Privacy, Efficiency & Fault Tolerance in Aggregate Computations on Massive Star Networks

Shantanu Rane, Julien Freudiger, Alejandro E. Brito, and Ersin Uzun
Palo Alto Research Center (PARC)
3333 Coyote Hill Road, Palo Alto, CA 94304.
Email: {srane,jfreudig,abrito,euzun}@parc.com

***Abstract*—We consider the challenge of performing efficient, fault-tolerant, privacy-preserving aggregate computations in a star topology, i.e., a massive number of participants connected to a single untrusted aggregator. The privacy constraints are that the participants do not discover each other's data, and the aggregator obtains the final results while remaining oblivious to each participant's individual contribution to the aggregate. In achieving these goals, previous approaches have either assumed a trusted dealer that distributes keys to the participants and the aggregator, or introduced additional parties that withhold the decryption key from the aggregator, or applied secret sharing with either pairwise communication amongst the participants or $O(N^2)$ ciphertext overhead at the aggregator. In contrast, we describe a protocol based on Shamir secret sharing and homomorphic encryption without assuming any additional parties. We also eliminate all pairwise communication amongst the participants and still require only $O(N^{1+\epsilon})$ overhead at the aggregator, where $\epsilon \ll 1$ can be achieved for massively multiparty computation. Our protocol arranges the star-connected participants into a logical hierarchy that facilitates parallelization, while allowing for user churn, i.e., a specified number of participants can go offline after providing their data, and new participants can join at a later stage of the computation.**

I. Introduction

Consider a multiparty computation scenario in which a server computes aggregate measures from individual contributions of a massive number of participants. Aggregate measures include summations, linear combinations, count queries, histograms and related functions. Such situations arise repeatedly in many applications where privacy of the participants is a concern. Examples include aggregation of power consumption data furnished by thousands of smart meters to a utility company, evaluating statistical measures on fitness data provided by wearables or smartphones of millions of users, obtaining statistics on browser activity from internet users, and so on. Although individuals may consent to the computation of aggregate measures in return for value-added services, they may be reluctant to share information about their individual behavior. In the above examples, power usage information reveals a home-owner's daily patterns, fitness apps may reveal sensitive medical information, and browser activity can reveal intimate details about an individual's life and values.

Owing to the above privacy concerns, solutions to such aggregation problems belong to the realm of secure multiparty computation. The main privacy constraint in this interaction between untrusted entities is that the data held by any individual participant should not be revealed to other participants, and especially not to the aggregator. The aggregator should discover only the desired aggregate measure such as the sum or the histogram of values, but nothing else. Moreover, to facilitate practical deployment, additional constraints may have to be imposed. One such constraint is that the computational complexity should be manageable; even with large computational resources available to cloud servers, a $O(N^2)$ overhead would be unreasonable when there are millions of participants. Another possible constraint is that the participants may not be able to communicate with one another. At first glance, this threatens to rule out the elegant approaches to secure computation based on secret sharing. A third practical constraint is that millions of participants, cannot all be expected to remain online simultaneously for the protocol to be executed. Thus, an aggregation protocol must be able to function correctly when users dynamically join and leave.

II. Overview of Related Work

Addressing all the aforementioned constraints is difficult predominantly because of the key management problem. Concretely, each participant should obfuscate its input such that all obfuscated inputs can later be homomorphically combined by the aggregator to reveal the aggregate function. However, this is not straightforward for the star network topology because each participant uses its own unique key for obfuscation. Thus, even a fully homomorphic cryptosystem will not resolve this issue. Below, we present a short overview of attempts to address this problem. For more details, we refer the reader to a comprehensive review by Erkin *et al.* [1]

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 [2], [3]. A similar approach is followed by Bilogrevic *et al.*, to compute means, variances and higher moments of distributions [4]. Rather than assuming a trusted dealer, Jawurek and Kerschbaum distribute the task of computation and decryption between an untrusted aggregator and an untrusted key managing authority [5]. This is an efficient approach with a single public key homomorphic cryptosystem, that has only $O(N)$ overhead. However, it introduces an extra participant, and carries 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 party called the collector [6]. Similar to the previous approach, their scheme forbids collusions between the aggregator and the collector.

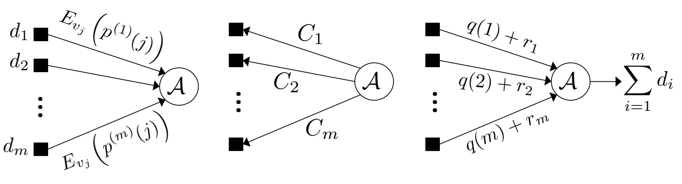
Work	Approach	Network Topology	No. of Rounds	Correctness under Node Failures	Ciphertext Computation Overhead	Ciphertext Transmission Overhead
Shi *et al.* [2], Bilogrevic *et al.* [4]	Differentially private aggregation with geometric distribution	Star network + trusted key dealer.	1	No	$O(N)$	$O(N)$
Chan *et al.* [3],	Differentially private aggregation with fault tolerance	Star network + trusted key dealer.	1	Yes	$O(N)$	$O(N)$
Joye and Libert [7]	Private aggregation with a large plaintext space using discrete logarithms	Star network + trusted key dealer.	1	No	$O(N)$	$O(N)$
Jawurek and Kerschbaum [5]	Practical scheme with a single additively homomorphic key-pair	Star network + untrusted key managing party	1	Yes	$O(N)$	$O(N)$
Leontiadis *et al.* [6]	Private Aggregation with Dynamic Group Management	Star network + untrusted "collector" party	1	Yes	$O(N)$	$O(N)$
Erkin and Tsudik [8]	Efficient homomorphic aggregation with inter-participant communication	Fully connected network to share random values	2	No	$O(N)$	$O(N)$
Ács and Castelluccia [9]	Differentially private aggregation with additive secret sharing	Fully connected network to share secrets	2	Yes, via differential privacy. This reduces accuracy.	$O(N)$	$O(N)$
Kursawe *et al.* [10]	Private aggregation using additive secret sharing	Star network + L "leaders".	2	No	$O(N)$	$O(N^2)$
Garcia and Jacobs [11]	Additive secret sharing with homomorphic encryption	Star network	2	No	$O(N^2)$	$O(N^2)$
This work	Shamir secret sharing and homomorphic encryption within cohorts	Star network	2	Yes	$O(N^{1+\epsilon})$ with $\epsilon < 1$	$O(N^{1+\epsilon})$ with $\epsilon < 1$

TABLE I. A CONCISE SUMMARY OF THE LITERATURE ON PRIVACY-PRESERVING DATA AGGREGATION.

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 [10]. The leaders add their own shares such that their effect vanishes upon combination, revealing only the sum of the participants' data. Going a step further, 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 [11]. 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. Unfortunately, both these approaches incur $O(N^2)$ ciphertext communication overhead at the aggregator.

We point to two other efforts on this topic, which relax the strict star network topology in exchange for a gain in the efficiency of aggregation. The first is the work of Erkin and Tsudik, which allows the participants (smart meters) to communicate with one another, exchanging random values before each smart meter communicates with the aggregator [8]. This is a mode of secret sharing, in which the randomized values are chosen to vanish when the aggregator computes the sum. The work of Ács and Castelluccia is also similar in that they allow each participant to communicate with a small number of other participants [9]. 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., $O(N)$. 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 $O(N)$ ciphertext overhead may not be achievable, in general, for fault-tolerant aggregation in a star-connected network. Indeed, we are not aware of any protocol that achieves this. This insight has informed our design of a fault-tolerant privacy-preserving aggregation scheme that utilizes Shamir secret sharing [12] and achieves $O(N^{1+\epsilon})$ overhead. To the best of our knowledge, this is the first scheme to achieve lower than $O(N^2)$ overhead with fault-tolerance for a strictly star-connected network, without introducing additional (trusted or untrusted) parties or requiring any of the N participants to explicitly share information with each other.

We note that in many of the above schemes [2]–[4], [9], the participants also add noise to their data in order to ensure differential privacy of the eventually computed aggregate function. The scheme that we will present herein, can also accommodate these approaches at achieving differential privacy for the participants. However, in this paper, we will concentrate only on ensuring the privacy constraints of the multiparty cryptographic protocols. Considerations of differential privacy and specific methods to achieve it within the proposed framework will be elaborated in a later work.



Steps 1, 2, 3 Step 4 Steps 5, 6

Fig. 1. Steps of the basic protocol of Section III. Participants send homomorphically encrypted shares to the aggregator $\mathcal{A}$ to indirectly implement Shamir secret sharing in a star network topology.

III. Secret Sharing for Private Aggregation

In the following, 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. Clearly, the data generated by a participant should not be revealed to the other participants. An important privacy requirement is referred to as *Aggregator Obliviousness* which means that the aggregator discovers only the aggregate function being computed and nothing else about the inputs of the individual participants. First, we consider the simplest aggregate function, namely the sum of the participants' data. Later, we will describe how to extend the summation protocols to the computation of other aggregate measures such as counts and histograms.

Let N be the total number of participants whose data is to be aggregated. In this section, we consider a protocol in which the aggregator interacts with a subset of m participants. For convenience, suppose that it is possible to choose $N = m^a$, where a is some positive integer. We now describe a protocol in which the aggregator obtains the partial sum of data held by m participants, incurring $O(m^2)$ complexity while supporting dynamic exits by a few of the participants. In the next section, we will show that, using many cohorts with m participants in each, the aggregator can obtain the final sum of data held by all N participants with complexity $O(N^{1+\frac{1}{a}})$. When a is large, this represents a significant saving over $O(N^2)$ protocols previously proposed for star networks.

Our initial protocol with m participants is based on Shamir Secret Sharing [12] 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. We assume that the aggregator and all the participants are *semi-honest* (*honest but curious*), i.e., they will follow the protocols but, at each step, they may attempt to discover the data held by honest participants. Thus, during privacy analysis, we will consider the view of semi-honest participants, and collusions of semi-honest participants.

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 of each participant. We assume that d_i is a non-negative integer and $d_i < d_{\max}$. The case of negative inputs can be accommodated simply by appropriately shifting all d_i, so that they are all non-negative. Associated with each $\mathcal{P}_i$ is a public-private key pair of a semantically secure additively homomorphic cryptosystem such as the Paillier cryptosystem [13], or its generalization, the Damgard-Jurik cryptosystem [14]. Denoting the public key for $\mathcal{P}_i$ by v_i, the additive homomorphic property ensures that $E_{v_i}(a)E_{v_i}(b) = E_{v_i}(a + b)$. Choose a positive integer $k < m$ as a fault tolerance parameter.

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

Protocol: Consider the following steps, also shown in Fig. 1:

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 $j \in \{1, 2, \ldots, m\}$,
$$q(j) = \sum_{i=1}^{m} p^{(i)}(j) \mod \beta$$
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: To prove that the protocol correctly evaluates the desired summation, note that by applying additively homomorphic encryption with appropriate public keys, the aggregator is able to distribute encrypted 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 [15]. Another way to verify correctness is to note that the underlying Shamir secret sharing is additively homomorphic modulo β.

Fault-Tolerance: Observe that the degree of the final "sum" polynomial is $k < m$. Hence, the protocol is fault tolerant: The aggregator can compute the summation even when up to $m-k-1$ parties 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 players. Secret sharing ensures that no honest participant discovers the data held by any other 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 players. Since the participants can only communicate through the aggregator, any non-trivial semi-honest coalition must contain the aggregator. In order to learn any further information about 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: We measure complexity in terms of the amount of ciphertext communication and computation. The communication complexity, as dominated by Step 3, is $O(m^2)$. The ciphertext computation complexity, as dominated by Step 4, is also $O(m^2)$. 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 [16].

IV. Logical Hierarchy to Mitigate $O(N^2)$ Overhead

In this section, we show how it is possible to avoid the $O(N^2)$ complexity blow-up, by grouping the N participants into smaller cohorts of m participants each. Note that this does not trivially follow by just repeating the protocol from Section III a total of $\lceil N/m \rceil$ times, because this would reveal extra information to the aggregator in the form of the partial sums $\sum_i d_i$ of each cohort of m participants. To avoid this, additive secret sharing is used again, as explained below.

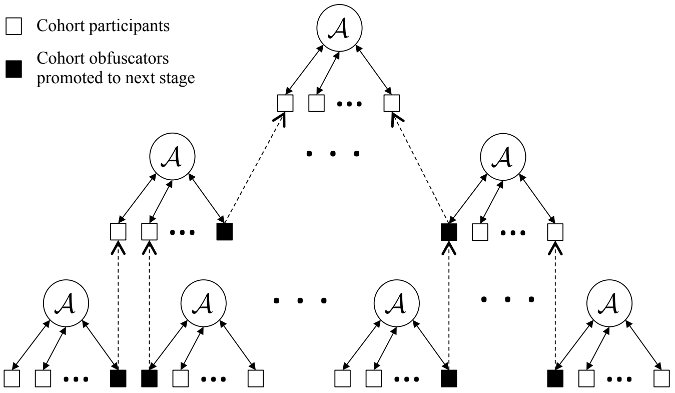


Fig. 2. The star-connected participants are arranged in a logical hierarchy to reduce the overall ciphertext overhead. This structure also ensures that, in each cohort, up to $m-k-1$ participants can go offline after Step 3 of the basic protocol in Section III. It also allows new participants to join a cohort at a later stage.

At a high level, the solution technique is as follows: One participant is chosen at random by the aggregator as an untrusted "obfuscator" for each cohort. The aggregator executes the protocol of Section III for each cohort, however, the obfuscator of each cohort additively blinds its input. As a result, the aggregator obtains an incorrect sum for each cohort. In the next stage, all obfuscators are again grouped into new cohorts of m participants each, and again, a new obfuscator is chosen at random for each cohort. At this stage, each participant, except the newly chosen obfuscator, treats as input a value that reverses the blinding operation from the previous round. Then, the aggregator again executes the protocol of Section III for all the newly formed cohorts. This process is repeated until only one cohort can be formed, containing a maximum of m participants. At this stage, no obfuscator is chosen and the protocol of Section III is executed as is, yielding the desired summation of the inputs of all N parties.

The protocol is drawn in Fig. 2 and explained more precisely below. Recall that we have chosen $N = m^a$ for convenience, to illustrate how the quadratic complexity of the summation protocol is mitigated when $N \gg m$.

1) The following steps are repeated until only one cohort of m participants remains:
   a) The aggregator divides the participants into cohorts of m participants and chooses an obfuscator at random within each cohort. Suppose there are T cohorts remaining. For $t = 1, 2, ..., T$, denote by $d_o^{(t)}$, the input data of the obfuscator of the t^{th} cohort.
   b) Each obfuscator chooses an integer value s_t uniformly at random from the set $\mathcal{I} = \{s\,;\ |s + d_o^{(t)}| < d_{\max}\}$, and uses $d_o^{(t)} + s$ as its new input.
   c) For each cohort, the aggregator executes the basic protocol of Section III and records the partial summation. Note that this summation is not the true sum of the inputs of the cohort's participants, because its obfuscator has introduced a random error in the previous step.
   d) The T cohort obfuscators are now promoted to serve as the participants in the next stage of the computation. For this stage, their input values are set to $d_o^{(t)} = -s_t$ in order to cancel for the effect of the masking above.

2) In the final remaining cohort, m participants interact with the aggregator to execute the basic protocol of Section III.
3) The aggregator adds up the partial sums of all cohorts.

Correctness: The correctness of the above protocol follows from that of the basic protocol of Section III. The difference is that instead of computing the true sum of the inputs of each of the cohort's participants, the partial sum for each cohort is masked by its obfuscator, in order to hide the partial sum from the aggregator. At the end of each stage, the masked value of the cohort's obfuscator is reversed in sign as described above, to serve as input data when that obfuscator becomes a participant in the next stage. It can be verified that, when the aggregator adds up the partial sums of all cohorts, it obtains the true summation given by $\sum_{i=1}^{N} d_i$.

Fault Tolerance: Within each cohort, fault tolerance carries over from Section III. However, the cost of the hierarchical approach is reduced fault tolerance with respect to the obfuscators. Since obfuscators are promoted to the next level of the hierarchy, as shown in Fig. 2, they need to remain online until Step 3 of the basic protocol is completed for the cohort in the new hierarchical level. If an obfuscator goes offline before this step, the aggregator has to discard the partial summation of the cohort from which the obfuscator was selected, as well as all cohorts below it in the hierarchy.

Privacy: For individual semi-honest players inside a cohort, privacy properties are the same as those discussed in the previous section. We re-iterate that none of the cohort-based aggregation operations (except the last one) reveals the true sum of the participants' data to the aggregator. Furthermore, the obfuscators do not have any extra visibility into the participants' data; they are merely designated to become participants in the next hierarchical level of the protocol.

Next, consider privacy against semi-honest collusions. An important consequence of the hierarchical construction is that, if the aggregator colludes with an obfuscator, the two colluders discover the summation of the inputs of all participants of that cohort. Choosing the obfuscator at random is important. Otherwise, a semi-honest aggregator can keep selecting the same participant as the obfuscator, and collude with that participant to obtain the partial summation of multiple cohorts.

Even in the above collusion attack, the colluders do not discover the individual participants' data. This type of collusion is a milder privacy compromise compared to collusion between the aggregator and the collector in [6] or between the aggregator and the key authority in [5], which compromises every honest participant's input. From Section III, in order to compromise the data of any honest participant, a coalition must comprise the aggregator and at least $k+1$ other participants *from the same cohort as the targeted victim*. Therefore, large coalitions are useless at discovering the data of honest participants unless any given cohort contains at least $k+1$ colluders.

Complexity: Recall that the ciphertext overhead of the basic protocol of Section III was $O(m^2)$. For $N = m^a$, the total number of times that the basic protocol is executed can be obtained as:

$$m^{a-1} + m^{a-2} + \ldots + m^2 + m + 1 = \frac{m^a - 1}{m - 1}$$

Thus, the ciphertext overhead for the full protocol becomes:

$$O\left(m^2 \frac{m^a - 1}{m - 1}\right) \equiv O(Nm) \equiv O\left(N^{1+\frac{1}{a}}\right)$$

Thus, the ciphertext overhead of the overall protocol is $O(N^{1+\epsilon})$, where $\epsilon = \frac{1}{a} < 1$. Here, $N = m^a$ was chosen to satisfy a hierarchy in which there are m-member cohorts at each stage. Even if this requirement is not satisfied, and the cohorts differ in size, a similar guarantee holds. For example, if the cohorts at the lower level of the hierarchy have m members, and some or all cohorts at the upper levels contain fewer than m members, the complexity is $O(N^{1+\frac{1}{a'}})$, where $a' = \log_m N$. For $N \gg m$, the computational complexity again becomes significantly better than $O(N^2)$.

V. Other Consequences of Hierarchical Aggregation

In addition to fault tolerance and privacy against semi-honest coalitions, the hierarchical protocol discussed above has some useful properties, which we discuss below.

Support for Dynamic Joins: The protocol structure enables new participants to join at later stages of the protocol. Concretely, as long as there is a cohort available, either consisting of the original participants, or the obfuscators chosen from the original participants, new participants can join, and provide their input to the aggregation function.

Parallelizability: The hierarchical structure readily lends itself to parallelized implementations, in which the aggregator simultaneously allocates the task of computing the partial summations to several cloud-based servers. It would have been most preferable for the computation of all cohorts to be simultaneously parallelized, and this is indeed possible if the number of participants N is fixed *a priori*, the cohorts and their obfuscators have been designated for each stage. However, as our goal is to support dynamic exits and joins, computation in all cohorts cannot be simultaneously parallelized. For example, if the hierarchy consists of $L \approx \lceil \log_m N \rceil$ levels, L stages of parallel computation of partial sums of cohorts are possible. In contrast, without any parallelization, the aggregator would have to compute partial summations for $\lceil N/m \rceil$ cohorts, one after the other. Thus, parallelization provides a significant speed-up for massively multiparty computation in which N grows much faster than m. For example, for N = 1 million participants, and cohort size, m = 100, we have $L = 3$ and $N/m = 10000$.

Graceful Degradation of Collusion Resistance: If all N users belonged to a single cohort, and a K-out-of-N secret sharing scheme was used, the collusion resistance property would show a “cliff” effect, i.e., for a single coalition of size $K-1$ or less, the privacy of the honest participants is protected. When one more member is added to the coalition, the privacy of all honest participants is compromised. In the hierarchical scheme described above, collusion resistance degrades more gracefully. i.e., honest participants are protected as long as their cohort contains k semi-honest colluders or less. As the coalition size increases, some cohorts will contain $k+1$ or more colluders, which is sufficient to compromise the privacy of all honest participants within those cohorts. In other words, under the influence of growing semi-honest coalitions, privacy loss is initially restricted to the most infected cohorts.

VI. Related Computations

The privacy-preserving summation protocol extends to the computation of other aggregate measures. Some illustrative examples are given below.

1) **Count queries**: Suppose that 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 above privacy-preserving protocol then results in a count query on the set $\mathcal{P}$.
2) **Histograms**: Suppose 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 $\mathbf{d}_i$.
3) **Linear classifiers**: For $i = 1, 2, ..., N$, suppose the aggregator wants to run a classifier with non-negative integer weights $c_i < c_{\text{max}}$ on the participants' inputs d_i to determine whether $\mathbf{c}^T\mathbf{d} \leqslant b$. This is achieved by computing cohort-wise linear combinations using a slightly modified version of the protocol in Section III. Concretely, in Step 4, the aggregator computes the ciphertexts C_j using:

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

As a consequence, in Step 5, the aggregator obtains $q(j) = \sum_{i=1}^{m} c_i p^{(i)}(j) \mod \beta$. Here, the large prime number is chosen as $\beta > mc_{\text{max}}d_{\text{max}}$. 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 $\mathbf{c}^T\mathbf{d} \leqslant b$, as the aggregator discovers the value of $\mathbf{c}^T\mathbf{d}$. One salient feature, however, is that the aggregator need not 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*. This is an advantage over the prior art. Concretely, although the prior art can also be extended to compute linear combinations, in most cases ([2]–[4], [6]–[10]), 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.

VII. Concluding Remarks

We have considered privacy preserving aggregation in constrained scenarios where inter-participant communication is not realistically possible. Introducing extra players such as collectors and key authorities certainly simplifies aggregation protocols, but this can be risky in practice, because a semi-honest coalition of the aggregator and an extra player can result in catastrophic privacy loss for all participants. To our knowledge, only the work of Garcia and Jacobs [11] utilized the strict star topology with no additional setup. Our proposal can be considered as providing two improvements to that approach: (1) Incorporation of fault tolerance, and (2) Reduction of protocol complexity by grouping star-connected participants into a logical hierarchy. We have also discussed the resistance of our protocol to various collusion scenarios.

In ongoing work, we are exploring the benefits of adding noise to the participants' data for differentially private aggregation. Examples include adding two-sided geometric noise [2], [3], or Gamma distributed noise [9] at the participants, in order to achieve the desired noise distribution after aggregation. We are also interested in aggregator-based services that enable one participant to compare herself against a population. Such computations involve non-linear operations (one or more privacy-preserving comparisons) which increase protocol complexity and make analysis of collusion resistance more challenging.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

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

[6] I. Leontiadis, R. Molva, and M. Onen. Privacy preserving statistics in the smart grid. In *Distributed Computing Systems Workshops (ICDCSW), 2014 IEEE 34th International Conference on*, pages 182–187, 2014.

[7] 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.

[8] 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.

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

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

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

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

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

[14] 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.

[15] 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.

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