2,j}(x)\big) : j \in [m]\right\}$.

Entity	# EXP	# Poly Eval	# MUL	# Const Poly Mul	# Bits Trans
SSP_i	$3m$	m	$2m$	m	$m\lambda$
SSP_{3-i}	$2m$	–	m	–	$2m\lambda$
Total	$5m$	m	$3m$	m	$3m\lambda$

5 OUR SECURE AGGREGATE SUM SCHEME

In our setting, the aggregate-sum protocol consists of two phases. In offline phase, each user receives two pairs of secret shared polynomials from two SSPs for deriving keying materials that will be used in the aggregate-sum protocol. Each user also receives from the server the information about its neighbors of ℓ-regular communication network. In online phase, the server S runs the aggregate-sum protocol with the users $\mathcal{U}$ where the only server receives the aggregate-sum as output. Note that the SSPs are not involved in the online phase. Below we describe the main components of the scheme, followed by presenting the scheme in Figure 4.

5.1 One-Way Key Chain for Pairwise Keys

For the frequent aggregation of user inputs, many one-time keys are needed for masking user inputs. We use the one-way chain technique to derive one-time pairwise keys from the pairwise master secret keys. One-way chain technique is used in many cryptographic applications, i,e., one-time password [31], TESLA [39]. We divide the time into small equal-length time intervals. For each time interval, we assign a pairwise key that is derived from the master pairwise key with some nonce (i.e., time stamp) and use that key to derive multiple one-time pairwise keys that are used in multiple aggregate-sum computations. For example, a time interval can be a day and in a day multiple aggregate-sums are computed. Let MK be a master pairwise key for two users and F be a one-way function. Figure 3 depicts a one-time pairwise key derivation using the one-way function F.

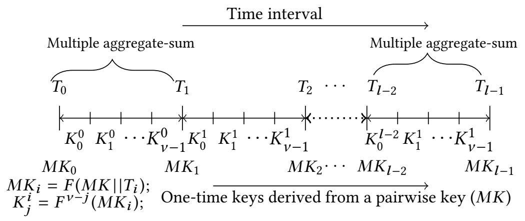

Time interval
Multiple aggregate-sum
Multiple aggregate-sum
T_0
T_1
T_2 $\cdots$
T_{l-2}
T_{l-1}
K_0^0 K_1^0 $\cdots K_{\nu-1}^0$ K_0^1 K_1^1 $\cdots K_{\nu-1}^1$
K_0^{l-2} K_1^1 $\cdots K_{\nu-1}^1$
MK_0
MK_1
$MK_2 \cdots$
MK_{l-2}
MK_{l-1}
$MK_i = F(MK||T_i)$;
$K_j^i = F^{\nu-j}(MK_i)$;
One-time keys derived from a pairwise key (MK)

Figure 3: One-way key chain for one-time pairwise keys. Let ν aggregate-sum computations to be performed in the time interval $[T_i, T_{i+1}]$. The keys for $[T_i, T_{i+1}]$ are $K_\tau^i = F^{\nu-\tau}(MK_i), 0 \le \tau \le \nu - 1$.

Deriving Pairwise Keys in Non-interactive Way. Assume that each user U_i has two pairs of secret shared polynomials $(s_{1,i}(x), s_{2,i}(x))$ and $(c_{1,i}(x), c_{2,i}(x))$ which are used to derive pairwise masking keys and authenticated encryption (AE) keys using NPKC.CompKey as follows.

- Masking key: $MK_{i,j} \leftarrow$NPKC.CompKey$(s_{1,i}(x), s_{2,i}(x), j, e)$
- AE key: $EK_{i,j} \leftarrow$NPKC.CompKey$(c_{1,i}(x), c_{2,i}(x), j, e)$

At τ-th session where $0 \le \tau \le \nu - 1$ in the time interval $[T_l, T_{l+1}]$, each user computes one-time masking key $(MK_{i,j}^{l,\tau})$ and AE key $(EK_{i,j}^{l,\tau})$ using the one-way function F as follows:

- $MK_{i,j}^{l} = F(MK_{ij}||TS_l)$ and $MK_{i,j}^{l,\tau} = F^{\nu-\tau}(MK_{i,j}^{l}||\nu||\tau)$
- $EK_{i,j}^{l} = F(EK_{ij}||TS_l)$ and $EK_{i,j}^{l,\tau} = F^{\nu-\tau}(EK_{i,j}^{l}||\nu||\tau)$

where TS_l is a common nonce sent by the server.

5.2 Efficient Mask-then-Encrypt Operation

Users perform the mask-then-encrypt operation to encode their private inputs $\mathbf{x}_i$, before sending to the server. This operation works as first mask the input with *coordinated* additive noise and then encrypt the masked input with a one-time key. Assume that the aggregate-sum protocol is run at τ-th session in the time interval $[T_l, T_{l+1}]$, and one-time keys are generated as above.

Mask and Create One-time Pairwise Key Shares. Our masking operation with *coordinated* additive noise to $\mathbf{x}_i$ is defined as $\boldsymbol{\gamma}_i = \mathbf{x}_i + \sum_{U_j \in \mathcal{N}_\mathcal{U}(U_i)} (-1)^{\delta_{i,j}} \mathbf{q}_{i,j}$ where the coordinated additive noise is $\sum_{U_j \in \mathcal{N}_\mathcal{U}(U_i)} (-1)^{\delta_{i,j}} \mathbf{q}_{i,j}$ derived using the neighboring users' one-time pairwise keys $MK_{i,j}^{l,\tau}$ at τ-th iteration in the time interval $[T_l, T_{l+1}]$, and $\delta_{i,j} = 0$ if $i \le j$ and 1 if $i > j$. When all the neighboring users of each user are present in the system, the sum of $\boldsymbol{\gamma}_i$'s gives the sum of $\mathbf{x}_i$'s, but if at least one $\boldsymbol{\gamma}_i$ is missing, equivalently if at least one of the users is dropped out, the sum of $\boldsymbol{\gamma}_i$'s does not give the sum of $\mathbf{x}_i$'s. Previous approach [7] based on the DH key exchange has employed a (t, m)-*tss* to share a DH private key to achieve the failure-recovery. However, a (t, m)-tss is computationally and communication-wise expensive for a large scale network of size m.

To make the coordinated noise component efficient, our main idea is to employ a 2-out-of-3 secret sharing ($\mathbf{Share}_3^2$, $\mathbf{Rec}_3^2$) scheme to share a one-time pairwise key $(MK_{i,j}^{l,\tau})$ among U_i, U_j and the server S where $MK_{i,j}^{l,\tau}$ is the one-time key at τ-th iteration in the time interval $[T_l, T_{l+1}]$. We use ($\mathbf{Comm}^2$, $\mathbf{SVerify}^2$) algorithms correspond to $(2, 3)$-tss for verifying the correctness of the shares sent to the server, where the commitments generated by $\mathbf{Comm}^2$ are sent to the server. This provides resistance against malicious behaviors of sending incorrect shares of one-time keys. If both U_i and U_j are present in the system or dropped out from the system, none of the secret shares needs to be sent to the server. If one of U_i and U_j is dropped out, the other user sends its share to the server so that the server can recover the pairwise key $(MK_{i,j}^{l,\tau})$. With a proper assignment, each user needs to perform at most $\lceil \frac{\ell}{2} \rceil$ $\mathbf{Share}_3^2$ operations, which is much faster than running a $\mathbf{Share}_m^t$ operation where $\ell = |\mathcal{N}_\mathcal{U}(U_i)|$ is the number of neighbors of a user. Moreover, for each $\mathbf{Share}_3^2$ operation, the user U_i needs to send one secret share to U_j and another one to the server by using $\mathcal{E}_K()$ where K is either $EK_{i,j}^{l,\tau}$ shared between U_i and U_j or $EK_{i,S}^{l,\tau}$ shared between U_i and S. Thus the total number of secret shares for the additive coordinated noise is ℓ, which is less than m.

Encrypt and Create One-time Key Shares. The user U_i encrypts $\boldsymbol{\gamma}_i$ using a freshly generated key (κ_i) as

$$\boldsymbol{\alpha}_i = \mathsf{Enc}_{\kappa_i}(\boldsymbol{\gamma}_i) = \boldsymbol{\gamma}_i + \mathbf{k}_i = \mathbf{x}_i + \mathbf{k}_i + \sum_{U_j \in \mathcal{N}_\mathcal{U}(U_i)} (-1)^{\delta_{i,j}} \mathbf{q}_{i,j}$$

where $\mathbf{q}_{i,j} = \mathsf{PRG}(MK_{i,j}^{l,\tau}, TS_\tau), U_j \in \mathcal{N}_\mathcal{U}(U_i)$, $\mathbf{k}_i = \mathsf{PRG}(\kappa_i, TS_\tau)$, TS_τ is a unique time stamp at τ, and κ_i is secret shared with all other users using $\mathbf{Share}_m^t$ and the commitments of (t, m)-tss generated by $\mathbf{Comm}^t$ are sent to the server. The user U_i delivers the shares of κ_i to the other users by encrypting using $\mathcal{E}_{EK_{i,j}^{l,\tau}}$ with a session key shared between $EK_{i,j}^{l,\tau}$. Each user sends $\boldsymbol{\alpha}_i$ and the encryption of the shares of pairwise keys used in $\boldsymbol{\gamma}_i$ to the server. The choice of the parameters such as ℓ, t and the number of colluding users in terms of m are given in Table 2.

5.3 Efficient Decrypt-then-Unmask Operation

To compute the aggregate sum, the server needs to decrypt all the received inputs. In doing so, the server needs to reconstruct the one-time secret key (κ_i) for each received inputs. For this recovery process, there are two cases to consider: 1) aggregation without drop out, when all users sent their inputs; and 2) aggregation with drop out, when some users drop out from the system after responding their participation in the aggregate sum. The server keeps track of current alive users and drop out users in the system, which can be obtained from the process that which users have submitted their inputs to the server.

Aggregation with No Dropout. Server sends a request to the users to send a share of each κ_i and collects at least t shares of each κ_i. Note that each user sends one share of each κ_i. The server then runs the $\mathbf{Rec}_m^t$ algorithm to recover κ_i. When the reconstruction of each κ_i is done, the server decrypts $\boldsymbol{\alpha}_i$, i.e.,

$\boldsymbol{\gamma}_i = \mathrm{Dec}_{\kappa_i}(\boldsymbol{\alpha}_i)$ and obtains the aggregate sum by summing up all $\boldsymbol{\gamma}_i$'s, i.e., $\sum_{i\in[m]} \boldsymbol{\gamma}_i = \sum_{i\in[m]} \mathbf{x}_i$.

Aggregation with Dropout. Assume that the server received $\boldsymbol{\alpha}_i$'s from the users in $\mathcal{U}_1$. Suppose that the protocol allows up to η_{max} users drop out from a user's neighbor. Upon receiving a request from the server, each user checks whether the total number of alive users in the system is at least t, and the total number of dropouts from its neighbor is less than η_{max} and then responds as follows: For all alive users (U_i) including itself, each alive user sends its share of κ_i, and for each neighboring dropout user, it sends the share of the corresponding one-time pairwise key. For all non-neighboring dropout users, it sends a special symbol $\perp$. After collecting inputs from the users in $\mathcal{U}_2 \subseteq \mathcal{U}_1$, if the number of received inputs is at least t, the server first reconstructs all κ_i's using $\mathbf{Rec}^t_m$ for users in $\mathcal{U}_2$ and then decrypts $\boldsymbol{\alpha}_i$ as $\boldsymbol{\gamma}_i = \mathrm{Dec}_{\kappa_i}(\boldsymbol{\alpha}_i)$, followed by computing the sum $\sum_{U_i\in\mathcal{U}_2} \boldsymbol{\gamma}_i$. Next, the server reconstructs one-time pairwise keys, using $\mathbf{Rec}^2_3$, with alive users and dropout users according their neighbors, followed by summing the expanded one-time pairwise keys with PRG and subtracting it with $\sum_{U_i\in\mathcal{U}_2} \boldsymbol{\gamma}_i$ and obtains $\sum_{U_i\in\mathcal{U}_2} \mathbf{x}_i$. Our aggregate-sum protocol for semi-honest and active adversaries is depicted in Figure 4.

5.4 Selection of the Parameters

We now present the parameters of the scheme. Recall that k is the degree of the secret shared polynomials and t is the threshold value in the (t, m)-tss. We choose the parameter $k = t$ to set the security of all keying materials same and to resist against colluding a set of users C, and colluding a set of users C and the server S. We parametrize the maximum number of dropouts, denoted by η_{max}, from the neighbor of each user and the size of the neighbor of each user (ℓ). Table 2 shows the selection of the parameters for three threat models.

Table 2: Parameters of the scheme for different adversarial models and the number of handled dropouts.

Threat Model	Minimum Threshold	Neighbor Size	Minimum inputs in Aggregate-sum	Number of Dropouts
Users-only collusion	1	$\ell = t$	t	–
Server-only	t	$\ell = \eta_{max} + 2$	t	η_{max}
Users-Server collusion	t	$\ell = \eta_{max} + t + 1$	$\ell - \lvert C \cup \{S\}\rvert$	η_{max}

If we set the maximum number of colluding users to $|C| = \lceil \frac{m}{3} \rceil - 1$ and $\eta_{max} = \lceil \frac{m}{3} \rceil - 1$, then, for the server-only model, the threshold is $t = \lceil \frac{m}{3} \rceil$ and the neighbor size is $\ell = \lceil \frac{m}{3} \rceil + 1$. For the users-server collusion up to $|C \cup \{S\}| = \lceil \frac{m}{3} \rceil - 1$ and $\eta_{max} = \lceil \frac{m}{3} \rceil - 1$, then the threshold is $t = \lceil \frac{m}{3} \rceil$ and the neighbor size is $\ell = 2\lceil \frac{m}{3} \rceil$.

5.5 Security against Semi-honest Adversaries

We consider three different adversarial models for the aggregate-sum scheme for users' input privacy as mentioned in Section 3.3. In Theorem 5.1, we prove that an adversary controlling a set of semi-honest users does not learn anything about the honest users' inputs. In Theorem 5.2, we proved that an adversary corrupting a set of users and the server cannot anything about individual inputs except the sum of honest users' inputs. Due to the space limit, the proofs are provided in Appendix B.

Theorem 5.1 (Users only Security against Semi-honest Adversaries). *Given v, t, η_{max}, $\mathcal{U}$ with $|\mathcal{U}| \geq t$, $\{\mathbf{x}_u\}_{u\in\mathcal{U}}$ and $C \subset \mathcal{U}$ with $|C| < t$, there exists a PPT simulator $\mathcal{S}$ such that*

$$\mathcal{S}_C^{v,t,\eta_{max},\mathcal{U}}\left(\{\mathbf{x}_u\}_{u\in C}, \mathcal{U}_0, \mathcal{U}_1, \mathcal{U}_2\right) \approx_c VIEW_C^{\pi_{AS},v,t,\mathcal{U}}\left(\{\mathbf{x}_u\}_{u\in\mathcal{U}}, \mathcal{U}_0, \mathcal{U}_1, \mathcal{U}_2\right).$$

Proof. See Appendix B □

Theorem 5.2 (Users-Server Security against Semi-honest Adversaries). *Given v, t, η_{max}, $\mathcal{U}$ with $\mathcal{U} \geq t$, $\{\mathbf{x}_u\}_{u\in\mathcal{U}}$ and $C \subset \mathcal{U} \cup \{S\}$ with $|C| < t$, there exists a PPT simulator $\mathcal{S}$ such that*

$$\mathcal{S}_C^{v,t,\eta_{max},\mathcal{U}}\left(\{\mathbf{x}_u\}_{u\in C}, SUM_{\mathcal{U}_2\setminus C}, \mathcal{U}_0, \mathcal{U}_1, \mathcal{U}_2\right) \approx_c \mathtt{VIEW}_C^{\pi_{AS},v,t,\mathcal{U}}\left(\{\mathbf{x}_u\}_{u\in\mathcal{U}}, \mathcal{U}_0, \mathcal{U}_1, \mathcal{U}_2\right)$$

where $SUM_{\mathcal{U}_2\setminus C} = \sum_{U\in\mathcal{U}_2\setminus C} \mathbf{x}_{id(U)}$.

Proof. See Appendix B □

5.6 Security against Active Adversaries

In this section, we consider the active or malicious adversaries in our protocol in which (1) a set of users is compromised by an adversary; (2) the adversary controls a set of users and the server. The adversary can abort the protocol at any step, can send arbitrarily chosen messages, arbitrary shares of (t, m)-tss and $(2, 3)$-tss ,and share its view with others during execution of the protocol. We emphasize that when the adversary sends an arbitrarily input ($\mathbf{x}_i$), our scheme cannot guarantee the correctness of the output (effective for user-only security).

In the following theorem, we prove that when the adversary controls a set of users C with $|C| < t$, it cannot learn anything about individual users' inputs. Our scheme provides stronger privacy guarantees in this threat model than the one in [7].

Theorem 5.3 (Users-Only Privacy against Active Adversaries). *Given v, t, η_{max}, $\mathcal{U}$ with $\mathcal{U} \geq t$, $\{\mathbf{x}_u\}_{u\in\mathcal{U}}$ and $C \subset \mathcal{U}$ with $|C| < t$, there exists a PPT simulator $\mathcal{S}$ such that*

$$\mathcal{S}_C^{v,t,\eta_{max},\mathcal{U}}\left(\{\mathbf{x}_u\}_{u\in C}, \mathcal{U}_0, \mathcal{U}_1, \mathcal{U}_2\right) \approx_c REAL_C^{\pi_{AS},v,t,\mathcal{U}}\left(\{\mathbf{x}_u\}_{u\in\mathcal{U}\setminus C}, \mathcal{U}_0, \mathcal{U}_1, \mathcal{U}_2\right).$$

Proof. Assume that the adversary $\mathcal{A}$ controls a set of users in C. $\mathcal{A}$ can send any arbitrary messages to the server on behalf of users in C, and can make some honest users to abort the protocol by sending wrong ciphertexts and can also record the abort pattern. The adversary does not receive any message related to honest users' inputs $\mathbf{x}_i$, the only messages $\mathcal{A}$ receives are encrypted shares of one-time keys used to mask-then-encrypt the inputs $\mathbf{x}_i$. The simulator $\mathcal{S}$ can generate the joint view of the corrupted users C by running the honest users on dummy inputs and the corrupted users on the true inputs.

We now show that the view of the adversary in committing sharing is indistinguishable from the simulated view. The simulator $\mathcal{S}$ simulates the commitments $\mathbf{Comm}^t(\cdot)$ and $\mathbf{Comm}^2(\cdot)$ of the randomly chosen key κ_i and one-time pairwise keys $MK^{l,\tau}_{i,j}$, respectively, and sends to $\mathcal{A}$ the encrypted shares of the corrupted users. In the Failure-recovery information exchange phase, $\mathcal{S}$ receives a set of $\boldsymbol{\beta}_i$'s for corrupted users. If any verification for the correctness

Protocol 2: Secure Aggregate-sum Protocol (π_{AS})

Private inputs: Each user U_i has two pairs of secret shared polynomials $(s_{1,i}(x), s_{2,i}(x))$ and $(c_{1,i}(x), c_{2,i}(x))$ received from SSP_1 and SSP_2. Each user U_i has a private input $\mathbf{x}_i$. Each user U_i has a signing key (sk_{U_i}) and a public verification key (pk_{U_i}). Server has a signing key $(\text{sk}_S, \text{pk}_S)$.

Public inputs: Communication ℓ-regular network, v, m, e, N, t, and q.

Output: The server receives the aggregate sum of all alive users' inputs $\sum_{i\in\mathcal{U}_2} \mathbf{x}_i$.

Round 0: Non-interactive Pairwise Key Computation (NIKE)

- Each user (U_i) has secret shared polynomials $(s_{1,i}(x), s_{2,i}(x))$ and $(c_{1,i}(x), c_{2,i}(x))$ received from SSP_1 and SSP_2.
- U_i constructs $g_i(x) = s_{1,i}(x)s_{2,i}(x)$ and $h_i(x) = c_{1,i}(x)c_{2,i}(x)$ over $\mathbb{Z}_N[x]$.
- For each user U_j in $\mathcal{N}(U_i)$, U_i computes $MK_{i,j} = \text{KDF}(g_i(j)g_i(0)^{e-1}) \in \mathbb{F}_q$ and $EK_{i,j} = \text{KDF}(h_i(j)h_i(0)^{e-1}) \in \mathbb{F}_q$.

Round 1: Initiate Aggregate-sum Computation by Server

- Server calls for aggregate-sum in the time interval $[T_l, T_{l+1}]$ and sends a message to users, and the set of alive users in $\mathcal{U}_0$ responds to participate in aggregate-sum.
- Server broadcasts $(\mathcal{U}_0, \tau, T_l, TS_l)$ along with $\sigma_S = \text{Sig}(sk_S, \mathcal{U}_0\|\tau\|T_l\|TS_l\|TS_\tau)$ to users in $\mathcal{U}_0$ where TS_l and TS_τ are unique time stamps for $[T_l, T_{l+1}]$ and the τ-th aggregate-sum, respectively.

Round 2: Mask-then-encrypt Input based on One-way Key Chain for Pairwise Keys

- Upon receiving $(\mathcal{U}_0, \tau, T_l, TS_l, TS_\tau, \sigma_S)$, U_i verifies σ_S, and if succeeds, checks whether $|\mathcal{N}_{\mathcal{U}_0}(U_i)| > \eta_{max}$. If yes, proceed to the next step, else abort the protocol.
- U_i computes $EK^l_{i,j} = F(EK_{ij}\|TS_l) \in \mathbb{F}_q$ and $EK^{l,\tau}_{i,j} = F^{v-\tau}(EK^l_{i,j}\|v\|\tau) \in \mathbb{F}_q$ at τ-th session where $0 \le \tau \le v-1$.
- For each $U_j \in \mathcal{N}_{\mathcal{U}_0}(U_i)$, U_i computes $MK^l_{i,j} = F(MK_{ij}\|TS_l) \in \mathbb{F}_q$ and $MK^{l,\tau}_{i,j} = F^{v-\tau}(MK^l_{i,j}\|v\|\tau) \in \mathbb{F}_q$ at τ-th session where $0 \le \tau \le v-1$.
- For each $U_j \in \mathcal{N}_{\mathcal{U}_0}(U_i)$, U_i computes three shares for each $MK^{l,\tau}_{i,j}$, i.e, $(s^{ij}_i, s^{ij}_j, s^{ij}_S) \leftarrow \mathbf{Share}^2_3(MK^{l,\tau}_{i,j})$ and generates commitments $\mathbf{Comm}^2(MK^{l,\tau}_{i,j})$ for the shares of $MK^{l,\tau}_{i,j}$.
- U_i generates one-time key κ_i uniformly at random and creates its m shares $\{\kappa_{i,j}, 1 \le j \le m\} \leftarrow \mathbf{Share}^t_m(\kappa_i)$ and generates commitments $\mathbf{Comm}^t(\kappa_i)$ for the shares of κ_i.
- If $U_j \in \mathcal{N}_{\mathcal{U}_0}(U_i)$, U_i sends $\mathcal{E}_{EK^{l,\tau}_{i,j}}(\kappa_{ij}\|s^{ij}_j)$ to U_j and sends $\mathcal{E}_{EK^{l,\tau}_{i,j}}(\kappa_{i,j})$ to U_j if $U_j \notin \mathcal{N}_{\mathcal{U}_0}(U_j)$ and sends $\mathcal{E}_{K_{i,S}}(s^{ij}_S)\|\mathcal{E}_{K_{i,S}}(\mathbf{Comm}^2(MK^{l,\tau}_{i,j}))$ and $\mathcal{E}_{K_{i,S}}(\mathbf{Comm}^t(\kappa_i))$ to server S.
- For $U \in \mathcal{N}_{\mathcal{U}_0}(U_i)$, U_i evaluates $\mathbf{q}_{i,\text{id}(U)} \leftarrow \text{PRG}\left(MK^{l,\tau}_{i,\text{id}(U)}, TS_\tau\right)$, TS_τ is a unique time stamp sent by the server.
- U_i encodes its input as $\boldsymbol{\alpha}_i = \mathbf{x}_i + \mathbf{k}_i + \sum_{U\in\mathcal{N}_{\mathcal{U}_0}(U_i)} (-1)^{\delta_{i,\text{id}(U)}} \mathbf{q}_{i,\text{id}(U)}$ where $\mathbf{k}_i \leftarrow \text{PRG}(\kappa_i)$ and sends $\boldsymbol{\alpha}_i$ to server S.

Round 3: Failure-recovery Information Exchange

- Upon receiving encrypted shares $\mathcal{E}_{EK^{l,\tau}_{i,j}}(\kappa_{ij}\|s^{ij}_j)$ or $\mathcal{E}_{EK^{l,\tau}_{i,j}}(\kappa_{i,j})$, respective users decrypt it and store the result. If any decryption fails, abort the protocol.
- Server receives inputs ($\boldsymbol{\alpha}_i$'s) from the users in $\mathcal{U}_1$. Check if $|\mathcal{U}_1| \ge t$ and then it broadcasts to the user list $\mathcal{U}_1$.
- On receiving $\mathcal{U}_1$, each U_i checks whether $U_i \in \mathcal{U}_1$, $|\mathcal{U}_1| \ge t$ and $\mathcal{U}_1 \subseteq \mathcal{U}_0$. If yes, it does the following, else abort the protocol.
  - U_i computes $\sigma_i = \text{Sig}(sk_{U_i}, \mathcal{U}_1)$ and sends to the server.
  - Server receives a collection of $\{\sigma_i\}_{U_i\in\mathcal{U}_2}$ from a set of users in $\mathcal{U}_2 \subseteq \mathcal{U}_1$ and broadcasts $\{\sigma_i\}_{U_i\in\mathcal{U}_2}$ to all users in $\mathcal{U}_2$.
- On receiving $\{\sigma_i\}_{U_i\in\mathcal{U}_2}$, each user in $\mathcal{U}_2$ does the following:
  - Check if $|\mathcal{U}_2| < t$ or $|\mathcal{N}_{\mathcal{U}_2}(U_i)| < \eta_{max}$, abort the protocol, else proceed to the next step.
  - U_i verifies σ_j for all $U_j \in \mathcal{U}_2$. If all signature verifications successful, proceed to the next step, else abort the protocol.
  - U_i constructs $\boldsymbol{\beta}_i = (\beta_{1,i}, \beta_{2,i}, \cdots, \beta_{m,i})$ where $\beta_{j,i} = s^{ij}_i$ if $U_j \in (\mathcal{U}_0\backslash\mathcal{U}_2)$ and $U_j \in \mathcal{N}_{\mathcal{U}_2}(U_i)$; $\beta_{j,i} = \perp$ if $U_j \in (\mathcal{U}_0\backslash\mathcal{U}_2)$ and $U_j \notin \mathcal{N}_{\mathcal{U}_2}(U_i)$; and $\beta_{j,i} = \kappa_{j,i}$ if $U_j \in \mathcal{U}_2$, and sends $\mathcal{E}_{K_{i,S}}(\boldsymbol{\beta}_i)$ to the server.

Round 4: Computing Aggregate-sum by Server (Decrypt-then-unmask)

- Server receives inputs $\{\boldsymbol{\beta}_i\}_{U_i\in\mathcal{U}_3}$ from users in $\mathcal{U}_3$ with $|\mathcal{U}_3| \ge t$. If $|\mathcal{U}_3| < t$, abort the protocol.

1. *When, $\mathcal{U}_0 = \mathcal{U}_3$, there is no drop out*
   - Using received inputs from users in $\mathcal{U}_2$, the server verifies each received share using $\mathbf{SVerify}^t$ and at least t shares verification is successful, it goes to next step, otherwise it discard the input $\boldsymbol{\gamma}_i$.
   - It reconstructs κ_i by running the $\mathbf{Rec}^t_m$ reconstruction algorithm from at least t collected shares from $\{\boldsymbol{\beta}_i\}$.
   - It then decrypts $\boldsymbol{\gamma}_i = \text{Dec}_{\kappa_i}(\boldsymbol{\alpha}_i), i \in [m]$.
   - It computes $\sum_{i\in[m]} \mathbf{x}_i = \sum_{i\in[m]} \boldsymbol{\gamma}_i$.

2. *When, $|\mathcal{U}_3| < |\mathcal{U}_0|$, there is drop out*
   - For each $U \in \mathcal{U}_3$, the server verifies each received share using $\mathbf{SVerify}^t$ and at least t shares verification is successful, it goes to next step, otherwise it discard the input $\boldsymbol{\gamma}_i$.
   - Server reconstructs $\kappa_{\mathrm{id}(U)}$ by running the $\mathbf{Rec}^t_m$ reconstruction algorithm from at least t collected shares in $\{\boldsymbol{\beta}_i\}$ and computes $\boldsymbol{\gamma}_{\mathrm{id}(U)} = \mathrm{Dec}_{\kappa_{\mathrm{id}(U)}}(\boldsymbol{\alpha}_{\mathrm{id}(U)}), i \in [m]$. If any decryption fails, abort the protocol.
   - Server verifies at least two shares including itself using $\mathbf{SVerify}^2$ and if verification is successful, it goes to next step.
   - Server reconstructs one-time pairwise keys $MK^{l,\tau}_{\mathrm{id}(U),\mathrm{id}(V)}$ using the $\mathbf{Rec}^2_3$ reconstruction algorithm where $U \in \mathcal{U}_3, V \in \mathcal{U}_0 \backslash \mathcal{U}_3$ and $V \in \mathcal{N}(U)$ and evaluates $\mathbf{q}_{\mathrm{id}(U),\mathrm{id}(V)} = \mathrm{PRG}\left(MK^{l,\tau}_{\mathrm{id}(U),\mathrm{id}(V)}, TS_\tau\right)$.
   - Server computes $\sum_{U \in \mathcal{U}_3} \mathbf{x}_{\mathrm{id}(U)} = \sum_{U \in \mathcal{U}_3} \boldsymbol{\gamma}_{\mathrm{id}(U)} + \sum_{(U,V) \in \mathcal{U}_3 \times (\mathcal{U}_0 \backslash \mathcal{U}_3) \text{ and } V \in \mathcal{N}(U)} (-1)^{\delta_{i,j}+1} \mathbf{q}_{\mathrm{id}(U),\mathrm{id}(V)}$.

Figure 4: Protocol for secure aggregate-sum for collecting mobile users' data by the server using non-interactive key generation. Red parts are required for ensuring the security against malicious/active adversaries.

of received shares from $\mathcal{A}$ fails by $\mathcal{S}$, it deletes the corresponding shares. $\mathcal{S}$ aborts if, for each pairwise one-time key, it has less than two valid shares and for each one time key κ_i, it has less than t valid shares (as in the real execution of the protocol the server can not recover the sum). This concludes the simulator. Thus, the corrupted parties view generated by $\mathcal{S}$ is identical to the joint of the corrupted users in the real execution of the protocol. □

We next prove the security of the protocol in the users-server threat model in the Random Oracle model. Our assumption about the random oracle model is standard. In a simulation security, a random oracle modeled as an ideal functionality is used in [36]. We model the KDF, one-way function and the PRG as a random oracle (RO), denoted by $\mathcal{F}_{\mathrm{RO}}$. For instance, in [30], the KDF is modeled as a random oracle. The $\mathcal{F}_{\mathrm{RO}}$ functionality works as follows. On input x from any user or the server to $\mathcal{F}_{\mathrm{RO}}$, it outputs a uniformly random string, and if the same x is inputed to $\mathcal{F}_{\mathrm{RO}}$, it outputs the same output by definition. We need to use the oracle $\mathrm{Ideal}^{\delta}_{\{\mathbf{x}_u\}_{u \in \mathcal{U} \backslash C}}$ in [7]. We define a new oracle $\mathrm{Ideal}^{\mathcal{N},\mathcal{U}}_{\eta_{max}}$ that works as follows: On an input user id i of U_i, it returns a set of random neighbors of U_i of size up to η_{max} associated with ℓ-regular network on the set of users $\mathcal{U}$. In the following theorem, we prove that an adversary controlling a set of users C and the server S cannot learn any individual honest users inputs, except their sum.

Theorem 5.4 (Users-Server Privacy against Active Adversaries). *Assume the KDF, one-way function, and PRG are implemented using a random oracle. Given v, t, η_{max}, $\mathcal{U}$ with $\mathcal{U} \geq t$, $\{\mathbf{x}_u\}_{u \in \mathcal{U}}$ and $C \subset \mathcal{U} \cup \{S\}$ with $|C| < t$, there exists a PPT simulator $\mathcal{S}$ such that*

$$\mathcal{S}_C^{v,t,\eta_{max},\mathcal{U},Ideal^{\delta}_{\{\mathbf{x}_u\}_{u \in \mathcal{U} \backslash C}},Ideal^{\mathcal{N}}_{\eta_{max}}}\left(\{\mathbf{x}_u\}_{u \in \mathcal{U}}, \mathcal{U}_0, \mathcal{U}_1, \mathcal{U}_2\right) \approx_c \mathsf{REAL}_C^{\pi_{AS},v,t,\mathcal{U}}\left(\{\mathbf{x}_u\}_{u \in \mathcal{U}}, \mathcal{U}_0, \mathcal{U}_1, \mathcal{U}_2\right).$$

Proof. We provide a construction of the simulator through a series of hybrids that are constructed by subsequent modifications. The simulator $\mathcal{S}$ emulates the honest parties and runs the adversary $\mathcal{A}$ internally and the adversary provides the inputs of the corrupted users C. $\mathcal{S}$ has access to $\mathcal{F}_{\mathrm{RO}}$ computing KDF, one-way function F, and PRG. Note that the simulator will never make any query to the oracle on input $F(x)$ to obtain information about x, otherwise, it will abort.

Hybrid 0: This hybrid is a random variable corresponding to the view of the adversary $\mathcal{A}$ in the real execution of the protocol.

Hybrid 1: This hybrid is identically same as the previous one except that $\mathcal{S}$ generates $MK_{i,j}$ at uniformly random for each pair of honest neighboring users U_i and U_j and sets $MK_{j,i} \leftarrow MK_{i,j}$. There are polynomial many such pairwise keys for all honest neighboring users. Since $|C| < t$, according to Theorem 4.6, the probability that $\mathcal{A}$ can recover a secret shared polynomial $g_i(x) = s_{1,i}(x)s_{2,i}(x)$ for an honest user U_i is negligible. The distribution of $\left\{g_i(j) : 1 \leq j \leq m\right\}$ is random as $g_i(x)$ was constructed from a random polynomial. Therefore, this hybrid is indistinguishable from the previous one.

Hybrid 2: This hybrid is identically same as the previous one except that $\mathcal{S}$ generates $EK_{i,j}$ at uniformly random for each pair of honest neighboring users U_i and U_j and sets $EK_{j,i} \leftarrow EK_{i,j}$. Since $|C| < t$, the probability that $\mathcal{A}$ can recover a secret shared polynomial $h_i(x) = c_{1,i}(x)c_{2,i}(x)$ for an honest user U_i is negligible. The distribution of $\left\{h_i(j) : 1 \leq j \leq m\right\}$ is random as $h_i(x)$ is a random polynomial. Therefore, this hybrid is indistinguishable from the previous one.

Hybrid 3: In this hybrid, $\mathcal{S}$ aborts if it receives an oracle query on input y such that $y = F(x)$ from $\mathcal{A}$ to obtain information about x where x is $MK^{l,\tau}_{i,j}$ or $EK_{i,j}$. Additionally, $\mathcal{S}$ aborts if $\mathcal{A}$ queries on input y such that $y = F^i(x)$ to obtain information about x for any i. This is equivalent to breaking the security of the one-way function. Thus the probability of distinguishing **Hybrids 2** and **3** is less than or equal to the advantage of breaking the random oracle.

Hybrid 4: $\mathcal{S}$ receives $(\mathcal{U}_0, \tau, T_l, TS_l, TS_\tau, \sigma_S)$ from $\mathcal{A}$ and it aborts if $|\mathcal{N}_{\mathcal{U}_0}(U_i)| > \eta_{max}$ (as an honest user would do in the real execution of the protocol). In this hybrid, for a pair of honest users, $\mathcal{S}$ randomly generates $EK^{l,\tau}_{i,j}$, instead of using $F^{v-\tau}(\cdot)$, for encryption (in **Mask-then-encrypt** phase) and decryption (in **Unmask-then-decrypt** phase) of messages. $\mathcal{S}$ varies one key for a pair of honest users, so there are polynomial many hybrids. The random oracle assumption guarantees that this hybrid is indistinguishable from the previous one.

Hybrid 5: In this hybrid, $\mathcal{S}$ aborts if $\mathcal{A}$ is successful in delivering a different message (which is different from the one $\mathcal{S}$ delivered before) for an honest user on behalf of another honest user. This can happen with a negligible probability due the INT-CTXT property of the encryption scheme. Therefore this hybrid is indistinguishable from the previous one.

Hybrid 6: In this hybrid, we change $\mathcal{S}$ to work as follows. It substitutes the encrypted shares for each pair of honest users with the encryption of **0** while preserving the length of ciphertexts as for an honest user, its neighboring honest users' ciphertext length is twice than non-neighboring honest users ciphertext lengths. Since in **Hybrid 4**, the encryption keys for each pair of honest users were chosen uniformly at random, the IND-CPA security of the encryption guarantees that this hybrid is indistinguishable from the previous one.

Hybrid 7: In this hybrid, $\mathcal{S}$ abort if it receives an input κ_i for an honest user U_i that $\mathcal{A}$ sends to $\mathcal{F}_{\mathrm{RO}}$ for PRG, before the Failure-recovery Information exchange phase. Since κ_i is information-theoretically hidden and $\mathcal{A}$ cannot recover it with $|C| < t$ as the shares produced by $\mathbf{Share}_m^t$ is uniformly distributed, which is independent of κ_i. Therefore, this hybrid is indistinguishable from the previous one.

Hybrid 8: This hybrid is identical to the previous one except that $\mathcal{S}$ runs $\mathbf{Comm}^t(\kappa_i)$ for each κ_i (κ_i is chosen uniformly at random in **Hybrid 7**), and sends $\mathcal{A}$ the encrypted commitments outputted by $\mathbf{Comm}^t(\kappa_i)$. Before the **Unmask-then-decrypt** phase, $\mathcal{A}$ does not learn anything about κ_i after decrypting the encrypted commitments because of the security of the commitment scheme. This hybrid is indistinguishable from the previous one.

Hybrid 9: In this hybrid, $\mathcal{S}$ abort if it receives an input $MK_{i,j}^{l,\tau}$ for a pair of honest users U_i and U_j that $\mathcal{A}$ sends to $\mathcal{F}_{\mathrm{RO}}$ for PRG output, before the Failure-recovery Information exchange phase. Since $MK_{i,j}^{l,\tau}$ is information-theoretically hidden and $\mathcal{A}$ cannot recover it as the shares produced by $\mathbf{Share}_3^2$ is uniformly distributed. Therefore, this hybrid is indistinguishable from the previous one.

Hybrid 10: This hybrid is identical to the previous one except that $\mathcal{S}$ runs the commitment $\mathbf{Comm}^2(MK_{i,j}^{l,\tau})$ for each honest pair $MK_{i,j}^{l,\tau}$, and sends $\mathcal{A}$ the encrypted commitments outputted by $\mathbf{Comm}^2(MK_{i,j}^{l,\tau})$. Before the **Unmask-then-decrypt** phase, $\mathcal{A}$ does not learn anything about $MK_{i,j}^{l,\tau}$ because of the security of the commitment scheme. This hybrid is indistinguishable from the previous one.

Hybrid 11: In this hybrid, $\mathcal{S}$ receives a set of signatures for honest users from $\mathcal{A}$ and it aborts if any signature verification is successful while the honest party did not produce the signature on $\mathcal{U}_1$. A signature forgery can happen only with a negligible probability. Thus, this hybrid is indistinguishable from the previous one due to the security of the signature scheme.

Hybrid 12: In this hybrid, for all honest users, $\mathcal{S}$ substitutes $\boldsymbol{\alpha}_i$ by a uniformly random string of same length while keeping the consistency of $\boldsymbol{\alpha}_i = \mathbf{x}_i + \mathbf{k}_i + \sum_{U_j \in \mathcal{N}_{\mathcal{U}_0}(U_i)} (-1)^{\delta_{i,j}} \mathbf{q}_{i,j}$. For any honest user U_i, $\mathcal{A}$ cannot make query to $\mathcal{F}_{\mathrm{RO}}$ with κ_i, otherwise the simulator will abort. Thus, $\boldsymbol{\alpha}_i$ looks uniformly random. Even after the **Unmask-then-decrypt** phase, when $\mathcal{A}$ learns $\mathbf{k}_i \leftarrow \mathsf{PRG}(\kappa_i)$ along with some of $\mathbf{q}_{i,j}$ where U_j is a corrupted user, $\boldsymbol{\alpha}_i$ still looks random. Therefore, this hybrid is indistinguishable from the previous one.

Hybrid 13: This hybrid is defined exactly same as the previous one except that for each honest user U_i, $\mathcal{S}$ calls $\mathsf{Ideal}_{\eta_{max}}^{\mathcal{N}}$ to receive a set of users H_i from the neighbor of U_i, i.e., $H_i \subset \mathcal{N}_{U_0}(U_i)$, and instead of sending

$$\begin{aligned}\mathsf{PRG}(\kappa_i) \leftarrow \boldsymbol{\alpha}_i - \mathbf{x}_i - \sum_{U_j \in H_i} (-1)^{\delta_{i,j}} \mathsf{PRG}\left(MK_{i,j}^{l,\tau}, TS_\tau\right) \\ - \sum_{U_j \in \mathcal{N}_{\mathcal{U}_0}(U_i) \setminus H_i} (-1)^{\delta_{i,j}} \mathsf{PRG}\left(MK_{i,j}^{l,\tau}, TS_\tau\right)\end{aligned}$$

to

$$\mathsf{PRG}(\kappa_i) \leftarrow \boldsymbol{\alpha}_i - \mathbf{w}_i - \sum_{U_j \in \mathcal{N}_{\mathcal{U}_0}(U_i) \setminus H_i} (-1)^{\delta_{i,j}} \mathsf{PRG}\left(MK_{i,j}^{l,\tau}, TS_\tau\right)$$

where $\sum_{u \in H_i \setminus C} \mathbf{x}_u$ is received from the oracle $\mathsf{Ideal}_{\{\mathbf{x}_u\}_{u \in \mathcal{U} \setminus C}}^{\delta}$ on input $H_i \setminus C$ and the collection $\{\mathbf{w}_u\}_{u \in H_i \setminus C}$ is uniformly distributed such that $\sum_{u \in H_i \setminus C} \mathbf{w}_u = \sum_{u \in H_i \setminus C} \mathbf{x}_u$. As for a pair of honest users U_i and U_j, $\mathcal{A}$ cannot query to $\mathcal{F}_{\mathrm{RO}}$ with $MK_{i,j}^{l,\tau}$, therefore, this hybrid is indistinguishable from the previous one.

Hybrid 14: This hybrid is exactly same as the previous one except that for each honest user U_i, $\mathcal{A}$ calls $\mathsf{Ideal}_{\eta_{max}}^{\mathcal{N}}$ to receive a set of honest users H_i (treated as dropout users) from the neighbor of U_i i.e., $H_i \subset \mathcal{N}_{U_0}(U_i)$ and sends H_i to $\mathcal{S}$. Instead of computing $\boldsymbol{\alpha}_i$ as

$$\begin{aligned}\boldsymbol{\alpha}_i \leftarrow \mathbf{x}_i + \mathbf{k}_i + \sum_{U_j \in H_i} (-1)^{\delta_{i,j}} \mathsf{PRG}\left(MK_{i,j}^{l,\tau}, TS_\tau\right) \\ + \sum_{U_j \in \mathcal{N}_C(U_i)} (-1)^{\delta_{i,j}} \mathsf{PRG}\left(MK_{i,j}^{l,\tau}, TS_\tau\right) \\ + \sum_{U_j \in \mathcal{N}_{\mathcal{U}_0}(U_i) \setminus H_i \setminus \mathcal{N}_C(U_i)} (-1)^{\delta_{i,j}} \mathsf{PRG}\left(MK_{i,j}^{l,\tau}, TS_\tau\right),\end{aligned}$$

$\mathcal{S}$ computes

$$\begin{aligned}\boldsymbol{\alpha}_i \leftarrow \mathbf{x}_i + \mathbf{k}_i + \sum_{U_j \in H_i} (-1)^{\delta_{i,j}} \mathsf{PRG}\left(MK_{i,j}^{l,\tau}, TS_\tau\right) \\ + \sum_{U_j \in \mathcal{N}_C(U_i)} (-1)^{\delta_{i,j}} \mathsf{PRG}\left(MK_{i,j}^{l,\tau}, TS_\tau\right) + \mathbf{z}_i,\end{aligned}$$

where $\sum_{U_j \in \mathcal{N}_{\mathcal{U}_0}(U_i) \setminus H_i \setminus \mathcal{N}_C(U_i)} (-1)^{\delta_{i,j}} \mathsf{PRG}\left(MK_{i,j}^{l,\tau}, TS_\tau\right)$ is substituted by a random string $\mathbf{z}_i$. Since $|H_i| \leq \eta_{max}$ and $C < t$, therefore, $|\mathcal{N}_{\mathcal{U}_0}(U_i) \setminus H_i \setminus \mathcal{N}_C(U_i)| > 1$ as we set $\mathcal{N}_{\mathcal{U}_0}(U_i) = \ell = \eta_{max} + t + 1$. Thus, $\boldsymbol{\alpha}_i$ looks random. Because of the security of PRG, this hybrid is indistinguishable from the previous one.

Hybrid 15: This hybrid is exactly same as the previous one except that for each honest user U_i, $\mathcal{A}$ calls $\mathsf{Ideal}_{\eta_{max}}^{\mathcal{N}}$ to receive a set of honest users H_i such that H_i is the same for all honest users in $\mathcal{N}_{\mathcal{U}_0}(U_i)\backslash H_i\backslash \mathcal{N}_C(U_i)$ and sends the simulator all H_i's. $\mathcal{S}$ simulates $\boldsymbol{\alpha}_i$ as done in **Hybrid 14** while ensuring $\sum_{U_j\in\mathcal{N}_{\mathcal{U}_0}(U_i)\backslash H_i\backslash\mathcal{N}_C(U_i)} \mathbf{z}_i = 0$ (as this can happen in the real execution of the protocol) and substitute $\mathbf{x}_i$ by $\mathbf{w}_i$ such that $\sum_{U_j\in\mathcal{N}_{\mathcal{U}_0}(U_i)\backslash H_i\backslash\mathcal{N}_C(U_i)} \mathbf{x}_i = \sum_{U_j\in\mathcal{N}_{\mathcal{U}_0}(U_i)\backslash H_i\backslash\mathcal{N}_C(U_i)} \mathbf{w}_i$ and sends to $\mathcal{A}$. In the **Unmask-then-decrypt** phase, $\mathcal{A}$ can obtain $\sum_{U_j\in\mathcal{N}_{\mathcal{U}_0}(U_i)\backslash H_i\backslash\mathcal{N}_C(U_i)} \mathbf{x}_i$. Thus the view of the adversary is identical to the view of the simulator. Hence the proof.

□

5.7 Performance of the Protocol

In this section, we present computation, communication and storage costs for each user and the server. Assume that λ and μ bits are needed to represent an element of $\mathbb{Z}_N$ and $\mathbb{F}_q$, respectively.

5.7.1 Performance for Each User. To compute all pairwise keys, each user performs $4(m-1)$ univariate polynomial evaluations for $(s_{i,1}(x), s_{i,2}(x))$ and $(c_{1,i}(x), c_{2,i}(x))$ and $4(m-1)$ modular multiplications over $\mathbb{Z}_N$, where the degree of the polynomials is $(k-1)$. We use the multipoint polynomial evaluation technique [2] in computing pairwise keys and $\mathbf{Share}^t_m$. The time complexity for computing the pairwise keys is $O(m\log^2(k))$. The time complexity for computing the shares of κ_i using $\mathbf{Share}^t_m$ is $O(m\log^2(t))$. The cost of $\lceil\frac{\ell}{2}\rceil$ $\mathbf{Share}^2_3$ operations is $\lceil\frac{\ell}{2}\rceil$ modular additions and multiplications over $\mathbb{F}_q$, and asymptotically $O(\ell)$. Assume that r is the dimension of $\mathbf{x}_i$ and each entry of $\mathbf{x}_i$ (e.g., $x_{i,j}$) is of n bits. For the mask-then-encrypt operation, to expand ℓ pairwise keys using a PRG and the encryption operation, the computational cost is $O(nr\ell)$. The overall time complexity for the mask-then-encrypt operation is $O(m\log^2(t) + nr\ell)$ when $k = t$.

The storage and communication overheads are measured in bits. As each user needs to store two pairs of secret shared polynomials of degree $(k-1)$, the storage cost for this is $4k\lambda$ bits. The storage cost for other users' shares for κ_i's and the shares of one-time neighboring pairwise keys is given by $\mu(m-1+\lceil\frac{\ell}{2}\rceil)$. An extra storage of $\mu(\ell+m-1)$ bits is needed if the user wishes to store the master pairwise keys for its neighbors and the master pairwise keys for encryption and decryption. Thus the total storage cost for a user is given by $4k\lambda+\mu(m+\ell+\lceil\frac{\ell}{2}\rceil)$ bits. The total communication cost involving receiving two pairs of secret shared polynomials, sending $\boldsymbol{\alpha}_i$ and $\boldsymbol{\beta}_i$, shares of κ_i, shares of one-time neighboring keys, and the commitments produced by $\mathbf{Comm}^t$ and $\mathbf{Comm}^2$ is $\mu(2m-1+2\ell+t)+4k\lambda$ bits.

5.7.2 Performance for Server. Since the dropout rate in the system is one of the key factors for the server's computational complexity, we include the dropout rate as a parameter in the server's computational complexity computation. Suppose that $\mathcal{U}_2$ is the set of users from whom the server received $\boldsymbol{\beta}_i$'s. To compute the aggregate sum of inputs received from users in $\mathcal{U}_2$, the server needs to first run the $\mathbf{Rec}^t_m$ algorithm $|\mathcal{U}_2|$ times to reconstruct κ_i's and then decrypt $\boldsymbol{\alpha}_i$'s. The computational complexity for this is $O(m^2)$ when the values of Lagrange basis polynomials at zero are pre-computed. The server needs to run the $\mathbf{Rec}^2_3$ algorithm $|\mathcal{U}_2|\cdot|\mathcal{U}_0\backslash\mathcal{U}_2|$ times for recovering all one-time pairwise keys between alive and dropped out users and also expand the keys using PRG. The computation cost for this process is $O(nr\cdot|\mathcal{U}_2|\cdot|\mathcal{U}_0\backslash\mathcal{U}_2|)$. The overall cost for the server with pre-computation of Lagrange basis polynomials is $O(m^2+nr\cdot|\mathcal{U}_2|\cdot|\mathcal{U}_0\backslash\mathcal{U}_2|)$. When the $\mathbf{Rec}^t_m$ algorithm recovers κ_i's on-the-fly, i.e., without precomputation, the overall cost for the server is $O(m^3+nr\cdot|\mathcal{U}_2|\cdot|\mathcal{U}_0\backslash\mathcal{U}_2|)$.

The server needs a storage of $O(\mu m)$ bits to store the inputs from users and precomputed Lagrange basis polynomials values. Moreover, in the worst case, the server needs $\frac{m\ell}{2}\mu$ bits to store secret shares of one-time pairwise keys over $\mathbb{F}_q$. The storage costs for the received inputs $\boldsymbol{\alpha}_i$ and $\boldsymbol{\beta}_i$ from users are mnr and $|\mathcal{U}_1|\cdot m\mu$ as each input is of nr bits. The storage cost for the commitments for the share verification is $m\mu(t+\ell)$ Therefore, the total storage cost is $(mnr+|\mathcal{U}_1|\cdot m\mu+\frac{m\ell\mu}{2}+m\mu(t+\ell))$ bits. The communication costs for the server involved in receiving inputs $\boldsymbol{\alpha}_i$ and $\boldsymbol{\beta}_i$ from users and secret shares for one-time pairwise keys and the commitments of shares are $|\mathcal{U}_1|\cdot nr$, $|\mathcal{U}_1|\cdot m\mu$, $\frac{m\ell}{2}$, and $m\mu(t+\ell)$ respectively. Table 3 presents a summary of the numbers of unit operations performed by each user and the server.

Table 3: Number of unit operations per user. † the modular sum is the operations over high-dimensional inputs.

	Operation	# Op		Operation	# Bits
Computation	**User**		Communication	**User**	
	# Poly Eval	$4(m-1)$		Sending $\boldsymbol{\alpha}_i$	nr
	# KDF Eval	$2(m-1)$		$\mathbf{Share}^t_m$	$\mu(m-1)$
	# Modular Exp ($\mathbb{Z}_N$)	2		$\mathbf{Share}^2_3$	$\frac{\ell\mu}{2}$
	# PRG Eval	ℓ		2 pairs of SS Poly	$4k\lambda$
	# Modular Sum† ($\mathbb{Z}_{2^n}$)	$\ell+1$		Fail. rec. $\boldsymbol{\beta}_i$	$m\mu$
	# $\mathbf{Share}^t_m$	1		$\mathbf{Comm}^t$	$t\mu$
	# $\mathbf{Share}^2_3$	$\lceil\frac{\ell}{2}\rceil$		$\mathbf{Comm}^2$	$\ell\mu$
	# $\mathrm{Enc}_{\kappa_i}()$	1			
	# $\mathcal{E}()$+# $\mathcal{D}()$	$2m+1$			
	# $F^{\nu-\tau}()$	2			
	Server			**Server**	
	# $\mathbf{Rec}^t_m$	$\lvert\mathcal{U}_2\rvert(\leq m)$		Receiving $\boldsymbol{\alpha}_i$	$\lvert\mathcal{U}_1\rvert\cdot nr$
	# $\mathrm{Dec}_{\kappa_i}()$	$\lvert\mathcal{U}_2\rvert$		Receiving $\boldsymbol{\beta}_i$	$\lvert\mathcal{U}_2\rvert\cdot m\mu$
	# $\mathbf{Rec}^2_3$	$\lvert\mathcal{U}_2\rvert\cdot\lvert\mathcal{U}_0\backslash\mathcal{U}_2\rvert$		# $\mathbf{Share}^2_3$	$\frac{m\ell}{2}$
	# PRG Eval	$\lvert\mathcal{U}_2\rvert\cdot\lvert\mathcal{U}_0\backslash\mathcal{U}_2\rvert$			
	# Modular Sum	$\lvert\mathcal{U}_2\rvert\cdot(\lvert\mathcal{U}_0\backslash\mathcal{U}_2\rvert+1)$			

5.8 Comparisons with Previous Work

We compare the performance of each user and the server of our scheme with the ones in [7] and in [1] because these are the closest ones to our work (see Table 4). We start by highlighting the key sharing phase. The schemes in [1, 7] used the DH key exchange scheme to establish pairwise keys in the online phase, which adds an overhead of a quadratic communication complexity in the number of users, but in our scheme, we employ a non-interactive key establishment scheme for generating pairwise keys, which does not need any communication between a pair of users, resulting in **zero communication** for establishing pairwise keys. It assumes an existence of two SSPs and one time communication cost between a user and two SSPs to obtain two pairs of secret shared polynomials in the offline phase. The advantage of our scheme is that the users never provide the master pairwise key to the server, it derives one-time pairwise keys using a one-way function. In the previous work [1, 7], the users need to establish new pairwise keys for each aggregate-sum computation, which is not efficient for multiple aggregate-sum computations for certain applications.

The scheme in [1] has extra computation and communication costs for the failure-recovery scenario and multi-dimensional inputs scenario because, for each dropping out phase, the alive users need to send the key material of length same as the input length. One difference between our scheme and the scheme in [1] is that we first mask the input and then encrypt it, but the scheme in [1] first encrypts with modular additions and then performs a mask operation.

When compared our scheme with the one in [7], the mask-then-encrypt operation in our scheme is much lighter than the **Masked-InputCollection** operation in [7] due to masking the input with coordinated noises based on an ℓ-regular network and lightweight $\mathbf{Share}_3^2$ operations. This reduces communication and computational costs for both users and the server (see Figures 7a and 7b). Table 4 presents a performance comparison with other approaches for the server and each user. A detailed comparison on the number of the unit operations in different schemes is given in Table 5. The storage complexity of our scheme for the server is the same as the one in [7], but for the users, our scheme needs extra storage of $4k\lambda$ bits for storing two pairs of secret shared polynomials. Notice that our system has two secret service providers that are offline entities and do not participate in the aggregate-sum computation protocol. Since the system models in [5, 12] are different, we do not compare our scheme with those. In summary, in our NIKE-based aggregation scheme: 1) for establishing pairwise keys, we save a communication overhead of $O(m^2)$ over the network, but there is a computational overhead of $O(\log^2(m))$ per user; and 2) we improve the state-of-art input encoding and decoding techniques that reduce the computational complexities for both users and the server.

6 EXPERIMENTAL EVALUATION

To evaluate the performance of the scheme, we implemented it against semi-honest adversaries on a smartphone as well as on a desktop. We use a Google Nexus 5 smartphone with Quad-core 2.3 GHz Krait 400 CPU and 2 GB RAM, running Android 5.0 Lollipop OS. For the desktop implementation, the experiments were conducted on a desktop with Ubuntu 16.04 OS with `Intel i7-6700 CPU@3.40GHz` with 8 GB RAM.

6.1 Implementation Details

Smartphone implementation. To evaluate the efficiency of the mask-then-encrypt operation on the smartphone, we implement the pairwise key computation algorithm over $\mathbb{Z}_N$, Shamir's (t, m)-tss over $\mathbb{F}_q$ for creating shares of secret κ_i, and $(2, 3)$-tss over $\mathbb{F}_q$ for creating shares of one-time pairwise keys. We use SHA256 as a KDF and AES in counter mode as a PRG from OpenSSL1.1 [38]. In our implementation, we generate input $\mathbf{x}_i$ randomly.

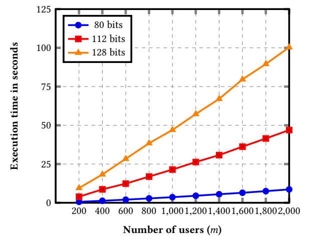
Desktop implementation. To measure the efficiency, we implement the scheme including all essential modules: 1) Secret shared polynomial generation by the SSPs; 2) Mask-then-encrypt algorithm for users; 3) Decrypt-then-unmask algorithm for the server. The schemes were implemented in C using the GMP [22] and FLINT [25] libraries to leverage large number operations and efficient multipoint polynomial evaluations. The codes were compiled using gcc 5.4.0 with -std=c99 -O1 -fomit-frame-pointer flag. Below are the implementation details of the crypto-primitives that offer 128-bit security.

– For the SSP implementation, we use the distributed RSA prime generation code of Mauland, written in Python for VIFF [46], for generating primes and hence the RSA modulus. We implement the basic modular inverse computation protocol of Catalano *et al.* [10] in two-party settings.
– For PRG, we use AES in counter mode implemented using AES-NI instructions to achieve the highest level of efficiency.
– For authenticated encryption $\mathcal{AE}$, we use AES-GCM from OpenSSL 1.0.2g.
– For KDF and one-way function, we use SHA256 from OpenSSL.
– For ECDH, we use the NIST P-256 curve from OpenSSL.
– For $(\mathbf{Share}_m^t, \mathbf{Rec}_m^t)$ and $(\mathbf{Share}_3^2, \mathbf{Rec}_3^2)$ operations, we use Shamir's secret sharing scheme implemented using the FLINT polynomial operation library, with NIST's 256-bit prime q.

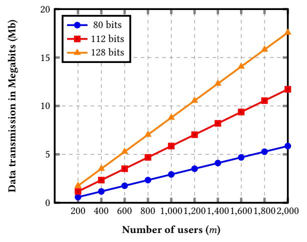
The ℓ-regular graphs are generated randomly, and a heuristic algorithm is designed and used for the assignments of $\mathbf{Share}_3^2$ as each user performs $\frac{\ell}{2}$ $\mathbf{Share}_3^2$ operations. Using two pairs of secret shared polynomials over $\mathbb{Z}_N[x]$, each user derives all master pairwise keys by applying a KDF and converting its output to elements of $\mathbb{F}_q$. We use SHA256 to implement one-way key chain for one-time pairwise key derivation. Note that the $(\mathbf{Share}_m^t, \mathbf{Rec}_m^t)$ and $(\mathbf{Share}_3^2, \mathbf{Rec}_3^2)$ operations for creating shares of κ_i and $MK_{i,j}^{l,\tau}$'s are defined over $\mathbb{F}_q$. Codes will be available upon a request.

6.2 Efficiency Results for Distributed SSP

We configured two desktops to build a distributed SSP with the same configuration, `Intel i7-6700 CPU@3.40GHz` with 8 GB RAM, connected through a LAN for measuring the performance of the distributed SSP. The speed of the LAN is 1 Gbps. We record the time for SSP_1 and SSP_2 for computing secret shared polynomials for all users. We chose the degree of the secret shared polynomials to be the same as the threshold value t in the (t, m)-tss that is decided based on the number of colluding users in the system. We choose $k = t = \lceil \frac{m}{3} \rceil$. We measure the performance by considering three security levels of 80, 112 and 128 bits, and the RSA modulus sizes are 1024, 2048 and 3072 bits [4]. Figure 5a shows the time (computation and network communication) needed to finish one of the SSPs secret shared polynomials computations. Figure 5b shows the amount of bits transmitted over the LAN for computing one of the SSPs secret shared polynomials. We omitted the computation and communication times for the authenticated encryption $\mathcal{AE}$ for delivering the shares.



(a) Distributed secret shared polynomial computation time of SSP_1



(b) Amount of data transfer over LAN between two SSPs

Figure 5: Execution time for $\left\{(s_{1,j}(x), s_{2,j}(x)) : j \in [m]\right\}$ computations and the amount of data exchange over LAN.

Table 4: A cost comparison with other schemes for each user and the server: (a) m denotes the number of users, t is the threshold value in (t, m)-tss; (b) ℓ denotes the degree of the nodes in the communication network; (c) $(k-1)$ is the degree of the secret shared polynomial; (d) r denotes the dimension of $\mathbf{x}_i$, $\mu = |q|$, $\lambda = |N|$; (e) γ is the number of bits to represent a DH group element; (f) $c(r, n)$ is the cost of computing PRG; (g) $\zeta = |\mathcal{U}_0 \backslash \mathcal{U}_2|$ is the number of dropouts. ‡ Each alive user may need to perform this operation multiple times based on drop our scenario.

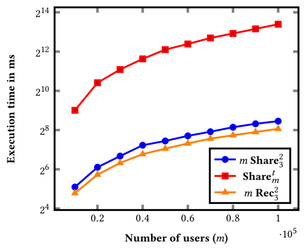
Entity	Communication	Computation	Storage	Reference
User	$O\Big((\lambda+\mu)m + nr\Big)$	$O\Big(m\log^2(m) + (\ell+1)c(r,n)\Big)$	$O\Big(4k\lambda + \mu(m + 3\cdot\lceil\frac{\ell}{2}\rceil) + nr\Big)$	**This paper**
	$O\Big(m\gamma + nr\Big)$	$O\Big(m^2 + (m-1)c(r,n)\Big)$	$O\Big((\gamma + 2\mu)m + nr\Big)$	Bonawitz et al. [7]
	$O\Big(m\gamma + nr\Big)^{‡}$	$O\Big((\ell+1)c(r,n)\Big)^{‡}$	$O\Big((\ell+1)\mu + nr\Big)$	Acs and Castelluccia [1]
Server	$O(m^2\mu + nmr + \frac{m\ell}{2})$	$O(m^2 + (m-\zeta)\cdot\ell\cdot c(r,n))$	$O(mnr + m^2\mu + \frac{m\ell\mu}{2})$	**This paper**
	$O(m^2\gamma + nmr)$	$O(m^2 + (m-\zeta)(m-1)c(r,n))$	$O(m^2\mu + mnr)$	Bonawitz et al. [7]
	$O(nmr)^{‡}$	$O(nmr)^{‡}$	$O(nmr)$	Acs and Castelluccia [1]

Table 5: A comparison of unit computation heavy operations and unit communications per user for an aggregate sum. $\mu = |q|$, $|\mathcal{U}_2| = \theta$, $\zeta = |\mathcal{U}_0 \backslash \mathcal{U}_2|$, **and** $\zeta + \theta = m$.

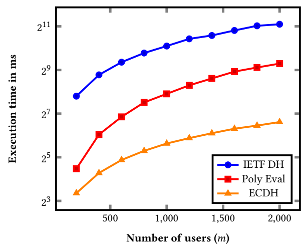
Computation in mask-then-encrypt					
Share$_m^t$	**Share**$_3^2$	# PRG eval	# MD Modular Sum	#Enc$_{\kappa_i}$	Ref.
1	$\frac{\ell}{2}$	ℓ	$\ell+1$	1	**This paper**
2	0	$(m-1)$	m	1	Bonawitz et al. [7]

Computation in decrypt-then-unmask					
Rec$_m^t$	**Rec**$_3^2$	# PRG eval	# MD Modular Sum	#Dec$_{\kappa_i}$	Ref.
θ	$\leq \theta\cdot\ell$	$\leq \theta\cdot\ell$	$\theta + \theta\cdot\ell$	θ	**This paper**
$\theta+\zeta$	0	$\zeta(m-1)$	$\theta + \zeta(m-1)$	θ	Bonawitz et al. [7]

Communication in bits			
Shares of **Share**$_m^t$	Shares of **Share**$_3^2$	Overall network cost	Ref.
$(m-1)\mu$	$\ell\mu$	$m(m+\ell-1)\mu$	**This paper**
$2(m-1)\mu$	0	$2(m^2-m)\mu$	Bonawitz et al. [7]



(a) One Share$_m^t$ and m Share$_3^2$ time



(b) Pairwise key computation time

Figure 6: Comparing Share$_m^t$ and m Share$_3^2$ ops and the pairwise key establishment times for the 128-bit security level.

6.3 Efficiency Results of the Aggregate-sum Scheme

We consider input data $\mathbf{x}_i = (x_{i,1}, \cdots, x_{i,r})$ for $r = p \times 10^5, 1 \leq p \leq 10$ and each $x_{i,j}$ is of 16 bits ($n = 16$). The size of q for $\mathbb{F}_q$ is 256 bits for 128 bit security. We chose t and ℓ depending upon the threat model. We evaluate the performance of the protocol for two different sets of parameter capturing three different threat models.

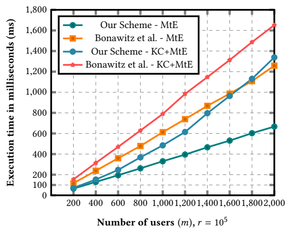
As **Share**$_m^t$, **Share**$_3^2$ and PRG are fundamental operations in our scheme, we provide a comparison of timings of one **Share**$_m^t$ and m **Share**$_3^2$ operations in Figure 6a. From Figure 6a we can see that m units of **Share**$_3^2$ operations is much faster than the **Share**$_m^t$ operation. **Share**$_3^2$ is one of the operations used to make the mask-then-encrypt algorithm lightweight. Figure 6b shows a comparison of pairwise key generation times for ECDH, polynomial evaluation based and IETF DH based [28]. For high-dimensional inputs, the PRG evaluation is an expensive operation when multiple of them are used to mask or encrypt one input. Using AES in counter mode implemented using AES-NI instructions (Skylake), on the desktop, we need about 4.51 ms to generate a keystream of length 2^{25} bits.

Execution time on desktop. Figure 7a presents the mask-then-encrypt (MtE) time for a user where the time includes the pairwise key computation, secret share computation of κ_i and the modular operation over $\mathbb{Z}_{2^n}^r$ and the dimension of $\mathbf{x}_i$ is set to $r = 10^5$. The masking time mainly depends on the size of the neighbor and the dimension of the input. Figure 7b displays the decrypt-then-unmask operation time by the server for no drop out and 33% drop out cases from the system. Since the dropouts of users are random, storing all possible pre-computed the Lagrange coefficients at zero needs an exponential amount of storage. So it is realistic to compute the Lagrange coefficients on-the-fly based on the dropouts in the system. When the number of users is $m = 10^3$ and the input dimension $r = 10^5$, the mask-then-encrypt operation takes about 484 ms per user, and the server needs about 549 ms to compute the aggregate sum with no drop out.

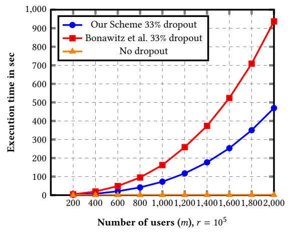
We also implemented **ShareKeys**, **MaskedInputCollection** and **Unmasking** operations of Bonawitz et al.'s scheme [7] to have a fair comparison on the same platform. To the best our of knowledge, there is no code of the scheme publicly available. Figure 7a compares the execution times of our scheme and Bonawitz et al.'s scheme for the user/server-only threat model where we set $t = \lceil\frac{m}{3}\rceil$ in **Share**$_m^t$. For the no dropout case, the performances of both schemes are the same because both schemes perform m **Rec**$_m^t$, m PRG evaluation and $2m$ modular addition operations over $\mathbb{Z}_{2^n}^r$. However, for the dropout case, our scheme outperforms than Bonawitz et al.'s scheme because our scheme needs to perform a small number of PRG evaluations, and does **Rec**$_3^2$, instead of **Rec**$_m^t$ for one-time pairwise keys. Figure 7b shows how our scheme outperforms Bonawitz et al.'s scheme, for instance, for 33% user dropouts. When $m = 2000$, for 33% dropout, our scheme takes about 470 sec, but Bonawitz et al.'s scheme takes about 937 sec.

We compute the decrypt-then-unmasking operation time of our scheme for different dropout rates, see Figure 7d. For instance, for

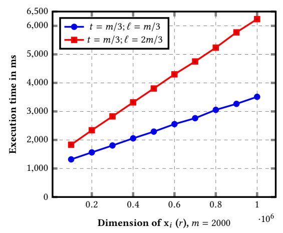
$m = 1000$ users, the server needs about 67 sec and 72 sec to calculate the sum for 5% and 30% drop out of users, respectively. Figure 7c presents the execution time for the mask-then-encrypt operation for different input dimensions and for the user/server-only and users-server threat models. Table 6 and 8 present the timings for the mask-then-encrypt, and decrypt-then-unmask operations for $m = 2000$ users for the user/server-only and users-server threat models, respectively. The results show that our scheme is faster than Bonawitz et al.'s scheme. From Table 6, we observe that, when the dropout is 30%, the computation time for the server is smaller than that for the server with 10% dropout. The reason is that there is a (time) tradeoff between the number of PRG evaluations and the number of $\mathbf{Rec}^t_m$ operations and as the amount of dropout increases, the server needs to run less $\mathbf{Rec}^t_m$ operations and run more PRG evaluations, which is much faster than one $\mathbf{Rec}^t_m$ operation for 2000 users. Detailed performance results of our scheme for the users-server threat model are provided in Appendix C.



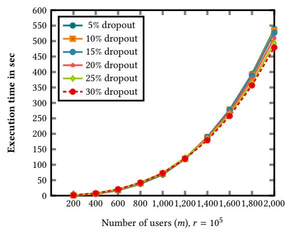
(a) Time for KeyComp (KC) and mask-then-encrypt (MtE) per user



(b) Time for decrypt-then-unmask with 33% Dropout



(c) KeyComp and mask-then-encrypt time for each user and $10^5 \le r \le 10^6$

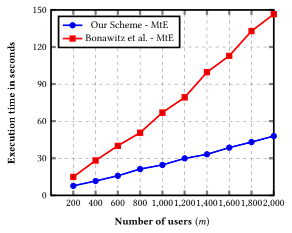


(d) Decrypt-then-unmask time for the server for different dropout rates

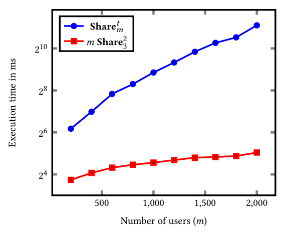
Figure 7: Comparison for mask-then-encrypt, decrypt-then-unmask with dropouts times on desktop and mask-then-encrypt with varied input dimensions.

Execution time on smartphone. Figure 8a illustrates the execution time for the mask-then-encrypt operation where the dimension of the input is set to $r = 10^5$. The mask-then-encrypt operation time includes times of $\mathbf{Share}^t_m$, $\mathbf{Share}^2_3$, PRG evaluation and modular addition operations. For $m = 10^3$ users, our mask-then-encrypt operation takes about 24 sec, but Bonawitz et al.'s **MaskedInput-Collection** takes 66 sec, whic is 2.75 times larger than ours. We also test the performance of one $\mathbf{Share}^t_m$ operation and $\lceil \frac{\ell}{2} \rceil$ $\mathbf{Share}^2_3$ operations. Figure 8b shows the timings for both operations. In Table 6, the pairwise keys in our scheme are dynamically computed and its computation time is larger compared to Bonawitz et al.'s, which is due to the $O((\log(m))^2)$ factor, but in our scheme, each user saves $O(m)$ in communication. We can save this key computation time by precomputing and storing $(m + \ell - 1)$ pairwise keys (called with storage "**w/ storage**" in Table 6).

Energy consumption of smartphone. For mobile devices, the energy consumption by cryptographic operations is an important factor to consider. We estimate the energy consumption of our scheme for mobile users. Table 7 presents the battery consumption of our scheme and Bonawitz et al.'s scheme. For $m = 2000$, our mask-then-encrypt operation consumes 2.03 mAh, on the other hand, Bonawitz et al.'s **MaskedInputCollection** consumes 7.43 mAh. Thus, the energy consumption results also demonstrate that our mask-then-encrypt operation is energy efficient than Bonawitz et al.'s scheme.



(a) Time for mask-then-encrypt operation ($r = 10^5$) on smartphone



(b) Comparing $\mathbf{Share}^t_m$ and m $\mathbf{Share}^2_3$ ops timings

Figure 8: Execution time on smartphone for mask-then-encrypt of one user, and times for $\mathbf{Share}^t_m$ and m $\mathbf{Share}^2_3$.

Interpreting of the efficiency of the scheme. For the users and the server, if the dimension of the inputs (r) is the dominating factor in the computational complexity and if we take a ratio between that of Bonawitz et al.'s scheme and our scheme, the ratio is roughly $\frac{(m-1)c(r,n)}{(\ell+1)c(r,n)} \approx \frac{m}{\ell}$ (see Tables 2- 5). Since in our scheme for different threat models we have different ℓ. This implies our scheme is roughly 1.5x to up to 3x faster that is achieved by saving 33% and 66% less PRG evaluations. If the number of users (m) is the dominating factor in the computational complexity, our scheme (for both the server and the user) is also faster because of employing $\mathbf{Share}^2_3$, one $\mathbf{Share}^t_m$ and ℓ PRG evaluations. Our experimental results also validate this phenomenon.

7 RELATED WORK

Secure data aggregation problem has been studied in the literature in different application perspectives such as wireless sensor networks, smart meters and smart grids, and machine learning applications by considering their underlying communication network's characteristics. Various protocols for secure data aggregation have been developed, e.g., [1, 7, 15, 19, 21, 23, 27, 29, 32, 42, 43].

Smart Meter Data Aggregation. Several schemes to securing smart metering applications have been developed based on cryptographic techniques such as homomorphic encryption, zero-knowledge proofs, stream cipher encryption and also based on anonymization (e.g., mixnet). However, the techniques based on anonymization do not provide strong security guarantees [29]. The work in [19, 32] proposed solutions based on additive homomorphic

Table 6: Timings (in millisecond) for different operations performed by the server, desktop user and mobile users for three different threat models. The parameters t and ℓ are different for different threat models. The dimension of $\mathbf{x}_i$ is 10^5.

	Timings for Users/Server-only threat model. Our scheme: $t = \lceil \frac{m}{3} \rceil, \ell = \lceil \frac{m}{3} \rceil$. Bonawitz et al.'s: $t = \lceil \frac{m}{3} \rceil, \ell = m - 1$.							
Entity	**# Users**	**Dropout**	**KeyComp**	**Mask-then-encrypt** (MtE)	**Decrypt-then-unmask**	**Total time w/o storage**	**Total time w/ storage**	Ref.
User	2000	0%	666.01 ms	670.03 ms	40.56 ms	1,376.60 ms	710.59 ms	**Our scheme**
User	2000	0%	91.09 ms	1528.25 ms	40.74 ms	1,660.08 ms	1568.99 ms	Bonawitz et al. [7]
Mobile user	2000	0%	71,9784 ms	47,994 ms	125.93 ms	767,903.93 ms	48,119.93 ms	**Our scheme**
Mobile user	2000	0%	29,474.7 ms	146,570 ms	127.21 ms	176,171.91 ms	146,697.22 ms	Bonawitz et al. [7]
Server	2000	0%	–	–	1,297.47 ms	1,297.47 ms	–	**Our Scheme**
Server	2000	10%	–	–	523,219.11 ms	523,219.11 ms	–	
Server	2000	20%	–	–	504,755.50 ms	504,755.50 ms	–	
Server	2000	30%	–	–	479,957.99 ms	479,957.99 ms	–	
Server	2000	0%	–	–	1,305.94 ms	1,305.94 ms	–	Bonawitz et al. [7]
Server	2000	10%	–	–	682,368.69 ms	682,368.69 ms	–	
Server	2000	20%	–	–	801,950.15 ms	801,950.15 ms	–	
Server	2000	30%	–	–	888,816.81 ms	888,816.81 ms	–	

Table 7: Energy consumption (in mAh) of the scheme and a comparison with other scheme.

# Users	200	400	600	800	1000	1200	1400	1600	1800	2000	Ref.
MtE	0.365	0.597	0.631	0.982	1.04	1.37	1.51	1.85	1.94	2.03	**Our scheme**
MaskedInput	0.74	1.10	1.81	2.39	2.58	3.14	5.16	6.31	7.02	7.43	Bonawitz et al.

encryptions for securing smart metering and grids applications. The work in [19] proposed a protocol for two rounds: 1) each meter first splits the input data into some number of measurements and encrypts the shares with other meters' public key and sends to the aggregator; 2) the aggregator multiplies all ciphertexts and sends to the meter to decrypt and add its share to the sum where each meter has a public key certificate. Acs and Castelluccia [1] present an aggregation scheme for securely collecting smart meter data. In their scheme, the smart meters encrypts data using a key shared between the meter and the aggregated and several pairwise keys with other meters. A Diffie-Hellman key exchange protocol is used to establish the pairwise keys among meters. In [5], Barthe *et al.* proposed an aggregation protocol for smart meter data collection where the system consists of a set of meters, a set of aggregators and a service provider where the service provider wants to learn the weighted aggregated sum. The meter data is masked with the sum of hashed pairwise keys with the meter and the set of aggregators. Their system model is different from ours.

WSN Data Aggregation. A large volume of publications in the literature have investigated the problem of secure data aggregation in wireless sensor networks by exploiting properties of networks such as multipath routing. In such settings, certain nodes may act as aggregators who collect information from the sensors within the network. Hu and Evans [27] propose a protocol for secure aggregation in sensor networks against a single adversary. The sensor nodes are arranged in a logical tree and each node uses a one-time MAC key to ensure integrity of the message. No confidentiality of the message is considered. Przydatek *et al.* [42] present a secure aggregation framework for the single aggregator model where a single node acts an aggregator to do aggregation. Chan *et al.* [11] develop a technique for secure data aggregation which supports several aggregated queries such as median, sum and average. Castelluccia *et al.* [9] propose a provably secure encryption scheme that enables additive aggregation of encrypted data in sensor networks. Roy *et al.* [43] propose a lightweight verification algorithm for securing the synopsis diffusion approach, which detects false subaggregate values computed at the base station.

Secure Data Aggregation. In [7], Bonawitz *et al.* proposed a protocol for secure high-dimensional data aggregation by a server which computes the aggregate sum for using it in machine learning algorithms. In their scheme, a user masks its data with the sum of all other users' keys. The robustness of the protocol when a set of users drop out from the system is considered. Goryczka and Xiong [21] consider a scenario for secure data aggregation in a distributed way and present two schemes that offer differential privacy. The user data is masked with a random noise where random noises are generate using distributed Laplace and geometric perturbation algorithms. Halevi *et al.* [23] proposed a one-pass protocol based on homomorphic encryption for computing the sum of user inputs.

Data Aggregation based on MPC and Mixed Approach. Secure multiparty computation (SMPC) allows a set of parties to privately compute a function without revealing anything about parties' inputs, except what can be learned from the output. SMPC protocols can be classified into two main categories: 1) 2-party computation protocols based on Yao's garbled circuit approach [47] and 2) multiparty computation based on secret sharing and homomorphic encryption [20, 44]. A generic secure multiparty computation technique can be applied to perform data aggregation, but that may be expensive in terms of computational and communications.

The work in [13] present a multi-user data aggregation based on homomorphic proxy re-authenticators. The scheme in [13] is based on computationally expensive cryptographic operations such as homomorphic signatures, which may not suitable for mobile applications. Elahi *et al.* [14] propose a system for collecting clients' aggregated statistics from anonymous network. Our work is different from theirs. Shi et al. [45] propose an aggregate sum protocol employing a homomorphic encryption scheme for time series data. The protocol guarantees the security against user input data and the distributed differential privacy for each user. Corrigan-Gibbs and Boneh [12] propose a system for computing aggregated statistics in a privacy preserving way. In their system model, the users secret share their input data among a set of servers and the set of servers compute aggregate statistics. At the core of their scheme is secret-shared non-interactive proofs for robustness.

8 CONCLUSIONS

This paper presented a system that allows a server to perform the aggregate sum of high-dimensional inputs from mobile users in a privacy-preserving way with low communication and computation overhead. We made the pairwise key generation task non-interactive by outsourcing it to a cryptographic service provider realized by two SSPs, which saves quadratic amount of communication costs. We designed an aggregate-sum protocol that has low communication and computation overheads and provides resistance against the failure recovery scenario. The security and performance of the scheme are analyzed. We evaluated and compared the performance of our scheme on a smartphone and on a desktop PC by conducting experiments on high-dimensional inputs. Our experiments show that our scheme is faster than existing ones.

In the mask-then-encrypt operation, each user needs to perform one $\mathbf{Share}_m^t$ operation, and for a large-scale network, which is computation heavy for mobile devices. We leave this as an open question that how to eliminate the (t, m)-tss operation while guaranteeing the (dropout scenario) robustness and security in the scheme.

REFERENCES

[1] Gergely Ács and Claude Castelluccia. 2011. I Have a DREAM!: Differentially Private Smart Metering. In *Proceedings of the 13th International Conference on Information Hiding (IH'11)*. Springer-Verlag, Berlin, Heidelberg, 118–132. http://dl.acm.org/citation.cfm?id=2042445.2042457

[2] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. 1974. *The Design and Analysis of Computer Algorithms* (1st ed.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[3] Joy Algesheimer, Jan Camenisch, and Victor Shoup. 2002. *Efficient Computation Modulo a Shared Secret with Application to the Generation of Shared Safe-Prime Products*. Springer Berlin Heidelberg, Berlin, Heidelberg, 417–432. https://doi.org/10.1007/3-540-45708-9_27

[4] Elaine Barker, William Barker, William Burr, William Polk, and Miles Smid. 2012. Recommendation for key management part 1: General (revision 3). *NIST special publication* 800, 57 (2012), 1–147.

[5] Gilles Barthe, George Danezis, Benjamin Gregoire, Cesar Kunz, and Santiago Zanella-Beguelin. 2013. Verified Computational Differential Privacy with Applications to Smart Metering. In *Proceedings of the 2013 IEEE 26th Computer Security Foundations Symposium (CSF '13)*. IEEE Computer Society, Washington, DC, USA, 287–301. https://doi.org/10.1109/CSF.2013.26

[6] Carlo Blundo and Paolo D'Arco. 2005. Analysis and Design of Distributed Key Distribution Centers. *Journal of Cryptology* 18, 4 (01 Sep 2005), 391–414. https://doi.org/10.1007/s00145-005-0407-0

[7] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical Secure Aggregation for Privacy-Preserving Machine Learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*. ACM, New York, NY, USA, 1175–1191. https://doi.org/10.1145/3133956.3133982

[8] Dan Boneh and Matthew Franklin. 1997. *Efficient generation of shared RSA keys*. Springer Berlin Heidelberg, Berlin, Heidelberg, 425–439. https://doi.org/10.1007/BFb0052253

[9] Claude Castelluccia, Aldar C-F. Chan, Einar Mykletun, and Gene Tsudik. 2009. Efficient and Provably Secure Aggregation of Encrypted Data in Wireless Sensor Networks. *ACM Trans. Sen. Netw.* 5, 3, Article 20 (June 2009), 36 pages. https://doi.org/10.1145/1525856.1525858

[10] Dario Catalano, Rosario Gennaro, and Shai Halevi. 2000. Computing Inverses over a Shared Secret Modulus. In *EUROCRYPT 2000*. Springer-Verlag, Berlin, Heidelberg, 190–206. http://dl.acm.org/citation.cfm?id=1756169.1756189

[11] Haowen Chan, Adrian Perrig, and Dawn Song. 2006. Secure Hierarchical In-network Aggregation in Sensor Networks. In *CCS 2006*. ACM, New York, NY, USA, 278–287. https://doi.org/10.1145/1180405.1180440

[12] Henry Corrigan-Gibbs and Dan Boneh. 2017. Prio: Private, Robust, and Scalable Computation of Aggregate Statistics. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation (NSDI'17)*. USENIX Association, Berkeley, CA, USA, 259–282. http://dl.acm.org/citation.cfm?id=3154630.3154652

[13] David Derler, Sebastian Ramacher, and Daniel Slamanig. 2017. Homomorphic Proxy Re-Authenticators and Applications to Verifiable Multi-User Data Aggregation. Cryptology ePrint Archive, Report 2017/086. (2017). https://eprint.iacr.org/2017/086.

[14] Tariq Elahi, George Danezis, and Ian Goldberg. 2014. PrivEx: Private Collection of Traffic Statistics for Anonymous Communication Networks. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS '14)*. ACM, New York, NY, USA, 1068–1079. https://doi.org/10.1145/2660267.2660280

[15] Zekeriya Erkin and Gene Tsudik. 2012. Private Computation of Spatial and Temporal Power Consumption with Smart Meters. In *ACNS 2012*. Springer-Verlag, Berlin, Heidelberg, 561–577. https://doi.org/10.1007/978-3-642-31284-7_33

[16] Paul Feldman. 1987. A Practical Scheme for Non-interactive Verifiable Secret Sharing. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science (SFCS '87)*. IEEE Computer Society, Washington, DC, USA, 427–438. https://doi.org/10.1109/SFCS.1987.4

[17] IMS Institute for Healthcare Informatics. Accessed April 6, 2017. Patient apps for improved healthcare from novelty to mainstream. (Accessed April 6, 2017). http://moodle.univ-lille2.fr/pluginfile.php/191819/mod_resource/content/0/patients%20apps%20for%20improved%20healthcare%20IMS.pdf

[18] Eduarda S. V. Freire, Dennis Hofheinz, Eike Kiltz, and Kenneth G. Paterson. 2013. Non-Interactive Key Exchange. In *Public-Key Cryptography – PKC 2013*, Kaoru Kurosawa and Goichiro Hanaoka (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 254–271.

[19] Flavio D. Garcia and Bart Jacobs. 2011. Privacy-friendly Energy-metering via Homomorphic Encryption. In *STM 2011*. Springer-Verlag, Berlin, Heidelberg, 226–238. http://dl.acm.org/citation.cfm?id=2050149.2050164

[20] O. Goldreich, S. Micali, and A. Wigderson. 1987. How to Play ANY Mental Game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing (STOC '87)*. ACM, New York, NY, USA, 218–229. https://doi.org/10.1145/28395.28420

[21] S. Goryczka and L. Xiong. 2017. A Comprehensive Comparison of Multiparty Secure Additions with Differential Privacy. *IEEE Transactions on Dependable and Secure Computing* 14, 5 (Sept 2017), 463–477. https://doi.org/10.1109/TDSC.2015.2484326

[22] Torbjörn Granlund et al. 1991. GMP, the GNU multiple precision arithmetic library. (1991).

[23] Shai Halevi, Yehuda Lindell, and Benny Pinkas. 2011. Secure Computation on the Web: Computing Without Simultaneous Interaction. In *CRYPTO 2011*. Springer-Verlag, Berlin, Heidelberg, 132–150. http://dl.acm.org/citation.cfm?id=2033036.2033047

[24] Lein Harn and Guang Gong. 2015. Conference key establishment protocol using a multivariate polynomial and its applications. *Security and Communication Networks* 8, 9 (2015), 1794–1800. https://doi.org/10.1002/sec.1143 SCN-14-0523.

[25] W. Hart, F. Johansson, and S. Pancratz. 2013. FLINT: Fast Library for Number Theory. (2013). Version 2.4.0, http://flintlib.org.

[26] Carmit Hazay, Gert Læssøe Mikkelsen, Tal Rabin, and Tomas Toft. 2012. *Efficient RSA Key Generation and Threshold Paillier in the Two-Party Setting*. Springer Berlin Heidelberg, Berlin, Heidelberg, 313–331. https://doi.org/10.1007/978-3-642-27954-6_20

[27] Lingxuan Hu and David Evans. 2003. Secure Aggregation for Wireless Networks. In *SAINT-W 2003*. IEEE Computer Society, Washington, DC, USA, 384–. http://dl.acm.org/citation.cfm?id=827275.829375

[28] Tero Kivinen. 2003. More modular exponential (modp) diffie-hellman groups for internet key exchange (IKE). (2003). https://tools.ietf.org/html/rfc3526.

[29] Aleksandra Korolova, Krishnaram Kenthapadi, Nina Mishra, and Alexandros Ntoulas. 2009. Releasing Search Queries and Clicks Privately. In *WWW 2009*. ACM, New York, NY, USA, 171–180. https://doi.org/10.1145/1526709.1526733

[30] Hugo Krawczyk. 2010. Cryptographic Extraction and Key Derivation: The HKDF Scheme. In *Advances in Cryptology – CRYPTO 2010*, Tal Rabin (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 631–648.

[31] Leslie Lamport. 1981. Password Authentication with Insecure Communication. *Commun. ACM* 24, 11 (Nov. 1981), 770–772. https://doi.org/10.1145/358790.358797

[32] F. Li, B. Luo, and P. Liu. 2010. Secure Information Aggregation for Smart Grids Using Homomorphic Encryption. In *2010 First IEEE International Conference on Smart Grid Communications*. 327–332. https://doi.org/10.1109/SMARTGRID.2010.5622064

[33] Ueli M. Maurer and Yacov Yacobi. 1991. Non-interactive Public-key Cryptography. In *Proceedings of the 10th Annual International Conference on Theory and Application of Cryptographic Techniques (EUROCRYPT'91)*. Springer-Verlag, Berlin, Heidelberg, 498–507. http://dl.acm.org/citation.cfm?id=1754868.1754924

[34] Payman Mohassel and Yupeng Zhang. 2017. SecureML: A System for Scalable Privacy-Preserving Machine Learning. Cryptology ePrint Archive, Report 2017/396. (2017). http://eprint.iacr.org/2017/396.

[35] Moni Naor, Benny Pinkas, and Omer Reingold. 1999. Distributed Pseudo-random Functions and KDCs. In *EUROCRYPT 1999*. Springer-Verlag, Berlin, Heidelberg, 327–346. http://dl.acm.org/citation.cfm?id=1756123.1756155

[36] Jesper Buus Nielsen. 2002. Separating Random Oracle Proofs from Complexity Theoretic Proofs: The Non-committing Encryption Case. In *Advances in Cryptology — CRYPTO 2002*, Moti Yung (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 111–126.

[37] Valeria Nikolaenko, Udi Weinsberg, Stratis Ioannidis, Marc Joye, Dan Boneh, and Nina Taft. 2013. Privacy-Preserving Ridge Regression on Hundreds of Millions of Records. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy (SP '13)*. IEEE Computer Society, Washington, DC, USA, 334–348. https://doi.org/10.1109/SP.2013.30

[38] OpenSSL. [n. d.]. The OpenSSL library. ([n. d.]). https://www.openssl.org/.

[39] Adrian Perrig and J. D. Tygar. 2003. *TESLA Broadcast Authentication*. Springer US, Boston, MA, 29–53. https://doi.org/10.1007/978-1-4615-0229-6_3

[40] David Pointcheval and Olivier Sanders. 2014. Forward Secure Non-Interactive Key Exchange. In *Security and Cryptography for Networks*, Michel Abdalla and Roberto De Prisco (Eds.). Springer International Publishing, Cham, 21–39.

[41] Ivens Portugal, Paulo Alencar, and Donald Cowan. 2018. The use of machine learning algorithms in recommender systems: A systematic review. *Expert Systems with Applications* 97 (2018), 205 – 227. https://doi.org/10.1016/j.eswa.2017.12.020

[42] Bartosz Przydatek, Dawn Song, and Adrian Perrig. 2003. SIA: Secure information aggregation in sensor networks. In *SenSys 2003*. ACM, 255–265.

[43] S. Roy, M. Conti, S. Setia, and S. Jajodia. 2014. Secure Data Aggregation in Wireless Sensor Networks: Filtering out the Attacker's Impact. *IEEE Transactions on Information Forensics and Security* 9, 4 (April 2014), 681–694. https://doi.org/10.1109/TIFS.2014.2307197

[44] Adi Shamir. 1979. How to Share a Secret. *Commun. ACM* 22, 11 (Nov. 1979), 612–613. https://doi.org/10.1145/359168.359176

[45] Elaine Shi, HTH Chan, Eleanor Rieffel, Richard Chow, and Dawn Song. 2011. Privacy-preserving aggregation of time-series data. In *Annual Network & Distributed System Security Symposium (NDSS)*. Internet Society.

[46] VIFF. [n. d.]. Distributed RSA. ([n. d.]). http://viff.dk/doc/applications.html.

[47] Andrew C. Yao. 1982. Protocols for Secure Computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (SFCS '82)*. IEEE Computer Society, Washington, DC, USA, 160–164. https://doi.org/10.1109/SFCS.1982.88

[48] Shuai Zhang, Lina Yao, and Aixin Sun. 2017. Deep Learning based Recommender System: A Survey and New Perspectives. *CoRR* abs/1707.07435 (2017). arXiv:1707.07435 http://arxiv.org/abs/1707.07435

A MODULAR EXPONENTIATION PROTOCOL

We present a new protocol for computing the modular exponentiation based on RSA.

Protocol 3: Modular Exponentiation (π_{EXP})

Private inputs: SSP_1 has private inputs d_1 and x. SSP_2 has a private input d_2.

Public parameters: e such that $ed = 1 \bmod \phi(N)$ and $d = d_1 + d_2$, $e = m$ (odd) or $e = m - 1$ (even).

Output: SSP_1 receives output $x^{d_1+d_2}$.

- SSP_1 encrypts x as $c = x^e \bmod N$ and sends it to SSP_2.
- SSP_2 computes $c' = c^{d_2} \bmod N$ and $c'' = c^{d_2^2} \bmod N$ and sends (c', c'') to SSP_1.
- After receiving (c', c''), SSP_1 computes $x^{d_1} \cdot c'^{d_1} \cdot c'' = x^{d_1} \cdot (x^{d_2})^{e(d_1+d_2)} = x^{d_1+d_2}$.

Figure 9: Protocol for modular exponentiation.

Theorem A.1. *The protocol π_{EXP} is secure assuming the factorization is hard.*

Proof. In [33], it is shown that computing discrete logarithms modulo N at least as hard as factoring N. Learning x from c is equivalent to solving the RSA problem, thus, SSP_2 can not learn anything about x. Again, learning d_2 from c' or c'' is equivalent to solving the DLP for a composite modulus. Thus SSP_1 and SSP_2 cannot learn about other's secret as long as the DLP for a composite modulus and the RSA problem are hard. □

B SECURITY PROOFS FOR SEMI-HONEST ADVERSARIES

B.1 Proof of Theorem 5.1

Proof of Theorem 5.1. Assume that the adversary controls a set of users in C and the set of users C is chosen before the protocol starts. The joint view of the adversary, denoted by $\texttt{VIEW}_C = \left\{\texttt{VIEW}_i\right\}_{i \in C}$, consists of all the corrupted users' views where each corrupted user's view $\texttt{VIEW}_i$ consists of the inputs of semi-honest users C, its internal random tape and encrypted messages (ciphertexts) received from honest users. Note that the ciphertexts are unrelated to $\boldsymbol{\alpha}_i$ or $\mathbf{x}_i$, i.e., no information about the inputs is used, and the internal random tapes used by the semi-honest users are the random coins in $\mathbf{Share}_m^t$, $\mathbf{Share}_3^2$ and $\mathcal{E}_{EK_{i,j}^{l,\tau}}(\cdot)$.

In the ideal model, the simulator $\mathcal{S}$ works as follows. It receives true inputs from the adversary and uses dummy inputs (e.g., $\mathbf{0}$ of appropriate length) for the honest users, and perfectly simulates the joint view of semi-honest users. From the real execution of the protocol, the semi-honest users know the identity of other users and common nonces used in the protocol, which are completely unrelated to the honest users' inputs. Therefore, the view of the simulator in the ideal model is identical to that of $\texttt{VIEW}_C$ in the real model. □

B.2 Proof of Theorem 5.2

Our construction of the simulator is quite similar to the one in [7] for semi-honest adversaries, except the one-time pairwise keys computation and the mask-then-encrypt based on the neighbor of users.

Proof of Theorem 5.2. We prove the theorem by the standard hybrid argument. We construct a simulator $\mathcal{S}$ though a series of hybrids which are constructed by subsequent modifications and show two subsequent hybrids are indistinguishable. Let $\mathcal{A}$ denote the adversary who sends messages on behalf of the corrupted (semi-honest) users.

Hybrid 0: This is the joint view $\texttt{VIEW}_C$ of the corrupted users C in the real execution of the protocol.

Hybrid 1: In this hybrid, the simulator replaces the one-time pairwise key $MK_{i,j}$ for a pair of honest users U_i and U_j by a uniformly random key. The security of KDF ensures that this hybrid is indistinguishable from the previous one.

Hybrid 2: In this hybrid, the simulator replaces the encryption and decryption key $EK_{i,j}$ for U_i and U_j by a uniformly random key. The output indistinguishability of the KDF ensures that this hybrid is indistinguishable from the previous one.

Hybrid 3: This hybrid is distributed identically same as the previous one except, all the ciphertexts for $\kappa_{i,j} \| s_j^{ij}$ or s_j^{ij} of honest users $\mathcal{U}_2 \backslash C$ is replaced by the encryption of $\mathbf{0}$ of appropriate lengths. However, in the failure-recovery information exchange phase, $\mathcal{S}$ responds with the correct shares of κ_i and $MK_{i,j}^{l,\tau}$ for honest users. The IND-CPA security of $\mathcal{E}$ guarantees that this hybrid is indistinguishable from **Hybrid 2**.

Hybrid 4: In this hybrid, the shares of κ_i for honest users are substituted with the shares of $\mathbf{0}$. Since $\mathcal{A}$ does not receive any information about the shares of the honest users in $\mathcal{U}_0 \backslash \mathcal{U}_2 \backslash C$, the combined view of the adversary contains only $|C|$ shares for each κ_i, which is less than t. Due to the perfect secrecy of Shamir's secret sharing ($\mathbf{Share}_m^t$), the distributions of the shares are identical. Therefore, this hybrid is indistinguishable from the previous one.

Hybrid 5: In this hybrid, we substitute the shares of $MK_{i,j}^{l,\tau}$ for each pair of honest users U_i and U_j with the shares of $\mathbf{0}$. In the Failure-recovery information exchange phase, the honest users do not reveal any shares of the honest users in $\mathcal{U}_0 \backslash \mathcal{U}_2 \backslash C$. Therefore, the joint view of the adversary contains only one share for each $MK_{i,j}^{l,\tau}$. The perfect secrecy of $\mathbf{Share}_3^2$ implies the distributions of the shares are identical. Thus, this hybrid is indistinguishable from the previous one.

Hybrid 6: This hybrid is defined exactly as the previous one, except that the encryption and decryption key $EK_{i,j}^{l,\tau}$ for each pair of honest users is replaced by a random key, instead of computed using $F^{v-\tau}(\cdot)$. Since the one-way function is implemented using a random oracle, the indistinguishability of the output ensures that this hybrid is indistinguishable from the previous one.

Hybrid 7: In this hybrid, for all honest users, we replace the key $\mathbf{k}_i \leftarrow \mathsf{PRG}(\kappa_i)$ by a uniform random string of the same length as of $\mathbf{k}_i$. The security of PRG ensures that this hybrid is indistinguishable from the previous one.

Hybrid 8: In this hybrid, for all honest users, the simulator performs the mask-then-encrypt operation on dummy inputs, say $\mathbf{0}$, i.e., $\boldsymbol{\alpha}_i = \mathbf{k}_i + \sum_{U \in \mathcal{N}_{\mathcal{U}_0}(U_i)} (-1)^{\delta_{i,\mathrm{id}(U)}} \mathbf{q}_{i,\mathrm{id}(U)}$ and sends to $\mathcal{A}$. Since, in **Hybrid 7**, $\mathbf{k}_i$ was chosen uniformly at random, so $\boldsymbol{\alpha}_i$ looks random. Thus, this hybrid is indistinguishable from the previous one.

Hybrid 9: In this hybrid, for a pair of honest users U_i, U_j in $\mathcal{U}_2 \backslash C$ and $U_j \in \mathcal{N}_{\mathcal{U}_0}(U_i)$, $\mathcal{S}$ substitutes the one-time pairwise key $MK_{i,j}^{l,\tau} = F^{v-\tau}(MK_{i,j}^{l} \| v \| \tau)$ generated using the one-way function with a uniformly random string $(R_{i,j})$ in $\mathbb{F}_q$ and computes $\boldsymbol{\alpha}'_i$ as

$$\boldsymbol{\alpha}'_i = \mathbf{x}_i + \mathbf{k}_i + \sum_{U_j \in \mathcal{N}_{\mathcal{U}_0}(U_i) \backslash \{U_j\}} (-1)^{\delta_{i,j}} \mathbf{q}_{i,j} + (-1)^{\delta_{i,j}} \mathsf{PRG}\left(R_{i,j}\right)$$

and for U_j as

$$\boldsymbol{\alpha}'_j = \mathbf{x}_j + \mathbf{k}_j + \sum_{U_u \in \mathcal{N}_{\mathcal{U}_0}(U_j)} (-1)^{\delta_{j,u}} \mathsf{PRG}\left(R_{j,u}\right).$$

There are polynomial many pairwise keys resulting in polynomial many hybrids. Since the one-way function is implemented using a random oracle, this hybrid is indistinguishable from the previous one.

Hybrid 10: This hybrid is the same as the previous one, except that $\mathbf{q}_{i,j} \leftarrow \mathsf{PRG}\left(MK_{i,j}^{l,\tau}, TS_\tau\right)$ for a pair of honest users U_i and U_j is replaced by a uniformly distributed random string of the same length. The security of PRG guarantees that this hybrid is indistinguishable from **Hybrid 9**.

Hybrid 11: We write the mask-then-encrypt operation as

$$\boldsymbol{\alpha}_i = \mathbf{x}_i + \mathbf{k}_i + \sum_{U_j \in \mathcal{N}_{\mathcal{U}_2}(U_i) \backslash \mathcal{N}_C(U_i)} (-1)^{\delta_{i,j}} \mathbf{q}_{i,j} + \sum_{U_j \in \mathcal{N}_{\mathcal{U}_0}(U_i) \backslash \mathcal{N}_{\mathcal{U}_2}(U_i) \backslash \mathcal{N}_C(U_i)} (-1)^{\delta_{i,j}} \mathbf{q}_{i,j}$$

In this hybrid, instead of sending above $\boldsymbol{\alpha}_i$, $\mathcal{S}$ sends to $\mathcal{A}$ the following

$$\boldsymbol{\alpha}'_i = \mathbf{y}_i + \mathbf{k}_i + \sum_{U_j \in \mathcal{N}_{\mathcal{U}_0}(U_i) \backslash \mathcal{N}_{\mathcal{U}_2}(U_i) \backslash \mathcal{N}_C(U_i)} (-1)^{\delta_{i,j}} \mathbf{q}_{i,j}$$

where $\left\{\mathbf{y}_i\right\}_{U_i \in \mathcal{U}_2 \backslash C}$ satisfying

$$\sum_{U_i \in \mathcal{U}_2 \backslash C} \mathbf{y}_j = \sum_{U_i \in \mathcal{U}_2 \backslash C} \mathbf{x}_j + \sum_{U_i \in \mathcal{U}_2 \backslash C} \sum_{U_j \in \mathcal{N}_{\mathcal{U}_2}(U_i) \backslash \mathcal{N}_C(U_i)} (-1)^{\delta_{i,j}} \mathbf{q}_{i,j} = \mathsf{SUM}_{\mathcal{U}_2 \backslash C}.$$

The adversary does not learn any information about the individual inputs of honest users, except the sum $\mathsf{SUM}_{\mathcal{U}_2 \backslash C}$. Thus, this hybrid is indistinguishable from the previous one.

This concludes the construction of the simulator. The output of the simulator is indistinguishable from the adversary's view in the real execution of the protocol. This completes the proof. □

C EXPERIMENTAL RESULTS FOR THE USERS-SERVER THREAT MODEL

In this section we present the performance of our scheme for the user-server threat model. In this threat model, the size of the neighbor is $\ell = 2\lceil \frac{m}{3} \rceil$ and $t = \lceil \frac{m}{3} \rceil$. Figures 10a and 10b present the timings of the mask-then-encrypt operation time on the desktop and on the mobile platform, respectively. Figure 10c shows the timings of the unmask-then-decrypt operation for different numbers of users. The timings for different dropout rates are given in Figure 10d.

D APPLICATION OF AGGREGATE-SUM IN MACHINE LEARNING

In this section, we present a concrete example how our aggregate-sum protocol can be used machine learning applications.

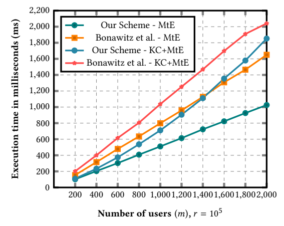
Application to Linear Regression. Suppose that there are m users in the system and each user holds an input $(\mathbf{x}_i, y_i), i \in [m]$ where $\mathbf{x}_i = (x_{1,i}, x_{2,i})$ and each $x_{i,j}$ and y_i are n bit integers over $\mathbb{Z}_{2^n}$. The server wishes to learn the linear regression model coefficient vector $\mathbf{c} = \left[c_0, c_1, c_2\right]^T$ on $\left\{(\mathbf{x}_i, y_i)\right\}_{i \in [m]}$ satisfying $y_i = L(\mathbf{x}_i) = c_0 + c_1 x_{1,i} + c_2 x_{2,i}$. The Least Squares Estimation for estimating $\mathbf{c}$ is given by $\mathbf{A}^{-1}\mathbf{b}$, i.e, $\mathbf{c}$ satisfies $\mathbf{A} \cdot \mathbf{c} = \mathbf{b}$ where $\mathbf{A} = Z^T Z$ and $\mathbf{b} = Z^T Y$ and $Y = (y_1, y_2, \cdots, y_m)$,

$$Z = \begin{bmatrix} 1 & x_{1,1} & x_{2,1} \\ 1 & x_{1,2} & x_{2,2} \\ \vdots & \vdots & \vdots \\ 1 & x_{1,m} & x_{2,m} \end{bmatrix}; \mathbf{A} = \begin{bmatrix} m & \sum_{i=1}^m x_{1,i} & \sum_{i=1}^m x_{2,i} \\ \sum_{i=1}^m x_{1,i} & \sum_{i=1}^m x_{1,i}^2 & \sum_{i=1}^m x_{2,i} x_{1,i} \\ \sum_{i=1}^m x_{2,i} & \sum_{i=1}^m x_{2,i} x_{1,i} & \sum_{i=1}^m x_{2,i}^2 \end{bmatrix}$$

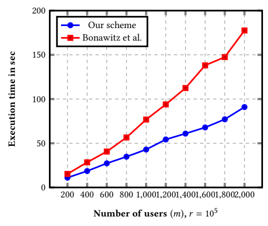
and $\mathbf{b} = \left[\sum_{i=1}^m y_i, \sum_{i=1}^m x_{1,i} y_i, \sum_{i=1}^m x_{2,i} y_i\right]^T$ and R^T denotes the transpose of R. Observe that each entry of matrix $\mathbf{A}$ and vector $\mathbf{b}$ is

Table 8: Timings (in millisecond) for different operations performed by the server, desktop user and mobile users for three different threat models. The parameters t and ℓ are different for different threat models. The dimension of $\mathbf{x}_i$ is 10^5.

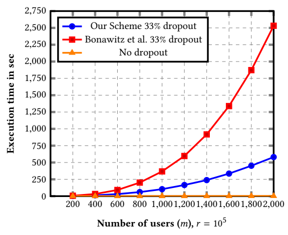
Entity	# Users	Dropout	KeyComp	Mask-then-encrypt (MtE)	Decrypt-then-unmask	Total time w/o storage	Total time w/ storage	Ref.
	Timings for Users-Server threat model. Our scheme: $t = \lceil \frac{m}{3} \rceil, \ell = 2\lceil \frac{m}{3} \rceil$. Bonawitz et al.'s: $t = 2\lceil \frac{m}{3} \rceil, \ell = m-1$							
User	2000	0%	834.96 ms	999.48 ms	40.56 ms	1,875.00 ms	1,040.04 ms	**Our scheme**
User	2000	0%	90.95 ms	1,909.98 ms	39.58 ms	2,040.51 ms	1,949.56 ms	Bonawitz et al. [7]
Mobile user	2000	0%	902,208 ms	90,931 ms	126.31 ms	993,265.31 ms	91,057.31 ms	**Our scheme**
Mobile user	2000	0%	29,448.8 ms	177,535 ms	126.605 ms	207,110.40 ms	177,661.60 ms	Bonawitz et al. [7]
Server	2000	0%	–	–	1,291.28 ms	1,291.28 ms	–	**Our Scheme**
Server	2000	10%	–	–	568,515.74 ms	568,515.74 ms	–	
Server	2000	20%	–	–	587,173.71 ms	587,173.71 ms	–	
Server	2000	30%	–	–	586,838.84 ms	586,838.84 ms	–	
Server	2000	0%	–	–	2,334.50 ms	2,334.50 ms	–	Bonawitz et al. [7]
Server	2000	10%	–	–	2292,084.38 ms	2292,084.38 ms	–	
Server	2000	20%	–	–	2418,686.95 ms	2418,686.95 ms	–	
Server	2000	30%	–	–	2509,194.84 ms	2509,194.84 ms	–	



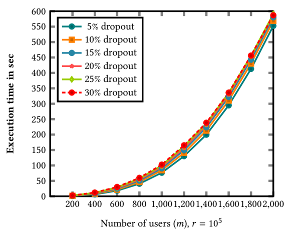
(a) Time for KeyComp and mask-then-encrypt per user on desktop



(b) Time for KeyComp and mask-then-encrypt per user on mobile



(c) Time for decrypt-then-unmask with 33% Dropout on desktop



(d) Decrypt-then-unmask time for the server for different dropout rates on desktop

Figure 10: Comparison for mask-then-encrypt, decrypt-then-unmask with dropouts times on desktop and mask-then-encrypt with varied input dimensions.

an aggregate-sum. Therefore, by knowing each aggregate-sum, the server can learn the linear regression model coefficients $\mathbf{c}$ by solving the system of linear equation. The scheme is secure as long as each aggregate-sum ensures privacy guarantees about individual users' inputs. In other words, the scheme is secure when aggregation as a privacy defense works. This is much efficient than solving a linear system in a privacy-preserving way.

SIMD-style Fast Computation of Linear Regression. One approach for the server to learn the linear regression model by running the aggregate-sum protocol *eight* times as there are total eight entries in $\mathbf{A}$ and $\mathbf{b}$. We propose a better solution where we need to run only one instance of our aggregate-sum protocol so that the server will obtain all eight sums as

$$\left(\sum_{i=1}^{m} x_{1,i}, \sum_{i=1}^{m} x_{2,i}, \sum_{i=1}^{m} x_{1,i}^2, \sum_{i=1}^{m} x_{2,i}x_{1,i}, \sum_{i=1}^{m} x_{2,i}^2, \sum_{i=1}^{m} y_i, \sum_{i=1}^{m} x_{1,i}y_i, \sum_{i=1}^{m} x_{2,i}y_i\right).$$

For this, each user will pack its input in form of a multidimensional vector as

$$\mathbf{y}_i = \left(x_{1,i}, x_{2,i}, x_{1,i}^2, x_{2,i}x_{1,i}, x_{2,i}^2, y_i, x_{1,i}y_i, x_{2,i}y_i\right),$$

where each component computations are on local private inputs. This is a single protocol run to process multiple data, which is like a Single Instruction and Multiple Data (SIMD) processing.

In general when $\mathbf{x}_i = (x_{1,i}, x_{2,i}, \cdots, x_{i,r})$ and the server wishes to learn the linear regression model on $\{(\mathbf{x}_i, y_i) : i \in [m]\}$ satisfying $y_i = L(\mathbf{x}_i) = c_0 + c_1x_{1,i} + c_2x_{2,i} + \cdots + c_rx_{2,r}$ based on the Least Squares Estimation, it obtains the aggregate-sum of each entry of the matrix $\mathbf{A}$ of dimension $(r+1)\times(r+1)$ and vector $\mathbf{b}$ of dimension $(r+1)\times 1$ and solves for $\mathbf{c} = \mathbf{A}^{-1}\mathbf{b}$ provided that $\mathbf{A}^{-1}$ exists. Each user locally constructs a vector $\mathbf{y}_i$ from its private input $\mathbf{x}_i$ as

$$\mathbf{y}_i = \Big(x_{1,i}, \cdots x_{r,i}, x_{1,i}^2, \cdots, x_{1,i}x_{r,i}, x_{2,i}^2, \cdots, x_{2,i}x_{r,i}, \cdots, x_{n,i}^2,$$
$$y_i, x_{1,i}y_i, \cdots x_{r,i}y_i\Big)$$

The dimension of the vector $\mathbf{y}_i$ is $\frac{(r+1)(r+2)}{2} + r$. After running the aggregate sum protocol with user inputs $\{\mathbf{y}_i\}_{i\in[m]}$, the server obtains $\sum_{i=1}^{m} \mathbf{y}_i$ and then construct the matrix $\mathbf{A}$ and vector $\mathbf{b}$ and then solves for $\mathbf{c}$ to learn the linear regression model coefficients.

Need for Multiple Aggregate-sum Computation. When the inverse of $\mathbf{A}$ does not exists, the linear regression model coefficient $\mathbf{c}$ can still be computed using the aggregate-sum protocol. The gradient descent approach is used to determine $\mathbf{c}$, but this needs multiple aggregate-sum computations. The error term to estimate $\mathbf{c}$ is $J_m = \frac{1}{m}\sum_{i=1}^{m}(y_i - L(\mathbf{x}_i))^2$. Using the gradient descent approach, the coefficient vector can estimated by iterative computation as

$$\begin{aligned}\mathbf{c} \leftarrow& \mathbf{c} - \zeta\nabla_{\mathbf{c}}J_m(\mathbf{c})\\ \leftarrow& \mathbf{c} + 2\zeta Z^T(y - Z\mathbf{c}).\end{aligned}$$

where ζ is the learning rate. Let at j-th iteration the function $L(x)$ be $L_j(x)$ and the coefficient vector be $\mathbf{c}^j$. Then, $Z^T(y - Z\mathbf{c}^j)$ can be written as

$$\Big(\sum_{i=1}^{m}(y_i - L_j(\mathbf{x}_i))), \sum_{i=1}^{m}(y_i - L_j(\mathbf{x}_i))x_{1,i}, \cdots, \sum_{i=1}^{m}(y_i - L_j(\mathbf{x}_i))x_{r,i}\Big)$$

Thus, **c** can be determined iteratively as

$$\mathbf{c}^{j+1} = \mathbf{c}^{j} + 2\zeta Z^T(y - Z\mathbf{c}^{j}), j = 0, \cdots, \omega$$

in ω iterations. It is easy to see that **c** can be determined using ω executions of the aggregate-sum protocol on $(r + 1)$ dimensional vectors as inputs.