Achieving Differential Privacy in Secure Multiparty Data Aggregation Protocols on Star Networks

Vincent Bindschaedler*
University of Illinois at Urbana-Champaign
bindsch2@illinois.edu

Shantanu Rane
Palo Alto Research Center
srane@parc.com

Alejandro Brito
Palo Alto Research Center
abrito@parc.com

Vanishree Rao
Palo Alto Research Center
vrao@parc.comt

Ersin Uzun
Palo Alto Research Center
euzun@parc.com

ABSTRACT

We consider the problem of privacy-preserving data aggregation in a star network topology, i.e., several untrusting participants connected to a single aggregator. We require that the participants do not discover each other's data, and the service provider remains oblivious to each participant's individual contribution. Furthermore, the final result is to be published in a differentially private manner, i.e., the result should not reveal the contribution of any single participant to a (possibly external) adversary who knows the contributions of all other participants. In other words, we require a secure multiparty computation protocol that also incorporates a differentially private mechanism.

Previous solutions have resorted to caveats such as postulating a trusted dealer to distribute keys to the participants, or introducing additional entities to withhold the decryption key from the aggregator, or relaxing the star topology by allowing pairwise communication amongst the participants. In this paper, we show how to obtain a noisy (differentially private) aggregation result using Shamir secret sharing and additively homomorphic encryption without these mitigating assumptions. More importantly, while we assume semi-honest participants, we allow the aggregator to be stronger than semi-honest, specifically in the sense that he can try to reduce the noise in the differentially private result.

To respect the differential privacy requirement, collusions of mutually untrusting entities need to be analyzed differently from traditional secure multiparty computation: It is not sufficient that such collusions do not reveal the data of honest participants; we must also ensure that the colluding entities cannot undermine differential privacy by reducing the amount of noise in the final result. Our protocols avoid this by requiring that no entity – neither the aggregator nor any participant – knows how much noise a participant contributes to the final result. We also ensure that if a cheating aggregator tries to influence the noise term in the differentially private output, he can be detected with overwhelming probability.

CODASPY'17, March 22-24, 2017, Scottsdale, AZ, USA

DOI: http://dx.doi.org/10.1145/3029806.3029829

Keywords

secret sharing, homomorphic encryption, differential privacy

1. INTRODUCTION

Aggregate computations are the among the most basic and widely used primitives in today's networked environments. These computation usually involve a server (aggregator) computing aggregate measures, e.g., histograms, weighted summations, averages, etc., using data gathered from several devices (participants). In this work we are concerned with application scenarios in which participants and the aggregator form a star network, and the privacy of the participants must be protected. The need for privacy-preserving aggregate computations arises in several applications: telemetry from Internet-of-Things (IoT) devices, analytics on medical data furnished by wearables, smart grid power aggregation, histograms of websites visited by users of a particular browser, to name a few.

As participants in the internet economy, we generally allow our data to be used by service providers (aggregators), as a fair exchange for the services they provide. However, there is a growing concern that our data may be used for purposes that we did not sanction. For example, smart meter readings can reveal a homeowner's life patterns, fitness apps may reveal private medical conditions, and browser activity and metadata reveal intimate details of a person's life and values. Furthermore, if the service provider is affected by a data breach, sensitive data belonging to unsuspecting individuals falls into the hands of an adversary.

Two kinds of privacy formulations may be used to express and resolve the above concerns. The first formulation, based on secure multiparty computation, is used to directly express which entities can communicate and how entities withhold knowledge of their data from other entities. In a star network, the main privacy constraint is that the data held by any individual participant should not be revealed to other participants, and also not to the aggregator. The aggregator should discover only the specified aggregate measure such as the summation or the probability distribution and nothing else. Additional constraints may be dictated by practical deployment. One such constraint, in a star network, is that the participants can communicate with the aggregator but may not be able to communicate with one another. A second constraint is that the participants cannot all be expected to remain online simultaneously while the protocol is being executed. Thus, an aggregation protocol must enable the aggregator to compute the correct result, while

*V. B. contributed to this work when he was an intern at PARC.

satisfying the privacy constraints and communication constraints, even when users dynamically join and leave.

The second formulation, based on differential privacy, is used to express how difficult it is for an adversary to observe the result of a computation, and make inferences about the protocol's participants. Suppose that the aggregator publishes the final result of the data aggregation protocol, e.g., the average power consumption, and makes it available to an analyst. Then, differential privacy quantifies the degree of confidence with which the analyst can claim that a particular participant, e.g., Alice, participated in the protocol. Seen another way, if the analyst repeated the protocol after Alice moved out of the neighborhood, he would discover Alice's power consumption, even if the protocol didn't directly reveal Alice's data to the analyst. A differentially private mechanism would modify the result, i.e., the average power consumption in a way that makes it hard for the analyst to discover Alice's power consumption. This is clearly a different notion of privacy compared to that encountered in secure multiparty computation. In this paper, we are concerned with situations where both kinds of privacy are desired.

2. OVERVIEW OF RELATED WORK

Addressing the privacy concerns related to multiparty computation is challenging predominantly because of the key management problem. Concretely, each participant should obfuscate its input so that all the obfuscated inputs can later be combined – for example, by means of a homomorphic cryptosystem – by the aggregator to reveal the aggregate function. However, this is not straightforward for the star network topology because each participant encrypts using its own unique key. Below, we present a short overview of attempts to solve this problem. For more details, we refer the reader to a comprehensive review by Erkin *et al*. [8]

Shi *et al.* considered an aggregation protocol that assumes a trusted dealer that distributes encryption keys to the participants, and ensures that the keys vanish when the aggregator combines the result in a prescribed way [4, 22]. A similar approach is followed by Bilogrevic *et al.*, to compute means, variances and higher moments of distributions [3]. Rather than assuming a trusted dealer, Jawurek and Kerschbaum distribute the task of computation and the decryption between an untrusted aggregator and an untrusted key managing authority [11]. This is an efficient approach with a single public key homomorphic cryptosystem, that has only $O(m)$ overhead, where m is the number of participants. However, it introduces an extra participant, and introduces the risk of catastrophic privacy loss if the key managing authority colludes with the aggregator. Leontiadis *et al.* proposed a different solution that allows each participant to use a different encryption key, while distributing the knowledge of the overall decryption key between the aggregator and an extra entity called the collector [16]. Similar to the previous approach, their scheme forbids collusions between the aggregator and the collector.

Though additive secret sharing requires pairwise information exchange amongst the participants, this approach can still be considered in star networks, by allowing the participants to communicate via the aggregator. Specifically, Kursawe *et al.* employed public-key encryption to send encrypted shares of the participants' data (or keys) to a subset of participants (termed "leaders") via the aggregator [15]. The leaders add their own shares such that their effect vanishes upon combination, revealing only the sum of the participants' data. Garcia and Jacobs presented a protocol in which a participant homomorphically encrypts each share by using the public-key of each share's intended recipient, but only sends it to the aggregator [10]. The aggregator then homomorphically combines the encrypted shares, requests the decryption of partial summations from each participant, and combines the partial sum to reveal the final sum, but nothing else. This approach has been generalized via the use of Shamir secret sharing [21], which provides fault tolerance in addition to collusion resistance [20]. In a strict star topology, all these approaches incur $O(m^2)$ ciphertext communication overhead at the aggregator.

We point to further efforts on this topic, which relax the strict star network topology in exchange for a gain in efficiency. Erkin and Tsudik allow the participants (smart meters) to communicate with one another, exchanging random values before each smart meter communicates with the aggregator [9]. This is a mode of secret sharing, in which the randomized values are chosen to vanish when the aggregator computes the sum. Ács and Castelluccia allow each participant to communicate with a small number of other participants [1]. These works show that by allowing a limited amount of communication amongst the participants, the ciphertext overhead of the protocol immediately becomes manageable, i.e., $O(m)$. On the other hand, if any participant leaves before the shares are combined, the final sum computed by the aggregator becomes error-prone. Though we do not have a proof, the above two papers suggest that a $O(m)$ ciphertext overhead may not be achievable, in general, for fault-tolerant aggregation with a star topology. Indeed, we are not aware of any protocol that achieves this. An improvement toward $O(m)$ overhead, has nevertheless, been attempted. Dividing the star-connected participants into a logical hierarchy composed of several "cohorts" of participants, it is possible to achieve a protocol complexity of $O(m^{1+\rho})$ [20]. By an appropriate choice of the cohort size, ρ can be made small.

Some of the schemes discussed above, have also proposed mechanisms to achieve differential privacy [1, 3, 4, 11, 22]. Recognizing that the aggregator has an incentive to obtain the most accurate aggregate value as possible, these studies have entrusted the task of noise addition to entities other than the aggregator. This is a departure from the differential privacy literature produced by the machine learning community, in which the database curator is trusted to add the correct amount of noise. For example, in Shi *et al.* [22], Chan *et al.* [4] and Bilogrevic *et al.* [3], the participants generate samples from the two-sided geometric distribution (a discrete version of the Laplace distribution), which add up to provide a differentially private summation of the participants' inputs. Similarly, in Acs and Castelluccia [1] the participants generate Gamma-distributed noise samples, which add up to a Laplace-distributed noise term at the aggregator.

Unfortunately, as we show later in the paper, this approach of having the participants add noise samples, is not sufficient to preserve differential privacy. In particular, we argue that collusions of semi-honest participants can subtract out their noise terms from the published result and reduce the differential privacy of honest participants. We present a protocol in which the participants and the aggregator must interact to determine the noise term. More importantly, our protocol is designed such that neither the participants nor the aggregator know the noise terms that have contributed to the differentially private output. Since the aggregator has an incentive to cause less noise to be added than necessary, we describe a protocol that can detect a cheating aggregator.

3. ADVERSARIAL MODEL

We consider a strict star topology, i.e., each of the participants can communicate only with the aggregator, and never with any of the other participants. Below, we explain several aspects of our adversarial model. The explanation below is intended to convey the broad requirements, thus notation is kept at a minimum. Later on, following the description of our protocols, we will explain in

Work	Approach	Network Topology	Correctness under Node Failures	Differential Privacy (DP)	Who knows the noise terms used for Differential Privacy?
Shi *et al*. [22], Bilogrevic *et al*. [3]	Differentially private aggregation with geometric distribution	Star network + trusted key dealer.	No	Yes	Each participant knows its own contribution.
Chan *et al*. [4],	Differentially private aggregation with fault tolerance	Star network + trusted key dealer.	Yes	Yes	Each participant knows its own contribution.
Joye and Libert [12]	Private aggregation with a large plaintext space using discrete logarithms	Star network + trusted key dealer.	No	No	N/A
Jawurek and Kerschbaum [11]	Practical scheme with a single additively homomorphic key-pair	Star network + key managing authority	Yes	Yes	Key managing authority knows the noise term in the final sum.
Leontiadis *et al*. [16]	Private Aggregation with Dynamic Group Management	Star network + untrusted "collector"	Yes	No	N/A
Erkin and Tsudik [9]	Efficient homomorphic aggregation with inter-participant communication	Fully connected network to share random values	No	No	N/A
Ács and Castelluccia [1]	Differentially private aggregation with additive secret sharing	Fully connected network to share secrets	Yes	Yes	Each participant knows its own contribution.
Kursawe *et al*. [15]	Private aggregation using additive secret sharing	Star network + L "leaders".	No	No	N/A
Garcia and Jacobs [10]	Additive secret sharing with homomorphic encryption	Star network	No	No	N/A
Rane *et al.* [20]	Shamir secret sharing and homomorphic encryption	Star network	Yes	No	N/A
This paper	Shamir secret sharing and homomorphic encryption, zero-knowledge proof to detect cheating aggregator	Star network	Yes	Yes	Neither participants nor aggregator know individual or final noise terms.

Table 1: Methods and Adversarial Models in the Privacy-preserving Aggregation Literature.

detail how these requirements are satisfied.

3.1 Participant & Aggregator Obliviousness

Consider that the participants' data elements d_i are collected in a vector denoted by $\mathbf{d}$. We require *Participant Obliviousness*, i.e., the input data, d_i, of a participant should not be revealed to the other participants. Further, we require *Aggregator Obliviousness*, which means that the aggregator discovers only the aggregate function being computed and nothing else about the inputs of individual participants. In our protocols, we first consider the simplest aggregate function, namely the sum of the participants' data. Later, we describe how to extend the summation protocols to the computation of other aggregate measures such as counts and histograms.

A typical assumption in studies of this kind is that the aggregator and all the participants are *semi-honest* (*honest but curious*). For our purposes, a semi-honest entity is one which follows the rules of the protocol, but based on the information it sees during each step of the protocol, it can attempt to discover the data held by other entities. In particular, the participants and aggregator do not provide false or spurious inputs to the protocol. During privacy analysis, it then becomes necessary to consider the view of semi-honest participants, and collusions of semi-honest participants.

We remark that, many published works assume semi-honest entities but do not allow certain collusions. For example, in the scheme of Jawurek *et al.* [11] , the aggregator may not collude with a key authority. In the scheme of Leontiadis *et al.* [16], the aggregator may not collude with the collector. While these caveats are made in the interest of practical realizability, they present potentially serious concerns, because the forbidden collusion results in catastrophic privacy loss for all honest participants. Our goal, therefore, is to design aggregation protocols that protect against privacy loss under collusions of all kinds. We go a step further than this by assuming that while the participants are semi-honest, the aggregator is "strictly stronger than semi-honest" in a particular sense, that we will clarify in Section 3.3.

3.2 Differential Privacy (DP)

Recent work on Differential Privacy [6] has shown that aggregator obliviousness is not enough when the aggregator (or an adversary that corrupts the aggregator) has side information about the participants. For example, if this side information consists of the sum of the inputs of $m-1$ out of m participants, then running a privacy-preserving aggregation protocol with m participants trivially reveals the remaining participant's input. Therefore, to protect attacks against adversaries armed with background information, we require a differentially private aggregation mechanism.

Let $\mathbf{d}'$ denote a vector that differs from $\mathbf{d}$ in only a single entry. This can be achieved by adding an element to $\mathbf{d}$, or removing a single element from $\mathbf{d}$, or by modifying the value of a single element in $\mathbf{d}$. In this case, $\mathbf{d}$ and $\mathbf{d}'$ are referred to as *adjacent* data sets. Then, $(\epsilon, \delta)-$Differential Privacy is defined as follows:

DEFINITION 1. (ϵ, δ)-***Differential Privacy*** *[7]: An algorithm or protocol or mechanism $\mathcal{M}$ is said to satisfy (ϵ, δ)-Differential Privacy if, for all adjacent data sets $\mathbf{d}$ and $\mathbf{d}'$, and for all sets $\mathcal{S} \subseteq$ Range$(\mathcal{M})$,*

$$P(\mathcal{M}(\mathbf{d}) \in \mathcal{S}) \leq e^{\epsilon} \cdot P(\mathcal{M}(\mathbf{d}') \in \mathcal{S}) + \delta$$

As a consequence of this definition, a differentially private protocol produces *nearly indistinguishable* outputs for adjacent data sets, thus preserving the privacy of the element that is different in $\mathbf{d}$ and $\mathbf{d}'$. The amount of indistinguishability is controlled by the parameters ϵ and δ with lower values implying greater privacy. Differential privacy can be achieved in many ways: One approach is *output perturbation*, which involves first computing a function of the data set and then adding noise to the function result. Another approach is *input perturbation*, which involves first adding noise to the data before evaluating the function result. In either case, a differentially private mechanism will ensure that the function value revealed to the aggregator at the end of the protocol, will contain some additive noise which will protect the participants against attackers equipped with background information. The amount of noise that needs to be added the final aggregate function f depends on the parameters ϵ, δ, and the "global sensitivity" of the function f, denoted by Δ and defined as

$$\Delta = \max_{\mathbf{d},\mathbf{d}'} \|f(\mathbf{d}) - f(\mathbf{d}')\|_1$$

for all adjacent data sets $\mathbf{d}$ and $\mathbf{d}'$.

For example, $(\epsilon, 0)$-Differential Privacy can be achieved by evaluating the function f and adding noise sampled from a Laplace distribution with scale parameter Δ/ϵ. Also, (ϵ, δ)-Differential Privacy can be achieved by evaluating the function f and adding noise sampled from a Gaussian distribution with zero mean and variance $\frac{\Delta}{\epsilon}\ln(1/\delta)$. Other mechanisms are possible in which the noise term is sampled from a different distribution, and is based on the "local sensitivity" rather than the "global sensitivity" [18]. In our scenario of computing aggregate functions on star networks, adding noise to a function value evaluated at the aggregator is not straightforward owing to the competing incentives of the participants and the aggregator which we now elaborate.

3.3 Secure Multiparty Differential Privacy

As we are interested in providing differential privacy to the participants, there are interesting adversarial scenarios that are not captured in pure secure multiparty computation or pure differentially private mechanisms. We now discuss these adversarial scenarios. Before that, we describe precisely the sense in which the aggregator is a more powerful adversary than a traditional semi-honest entity.

Aggregator can influence the noise term: The aggregator wants the revealed aggregate result to be as accurate as possible. Thus, he has an incentive to add a small amount of noise (even no noise at all), which would result in insufficient differential privacy for the participants. In most of the prior literature on differentially private machine learning, the database curator is the entity that adds noise to the data. We contend that when the curator (or analyst, or aggregator) himself has the incentive to obtain accurate measurements, he cannot be trusted to add the correct amount of noise to ensure differential privacy for the participants. In our model, the aggregator can try to influence (e.g., reduce) the level of noise that is added to the aggregate before it is published. By doing this, he can try to force a higher value of ϵ, thus resulting in lower differential privacy. In this respect, he is stronger than the semi-honest entity encountered in traditional secure multiparty computation[1].

Collusions of semi-honest participants: We assume that each participant is semi-honest. It may seem that the way to prevent the aggregator from influencing the noise term is to ask the participants, not the aggregator, to generate noise values for differential privacy. Since the participants are semi-honest, they do not present spurious data or wrongly distributed noise samples to the protocol. However, their curiosity has consequences beyond those normally observed in secure multiparty computation, where the goal of semi-honest entities is to discover the data of honest entities. In our adversarial model, the goal of semi-honest entities is, additionally, to make *statistical* inferences about the honest entities that are more accurate than what the specified differential privacy parameter allows.

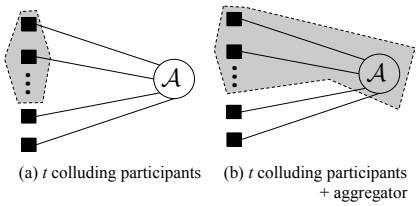

Figure 1: Our adversarial model permits a collusion between a subset of the semi-honest participants (a), or between a subset of the participants and a corrupted aggregator (b).

Consider, for example, the case in Fig. 1(a), in which the aggregator is following the protocol and t participants are honest, but all the remaining $m - t$ participants form a collusion. The colluding participants will now discover the noisy summation output by the aggregator, from which they can subtract their own inputs and their own noise contributions. This leaves the noisy contribution of the t honest participants, which may not guarantee sufficient differential privacy, i.e., a low enough value of ϵ. Thus, it is not sufficient to trust the semi-honest participants to generate noise values. Of course, the adversarial model also allows for the more serious collusion between a corrupted aggregator and a subset of the participants, as shown in Fig. 1(b).

In earlier work, e.g. [3, 22], the above situation is avoided by requiring that some fraction of the participants must remain honest, and that these honest participants must add enough noise to ensure differential privacy. Other earlier work, e.g. [20], does not consider differential privacy, hence a collusion of $m-1$ out of m participants trivially reveals the data of the remaining honest participant even if the aggregator remains honest. We claim that, with a differentially private aggregation protocol, it is possible to protect the lone honest participant even when all other participants collude (provided the aggregator stays honest). Intuitively, the way to achieve this is to have the participants generate noise samples, and make the aggregator blindly pick only a subset of those samples, unbeknownst to the participants. Our protocols will make this notion clear. In particular, we will ensure that an aggregator who tries to choose fewer than the prescribed number of noise samples is caught with overwhelming probability.

Our view is that the adversarial model described here is more realistic than a semi-honest model, and can protect participants in many real-world situations. For example, companies might want to *disaggregate* power usage data gathered from a neighborhood. Noising the power usage can provide some privacy against disaggregation. A ride-sharing service might want to extract ratings received by individual drivers, in order to penalize low scorers. Noising the scores provides some protection for drivers, encouraging them to participate.

1. We clarify that the aggregator is *not* as powerful as an "active" or "malicious" adversary in traditional secure multiparty computation. Such an adversary can arbitrarily deviate from the protocol. The corresponding defense mechanisms are significantly more complicated, and possibly impractical to implement.

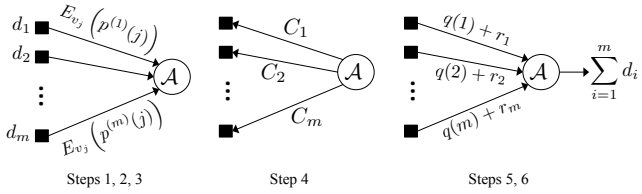

Figure 2: The protocol of Section 4 as described in [20]. Participants send homomorphically encrypted shares to the aggregator $\mathcal{A}$ to indirectly implement Shamir secret sharing in a star network.

4. SECRET SHARING FOR PRIVATE AGGREGATION

We first describe the basic protocol that is used to perform privacy-preserving aggregation in a star topology, without consideration of differential privacy. For this, we leverage existing work in the literature [20], and in subsequent sections, show how to achieve the desired level of differential privacy, even with an adversarial aggregator. As the steps for achieving differential privacy crucially depend on this basic protocol, we will describe it in some detail. In this section, assume that all entities, including the aggregator are semi-honest. Later on, when differentially private mechanisms are considered, we will make the aggregator more powerful, in the sense that he can influence the amount the noise added to the final result output by the protocol. At this point, we are still in the realm of traditional secure multiparty computation and there is no noise being added by any entity. Thus, nothing is lost by assuming a semi-honest aggregator, just for this section.

The protocol with m participants is based on Shamir Secret Sharing [21] and additively homomorphic encryption. The high-level idea is that each of the m participants generates a polynomial with secret coefficients whose constant coefficient is their input data. Each participant evaluates its polynomial at m distinct known points, encrypts the resulting values using the public keys of relevant participants, and sends them to the aggregator. The aggregator homomorphically combines the encrypted shares received from all participants, to obtain encrypted evaluations of a "sum" polynomial at those m points. Upon decrypting these evaluations, the aggregator performs polynomial interpolation to obtain the coefficients of the sum polynomial, evaluates the sum polynomial at $x = 0$ to discover the desired sum of inputs of all participants. A more precise description of the protocol follows below.

Inputs: Denote the aggregator by $\mathcal{A}$, and each participant by $\mathcal{P}_i$, $i = 1, 2, \ldots, m$. Let d_i be the input data held by each participant, such that d_i is a non-negative integer and $d_i < d_{\max}$. Associated with each $\mathcal{P}_i$ is the public-private key pair of a semantically secure additively homomorphic cryptosystem [5, 19]. Denoting the public key for $\mathcal{P}_i$ by v_i, the additive homomorphic property ensures that $E_{v_i}(a)E_{v_i}(b) = E_{v_i}(a + b)$. The semantic security property implies that a given plaintext maps to a different ciphertext at every encryption, thus providing protection against Chosen Plaintext Attacks (CPA).

Output: The aggregator discovers $\sum_{i=1}^{m} d_i$. The participants discover nothing else.

Protocol: Consider the following steps, also shown in Fig. 2:

1. The aggregator broadcasts a large prime number $\beta > md_{\max}$ to all participants.

2. Each participant, $\mathcal{P}_i, i = 1, 2, ..., m$ generates a polynomial of degree $k < m$ given by:

$$p^{(i)}(x) = d_i + p_1^{(i)}x + p_2^{(i)}x^2 + \ldots + p_k^{(i)}x^k \mod \beta$$

where the coefficients $p_s^{(i)}$ where $s = 1, 2, ..., k$, are chosen uniformly at random from the interval $[0, \beta)$. By construction, note also that $p^{(i)}(0) = d_i < \beta$, i.e., evaluating the polynomial at zero yields each participant's input data.

3. Each participant $\mathcal{P}_i$ evaluates the polynomial at m known, distinct points. Without loss of generality, let these points be the integers $j = 1, 2, \ldots, m$. Then, each $\mathcal{P}_i$ encrypts $p^{(i)}(j)$ using the public key v_j of the participants $\mathcal{P}_j, j = 1, 2, ..., m$, and sends the ciphertexts $E_{v_j}(p^{(i)}(j))$ to the aggregator, $\mathcal{A}$.

4. For each $i = 1, 2, \ldots, m$, the aggregator computes

$$\begin{aligned} E_{v_j}(r_j)\prod_{i=1}^{m} E_{v_j}(p^{(i)}(j)) &= E_{v_j}\left(r_j + \sum_{i=1}^{m} p^{(i)}(j)\right) \\ &= E_{v_j}\left(r_j + q(j)\right) = C_j \end{aligned}$$

The aggregator then sends each C_j, $j = 1, 2, \ldots, m$ to participant $\mathcal{P}_j$ for decryption. Here, the constant r_j is chosen at random to hide the summation term from $\mathcal{P}_j$.

5. The participants $\mathcal{P}_j$ who are still online, decrypt the respective C_j and returns it to the aggregator. The aggregator subtracts r_j and obtains, for $j \subset \{1, 2, \ldots, m\}$, the values

$$q(j) = \sum_{i=1}^{m} p^{(i)}(j) \mod \beta$$

6. By construction, the above steps have enabled the aggregator to evaluate the polynomial,

$$q(x) = q_1x + q_2x^2 + \ldots + q_kx^k + \sum_{i=1}^{m} d_i \mod \beta$$

at some points in the set $\{1, 2, \ldots, m\}$. In order to recover the coefficients $q_1, q_2, \ldots, q_k$ and the desired summation, the aggregator needs the polynomial $q(x)$ to be evaluated at $k + 1$ or more points, i.e., the aggregator needs at least $k+1$ participants to be online. If this requirement is satisfied, the aggregator can perform polynomial interpolation to obtain $q_1, q_2, \ldots, q_k$, and recover the value of $q_0 = \sum_{i=1}^{m} d_i$, which is the quantity of interest.

Correctness: The use of additively homomorphic encryption with the appropriate participant's public keys distributes shares of the desired summation to the participants who are still online. Functionally, this is equivalent to distributing polynomial secret shares, and performing additions in the BGW protocol [2]. Alternatively, correctness follows from the realization that Shamir secret sharing is additively homomorphic modulo β.

Fault-Tolerance: The degree of the "sum" polynomial is $k < m$. Hence, the protocol is fault tolerant: The aggregator can compute the summation even when up to $m - k - 1$ participants go offline after Step 3, i.e., before polynomial interpolation is used to extract the final sum from the shares.

Privacy: First, consider privacy against individual semi-honest entities. Secret sharing ensures that no participant discovers the data held by an honest participant. Furthermore, the homomorphic cryptosystem ensures that the aggregator only discovers the coefficients

of the "sum" polynomial $q(x)$, but does not discover the coefficients of the component polynomials $p^{(i)}(x)$. The privacy guarantee against individual semi-honest participants is information-theoretic, while that against the aggregator is computational.

Next, consider privacy against collusions of semi-honest entities. An adversary that corrupts $m-1$ out of m participants, can examine the published summation and discover the data of the remaining honest participant (We *can* protect the lone honest participant using a differentially private mechanism that we will describe in Section 5). Next, consider semi-honest coalitions that also contain the aggregator. In order to discover the data d_i of an honest participant $\mathcal{P}_i$, the coalition needs to access at least $k+1$ decrypted polynomial secret shares $p^{(i)}(j)$ for $j \in \{1, 2, ..., m\}$ and perform polynomial interpolation. To achieve this, the coalition must comprise the aggregator and at least $k+1$ other semi-honest participants. In other words, the protocol preserves privacy of an honest participant against coalitions consisting of the aggregator and up to k other participants.

Complexity: The ciphertext communication complexity of the protocol is $O(m^2)$ as determined by Step 3. Similarly, the ciphertext computation complexity is also $O(m^2)$ as determined by Step 4. Note that, the aggregator has to perform polynomial interpolation in Step 5. This can be accomplished using Lagrange interpolation, which has $O(m^2)$ complexity [13].

5. DIFFERENTIALLY PRIVATE AGGREGATION PROTOCOLS

Our approach is to make the participants and the aggregator generate and add noise to the aggregate function through interaction. This becomes challenging under the adversarial model described earlier. The aggregator has an incentive to add as little noise as possible. Furthermore, even if the aggregator is honest, it is very difficult to ensure differential privacy of a participant when all other participants are colluding. Our approach is to design protocols in which neither the aggregator nor any participant finds out the noise value that has been added to the final summation. In fact, we have a slightly stronger requirement: No single participant can discover how much noise she herself contributed to the final summation.

Broadly, the protocol has two phases: (1) Cheating-proof noise generation, and (2) Secure aggregation. In the first phase, the noise term is generated via interactions between the entities, in such a way that cheating attempts can be detected. The second phase executes the secure aggregation protocol of Section 4 such that the aggregated result will incorporate the noise term to guarantee differential privacy. We assume the following setup for all of the protocols (and sub-protocols) of this section.

Setup & Public Parameters: As before, denote the aggregator by $\mathcal{A}$, and each participant by $\mathcal{P}_i$, $i = 1, 2, \ldots, m$. The aggregator has a public-private key pair of a semantically secure additively homomorphic cryptosystem. The public key of the aggregator is denoted by v_A. The plaintext domain of the aggregator is $\mathbb{D}_A$. Let F, F_S denote noise distributions over $\mathbb{R}$. (For example, F is the Laplacian distribution with parameter $\frac{\Delta}{\epsilon}$, denoted by $\text{Lap}(\Delta/\epsilon)$.)

Inputs: Each $\mathcal{P}_i$ has input data d_i. The aggregator has no inputs.

Output: The aggregator discovers $\sum_{i=1}^{m} d_i + \xi$, where $\xi \sim F$ is the noise term. The participants do not discover anything.

Protocol: Consider the following steps:

1. The participants and the aggregator jointly execute the cheating-proof noise generation protocol, to be described in Section 5.2. Participant i obtains blinding term r_i. The aggregator obtains (from $\mathcal{P}_i$) a value $r_i + \xi_i$ for every participant, and computes the sum, $\sum_{i=1}^{m}(r_i + \xi_i)$.

2. The participants and the aggregator jointly execute the private aggregation protocol of Section 4, with every $\mathcal{P}_i$'s input set to $d_i - r_i$. The aggregator obtains the value $\sum_{i=1}^{m}(d_i - r_i)$ via interactions with the participants who are still online during Step 5 of the protocol in Section 4. It then calculates:

$$\sum_{i=1}^{m}(d_i - r_i) + \sum_{i=1}^{m}(r_i + \xi_i) = \sum_{i=1}^{m} d_i + \xi$$

which is the desired noised sum with $\xi = \sum_{i=1}^{m} \xi_i$.

In the subsections that follow, we describe the protocol in more detail. We first explain how the noise term needed for differential privacy can be computed by aggregating noise samples generated by the participants. Then, we describe the protocol to achieve differential privacy and analyze it with respect to correctness, fault-tolerance, and privacy under the threat model described in Section 3. [2]

5.1 Sampling Noise for Differential Privacy

There are several ways to sample noise in order to satisfy differential privacy. The popular Laplacian mechanism adds noise from the Laplace distribution, but other distributions (e.g., Gaussian) can be used as well [7]. In our setting, it is not sufficient to sample a single noise term from an appropriate distribution. Indeed, neither the participants nor the aggregator should find out the sampled noise value, and so no entity should perform the sampling on its own. Instead, we propose to sample the noise as the sum of several noise terms generated by the participants. Formally, we model the noise term ξ as a random variable $X \sim F$. The idea is to generate X as the sum of some i.i.d. random variables X_i.

Consider first the case where F is the Gaussian distribution. Write $S_n = \frac{1}{n}\sum_{i=1}^{n} X_i$, where X_i are i.i.d. random variables from some distribution F_S with finite variance σ^2. We know by the Central Limit Theorem (CLT) that $\sqrt{n}(S_n - E[S_n])$ converges in distribution to $\mathcal{N}(0, \sigma^2)$. This observation allows us to generate noise that approximately follows a Gaussian distribution with a given variance. Note that (ϵ, δ)-differential privacy can be achieved using Gaussian noise (see Section 3.2).

What if, instead, we want to achieve ϵ-differential privacy (i.e., $\delta = 0$)? This requires F to be a Laplace distribution, specifically $F = \text{Lap}(\Delta/\epsilon)$. This can be accomplished using a similar idea for generating X by exploiting the infinite divisibility of the Laplace distribution [14]. Concretely, if X is a Laplace random variable with mean zero and scale parameter b, then for any $n \geq 1$, there exist i.i.d. random variables X_i such that $X = \sum_{i=1}^{n} X_i$. For example, this holds if $X_i = Y_{1,i} - Y_{2,i}$, where the random variables $Y_{1,i}, Y_{2,i}$ are distributed as $\text{Gamma}(1/n, b)$.

More generally, we consider generating noise from any distribution F which can satisfy differential privacy, as long as we can sample $X \sim F$ by sampling i.i.d. X_i's for some distribution F_S such that $X = a_n \sum_{i=1}^{n} X_i$, for some constant a_n. It may also be

2. In addition to the adversarial actions described in Section 3, other deviations are possible. For example, a participant can be online but could refuse to provide the decryption key in Step 5 in the protocol of Section 4. In our construction, such deviations fall into the category of failures, from which recovery is possible owing to the fault tolerance provided by Shamir Secret Sharing. We also remark that the aggregator has no incentive to deviate in a way that would result in a failure to complete the protocol because that implies a failure to obtain the aggregate sum.

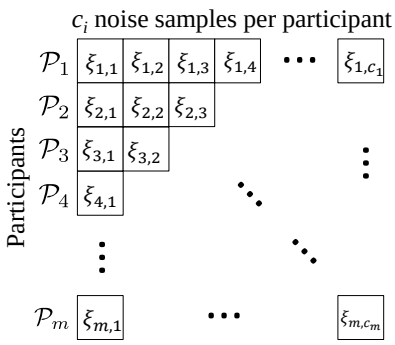

Figure 3: Every participant i generates c_i noise terms of the form $\xi_{i,j}$. The noise generation protocol ensures that, for the purpose of providing differential privacy, n out of these terms are aggregated into the final noise term ξ, without the participants discovering which n terms were included.

possible to exploit the geometric stability of the Laplace distribution, although we do not consider this explicitly here.

To deal with collusions, we will further set the parameters such that if only $n' < n$ terms are added to the sum, the noise is sufficient to ensure differential privacy. That is, if F_S is the distribution of the X_i's, we will choose the parameters such that $\sum_{i=1}^{n'} X_i = X \sim F$. This satisfies differential privacy since the final noise term will be $X + \hat{X}$, where $\hat{X}$ is a sum of up to $n - n'$ random variables distributed according to F_S independent of X. Clearly, if adding X is sufficient to satisfy differential privacy, then so is adding $X + \hat{X}$.

As discussed in the related work section, we are not the first to propose the idea of having participants generate noise samples that are eventually accumulated into a noise term from a desired distribution. The novel aspect of our treatment is, rather, in the adversarial incentives of the participants and the aggregator, and in the measures taken to detect such adversarial actions. These adversarial incentives, and detection measures are not considered in the prior art.

5.2 Noise Generation Phase

We present a double-blind noise addition protocol in which the participants generate noise components and the aggregator obliviously computes an overall noise term. This is achieved without the aggregator learning the noise term or being able to influence it (without being detected). To ensure that the aggregator does not cheat and influence the noise sampled, we use a lightweight verifiable computation subprotocol. In general, there are two challenges in such proofs of computation: (a) the proof must not reveal information (e.g., amount of noise), and (b) the protocol must be efficient. While the first concern could be addressed by using generic zero-knowledge proof techniques, the second concern (i.e., efficiency) remains. Therefore, we design a custom solution that exploits the structure and statistical properties of the computation. We first explain the protocol used to derive the overall noise term. The sub-protocol used to detect a cheating aggregator is explained immediately afterward in this section.

Setup: Let t, l be positive integers (security parameters). Let n be a positive integer ($n \geq m$), and F_S be a distribution such that $(\sum_{i=1}^{n} X_i) \sim F$, where $X_i \sim F_S$.

Inputs: The participants and the aggregator have no inputs.

Output: The aggregator obtains $\xi + r$, where ξ is a noise term distributed according to F. The participants have no output.

Protocol: Consider the following steps:

1. Each participant $\mathcal{P}_i$ chooses values $r_{i,j} \in \mathbb{D}_A$, and samples values $\xi_{i,j}$ from F_S, for $j = 1, 2, \ldots, t \cdot \lceil \frac{n}{m} \rceil$. This is depicted in Fig. 3.

2. For $i = 1, 2, \ldots, m$, the aggregator sets $c_i = t \cdot s_i$, where $s_i = \lfloor \frac{n}{m} \rfloor + \mathbb{1}_{\{i \leq n \pmod m\}}$. The aggregator then generates the binary sequence $b_{i,j} \in \{0, 1\}$, for $i = 1, 2, \ldots, m$, and $j = 1, 2, \ldots, c_i$, such that:

   - $\sum_{i,j} b_{i,j} = n$
   - $\sum_{t'=1}^{t} b_{i,t \cdot s + t'} = 1$, for $s = 0, 1, \ldots, s_i - 1$:

   That is, the binary sequence $b_{i,j}$ has n ones (and $tn - n$ zeros), and for each i, $b_{i,j}$ consists of s_i sub-sequences each containing a single 1. An example of the binary sequence is depicted in Fig. 4. For $i = 1, 2, \ldots, m$, $\mathcal{A}$ sends $E_{v_A}(b_{i,j})$ to $\mathcal{P}_i$, for $j = 1, 2, \ldots, c_i$. In this step, a cheating aggregator can attempt to reduce the number of 1's in the selector vector of one or more participants. As we shall see, the fewer the number of 1's, the lower the amount of noise in the final computed aggregate. To detect such cheating, we require that the aggregator prove to every participant $\mathcal{P}_i$, that the above conditions on the $b_{i,j}$ are satisfied. However, the participants must not know the binary values $b_{i,j}$, so this proof should be carried out in zero knowledge. In Section 5.4, we describe an efficient method to accomplish this.

3. Each participant $\mathcal{P}_i$ computes for $j = 1, 2, \ldots, c_i$:

   $$e_{i,j} = E_{v_A}\left(b_{i,j}\right)^{\xi_{i,j}} \cdot E_{v_A}\left(r_{i,j}\right) = E_{v_A}\left(b_{i,j}\xi_{i,j} + r_{i,j}\right).$$

   $\mathcal{P}_i$ then sends the values $e_{i,j}$ to the aggregator.

4. For $i = 1, 2, \ldots, m$, the aggregator computes $\prod_{j=1}^{c_i} e_{i,j} = E_{v_A}(\xi_i + r_i)$, where $\xi_i = \sum_{j:b_{i,j}=1} \xi_{i,j}$ and $r_i = \sum_{j=1}^{c_i} r_{i,j}$. Note that ξ_i is sum of only those $\xi_{i,j}$'s such that $b_{i,j} = 1$. $\mathcal{A}$ decrypts each term $\xi_i + r_i$ and computes its output as: $\sum_{i=1}^{m}(\xi_i + r_i) = \xi + r$.

Correctness: We show that the protocol correctly computes the sum $\xi + r$, where $\xi = \sum_{s=1}^{n} \xi_s$, and $r = \sum_{i=1}^{m} r_i$. Notice that, for j such that $b_{i,j} = 1$, participant i homomorphically computes the encryption of $\xi_{i,j} + r_{i,j}$. Thus, as there are n pairs (i, j) such that $b_{i,j} = 1$, the output that the aggregator computes is: $\sum_{(i,j):b_{i,j}=1} \xi_{i,j} + \sum_i (\sum_j r_{i,j})$. This quantity is $\sum_{s=1}^{n} \xi_s + r$, for some random r. The result follows by noting that each summand of $\sum_{s=1}^{n} \xi_s$ contains only one noise sample and is distributed according to F_S.

5.3 Secure Aggregation Phase

Once the noise term has been generated, the participants run the private aggregation protocol of Section 4 with the aggregator. Specifically, each online participant i sets his input to be $d_i - r_i$, where $r_i = \sum_j r_{i,j}$ generated in the first phase.

Correctness: This follows from the fact that

$$\sum_{i=1}^{m}(d_i - r_i) + \sum_{i=1}^{m}(r_i + \xi_i) = \sum_{i=1}^{m} d_i + \xi.$$

Fault-tolerance: We require that the participants send all their values $e_{i,j}$ (step 3 of the noise generation phase) and their evaluated

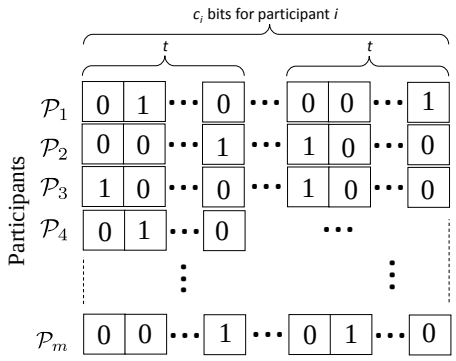

Figure 4: The aggregator generates a c_i-length selector vector for each participant i. The binary values, $b_{i,j}$, are encrypted and used to select which of the noise samples $\xi_{i,j}$ generated by the participants (See Fig. 3) will be added up to generate the final noise term ξ.

polynomial points (step 3 of the private aggregation protocol of Section 4) in a single message (i.e., an atomic operation) to the aggregator. This ensures that, for each participant i, the aggregator gets all the $e_{i,j}$'s and the $E_{v_j}(p^{(i)}(j))$'s in one step. If he does not (e.g., if $\mathcal{P}_i$ fails) then participant i's values are ignored from the protocol. This is sufficient to ensure that fault-tolerance follows from the private aggregation protocol of Section 4. Indeed, if a participant $\mathcal{P}_i$ fails (or goes offline) before delivering its $e_{i,j}$'s and the $E_{v_j}(p^{(i)}(j))$'s, then none of r_i, ξ_i, or d_i will be included in the sum $\sum_j (d_j - r_j) + \sum_j (r_j + \xi_j) = \sum_j d_j + \xi$, which only includes the participants who are online until that step. If $\mathcal{P}_i$ fails after that step, then the blinded noise term $r_i + \xi_i$ will be known to the aggregator, and the fault-tolerance property of the private aggregation protocol ensures that his input (i.e., $d_i - r_i$) is included in the final result.

We emphasize that while $d_i - r_i$ is blinded (i.e., independent of d_i if r_i is not known), it cannot directly be sent to the aggregator. Indeed, given $r_i + \xi_i$ (obtained after the first phase) and $d_i - r_i$, the aggregator can compute $d_i + \xi_i$, but ξ_i is not enough noise (in general) to satisfy differential privacy. This is why the private aggregation protocol of Section 4 is used.

5.4 Detecting a Cheating Aggregator

As described above, the aggregator has no control over the noise terms generated by the participants. However, the aggregator generates the bit selector vector, i.e., the $b_{i,j}$'s. Thus the participants need to ensure that the $b_{i,j}$'s were correctly generated such that enough noise may be added. To do this, we use a protocol proposed by Stern [23], for proving in zero-knowledge that an encrypted vector includes only a single 1 and that the other elements are all 0s. We reproduce this protocol here for convenience.

Inputs: The prover $\mathcal{P}$ has a t bit vector $b_1, b_2, \ldots, b_t$, and an index $x \in \{1, 2, \ldots, t\}$ such that $b_x = 1$, but $b_{x'} = 0$, for $x' \neq x$. There is a public-private key pair of semantically secure additively homomorphic cryptosystem associated with $\mathcal{P}$. The public key is known to the verifier and is denoted by v_P. The verifier $\mathcal{V}$ has no inputs.

Output: The verifier determines that the prover's bit vector contains exactly $t - 1$ zeros and a single one, with probability $1 - (4/5)^l$, for some positive integer l.

Protocol (from [23]):

1. $\mathcal{P}$ computes $E_{v_P}(b_i)$, for $i = 1, 2, \ldots, t$, and sends the results to $\mathcal{V}$.
2. $\mathcal{P}$ and $\mathcal{V}$ repeat l times the following steps:
   (a) $\mathcal{P}$ computes $e_0 = E_{v_P}(0)$ and $e_1 = E_{v_P}(1)$, and sends the results to $\mathcal{V}$
   (b) $\mathcal{V}$ picks $r \in_R \{1, 2, 3, 4, 5\}$.
   (c) If $r = 1$, then $\mathcal{V}$ asks $\mathcal{P}$ to reveal the plaintexts of e_0 and e_1.
   (d) Otherwise, $\mathcal{V}$ randomly partitions $\{1, 2, \ldots, t\}$ into (disjoint) subsets A and B, and calculates:
   - $e_a = \prod_{i \in A} E_{v_P}(b_i) = E_{v_P}(\sum_{i \in A} b_i)$, and
   - $e_b = \prod_{i \in B} E_{v_P}(b_i) = E_{v_P}(\sum_{i \in B} b_i)$.

   $\mathcal{V}$ then sends A and B to $\mathcal{P}$.
   (e) $\mathcal{P}$ proves that e_a and e_b represent ciphertexts of the same numbers as those of e_0 and e_1.

Using this protocol, we describe below, the sub-protocol to detect a cheating aggregator. As before, the steps in the protocol below are numbered according to the stage at which they occur in the protocol described in Section 5.2.

Inputs: There are no inputs.

Output: The participants determine if the aggregator is cheating.

(Sub)-Protocol:

2b Each participant $\mathcal{P}_i$ runs step 2 of the Stern protocol for $s = 0, 1, \ldots, s_i$ to verify that $E_{v_A}(b_{i,t \cdot s + t'})$, for $t' = 1, 2, \ldots, t$, is the encryption of $t - 1$ zeros, and 1 one. The participant plays the role of the verifier and the aggregator plays the role of the prover.

If $\mathcal{P}_i$ detects cheating from $\mathcal{A}$, it aborts and attempts to notify the other participants.

Security: Follows directly from [23]. Note that this protocol can also detect whether the aggregator has chosen non-binary values for any of the $b_{i,j}$'s.

5.5 Privacy Analysis

For non-colluding entities we evaluate the privacy against a single semi-honest participant and against the aggregator. For colluding entities, we evaluate privacy when semi-honest participants collude among themselves and when some participants collude with the cheating aggregator as shown in Fig. 1.

Privacy against a non-colluding semi-honest participant: Each $\mathcal{P}_i$ observes only semantically secure encryptions of the $b_{i,j}$'s. Because $\mathcal{P}_i$ does not know which of the $b_{i,j}$'s are encryptions of 1s, all he knows is that ξ_i will be one of the t^{s_i} possible combinations of the $\xi_{i,j}$. Any other participant $\mathcal{P}_{i'}$, $i' \neq i$ cannot discover the data $d_{i'}$ or the noise term $\xi_{i'}$.

Privacy against colluding semi-honest participants: Though the participants cannot directly communicate with each other, one can consider an attack in which a single entity compromises several participants (e.g., readings from several sensors are sent to a remote server engaging in industrial espionage). Once the protocol is completed, and the aggregator publishes the noisy summation, this entity can subtract the data d_i held by the participants in the colluding set. However, in the protocol of Section 5.2, none of the participants can tell which of their generated noise samples were

aggregated into the final summation. This is because, the bits $b_{i,j}$ that determine which samples are chosen and which are discarded, are encrypted under the aggregator's key.

Let $\mathcal{S}$ be the colluding set and $\widetilde{\mathcal{S}}$ be the non-colluding set, where $|\mathcal{S} \bigcup \widetilde{\mathcal{S}}| = m$, and $|\mathcal{S} \bigcap \widetilde{\mathcal{S}}| = \phi$. Participants in $\mathcal{S}$ can recover:

$$\sum_{i=1}^{m}(d_i + \xi_i) - \sum_{j \in \mathcal{S}} d_j = \sum_{i \in \widetilde{\mathcal{S}}} d_i + \sum_{i=1}^{m} \xi_i$$

Thus, the colluders cannot cancel out their own noise samples from the published aggregate, so the term on the right hand side contains more noise than is needed to ensure differential privacy of the honest participants. For $1 \leq |\mathcal{S}| \leq m-1$, the colluding participants cannot reduce the differential privacy of the honest participants.

Privacy against the aggregator: Since participants follow the protocol honestly, the noise added will be of the correct distribution F. The aggregator observes $\sum_i d_i - r_i$, and for every participant i, he observes the $e_{i,j}$'s. By itself, the summation $\sum_i d_i - r_i$ conveys no information about the participants' inputs d_i owing to the blinding terms r_i. Recall that whenever $b_{i,j} = 1$, $e_{i,j}$ is an encryption of $\xi_{i,j} + r_{i,j}$, which conveys no information about $\xi_{i,j}$, because $r_{i,j}$ is chosen uniformly at random within its domain. (In other words, $\xi_{i,j} + r_{i,j}$ is statistically independent of $\xi_{i,j}$). When $b_{i,j} = 0$, then $e_{i,j}$ is an encryption of $r_{i,j}$, and $\xi_{i,j}$ will not be included in ξ_i. Finally, from $\xi_i + r_i = \sum_{j:b_{i,j}=1} \xi_{i,j} + r_i$, and $\sum_i d_i - r_i$, the aggregator learns no information about d_i or ξ_i, other than what it can learn from $\sum_i d_i + \xi_i$. Thus, the aggregator learns neither the individual noise terms ξ_i, nor the amount of noise added, $\sum_i \xi_i$, nor the participant's private inputs d_i, nor their noise-free sum $\sum_i d_i$.

As we have argued in Section 5.4, the aggregator cannot convincingly cheat by preparing a bit vector $b_{i,j}$ which is not of the proper form. A cheating aggregator might also want to exclude some participants in an effort to learn more about the remaining participants inputs. To see why he cannot exclude too many participants, recall that he needs at least $k+1$ out of m participants to obtain the summation, where k is the degree of the polynomial used for Shamir Secret Sharing.

Privacy against aggregator colluding with (some) participants: Unfortunately, the above favorable situation is lost if the aggregator is part of the colluding set. Suppose the aggregator colludes with some number k' of participants. Through such a coalition, the aggregator is able to subtract the noise generated by the colluding participants and also their private inputs. Thus, it is as if only $m-k'$ participants participated in the protocol, since the noise terms (and the inputs) of the honest participants are independent of those of the colluding participants. Furthermore, by exploiting the fault tolerance property, the aggregator can exclude up to $m-(k+1)$ participants from the protocol and still be able to get a noised sum over the inputs of the non-excluded participants. This means that, at least $k+1-k'$ honest participants will take part in the protocol. In principle, differential privacy can be guaranteed in this situation if the variance of the noise terms ξ_i contributed by those honest participants is large enough. For example, if $k+1 = \frac{2m}{3}$, and $k' = \frac{m}{3}$, then we need to choose the individual noise contributions such that the $k+1-k' = \frac{m}{3}$ honest (and non-failing) participants' noise terms add up to a sample from the distribution F. In practice, however, this defensive strategy is hard to implement because an honest participant does not know how many participants are colluding with the aggregator, and thus might not be able to choose his noise terms $\xi_{i,j}$ correctly.

Limitation: The star topology implies that honest participants can only communicate through (i.e., with the help of) the aggregator. Thus, a participant unfairly excluded from the protocol or one who detects the aggregator cheating is not guaranteed to be able to inform other participants. We remark that this is not a problem for this participant's privacy since he may simply abort the protocol thereby not revealing anything about his private input.

Parameter Set	t	s	$t \cdot s$	l	p_{cheat}	t^s
Resource Bound	2	48	96	62	$\approx 2^{-20}$	2^{48}
High Security	2	80	160	125	$\approx 2^{-40}$	2^{80}

Table 2: Suggested security parameters. The probability p_{cheat} is the probability that the aggregator successfully fools a participant when running the cheating detection protocol. The quantity t^s is the number of possible noise terms.

5.6 Choosing Security Parameters

We now discuss how to set the security parameters t, l, and n. In the noise generation protocol, $\mathcal{P}_i$ calculates c_i encrypted values. For simplicity, it is convenient to take $s = s_i$ to be fixed for all i, so that we have that $c_i = t \cdot s$ for all participants. In such cases, we have $s = \frac{n}{m}$, as n is a multiple of m. Naturally, the larger the values of t, l, and s, the higher the security. That said, both the communication and computation costs of each participant depend on these parameters. In fact, this cost is $O(t \cdot s \cdot l)$, where $O(t \cdot s)$ is the communication cost and computation cost for the $e_{i,j}$'s, and $O(l \cdot s \cdot t)$ is the cost of the protocol to detect a cheating aggregator.

Thus, parameter selection is a trade-off between the communication and computation costs and the level of security achieved. In terms of security, we want to maximize: (1) the probability of detecting a cheating aggregator, and (2) the number of possible final noise terms ξ_i for each participant given his noise components $\xi_{i,j}$. For (1) recall that the probability of detecting a cheating aggregator is $1-(4/5)^l$ using the protocol of Section 5.4.[3] For (2) we want to ensure that the space of all possible final noise terms ξ_i produced by each participant given the components $\xi_{i,j}$ cannot feasibly be explored. (If a participant can explore a non-negligible fraction of that space, he gets information about the distribution of his noise term ξ_i.) Recall that there are t^s possible such combinations.

Table 2 shows two sets of parameters: one for resource-constrained devices which minimizes the communication and computation costs given an acceptable security level, and one providing a higher level of security but with increased communication and computation costs.

5.7 Implementation Considerations

Our noise adding technique (Section 5.1) consists in summing i.i.d. random variables. It is easier to work with integers, since the plaintext domain of additive homomorphic cryptosystems is typically the set of integers modulo some large positive integer. When the noise term is a real number, we can encode it into the plaintext domain using a fixed point representation. Concretely, we take a large positive integer exponent a. We then encode the number $x \in \mathbb{R}$ as the positive integer $x_a = \lfloor x \cdot 2^a \rfloor$. If the maximum al-

3. For the noise of a participant $\mathcal{P}_i$ to be eliminated or significantly reduced, the number of cheating attempts made by the aggregator must be comparable to the number of times the detection protocol is run, i.e., s. Otherwise, only a small fraction of the noise components $\xi_{i,j}$ will be affected. This, and the fact that the aggregator's cheating need only be detected once (as any failure leads to the protocol being aborted), implies that the *effective* probability that the aggregator fools participant i by cheating $s' \leq s$ times is the probability that the aggregator fools participant i all s' times. Therefore, we can drastically reduce the value of l (reducing the communication and computation costs) while keeping (almost) the same security.

lowable number of bits in the plaintext domain is b, then we must take $a < b$. This encodes exactly all numbers, x, whose decimal part can be fully represented in a digits or less, and allows us to represent numbers whose integer part is as large as 2^{b-a}. We can perform homomorphic additions as usual, provided all ciphertexts involved have had their plaintext encoded in the same fashion (with the same exponent a). To recover x after decryption, we simply divide the integer by 2^a and take the result as a real number.

Recall that multiplication of two numbers x and y can be performed with an additively homomorphic cryptosystem provided one number is encrypted, and the other number is available in plain text. Here, two cases arise in our protocols: Either the encrypted number (say x) is an integer and the plaintext number (say y) is a real number, or vice versa. Let v be a public key. Then, if x is a real number, we can simply perform $E_v(x_a)^y = E_v((xy)_a)$. On the other hand, if x is an integer and y is a real number, then given the ciphertext $E_v(x)$, we need to perform: $E_v(x)^{y_a} = E_v((xy)_a)$.

Due to the fixed point representation, not all real numbers can be exactly represented, but this is not a significant concern because the plaintext domain used in practice is quite large (e.g., 2^{1024} or even 2^{2048}) as it is related to the key size in bits. Another concern is that differential privacy might not be achieved due to the approximate nature of the noise sampled, but this is not the case. In fact, textbook floating point implementations are vulnerable to attacks as demonstrated by Mironov [17] while integer or fixed-point representations are not vulnerable to such attacks.

Negative integers can be represented by using the upper half of the plaintext domain: i.e., if the plaintext domain is $\{0, 1, \ldots, M-1\}$, then $-x$, for $1 \leq x \leq \lfloor \frac{M}{2} \rfloor$, is represented as the positive integer $M - x$. When decrypting or decoding we interpret a result as negative if it is greater than or equal $\lceil \frac{M}{2} \rceil$.

6. RELATED PROTOCOLS

The privacy-preserving summation protocol extends to other queries as we describe below. These extensions have also been considered in [20], but without regard for differential privacy as we do here.

Count queries: The aggregator wants to count the number of participants $\mathcal{P}_i$, whose data x_i falls in a set $\mathcal{P}$. The aggregator broadcasts $\mathcal{P}$, and each participant sets their input to the protocol as $d_i = 1$ if $x_i \in \mathcal{P}$, and $d_i = 0$ otherwise. Running the proposed protocol then results in a count query on the set $\mathcal{P}$. To determine the distribution of the noise term for achieving differential privacy in the final computed count, note that the global sensitivity of the count query is $\Delta = 1$.

Histograms: The aggregator wants to compute a histogram based on data x_i held by the participants. It broadcasts a set of disjoint bins $\mathcal{B}_1, \mathcal{B}_2, ..., \mathcal{B}_h$ to the participants. Each participant $\mathcal{P}_i$ constructs a binary vector $\mathbf{d}_i \in \{0,1\}^h$ where the j^{th} element $d_{ij} = 1$ if $x_i \in \mathcal{B}_j$, otherwise $d_{ij} = 0$. Then the participants and the aggregator run a count query for each of the h bins, at the end of which the aggregator obtains the desired histogram without discovering the individual vectors $\mathbf{d}_i$. As a histogram is a generalization of a count query, the distribution of the noise term for achieving differential privacy in the aggregator's output histogram is again based on global sensitivity of a count query, i.e., $\Delta = 1$.

Linear combinations: For $i = 1, 2, ..., N$, the aggregator wants to run a classifier with non-negative weights $c_i < c_{\max}$ on the participants' inputs d_i to determine whether $\mathbf{c}^T\mathbf{d} \lesseqgtr b$. This is achieved by using a slightly modified version of the protocol in Section 4. Concretely, in Step 4, the aggregator computes the ciphertexts C_j using:

$$E_{v_j}(r_j)\prod_{i=1}^{m} E_{v_j}(p^{(i)}(j))^{c_i} = E_{v_j}\left(r_j + \sum_{i=1}^{m} c_i p^{(i)}(j)\right)$$

Consequently, in Step 5, the aggregator gets $q(j) = \sum_{i=1}^{m} c_i p^{(i)}(j) \mod \beta$. Here, the large prime number is chosen as $\beta > m c_{\max} d_{\max}$. Then, in Step 6, it evaluates the polynomial,

$$q(x) = q_1 x + q_2 x^2 + \ldots + q_k x^k + \sum_{i=1}^{m} c_i d_i \mod \beta$$

at $k+1$ or more points. While participant privacy is maintained as before, this process leaks more information than just the comparison $\mathbf{c}^T\mathbf{d} \lesseqgtr b$, as the aggregator discovers the value of $\mathbf{c}^T\mathbf{d}$.

One salient feature though, is that the aggregator does not need to reveal the classifier weights c_i to any of the participants. Multiple linear combinations (hence classifiers) can thus be realized, *without repeating the polynomial secret sharing step* in Section 4. This is an advantage over the prior art. Specifically, although the prior art could also be extended to compute linear combinations, in most cases ([1, 3, 4, 9, 12, 15, 16, 22]), the protocols have to reveal the weights c_i to the participants. Moreover, they have to repeat the secret sharing step whenever a new linear combination is computed.

In a differentially private realization, the classifier boundaries will be perturbed because of the noise term that is added to the inner product $\mathbf{c}^T\mathbf{d}$. The privacy/utility tradeoff is then based on the ϵ and δ values, and the classification error that occurs as a consequence of adding noise. Recall that, the distribution of the noise term ξ to be added to $\mathbf{c}^T\mathbf{d}$ is based on the global sensitivity of the linear combination, which is $\Delta = c_{\max} d_{\max}$. In this case, the aggregator has to multiply not just the perturbed data terms $d_i - r_i$, but also the perturbed noise terms $r_i + \xi_i$ of participant $\mathcal{P}_i$ with the constant c_i, to ensure that:

$$\sum_{i=1}^{m} c_i(d_i - r_i) + \sum_{i=1}^{m} c_i(r_i + \xi_i) = \sum_{i=1}^{m} c_i d_i + \xi$$

Note that, as the distribution of noise term ξ depends on $c_{\max}$, the aggregator must reveal $c_{\max}$ (but fortunately not the individual c_i's) to all participants, so that they can generate the appropriate candidate noise samples.

7. CONCLUDING REMARKS

We have considered privacy preserving aggregation in constrained scenarios where inter-participant communication is not realistically possible. We employed Shamir secret sharing within a star network to provide collusion resistance and fault tolerance. We developed protocols to add noise to the computed aggregate function, thereby ensuring differential privacy for the participants. The differential privacy requirement requires us to appreciate that collusions are capable of reducing privacy without necessarily discovering the honest participants' data.

To add the correct amount of noise, we make the participants generate several noise samples, with the aggregator deciding (obliviously) which of those samples are utilized in the differentially private mechanism. In this way, neither the cheating aggregator nor the semi-honest participants know which noise values are used to achieve differential privacy. This ensures that semi-honest colluding participants cannot reduce the differential privacy guarantees of honest participants. Our adversarial model allows the aggregator to influence (e.g., reduce) the noise term, so we describe a protocol to catch a cheating aggregator with overwhelming probability.

References

[1] G. Ács and C. Castelluccia. I have a DREAM! (differentially private smart metering). In *Information Hiding*, pages 118–132, 2011.

[2] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 1–10, 1988.

[3] I. Bilogrevic, J. Freudiger, E. De Cristofaro, and E. Uzun. What's the gist? privacy-preserving aggregation of user profiles. In *Computer Security-ESORICS 2014*, pages 128–145. 2014.

[4] T.-H. H. Chan, E. Shi, and D. Song. Privacy-preserving stream aggregation with fault tolerance. In *Financial Cryptography and Data Security*, pages 200–214. 2012.

[5] I. Damgård, M. Jurik, and J. Nielsen. A generalization of Paillier's public-key system with applications to electronic voting. *International Journal of Information Security*, 9(6):371–385, 2010.

[6] C. Dwork. Differential privacy: A survey of results. In *Theory and applications of models of computation*, pages 1–19. Springer, 2008.

[7] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Theoretical Computer Science*, 9(3-4):211–407, 2013.

[8] Z. Erkin, J. R. Troncoso-Pastoriza, R. Lagendijk, and F. Perez-Gonzalez. Privacy-preserving data aggregation in smart metering systems: An overview. *Signal Processing Magazine, IEEE*, 30(2):75–86, 2013.

[9] Z. Erkin and G. Tsudik. Private computation of spatial and temporal power consumption with smart meters. In *Applied Cryptography and Network Security*, pages 561–577, 2012.

[10] F. Garcia and B. Jacobs. Privacy-friendly energy-metering via homomorphic encryption. In *Security and Trust Management*, pages 226–238. 2011.

[11] M. Jawurek and F. Kerschbaum. Fault-tolerant privacy-preserving statistics. In *Privacy Enhancing Technologies*, pages 221–238, 2012.

[12] M. Joye and B. Libert. A scalable scheme for privacy-preserving aggregation of time-series data. In *Financial Cryptography and Data Security*, pages 111–125. 2013.

[13] D. E. Knuth. Seminumerical Algorithms, The art of computer programming, Vol. 2, Section 4.6, 1981.

[14] S. Koltz, T. Kozubowski, and K. Podgorski. The laplace distribution and generalizations, 2001.

[15] K. Kursawe, G. Danezis, and M. Kohlweiss. Privacy-friendly aggregation for the smart-grid. In *Privacy Enhancing Technologies*, pages 175–191, 2011.

[16] I. Leontiadis, K. Elkhiyaoui, and R. Molva. Private and dynamic time-series data aggregation with trust relaxation. In *Cryptology and Network Security*, pages 305–320. Springer, 2014.

[17] I. Mironov. On significance of the least significant bits for differential privacy. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 650–661. ACM, 2012.

[18] K. Nissim, S. Raskhodnikova, and A. Smith. Smooth sensitivity and sampling in private data analysis. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 75–84. ACM, 2007.

[19] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in cryptology, EUROCRYPT99*, pages 223–238, 1999.

[20] S. Rane, J. Freudiger, A. Brito, and E. Uzun. Privacy, efficiency and fault tolerance in aggregate computations on massive star networks. In *IEEE Workshop on Information Forensics and Security (WIFS 2015)*, Rome, Italy, November 2015.

[21] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[22] E. Shi, T.-H. H. Chan, E. Rieffel, R. Chow, and D. Song. Privacy-preserving aggregation of time-series data. In *NDSS*, volume 2, page 4, 2011.

[23] J. Stern. A new and efficient all-or-nothing disclosure of secrets protocol. In *Advances in Cryptology—ASIACRYPT'98*, pages 357–371. Springer, 1998.