Practical Secure Aggregation for Privacy-Preserving Machine Learning

Keith Bonawitz*, Vladimir Ivanov*, Ben Kreuter*, Antonio Marcedone†*,
H. Brendan McMahan*, Sarvar Patel*, Daniel Ramage*, Aaron Segal* and Karn Seth*
*Google, 1600 Amphitheatre Parkway Mountain View, California 94043
{bonawitz, vlivan, benkreuter, mcmahan, sarvar, dramage, asegal, karn}@google.com
†Cornell University, Ithaca, New York 14853
marcedone@cs.cornell.edu

***Abstract*—We design a novel, communication-efficient, failure-robust protocol for secure aggregation of high-dimensional data. Our protocol allows a server to compute the sum of large, user-held data vectors from mobile devices in a secure manner (i.e. without learning each user's individual contribution), and can be used, for example, in a federated learning setting, to aggregate user-provided model updates for a deep neural network. We prove the security of our protocol in the honest-but-curious and malicious settings, and show that security is maintained even if an arbitrarily chosen subset of users drop out at any time. We evaluate the efficiency of our protocol and show, by complexity analysis and a concrete implementation, that its runtime and communication overhead remain low even on large data sets and client pools. For 16-bit input values, our protocol offers $1.73\times$ communication expansion for 2^{10} users and 2^{20}-dimensional vectors, and $1.98\times$ expansion for 2^{14} users and 2^{24}-dimensional vectors over sending data in the clear.**

I. Introduction

Machine learned models trained on sensitive real-world data promise improvements to everything from medical screening [1] to disease outbreak discovery [2]. And the widespread use of mobile devices means even richer—and more sensitive—data is becoming available [3]. However, large-scale collection of sensitive data entails risks. A particularly high-profile example of the consequences of mishandling sensitive data occurred in 1988, when the video rental history of a nominee for the US Supreme Court was published without his consent [4]. The law passed in response to that incident remains relevant today, limiting how online video streaming services can use their user data [5].

This work outlines an approach to advancing privacy-preserving machine learning by leveraging secure multiparty computation (MPC) to compute weighted averages of model parameter updates from individual users' devices in a secure manner. The problem of computing a multiparty weighted average where no party reveals its update in the clear—even to the aggregator—is referred to as *Secure Aggregation*. As described in Section II, the secure aggregation primitive can be used to privately combine the outputs of local machine learning on user devices, in order to update a global model. Training models in this way offers tangible benefits—a user's device can share an update knowing that the service provider will only see that update after it has been averaged with those of other users.

While secure aggregation alone may suffice for some applications, for other applications stronger guarantees may be needed, as indicated by the failures of ad-hoc anonymization techniques [6], [7], [8], and by the demonstrated capability to extract information about individual training data from fully-trained models (which are essentially aggregates) [9], [10], [11]. In such cases, secure aggregation composes well with *differential privacy* [12]. This is particular advantageous in the *local privacy* setting [13], which offers provable guarantees for the protection of individual training examples [14], [15] even when the data aggregator is not assumed to be trusted [16], [17]. For example, when computing averages, partial averages over subgroups of users may be computed and privacy-preserving noise may be incorporated [18] before revealing the results to the data aggregator. Under some privatization schemes, for a fixed total number of users and for secure aggregation subgroups of size n, the same amount of differential privacy may be offered to each user while reducing the standard deviation of the estimated average across all users by a factor of $\sqrt{n}$ relative to providing local differential privacy without secure aggregation[1]. Thus, secure aggregation over just 1024-user subgroups holds the promise of a $32\times$ improvement in differentially private estimate precision. We anticipate that these utility gains will be crucial as methods for differentially private deep learning in the trusted-aggregator setting [14] are adapted to support untrusted aggregators, though a detailed study of the integration of differential privacy, secure aggregation, and deep learning is beyond the scope of the current work.

The secure aggregation problem has been a rich are of research: different approaches include works based on generic secure multi-party computation protocols, works based on DC-nets, works based on partially- or fully-homomorphic threshold encryption, and works based on pairwise masking. We discuss these existing works in more detail in Section IV, and compare them to our approach.

We are particularly focused on the setting of mobile devices, where communication is extremely expensive, and dropouts are common. Given these constraints, we would like our protocol to incur no more than twice as much communication as sending the data vector to be aggregated in the clear, and woudl also

[1]See Appendix D for details.

like the protocol to be fully robust to users dropping at any point. We believe that previous works do not address this mixture of constraints, which is what motivates our work.

A. Our Results

We present a protocol for securely computing sums of vectors, which has a constant number of rounds, low communication overhead, robustness to failures, and which requires only one server with minimal trust. In our design the server has two roles: it routes messages between the other parties, and it computes the final result. We present two variants of the protocol: one is more efficient and can be proven secure against honest but curious adversaries, in the plain model. The other guarantees privacy against malicious adversaries (including a malicious server), but requires an extra round, and is proven secure in the random oracle model. In both cases, we can show formally that the server only learns users' inputs in aggregate, using a simulation-based proof as is standard for MPC protocols. Both variants we present are practical and we present benchmark results from our prototype implementation.

B. Organization

In Section II we describe the machine learning application that motivates this work. In Section III, we give a high-level description of our protocol design. We discuss related work in Section IV. We present our formal protocol description in Section V. In Section VI we show security against honest-but-curious (passive) adversaries in Sections VI. In that section, we also include a high-level discussion of privacy against malicious (active) adversaries, and give a full proof of the same in Appendix C. In Section VII, we give performance numbers based on a prototype implementation. Finally, we discuss some issues surrounding practical deployments and future work in Section VIII. We also provide an Appendix that includes the aforementioned proof of privacy against active adversaries, formalizes the cryptographic primitives used throughout the paper, and provides some technical lemmas.

II. Secure Aggregation for Federated Learning

Consider training a deep neural network to predict the next word that a user will type as she composes a text message. Such models are commonly used to to improve typing efficacy for a phone's on-screen keyboard [19]. A modeler may wish to train such a model on all text messages across a large population of users. However, text messages frequently contain sensitive information; users may be reluctant to upload a copy of them to the modeler's servers. Instead, we consider training such a model in a *Federated Learning* setting, wherein each user maintains a private database of her text messages securely on her own mobile device, and a shared global model is trained under the coordination of a central server based upon highly processed, minimally scoped, ephemeral updates from users [20], [10].

A neural network represents a function $f(\boldsymbol{x}, \boldsymbol{\Theta}) = \boldsymbol{y}$ mapping an input $\boldsymbol{x}$ to an output $\boldsymbol{y}$, where f is parameterized by a high-dimensional vector $\boldsymbol{\Theta} \in \mathbb{R}^m$. For modeling text message composition, $\boldsymbol{x}$ might encode the words entered so far and $\boldsymbol{y}$ a probability distribution over the next word. A training example is an observed pair $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$ and a training set is a collection $\mathcal{D} = \{\langle \boldsymbol{x}_i, \boldsymbol{y}_i \rangle; i = 1, \ldots, m\}$. A loss is defined on a training set $\mathcal{L}_f(\mathcal{D}, \boldsymbol{\Theta}) = \frac{1}{|\mathcal{D}|} \sum_{\langle \boldsymbol{x}_i, \boldsymbol{y}_i \rangle \in \mathcal{D}} \mathcal{L}_f(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\Theta})$, where $\mathcal{L}_f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\Theta}) = \ell(\boldsymbol{y}, f(\boldsymbol{x}, \boldsymbol{\Theta}))$ for a loss function ℓ, e.g., $\ell(\boldsymbol{y}, \hat{\boldsymbol{y}}) = ||\boldsymbol{y} - \hat{\boldsymbol{y}}||_2$.

Training a neural net consists of finding parameters $\boldsymbol{\Theta}$ that achieve small $\mathcal{L}_f(\mathcal{D}, \boldsymbol{\Theta})$, typically by iterating a variant of a minibatch stochastic gradient descent rule [21], [22]:

$$\boldsymbol{\Theta}^{t+1} \leftarrow \boldsymbol{\Theta}^t - \eta \nabla \mathcal{L}_f(\mathcal{D}^t, \boldsymbol{\Theta}^t)$$

where $\boldsymbol{\Theta}^t$ are the parameters after iteration t, $\mathcal{D}^t \subseteq \mathcal{D}$ is a randomly selected subset of the training examples, and η is a learning rate parameter.

In the Federated Learning setting, each user $u \in \mathcal{U}$ holds a private set $\mathcal{D}_u$ of training examples with $\mathcal{D} = \bigcup_{u \in \mathcal{U}} \mathcal{D}_u$. To run stochastic gradient descent, for each update we select a random subset of users $\mathcal{U}^t \subseteq \mathcal{U}$ (in practice we might have say $|\mathcal{U}^t| = 10^4$ while $|\mathcal{U}| = 10^7$) and for each user $u \in \mathcal{U}^t$ we select a random subset of that user's data $\mathcal{D}_u^t \subseteq \mathcal{D}_u$. We then form a (virtual) minibatch $\mathcal{D}^t = \bigcup_{u \in \mathcal{U}^t} \mathcal{D}_u^t$. The minibatch loss gradient $\nabla \mathcal{L}_f(\mathcal{D}^t, \boldsymbol{\Theta}^t)$ can be rewritten as a weighted average across users:

$$\nabla \mathcal{L}_f(\mathcal{D}^t, \boldsymbol{\Theta}^t) = \frac{1}{|\mathcal{D}^t|} \sum_{u \in \mathcal{U}^t} \boldsymbol{\delta}_u^t$$

where $\boldsymbol{\delta}_u^t = |\mathcal{D}_u^t| \nabla \mathcal{L}_f(\mathcal{D}_u^t, \boldsymbol{\Theta}^t)$. A user can thus share just $\langle |\mathcal{D}_u^t|, \boldsymbol{\delta}_u^t \rangle$ with the server, from which a gradient descent step:

$$\boldsymbol{\Theta}^{t+1} \leftarrow \boldsymbol{\Theta}^t - \eta \frac{\sum_{u \in \mathcal{U}^t} \boldsymbol{\delta}_u^t}{\sum_{u \in \mathcal{U}^t} |\mathcal{D}_u^t|}$$

may be taken.

Although each update $\langle |\mathcal{D}_u^t|, \boldsymbol{\delta}_u^t \rangle$ is ephemeral and contains no more (and typically significantly less) information then the raw $\mathcal{D}_u^t$, a user might still wonder what information remains. There is evidence that a trained neural network's parameters sometimes allow reconstruction of training examples [9], [10], [11], [14]; might the parameter updates be subject to similar attacks? For example, if the input x is a one-hot vocabulary-length vector encoding the most recently typed word, common neural network architectures will contain at least one parameter θ_w in $\boldsymbol{\Theta}$ for each word w such that $\frac{\partial \mathcal{L}_f}{\partial \theta_w}$ is non-zero only when x encodes w. Thus, the set of recently typed words in $\mathcal{D}_u^t$ would be revealed by inspecting the non-zero entries of $\boldsymbol{\delta}_u^t$. However, the server does not need to access any *individual* user's update in order to perform stochastic gradient descent; it requires only the *sums* $\sum_{u \in \mathcal{U}^t} |\mathcal{D}_u^t|$ and $\sum_{u \in \mathcal{U}^t} \boldsymbol{\delta}_u^t$. Using a Secure Aggregation protocol to compute these sums would ensure that the server learns only that *one or more* users in $\mathcal{U}^t$ wrote the word w, but not *which* users.

Federated Learning systems face several practical challenges. Mobile devices have only sporadic access to power and network connectivity, so the set $\mathcal{U}^t$ participating in each update step is unpredictable and the system must be robust to users dropping

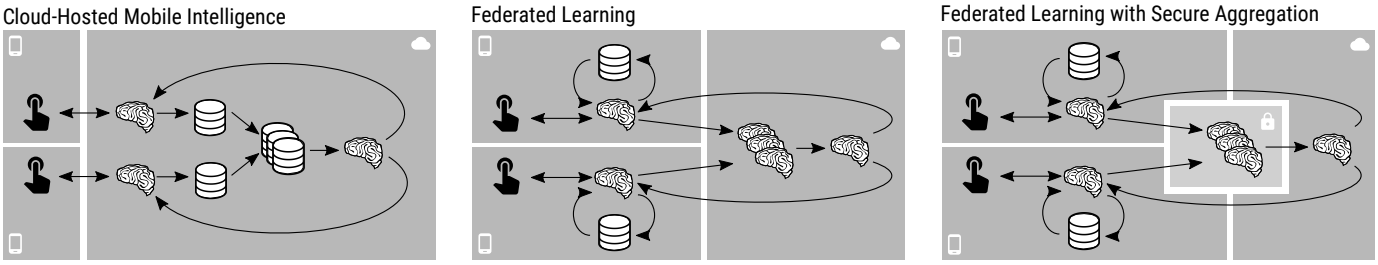


Fig. 1. **Left:** In the cloud-centric approach to machine intelligence, user devices interact with cloud-hosted models generating logs that can be used as potential training examples; the logs from many users are combined and used to improve the model, which is then used to serve future requests from users. **Middle:** In Federated Learning, machine intelligence models are shipped to users' devices where they are evaluated locally. Potential training examples are written to local storage and used to improve the model on the basis of local data. Summaries of improved models are shared with the server, where they are aggregated into a new model which is then deployed to user devices to power future interactions. In practice, it is necessary to perform many iterations of Federated Learning in sequence in order for training to converge to an optimal model; each of these iterations will be short-lived and involve a different randomly selected subset of participating user devices. **Right:** When Secure Aggregation is added to Federated Learning, the aggregation of model updates is logically performed by the virtual, incorruptible third party induced by the secure multiparty communication protocol. Otherwise the protocol remains the same. The cloud provider learns only the aggregated model update.

out. Because Θ may contain millions of parameters, updates $\boldsymbol{\delta}_u^t$ may be large, representing a direct cost to users on metered network plans. Mobile devices also generally cannot establish direct communications channels with other mobile devices (relying on a server or service provider to mediate such communication) nor can they natively authenticate other mobile devices.

Thus, Federated Learning motivates a need for a Secure Aggregation protocol that:

1) operates on high-dimensional vectors,
2) is highly communication efficient, even with a novel set of users on each instantiation,
3) is robust to users dropping out, and
4) provides the strongest possible security under the constraints of a server-mediated, unauthenticated network model.

III. Technical Intuition

In this section, we will give the technical intuition for the design of our protocol, in keeping with the contraints of the application described in Section II. We note that our protocol is quite similar the work of Ács and Castelluccia [15], and we give a detailed comparison between our approaches in Section IV.

We divide the parties into two classes: a single server S that *aggregates* inputs from n client parties $\mathcal{U}$. Each user[2] $u \in \mathcal{U}$ holds a private vector $\boldsymbol{x}_u$ of dimension m; for simplicity we assume the elements are in $\mathbb{Z}_R$. The goal of the protocol is to compute $\sum_{u\in\mathcal{U}} \boldsymbol{x}_u$ in a secure fashion: at a high level, we guarantee the server only learns a sum of the clients' inputs containing contributions from at least a large fraction of the users and that users learn nothing.

1) Masking with One-Time Pads: The first observation is that $\sum_{u\in\mathcal{U}} \boldsymbol{x}_u$ can be computed with perfect secrecy if $\boldsymbol{x}_u$ is masked in a particular way. Suppose each pair of users (u, v) agree on some random vector $\boldsymbol{s}_{u,v}$. If u adds this to $\boldsymbol{x}_u$ and v subtracts it from $\boldsymbol{x}_v$, then the mask will be canceled when their vectors are added, but their actual inputs will not be revealed. In other words, each user u computes:

$$\boldsymbol{y}_u = \boldsymbol{x}_u + \sum_{v\in\mathcal{U}:u<v} \boldsymbol{s}_{u,v} - \sum_{v\in\mathcal{U}:u>v} \boldsymbol{s}_{v,u} \pmod R$$

and sends $\boldsymbol{y}_u$ to the server, and the server computes:

$$\begin{aligned}\boldsymbol{z} &= \sum_{u\in\mathcal{U}} \boldsymbol{y}_u \\ &= \sum_{u\in\mathcal{U}} \left(\boldsymbol{x}_u + \sum_{v\in\mathcal{U}:u<v} \boldsymbol{s}_{u,v} - \sum_{v\in\mathcal{U}:u>v} \boldsymbol{s}_{v,u} \right) \\ &= \sum_{u\in\mathcal{U}} \boldsymbol{x}_u \pmod R\end{aligned}$$

There are two shortcomings to this approach. The first is that the users must exchange the random vectors $\boldsymbol{s}_{u,v}$, which, if done naively, would require quadratic communication overhead ($|\mathcal{U}| \times |\boldsymbol{x}|$). The second is that there is no tolerance for a party failing to complete the protocol: if a user u drops out after exchanging vectors with other users, but before submitting $\boldsymbol{y}_u$ to the server, the vector masks associated with u would not be canceled in the sum $\boldsymbol{z}$.

2) Efficient Communication and Handling Dropped Users: We notice that we can reduce the communication complexity by having the parties agree on common seeds for a pseudorandom generator (PRG) rather than on the entire mask $\boldsymbol{s}_{v,u}$. These shared seeds will be computed by having the parties exchange Diffie-Hellman public keys and engaging in a key agreement.

One approach to handling dropped-out users would be to notify the surviving users of the drop-out, and to have them each reply with the common seed they computed with the dropped user. This approach still has a problem: additional users may drop out in the recovery phase before replying with the seeds, which would thus require an additional recovery phase for the newly dropped users' seeds to be reported, and so on, leading the number of rounds up to at most the number of users.

[2]We use the terms user and client interchangeably.

We resolve this problem by using a threshold secret sharing scheme and having each user send shares of their Diffie-Hellman secret to all other users. This allows pairwise seeds to be recovered even if additional parties drop out during the recovery, as long as some minimum number of parties (equal to the threshold) remain alive and respond with the shares of the dropped users' keys.

This approach solves the problem of unbounded recovery rounds, but still has an issue: there is a possibility that a user's data might accidentally be leaked to the server. Consider a scenario where a user u is too slow in sending its $\boldsymbol{y}_u$ to the server. The server assumes that the user has dropped, and asks all other users to reveal their shares of u's secret key, in order to remove u's uncancelled masks from $\boldsymbol{z}$. However, just after receiving these shares and computing each of the $\boldsymbol{s}_{u,v}$ values, the server may receive the delayed $\boldsymbol{y}_u$ from u. The server is now able to remove all the masks from $\boldsymbol{y}_u$, and learn $\boldsymbol{x}_u$ in the clear, breaking security for u. Moreover, a malicious server can similarly learn $\boldsymbol{x}_u$ simply by lying about whether user u has dropped out.

3) Double-Masking to Protect Security: To resolve this new security problem, we introduce a double-masking structure that protects $\boldsymbol{x}_u$ even when the server can reconstruct u's masks.

First, each user u samples an additional random value $\boldsymbol{b}_u$ during the same round as the generation of the $\boldsymbol{s}_{u,v}$ values. During the secret sharing round, the user also generates and distributes shares of $\boldsymbol{b}_u$ to each of the other users. When generating $\boldsymbol{y}_u$, users also add this secondary mask:

$$\boldsymbol{y}_u = \boldsymbol{x}_u + \boldsymbol{b}_u + \sum_{v \in \mathcal{U}: u<v} \boldsymbol{s}_{u,v} - \sum_{v \in \mathcal{U}: u>v} \boldsymbol{s}_{v,u} \pmod R$$

During the recovery round, the server must make an explicit choice with respect to each user u: from each surviving member v, the server can request *either* a share of the common secret $\boldsymbol{s}_{u,v}$ associated with u *or* a share of the $\boldsymbol{b}_u$ for u; an honest user v will never reveal both kinds of shares for the same user. After gathering at least t shares of $\boldsymbol{s}_{u,v}$ for all dropped users and t shares of $\boldsymbol{b}_u$ for all surviving users, the server can subtract off the remaining masks to reveal the sum.

4) Putting it all Together: We summarize our protocol in Figure 2 and its asymptotic costs in Figure 3. The computational cost is quadratic for the users, and cubic for the server. As the size of the data vector gets large, the communication and storage overhead for each of the clients and the server using our protocol approaches a multiplicative constant over sending the data in the clear.

IV. Related work

In this section, we review the many existing works in the field and briefly dicuss how they compare to our work.

As noted in Section II, we emphasize that our focus is on mobile devices, where bandwidth is expensive, and dropouts are common. Consequently, our main goal is to minimize communication while guaranteeing robustness to dropouts. Computational cost is an important, but secondary, concern. These constraints will motivate our discussion of, and comparison with, existing works.

A. Works based on Generic Secure Multiparty Computation:

As noted in Section I, there is a long line of work showing how multiple parties can securely compute any function using generic secure MPC [23], [24], [25], [26], [27]. These works generally fall into two categories: those based on Yao's garbled circuits, and those based on homomorphic secret sharing. The protocols based on Yao's garbled circuits are better suited to 2- or 3-party secure computation, and do not extend easily to hundreds of users.

The MPC protocols based on secret sharing, however, easily extend to hundreds of users. In addition, these protocols have become relatively computationally efficient, and can be made easily robust against dropouts. Some works, notably [28], optimize these generic techniques for the specific task of secure summation, and have publicly available implementations.

However, the key weakness of the secret-sharing based generic MPC approach is communication cost. In all such protocols, each user broadcasts a secret-share of its entire data vector to some subset of the other users. To guarantee robustness, this subset of users must be relatively large: robustness is essentially proportional to the size of the subset. Additionally, each secret share is as long the size of the entire data vector.

This approach thus becomes prohibitively expensive in our setting, where dropouts are common and data vectors are very large. We defer a detailed cost comparison to an upcoming version of this paper.

B. Works based on Dining Cryptographers Networks:

Dining cryptographers networks, or DC-nets, are a type of communication network which provide anonymity by using pairwise blinding of inputs [29], [30], similarly to our secure aggregation protocol. The basic version of DC-nets, in which a single participant at a time sends an anonymous message, can be viewed as the restricted case of secure aggregation in which all users except for one have an input of 0. Recent research has examined increasing the efficiency of DC-nets protocols and allowing them to operate in the presence of malicious users [31]. But previous DC-nets constructions share the flaw that, if even one user aborts the protocol before sending its message, the protocol must be restarted from scratch, which can be very expensive [32].

C. Works based on Pairwise Additive Masking:

Pairwise blinding using additive stream ciphers has been explored in previous work [15], [18], [33], [34], and deal with dropouts/ failures in different ways.

[33], [34] rely on multiple non-colluding servers to provide robustness to client failures, but these schemes still abort in the case of a single server failure.

The work of Ács and Castelluccia [15], and the modification suggested by [18], are the most closely related to our scheme,

[3]We reconstruct n secrets from aligned (t, n)-Shamir shares in $O(t^2 + nt)$ by caching Lagrange coefficients; see section VII-B for details.

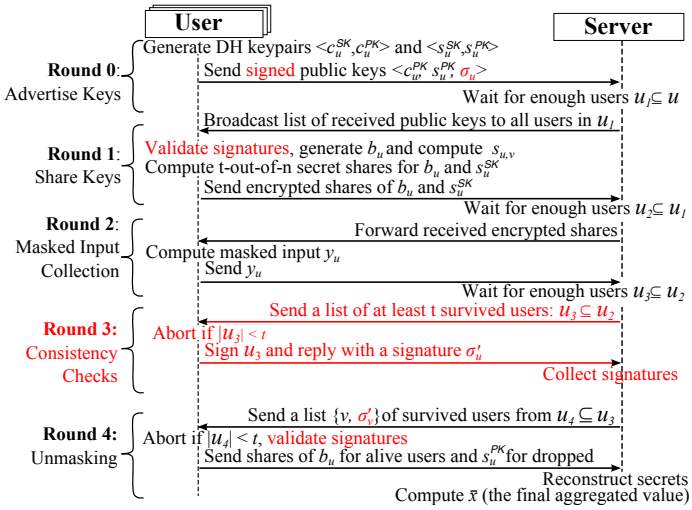


Fig. 2. High-level view of our protocol. Red parts are required to guarantee security in the fully malicious model (and not necessary in the honest but curious one).

computation	
User	$O(n^2 + mn)$
Server[3]	$O(mn^2)$
communication	
User	$O(n + m)$
Server	$O(n^2 + mn)$
storage	
User	$O(n + m)$
Server	$O(n^2 + m)$

Fig. 3. Cost summary for the protocol.

and have an explicit recovery round to deal with failures. Their protocols operate very similarly to ours: pairs of clients use Diffie-Hellman key exchange to agree on pairwise masks, and send the server their data vectors, summed with each of their pairwise masks and also a "self-mask". In the recovery step, the server tells all un-dropped clients which other clients dropped out, and each un-dropped client responds with the sum of their (uncancelled) pairwise masks with the dropped users, added to their "self-mask". The server subtracts these "recovery" values from the masked vectors received earlier, and correctly learns the sum of the undropped users' data.

However, their recovery phase is brittle: if additional users drop out during the recovery phase, the protocol cannot continue. Simply repeating the recovery round is not sufficient, since this has the potential to leak the "self-masks" of the surviving users, which in turn can leak their data vectors. Furthermore, the recovery round takes a significant portion of the protocol running time. This is because the clients send the sum of *expanded* masks, which are as long as the plaintext data vector. This means that almost half of the communication of the entire protocol occurs during the recovery round. In the volatile setting of mobile devices that can lose power or network, this leads to a significant risk of dropouts during that phase.

Adding a more robust recovery phase is essential for the mobile setting, and we view this as one of our key improvements over their works.

D. Schemes based on (Threshold) Homomorphic Encryption

Schemes based on threshold additively-homomorphic cryptosystems (e.g. the Paillier cryptosystem [35]) can handle client dropouts, but are either computationally expensive, or require additional trust assumptions. For example, Paillier-based schemes require an expensive-to-generate set of threshold decryption keys, that must either be generated and distributed by a trusted third party who then has the ability to break the entire security of the protocol, or be obliviously generated collaboratively by the parties in the protocol, which is very expensive.

The difficulty with threshold key-generation likewise affects schemes based on Fully Homomorphic Encryption.

Halevi, Lindell and Pinkas [36] present a protocol that uses homomorphic encryption to securely compute the sum in just one round of interaction between the server and each of the clients (assuming a PKI is already in place). Their protocol has the substantial advantage that all parties do not need to be online simultaneously for the protocol to execute. However, the protocol also requires the communication rounds to be carried out sequentially (i.e. messages cannot be sent in parallel, as the server needs to wait for each client's response before he can send the next message to the following client). More importantly for our setting, their protocol does not deal with clients dropping out: all clients included in the protocol must respond before the server can learn the decrypted sum.

V. A Practical Secure Aggregation Protocol

The protocol is run (in a synchronous network) between a server and a set of n users, and consists of four rounds. Each user u holds as input a vector $\boldsymbol{x}_u$ (of equal length m) consisting of elements from $\mathbb{Z}_R$ for some R. The server has no input, but can communicate with the users through secure (private and authenticated) channels. At any point, users can drop out of the protocol (in which case they stop sending messages completely), and the server will be able to produce a correct output as long as t of them survive until the last round. To simplify the notation we assume that each user u is assigned a unique "logical identity" (also denoted with u) in the form of an integer between 1 and n, so that no two honest

users share the same index[4].

A complete description is provided in Figure 4. We assume the following cryptographic primitives (detailed descriptions can be found in the appendix):

- *Threshold Secret Sharing*: (**SS.share**, **SS.reconstruct**)
- *Key Agreement*: (**KA.paramGen**, **KA.gen**, **KA.agree**)
- *Authenticated Encryption*: (**AE.enc**, **AE.dec**)
- *Pseudorandom Generator*: (**PRG**)
- *Signature Scheme*: (**SIG.gen**, **SIG.sign**, **SIG.verify**)

We stress that, in Figure 4, when we say that the server "collects messages from *at least* t users", we mean that the server receives the messages from all users that have not dropped out/aborted in that round (recall that we prove our results in the synchronous setting), and aborts if the number of messages received is less than t. In a practical implementation, the server would wait until a specified timeout (considering all users who did not respond in time to have dropped out), and abort itself if not enough messages are received before such timeout.

To prove security in the malicious model, we also assume the existence of a Public Key Infrastructure, which for simplicity we abstract away by assuming all clients receive as input (from a trusted third party) public signing keys for all other clients.

Overall, the protocol is parameterized over a security parameter k, which can be adjusted to bound the success probability of any attacker. In all theorems, we implicitly assume that the number of clients n is polynomially bounded in the security parameter. Moreover, some of the primitives also require additional global parameters.

We note that Figure 4 presents both variants of the protocol: in the honest but curious case, since all parties are following the protocol honestly, we can avoid the use of signatures and the need for a PKI (which, most notably, allows us to avoid the **ConsistencyCheck** round entirely).

VI. Security Analysis

In our security arguments, we will make use of an important technical lemma (Lemma 1, presented in the Appendix). It says that if users' values have uniformly random pairwise masks added to them, then the resulting values look uniformly random, conditioned on their sum being equal to the sum of the users' values. In other words, the pairwise masks hide all information about users' individual inputs, except for their sum.

A. Honest but Curious Security

Here, we argue that our protocol is a secure multiparty computation in the honest but curious setting, regardless of how and when parties abort. In particular, we prove that when executing the protocol with threshold t, the joint view of the server and any set of less than t (honest) users does not leak any information about the other users' inputs, besides what can be inferred from the output of the computation. Before formally stating our result, we introduce some notation.

[4]These identities will be bound to the users' keys by a PKI. We rely on this in the malicious setting.

We will consider executions of our secure aggregation protocol where the underlying cryptographic primitives are instantiated with security parameter k, a server S interacts with a set $\mathcal{U}$ of n users (denoted with logical identities $1, \ldots, n$) and the threshold is set to t. In such executions, users might abort at any point during the execution, and we denote with $\mathcal{U} \supseteq \mathcal{U}_1 \supseteq \mathcal{U}_2 \supseteq \mathcal{U}_3 \supseteq \mathcal{U}_4 \supseteq \mathcal{U}_5$ subsets of the users that correctly sent their message to the server at round $i-1$. For example, users in $\mathcal{U}_2 \setminus \mathcal{U}_3$ are exactly those that abort before sending the message to the server in Round 2, but after sending the message of Round 1. If Round **ConsistencyCheck** has been omitted, define $\mathcal{U}_4 := \mathcal{U}_3$.

Denote the input of each user u with $\boldsymbol{x}_u$, and with $\boldsymbol{x}_{\mathcal{U}'} = \{\boldsymbol{x}_u\}_{u \in \mathcal{U}'}$ the inputs of all users in $\mathcal{U}'$.

In such a protocol execution, the *view* of a party consists of its internal state (including its input and randomness) and all messages this party received from other parties (the messages sent by this party do not need to be part of the view because they can be determined using the other elements of its view). Moreover, if the party aborts, it stops receiving messages and the view is not extended past the last message received.

Given any subset $\mathcal{C} \subseteq \mathcal{U} \cup \{S\}$ of the parties, let $\texttt{REAL}_{\mathcal{C}}^{\mathcal{U},t,k}(\boldsymbol{x}_{\mathcal{U}}, \mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3, \mathcal{U}_4, \mathcal{U}_5)$ be a random variable representing the combined views of all parties in $\mathcal{C}$ in the above protocol execution, where the randomness is over the internal randomness of all parties, and the randomness in the setup phase.

Our first theorem shows that the joint view of any subset of honest users (excluding the server) can be simulated given only the knowledge of the inputs of those users. Intuitively, this means that those users learn "nothing more" than their own inputs.

Theorem 1 (Honest But Curious Security, against clients only)**.** *There exists a PPT simulator* $\texttt{SIM}$ *such that for all* $k, t, \mathcal{U}$ *with* $t \leq |\mathcal{U}|, \boldsymbol{x}_{\mathcal{U}}, \mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3, \mathcal{U}_4, \mathcal{U}_5$ *and* $\mathcal{C}$ *such that* $\mathcal{C} \subseteq \mathcal{U}$*,* $\mathcal{U} \supseteq \mathcal{U}_1 \supseteq \mathcal{U}_2 \supseteq \mathcal{U}_3 \supseteq \mathcal{U}_4 \supseteq \mathcal{U}_5$*, the output of* $\texttt{SIM}$ *is perfectly indistinguishable from the output of* $\texttt{REAL}_{\mathcal{C}}^{\mathcal{U},t,k}$*:*

$$\texttt{REAL}_{\mathcal{C}}^{\mathcal{U},t,k}(\boldsymbol{x}_{\mathcal{U}}, \mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3, \mathcal{U}_4, \mathcal{U}_5) \equiv \texttt{SIM}_{\mathcal{C}}^{\mathcal{U},t,k}(\boldsymbol{x}_{\mathcal{C}}, \mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3, \mathcal{U}_4, \mathcal{U}_5)$$

Proof. Note that, since the view of the server is omitted, the joint view of the parties in $\mathcal{C}$ does not depend (in an information theoretic sense) on the inputs of the parties not in $\mathcal{C}$. The simulator can therefore produce a perfect simulation by running the honest but curious users on their true inputs, and all other users on a dummy input (for example, a vector of 0s), and outputting the simulated view of the users in $\mathcal{C}$. In more detail, the only value sent by the honest parties which depend on their input is $\boldsymbol{y}_u$ (sent to the server in round **MaskedInputCollection**). One can easily note that the response sent by the server to the users in round **MaskedInputCollection** just contains a list of user identities which depends on which users responded on the previous round, but not on the specific $\boldsymbol{y}_u$ values of the responses. This means

Secure Aggregation Protocol

- **Setup**:
  - All parties are given the security parameter k, the number of users n and a threshold value t, honestly generated $pp \leftarrow \textbf{KA.gen}(k)$, parameters m and R such that $\mathbb{Z}_R^m$ is the space from which inputs are sampled, and a field $\mathbb{F}$ to be used for secret sharing. All users also have a private authenticated channel with the server.
  - All users u receive their signing key d_u^{SK} from the trusted third party, together with verification keys d_v^{PK} bound to each user identity v.
- **Round** 0 **(AdvertiseKeys)**:
  User u:
  - Generate key pairs $(c_u^{PK}, c_u^{SK}) \leftarrow \textbf{KA.gen}(pp)$, $(s_u^{PK}, s_u^{SK}) \leftarrow \textbf{KA.gen}(pp)$, and generate $\sigma_u \leftarrow \textbf{SIG.sign}(d_u^{SK}, c_u^{PK}||s_u^{PK})$.
  - Send $(c_u^{PK}||s_u^{PK}||\sigma_u)$ to the server (through the private authenticated channel) and move to next round.

  Server:
  - Collect at least t messages from individual users in the previous round (denote with $\mathcal{U}_1$ this set of users). Otherwise, abort.
  - Broadcast to all users in $\mathcal{U}_1$ the list $\{(v, c_v^{PK}, s_v^{PK}, \sigma_v)\}_{v \in \mathcal{U}_1}$ and move to next round.
- **Round** 1 **(ShareKeys)**:
  User u:
  - Receive the list $\{(v, c_v^{PK}, s_v^{PK}, \sigma_v)\}_{v \in \mathcal{U}_1}$ broadcasted by the server. Assert that $|\mathcal{U}_1| \geq t$, that all the public key pairs are different, and that $\forall v \in \mathcal{U}_1$, $\textbf{SIG.verify}(d_v^{PK}, c_v^{PK}||s_v^{PK}, \sigma_u) = 1$.
  - Sample a random element $b_u \leftarrow \mathbb{F}$ (to be used as a seed for a **PRG**).
  - Generate t-out-of-$|\mathcal{U}_1|$ shares of s_u^{SK}: $\{(v, s_{u,v}^{SK})\}_{v \in \mathcal{U}_1} \leftarrow \textbf{SS.share}(s_u^{SK}, t, \mathcal{U}_1)$
  - Generate t-out-of-$|\mathcal{U}_1|$ shares of b_u: $\{(v, b_{u,v})\}_{v \in \mathcal{U}_1} \leftarrow \textbf{SS.share}(b_u, t, \mathcal{U}_1)$
  - For each other user $v \in \mathcal{U}_1 \setminus \{u\}$, compute $e_{u,v} \leftarrow \textbf{AE.enc}(\textbf{KA.agree}(c_u^{SK}, c_v^{PK}), u||v||s_{u,v}^{SK}||b_{u,v})$
  - If any of the above operations (assertion, signature verification, key agreement, encryption) fails, abort.
  - Send all the ciphertexts $e_{u,v}$ to the server (each implicitly containing addressing information u, v as metadata).
  - Store all messages received and values generated in this round, and move to the next round.

  Server:
  - Collect lists of ciphertexts from at least t users (denote with $\mathcal{U}_2 \subseteq \mathcal{U}_1$ this set of users).
  - Sends to each user $u \in \mathcal{U}_2$ all ciphertexts encrypted for it: $\{e_{u,v}\}_{v \in \mathcal{U}_2}$ and move to the next round.
- **Round** 2 **(MaskedInputCollection)**:
  User u:
  - Receive (and store) from the server the list of ciphertexts $\{e_{u,v}\}_{v \in \mathcal{U}_2}$ (and infer the set $\mathcal{U}_2$). If the list is of size $< t$, abort.
  - For each other user $v \in \mathcal{U}_2 \setminus \{u\}$, compute $s_{u,v} \leftarrow \textbf{KA.agree}(s_u^{SK}, s_v^{PK})$ and expand this value using a **PRG** into a random vector $\boldsymbol{p}_{u,v} = \Delta_{u,v} \cdot \textbf{PRG}(s_{u,v})$, where $\Delta_{u,v} = 1$ when $u > v$, and $\Delta_{u,v} = -1$ when $u < v$ (note that $\boldsymbol{p}_{u,v} + \boldsymbol{p}_{v,u} = 0 \; \forall u \neq v$). Additionally, define $\boldsymbol{p}_{u,u} = 0$.
  - Compute the user's own private perturbation vector $\boldsymbol{p}_u = \textbf{PRG}(b_u)$. Then, Compute the masked input vector

$$\boldsymbol{y}_u \leftarrow \boldsymbol{x}_u + \boldsymbol{p}_u + \sum_{v \in \mathcal{U}_2} \boldsymbol{p}_{u,v} \pmod R$$

  - If any of the above operations (key agreement, PRG) fails, abort. Otherwise, Send $\boldsymbol{y}_u$ to the server and move to the next round.

  Server:
  - Collect $\boldsymbol{y}_u$ from at least t users (denote with $\mathcal{U}_3 \subseteq \mathcal{U}_2$ this set of users). Send to each user in $\mathcal{U}_3$ the list $\mathcal{U}_3$.
- **Round** 3 **(ConsistencyCheck)**:
  User u:
  - Receive from the server a list $\mathcal{U}_3 \subseteq \mathcal{U}_2$ consisting of at least t users (including itself). If $\mathcal{U}_3$ is smaller than t, abort.
  - Send to the server $\sigma'_u \leftarrow \textbf{SIG.sign}(d_u^{SK}, \mathcal{U}_3)$.

  Server:
  - Collect σ'_u from at least t users (denote with $\mathcal{U}_4 \subseteq \mathcal{U}_3$ this set of users). Send to each user in $\mathcal{U}_4$ the set $\{v, \sigma'_v\}_{v \in \mathcal{U}_4}$.
- **Round** 4 **(Unmasking)**:
  User u:
  - Receive from the server a list $\{v, \sigma'_v\}_{v \in \mathcal{U}_4}$. Verify that $\mathcal{U}_4 \subseteq \mathcal{U}_3$, that $|\mathcal{U}_4| \geq t$ and that $\textbf{SIG.verify}(d^{PK}, \mathcal{U}_3, \sigma'_v) = 1$ for all $v \in \mathcal{U}_4$ (otherwise abort).
  - For each other user v in $\mathcal{U}_2 \setminus \{u\}$, decrypt the ciphertext $v'||u'||s_{v,u}^{SK}||b_{v,u} \leftarrow \textbf{AE.dec}(\textbf{KA.agree}(c_u^{SK}, c_v^{PK}), e_{v,u})$ received in the **MaskedInputCollection** round and assert that $u = u' \wedge v = v'$.
  - If any of the decryption operations fail (in particular, the ciphertext does not correctly authenticate), abort.
  - Send a list of shares to the server, which consists of $s_{v,u}^{SK}$ for users $v \in \mathcal{U}_2 \setminus \mathcal{U}_3$ and $b_{v,u}$ for users in $v \in \mathcal{U}_3$.

  Server (generating the output):
  - Collect responses from at least t users (denote with $\mathcal{U}_5$ this set of users).
  - For each user in $u \in \mathcal{U}_2 \setminus \mathcal{U}_3$, reconstruct $s_u^{SK} \leftarrow \textbf{SS.reconstruct}(\{s_{u,v}^{SK}\}_{v \in \mathcal{U}_5}, t)$ and use it (together with the public keys received in the **AdvertiseKeys** round) to recompute $\boldsymbol{p}_{v,u}$ for all $v \in \mathcal{U}_3$ using the PRG.
  - For each user $u \in \mathcal{U}_3$, reconstruct $b_u \leftarrow \textbf{SS.reconstruct}(\{b_{u,v}\}_{v \in \mathcal{U}_5}, t)$ and then recompute $\boldsymbol{p}_u$ using the PRG.
  - Compute and output $\boldsymbol{z} = \sum_{u \in \mathcal{U}_3} \boldsymbol{x}_u$ as

$$\sum_{u \in \mathcal{U}_3} \boldsymbol{x}_u = \sum_{u \in \mathcal{U}_3} \boldsymbol{y}_u - \sum_{u \in \mathcal{U}_3} \boldsymbol{p}_u + \sum_{u \in \mathcal{U}_3, v \in \mathcal{U}_2 \setminus \mathcal{U}_3} \boldsymbol{p}_{v,u}$$

Fig. 4. Detailed description of the Secure Aggregation protocol. Red parts are required to guarantee security in the fully malicious model (and not necessary in the honest but curious one).

that the simulator can use dummy values for the inputs of all honest parties not in $\mathcal{C}$, and the joint view of users in $\mathcal{C}$ will be identical to that in $\texttt{REAL}^{\mathcal{U},t,k}$. □

In our next theorem, we consider security against an honest-but-curious server, who can additionally combine knowledge with some honest-but-curious clients. We show that any such a group of honest-but-curious parties can be simulated given the inputs of the clients in that group, and only the *sum* of the values of the remaining clients. Intuitively, this means that those clients and the server learn "nothing more" than their own inputs, and the sum of the inputs of the other clients. Additionally, if too many clients abort before Round **Unmasking**, then we show that we can simulate the view of the honest-but-curious parties given *no information* about the remaining clients' values. Thus, in this case, the honest-but-curious parties learn *nothing* about the remaining clients' values.

Importantly, the view to be simulated must contain fewer than t honest-but-curious clients, or else we cannot guarantee security.

Theorem 2 (Honest But Curious Security, with curious server)**.** *There exists a PPT simulator* SIM *such that for all* $t,\mathcal{U},\boldsymbol{x}_{\mathcal{U}},\mathcal{U}_1,\mathcal{U}_2,\mathcal{U}_3,\mathcal{U}_4$*, and* $\mathcal{C}$ *such that* $\mathcal{C} \subseteq \mathcal{U} \cup \{S\}$*,* $|\mathcal{C} \setminus \{S\}| < t$*,* $\mathcal{U} \supseteq \mathcal{U}_1 \supseteq \mathcal{U}_2 \supseteq \mathcal{U}_3 \supseteq \mathcal{U}_4 \supseteq \mathcal{U}_5$*, the output of* SIM *is computationally indistinguishable from the output of* $\texttt{REAL}_{\mathcal{C}}^{\mathcal{U},t,k}$*:*

$$\texttt{REAL}_{\mathcal{C}}^{\mathcal{U},t,k}(\boldsymbol{x}_{\mathcal{U}},\mathcal{U}_1,\mathcal{U}_2,\mathcal{U}_3,\mathcal{U}_4,\mathcal{U}_5) \approx_c \texttt{SIM}_{\mathcal{C}}^{\mathcal{U},t,k}(\boldsymbol{x}_{\mathcal{C}},\boldsymbol{z},\mathcal{U}_1,\mathcal{U}_2,\mathcal{U}_3,\mathcal{U}_4,\mathcal{U}_5)$$

where

$$\boldsymbol{z} = \begin{cases} \sum_{u\in\mathcal{U}_3\setminus\mathcal{C}} \boldsymbol{x}_u & \text{if } |\mathcal{U}_3| \geq t \\ \perp & \text{otherwise.} \end{cases}$$

Proof. We prove the theorem by a standard hybrid argument. We will define a simulator SIM through a series of (polynomially many) subsequent modifications to the random variable REAL, so that any two subsequent random variables are computationally indistinguishable.

- Hyb_0 This random variable is distributed exactly as REAL, the joint view of the parties $\mathcal{C}$ in a real execution of the protocol.
- Hyb_1 In this hybrid, we change the behavior of simulated honest parties in the set $\mathcal{U}_2 \setminus \mathcal{C}$, so that instead of using **KA.agree**(c_u^{SK}, c_v^{PK}) to encrypt and decrypt messages to other users v in the same set, they use a uniformly random encryption key $c_{u,v}$ chosen by the simulator. The Decisional Diffie-Hellman assumption (as recalled in Definition 1) guarantees that this hybrid is indistinguishable from the previous one.
- Hyb_2 In this hybrid, we substitute all ciphertexts encrypted by honest parties in the set $\mathcal{U}_2 \setminus \mathcal{C}$ and sent to other honest parties with encryptions of 0 (padded to the appropriate length) instead of shares of s_u^{SK} and b_u. However, the honest clients in that set continue to respond with the correct shares of s_u^{SK} and b_u in Round **Unmasking**. Since only the contents of the ciphertexts have changed, IND-CPA security of the encryption scheme guarantees that this hybrid is indistinguishable from the previous one.
- Hyb_3 Define:

$$\mathcal{U}^* = \begin{cases} \mathcal{U}_2 \setminus \mathcal{C} & \text{if } \boldsymbol{z} = \perp \\ \mathcal{U}_2 \setminus \mathcal{U}_3 \setminus \mathcal{C} & \text{otherwise.} \end{cases}$$

  This hybrid is distributed exactly as the previous one, but here we substitute all shares of b_u generated by parties $u \in \mathcal{U}^*$ and given to the corrupted parties in Round **ShareKeys** with shares of 0 (using a different sharing of 0 for every $u \in \mathcal{U}^*$). Note that, in this hybrid and the previous one, the adversary does not receive any additional shares of b_u for users u in the set $\mathcal{U}^*$ in Round **Unmasking**, either because the honest clients do not shares of b_u for such u, or because all honest clients abort (when $|\mathcal{U}_3| < t$, which happens exactly when $\boldsymbol{z} = \perp$). Thus, $M_{\mathcal{C}}$'s joint view contains only $|\mathcal{C}| < t$ shares of each b_u. The properties of Shamir's secret sharing thus guarantee that the distribution of any $|\mathcal{C}|$ shares of 0 is identical to the distribution of an equivalent number of shares of any given secret b_u, making this hybrid identically distributed to the previous one.
- Hyb_4 In this hybrid, for all parties $u \in \mathcal{U}^*$, instead of computing $\boldsymbol{p}_u \leftarrow$ **PRG**(b_u), we set it to be a uniformly random vector (of the appropriate size). Note that, in the previous hybrid, since b_u is chosen uniformly at random and its shares given to the adversary are substituted with shares of 0, the output of the random variable does not depend on the seed of the **PRG** except through the **PRG**'s output. Therefore, the only change in this hybrid boils down to substituting the output of a **PRG** (on a randomly generated seed otherwise independent from the joint view of parties in $\mathcal{C}$) with a uniformly random value. Therefore, leveraging the security of the **PRG**, we can argue that this hybrid is indistinguishable from the previous one.
- Hyb_5 For all parties $u \in \mathcal{U}^*$, in Round **MaskedInputCollection**, instead of sending:

$$\boldsymbol{y}_u \leftarrow \boldsymbol{x}_u + \boldsymbol{p}_u + \sum_{v\in\mathcal{U}_2} \boldsymbol{p}_{u,v}$$

  we send:

$$\boldsymbol{y}_u \leftarrow \boldsymbol{p}_u + \sum_{v\in\mathcal{U}_2} \boldsymbol{p}_{u,v}$$

  Since $\boldsymbol{p}_u$ was changed in the previous hybrid to be uniformly random and independent of any other values, $\boldsymbol{x}_u + \boldsymbol{p}_u$ is also uniformly random, and so this hybrid and the previous hybrid are identically distributed. Further, this hybrid and all subsequent hybrids do not depend on the values $\boldsymbol{x}_u$ for $u \in \mathcal{U}^*$. **Note:** If $\boldsymbol{z} = \perp$, then we can ignore the further hybrids, and let SIM be as described in Hyb_5, since

SIM can already simulate REAL without knowing $\boldsymbol{x}_u$ for any $u \notin \mathcal{C}$. Therefore in the following hybrids we assume $\boldsymbol{z} \neq \perp$.

Hyb_6 This random variable is distributed exactly as the previous one, but here we substitute all shares of s_u^{SK} generated by parties $u \in \mathcal{U}_3 \setminus \mathcal{C}$ and given to the corrupted parties in Round **ShareKeys** with shares of 0 (using a different sharing of 0 for every $u \in \mathcal{U}_3 \setminus \mathcal{C}$). Following an analogous argument to that for Hyb_3, the properties of Shamir's secret sharing guarantee that this hybrid identically distributed to the previous one.

Hyb_7 We fix a specific user $u' \in \mathcal{U}_3 \setminus \mathcal{C}$. For this user, and each other user $u \in \mathcal{U}_3 \setminus \mathcal{C}$, in order to compute the value $\boldsymbol{y}_u$ sent to the server, we substitute the joint noise key (which would be computed by u' and u as $s_{u',u} = s_{u,u'} \leftarrow \textbf{KA.agree}(s_{u'}^{SK}, s_u^{PK})$) with a uniformly random value (which will used by both parties as a **PRG** seed).

In more detail, for each user $u \in \mathcal{U}_3 \setminus \mathcal{C} \setminus \{u'\}$, a value $s'_{u',u}$ is sampled uniformly at random and, instead of sending

$$\boldsymbol{y}_u \leftarrow \boldsymbol{x}_u + \boldsymbol{p}_u + \sum_{v \in \mathcal{U}_2} \boldsymbol{p}_{u,v}$$

SIM sends

$$\boldsymbol{y}'_u \leftarrow \boldsymbol{x}_u + \boldsymbol{p}_u + \sum_{v \in \mathcal{U}_2 \setminus \{u'\}} \boldsymbol{p}_{u,v} + \Delta_{u,u'} \cdot \textbf{PRG}(s'_{u',u})$$

and accordingly

$$\boldsymbol{y}'_{u'} \leftarrow \boldsymbol{x}_{u'} + \boldsymbol{p}_{u'} + \sum_{v \in \mathcal{U}_2} \Delta_{u',v} \cdot \textbf{PRG}(s'_{u',v})$$

It is easy to see that the Decisional Diffie-Hellman Assumption (Definition 1) guarantees that this hybrid is indistinguishable from the previous one[5].

Hyb_8 In this hybrid, for the same party u' chosen in the previous hybrid and all other parties $v \in \mathcal{U}_3 \setminus \mathcal{C}$, instead of computing $\boldsymbol{p}_{u',v} \leftarrow \Delta_{u',v} \cdot \textbf{PRG}(s'_{u',v})$, we compute it using fresh randomness $r_{u',v}$ (of the appropriate size) as $\boldsymbol{p}_{u',v} \leftarrow \Delta_{u',v} \cdot r_{u',v}$.

Note that, in the previous hybrid, since $s'_{u',v}$ is chosen uniformly at random (and independently from the Diffie-Hellman keys), the output of the random variable does not depend on the seed of the **PRG** except through the **PRG**'s output. Therefore, the only change in this hybrid boils down to substituting the output of a **PRG** (on an randomly generated seed otherwise independent from the joint view of parties in $\mathcal{C}$) with a uniformly random value. Therefore, leveraging the security of the **PRG**, we can argue that this hybrid is indistinguishable from the previous one.

[5]It is important to note here that, in the previous hybrids, we removed all shares of s_u^{SK} for $u \in \mathcal{U}_3 \setminus \mathcal{C}$ from the joint view of parties in $\mathcal{C}$. Without doing so, we could not reduce to the security of DH Key Agreement.

Hyb_9 In this hybrid, for all users $u \in \mathcal{U}_3 \setminus \mathcal{C}$, in round **MaskedInputCollection** instead of sending:

$$\begin{aligned}\boldsymbol{y}_u &\leftarrow \boldsymbol{x}_u + \boldsymbol{p}_u + \sum_{v \in \mathcal{U}_2} \boldsymbol{p}_{u,v} \\ &= \boldsymbol{x}_u + \boldsymbol{p}_u + \sum_{v \in \mathcal{U}_3 \setminus \mathcal{C}} \boldsymbol{p}_{u,v} + \sum_{v \in \mathcal{U}_2 \setminus \mathcal{U}_3 \setminus \mathcal{C}} \boldsymbol{p}_{u,v}\end{aligned}$$

we send:

$$\boldsymbol{y}_u \leftarrow \boldsymbol{w}_u + \boldsymbol{p}_u + \sum_{v \in \mathcal{U}_2 \setminus \mathcal{U}_3 \setminus \mathcal{C}} \boldsymbol{p}_{u,v}$$

Where $\{\boldsymbol{w}_u\}_{u \in \mathcal{U}_3 \setminus \mathcal{C}}$ are uniformly random, subject to $\sum_{\mathcal{U}_3 \setminus \mathcal{C}} \boldsymbol{w}_u = \sum_{\mathcal{U}_3 \setminus \mathcal{C}} \boldsymbol{x}_u = \boldsymbol{z}$. Invoking Lemma 1 with $n = |\mathcal{U}_3 \setminus \mathcal{C}|$, we have that this hybrid is identically distributed to the previous one. Moreover, note that to sample from the random variable described by this hybrid, knowledge of the individual $\boldsymbol{x}_u$ for $u \in \mathcal{U}_3 \setminus \mathcal{C}$ is not needed, and their sum $\boldsymbol{z}$ is sufficient.

We can thus define a PPT simulator SIM that samples from the distribution described in the last hybrid. The argument above proves that the output of the simulator is computationally indistinguishable from the output of REAL, completing the proof. □

B. Privacy against Active Adversaries

In this section, we discuss our argument showing privacy against active adversaries (detailed proofs are available in Appendix C).

By active, or malicious, adversaries, we mean parties (clients or the server) that deviate from the protocol, sending incorrect and/or arbitrarily chosen messages to honest users, aborting, omitting messages, and sharing their entire view of the protocol with each other, and also with the server (if the server is also malicious).

In Appendix C, we show that even when the server and a subset of users act *maliciously*, colluding and deviating arbitrarily from the protocol, that privacy for the remaining honest users is preserved. That is, no party learns anything more than the sum of the inputs of a single subset of honest users of large size.

We note that we only show *input privacy* for honest users: it is much harder to additionally guarantee *correctness* and *availability* for the protocol when some users are malicious. Malicious users can distort the output of the protocol by setting their input values $\boldsymbol{x}_u$ to be uniformly random, by sending inconsistent Shamir shares to other users in Round **ShareKeys**, or by reporting incorrect shares to the server in Round **Unmasking**. Making such deviations efficient to detect and possibly recover from is left to future work.

We note some key differences between the argument for honest-but-curious security, and the argument for privacy against active adversaries.

The first key difference is that, for the proof against malicious adversaries, we assume that there exists a public-key infrastructure (PKI), which guarantees to users that messages

they receive came from other users (and not the server). Without this assumption, the server can perform a Sybil attack on the users in Round **ShareKeys**, by simulating for a specific user u all other users v in the protocol and thus receiving all u's key shares and recovering that users' input. Alternatively, as mentioned in section V, we can require the server to *act honestly in its first message* (in Round **ShareKeys**). Specifically, the server must honestly forward the Diffie-Hellman public keys it receives to all other users, allowing them to set up pairwise private and authenticated channels amongst themselves.

However, if we assume a PKI, then we observe that the server's power in the remainder of the protocol is reduced to lying to users about which other users have dropped out: since all user-to-user messages (sent in round **ShareKeys**) are authenticated through an authenticated encryption scheme, the server cannot add, modify or substitute messages, but rather, can only fail to deliver them. Note, importantly, that the server can try to give a different view to each user of which other users have dropped out of the protocol. In the worst case, this could allow the server to learn a different set of shares from each user in Round **Unmasking**, allowing it to potentially reconstruct more secrets than it should be allowed to. The **ConsistencyCheck** round is included in the protocol to deal with this issue. The inclusion of the **ConsistencyCheck** round is the second key difference with the honest-but-curious proof.

The final key difference is that we need the proof to be in the random oracle (RO) model. To see why, notice that honestly acting users essentially "commited" to their secrets and input by the end of the **MaskedInputCollection** round. However, the server can adaptively choose which users drop after the **MaskedInputCollection** round. This causes problems for a simulation proof, because the simulator doesn't know honest users' real inputs, and must use dummy information in the earlier rounds, thus "committing" itself to wrong values that are potentially easily detectable. The random oracle adds a trapdoor for the simulator to equivocate, so that even if it commits to dummy values in early rounds, it can reprogram the random oracle to make the dummy values indistinguishable from honest users' values. More details can be seen in the proof of Theorem 4 in Appendix C.

C. Interpretation of Results

We summarize our system for the different security models we consider in Figure 5.

Threat model	Minimum threshold	Minimum inputs in sum
Client-only adversary	1	t
Server-only adversary	$\lfloor \frac{n}{2} \rfloor + 1$	t
Clients-Server collusion	$\lfloor \frac{2n}{3} \rfloor + 1$	$t - n_{\mathcal{C}}$

Fig. 5. Parameterization for different threat models. "Minimum threshold" denotes the minimum value of t required for security in the given threat model. "Minimum inputs in the sum" denotes a lower bound on the number of users' values that are included in the sum learned by the server.

1) Security against only clients: In each of Theorems 1 and 3, we see that the joint view of any subset of clients, honest or malicious, can be simulated given *no information* about the values of the remaining clients. This means, no matter how we set our t parameter, clients on their own learn nothing about other clients.

2) Security against only the server: From Theorems 2 and 4, we see that if we set $n_{\mathcal{C}} = 0$, that is, there are no clients who cheat or collaborate with the server, then setting $t \geq \lfloor \frac{n}{2} \rfloor + 1$ guarantees that the sum learned by the server contains the values of at least $t > \frac{n}{2}$ clients, and the protocol can deal with up to $\lceil \frac{n}{2} \rceil - 1$ dropouts.

3) Security against a server colluding with clients: From Theorems 2 and 4, we see that we can allow a server (honest or malicious) to collaborate with up to $n_{\mathcal{C}} = \lceil \frac{n}{3} \rceil - 1$ users (honest or malicious), if we set $t \geq \lfloor \frac{2n}{3} \rfloor + 1$, at the same time guaranteeing that the sum learned by the server contains the values of at least $\frac{n}{3}$ clients. Additionally, the protocol is robust to up to $\lceil \frac{n}{3} \rceil - 1$ users dropping out.

For all the results above, if we reiterate that if we want security against malicious servers (whether or not they collaborate with clients), we must use include the protocol features highlighted in Figure 4.

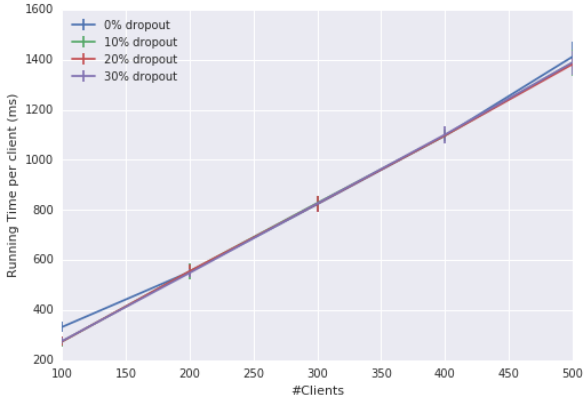
VII. Evaluation

We summarize the protocol's performance in Table 3. All calculations below assume a single server and n users, where each user holds a data vector of size m. We evaluate the honest-but-curious version of the protocol, and ignore the cost of the PKI, all signatures, and Round **ConsistencyCheck**. We note that including their cost does not change any of the asymptotics, and only slightly increases the computation and communication costs.

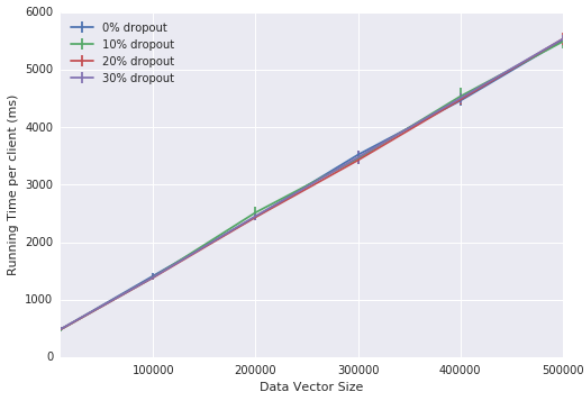
A. Performance Analysis of Client

Computation cost: $O(n^2+mn)$**.** Each user u's computation cost can be broken up as (1) Performing the $2n$ key agreements, which take $O(n)$ time, (2) Creating t-out-of-n Shamir secret shares of s_u^{SK} and b_u, which is $O(n^2)$ and (3) Generating values $\boldsymbol{p}_u$ and $\boldsymbol{p}_{u,v}$ for every other user v for each entry in the input vector by stretching one **PRG** seed each, which takes $O(mn)$ time in total. Overall, each user's computation is $O(n^2 + mn)$.

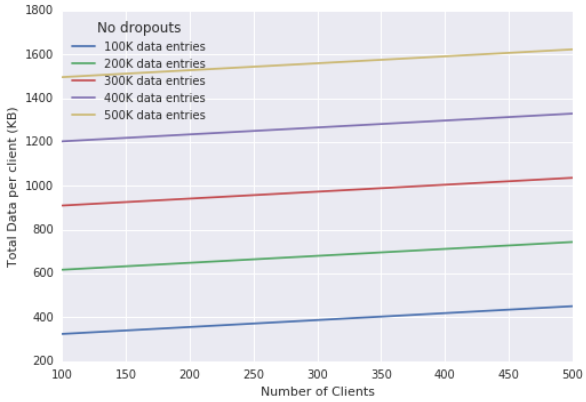
Communication cost: $O(n+m)$**.** The communication costs of each user can be broken up into 4 parts: (1) Exchanging keys with each other user by sending 2 and receiving $2(n-1)$ public keys, (2) Sending $2(n-1)$ and receiving $2(n-1)$ encrypted secret shares, (3) Sending a perturbed data vector of size $m\lceil \log_2 R \rceil$ to the server, and (4) Sending the server n secret shares, for an overall communication cost of $2na_K + (5n-4)a_S + m\lceil \log_2 R \rceil$, where a_K and a_S are the number of bits in a key exchange public key and the number of bits in a secret share, respectively. Overall, the user's communication complexity is $O(n+m)$. Assuming inputs for each user are on the same range $[0, R_U - 1]$, we require $R = n(R_U - 1) + 1$ to avoid overflow. A user could transmit its raw data using



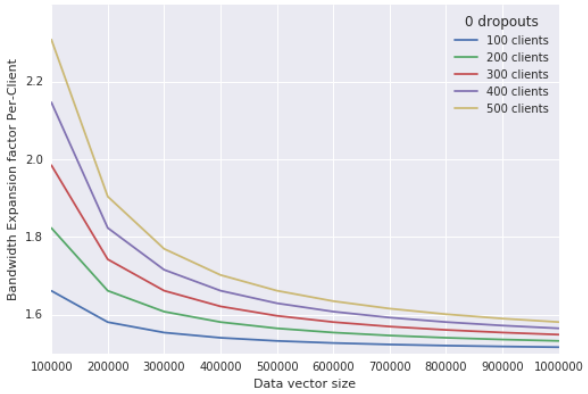
(a) Wall-clock running time per client, as the number of clients increases. The data vector size is fixed to 100K entries.



(b) Wall-clock running time per client, as the size of the data vector increases. The number of clients is fixed to 500.



(c) Total data transfer per client, as the number of clients increases. Different lines show different data vector sizes. Assumes no dropouts.



(d) Total data expansion factor per client, as compared to sending the raw data vector to the server. Different lines represent different values of n. Assumes no dropouts.

Fig. 6. Client Running Time and Data Transfer Costs. All wall-clock running times are for a single-threaded client implemented in Java, and ignore communication latency. Plotted points represent averages over 10 end-to-end iterations, and error bars represent 95% confidence intervals. (Error bars are omitted where measured standard deviation was less than 1%).

$m\lceil \log_2 R_U \rceil$ bits. Taking $a_K = a_S = 256$ bits implies a communication expansion factor of $\frac{256(7n-4)+m\lceil \log_2 R \rceil}{m\lceil \log_2 R_U \rceil}$. For $R_U = 2^{16}$ (i.e. 16-bit input values), $m = 2^{20}$ elements, and $n = 2^{10}$ users, the expansion factor is $1.73\times$; for $n = 2^{14}$ users, it is $3.62\times$. For $m = 2^{24}$ elements and $n = 2^{14}$ users, the expansion factor is $1.98\times$.

Storage cost: $O(n + m)$. The user must store the keys and secret-shares sent by each other user, which are $O(n)$ in total, and the data vector (which it can perturb in-place), which has size $O(m)$.

B. Performance Analysis of Server

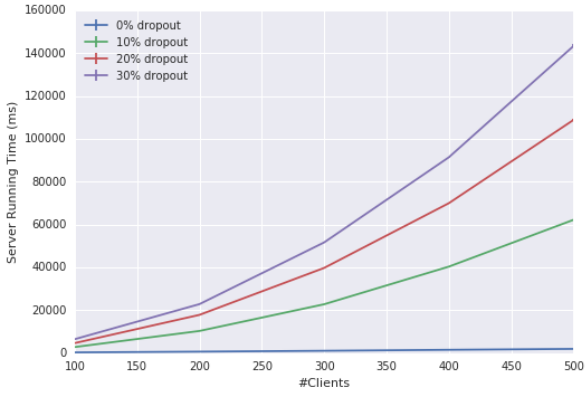
Computation cost: $O(mn^2)$**.** The server's computation cost can be broken down as (1) Reconstructing n t-out-of-n Shamir secrets (one for each user), which takes total time $O(n^2)$, and (2) generating and removing the appropriate $\boldsymbol{p}_{u,v}$ and $\boldsymbol{p}_u$ values from the sum of the $\boldsymbol{y}_u$ values received, which takes time $O(mn^2)$ in the worst case.

We note that reconstructing n secrets in the Shamir scheme takes $O(n^3)$ time in the general case: each secret reconstruction **SS.reconstruct**$(\{(u, s_u)\}_{u\in\mathcal{U}'}, t) \to s$ amounts to interpolating a polynomial L over the points encoded by the shares and then evaluating at 0, which can be accomplished via Lagrange polynomials:

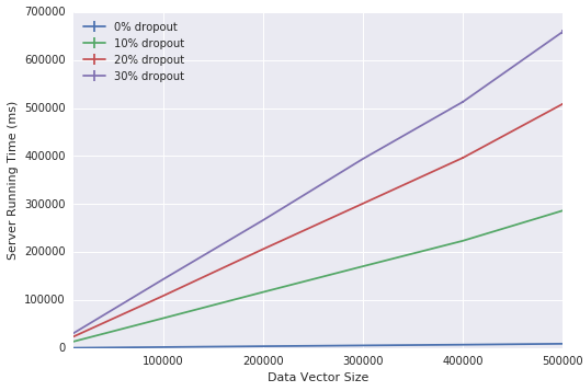
$$s = L(0) = \sum_{u\in\mathcal{U}'} s_u \prod_{v\in\mathcal{U}'\setminus\{u\}} \frac{v}{v-u} \pmod p$$

Each reconstruction requires $O(n^2)$ computation and we must perform n reconstructions, implying $O(n^3)$ total time. However, in our setting, we can perform all of the reconstructions in in $O(n^2)$ time by observing that all of our secrets will be reconstructed from identically-indexed sets of secret shares – that is, $\mathcal{U}'$ is fixed across all secrets, because in round **Unmasking**, each user that is still alive sends a share of *every* secret that needs to be reconstructed. Therefore, we can precompute the Lagrange basis polynomials

$$\ell_u = \prod_{v\in\mathcal{U}'\setminus\{u\}} \frac{v}{v-u} \pmod p$$



(a) Wall-clock running time for the server, as the number of clients increases. The data vector size is fixed to 100K entries.



(b) Wall-clock running time for the server, as the size of the data vector increases. The number of clients is fixed to 500.

Fig. 7. Server Running Time and Data Transfer Costs. All wall-clock running times are for a single-threaded server implemented in Java, and ignore communication latency. Plotted points represent averages over 10 end-to-end iterations. Error bars are omitted where measured standard deviations are less than 1%.

in $O(n^2)$ time and $O(n)$ space, then reconstruct each of n secrets in $O(n)$ time as $L(0) = \sum_{u \in \mathcal{U}'} s_u \ell_u \pmod p$ resulting in a total computational cost of $O(n^2)$ to reconstruct all the secrets.

We also note that the $O(mn^2)$ term can be broken into $O(m(n-d) + md(n-d))$, where d is the number of users that dropped from the protocol. In practice, d may be significantly smaller than n, which would also reduce the server's computation cost.

Communication cost: $O(n^2 + mn)$**.** The server's communication cost is dominated by its mediation of all pairwise communications between users, which is $O(n^2)$, and also for receiving perturbed data vectors from each user, which is $O(mn)$ in total.

Storage cost: $O(n^2+m)$**.** The server must store t shares for each user, which is $O(n^2)$ in total, along with an m-element buffer in which to maintain a running sum of y_u as they arrive.

C. Prototype Performance

In order to measure performance, we implemented a prototype in Java, with the following cryptographic primitives:

- For Key Agreement, we used Elliptic-Curve Diffie-Hellman over the NIST P-256 curve, composed with a SHA-256 hash.
- For Secret Sharing, we used standard t-out-of-n Shamir Sharing.
- For Authenticated Encryption, we used AES-GCM with 128-bit keys.
- For the Pseudorandom Number Generator, we used AES in counter mode.

We assume an honest-but-curious setting, and thus omitted the portions of Figure 4 special to malicious clients from our simulations. We note that these omissions would not change the overall shape of our results in practice, since, as we discuss below, the bulk of the costs involve masking, storing and sending the large data vector.

Additionally, we assume that when clients drop out of the protocol, that they drop after sending their shares to all other clients, but before sending their masked input to the server. This is essentially the "worst case" dropout, since all other clients have already incorporated the dropped clients perturbations, and the server must perform an expensive recovery computation to remove them. We also assumed that client's data vectors had entries of 2 bytes each, such that 3 bytes are required to store the sum of up to 500 clients' values without overflow.

We ran single-threaded simulations on a Linux workstation with an Intel Xeon CPU E5-1650 v3 (3.50 GHz), with 32 GB of RAM. Wall-clock running times and communication costs for clients are plotted in Figure 6. Wall clock running times for the server are plotted in Figure 7, with different lines representing different percentages of clients dropping out. Figure 8 shows wall-clock times per round for both the client and the server. We omit data transfer plots for the server, as they are essentially identical to those for the client, except higher by a factor of n. This is because the incoming data of the server is exactly the total outgoing data of all clients, and vice versa. We also do not plot bandwidth numbers for different numbers of dropouts, as the number of dropouts does not have a significant impact on this metric.

In our simulations, for both the client and the server, almost all of the computation cost comes from expanding the various PRG seeds to mask the data vector. Compared to this, the computational costs of key agreement, secret sharing and reconstruction, and encrypting and decrypting messages between clients, are essentially negligible, especially for large choices of n and data vector size. This suggests that using an optimized PRG implementation would yield a significant running-time improvement over our prototype.

As seen in Figures 6a and 6b, the running time of each client increases linearly with both the total number of clients and the number of data vector entries, but does not change significantly when more clients drop out. In Figure 6c, the

	Dropouts	AdvertiseKeys	ShareKeys	MaskedInputCollection	Unmasking	Total
User	0%	1 ms	376 ms	1030 ms	1 ms	1413 ms
Server	0%	1 ms	26 ms	723 ms	1268 ms	2018 ms
Server	10%	1 ms	29 ms	623 ms	61586 ms	62239 ms
Server	30%	1 ms	28 ms	514 ms	142847 ms	143389 ms

Fig. 8. Wall clock times per round. All wall-clock running times are for a single-threaded servers and clients implemented in Java, and ignore communication latency. Each entry represents the average over 10 iterations. Number of clients is fixed to 500, and the data vector size is fixed to 100K entries.

communication expansion factor for each client increases as the total number of clients increases, but this increase is relatively small compared to the impact of increasing the size of the data vector. This is also reflected in Figure 6d, where the communication expansion factor for each client increases as the total number of clients increases, but falls quickly as the size of the data vector increases. This shows the the cost of messages between clients amortizes well as the size of the data vector increases.

In the case of the server, Figures 7a and 7b show that the running time of the server increases significantly with the fraction of dropouts. This is because, for each dropped client u, the server must remove that client's pairwise perturbations $\boldsymbol{p}_{u,v}$ from each other surviving client v, which requires $(n-d)$ PRG expansions, where d is the number of dropped users. In contrast, each undropped user needs only a single PRG expansion, to remove its self-perturbation. The high cost of dealing with dropped users is also reflected in the server running times in Figure 8.

VIII. Discussion and Future Work

1) Identifying and Recovering from Abuse: The security proof in Theorem 4 guarantees that when users' inputs are learned by the server, they are always in aggregate with the values of other users. However, we do not protect against malicious clients that try to prevent the server from learning *any* sum at all. For example, an attacker-controlled client could send malformed messages to other clients, causing enough of them to abort that the protocol fails before the server can compute its output. Ideally, we would like such abuse by malicious clients to be efficiently identifiable, and the protocol to gracefully recover from it. However, the problem of assigning blame for abuse is subtle, and often adds several rounds to protocols. We leave this problem to future work.

2) Enforcing Well-formed Inputs: Our protocol also does not verify that users' inputs are well-formed or within any particular bounds, so malicious users could send arbitrary values of their choice, that can cause the output learned by the server to also be ill-formed. For our specific machine learning application, we will be able to detect obviously malformed outputs and can simply run the protocol again with a different set of clients. However, a malicious client may be able to supply "slightly" malformed input values, that are hard to detect, for example, using double its real values.

A possible solution is to use zero-knowledge proofs that the client inputs are in the correct range. Unfortunately, even using the best-known garbled circuit techniques [37], even one such proof would be more costly than the entire protocol. We leave the problem of guaranteeing well-formed inputs from the clients to future work.

3) Reducing Communication Further: In the protocol we describe, all clients exchange pairwise perturbation with all other clients. However, it may be sufficient to have the clients exchange perturbations with only a subset of other clients, as long as these subsets of clients do not form disjoint clusters. In fact, previous works (notably [15]) use this approach already. However, in our setting, this requires extra care because the server facilitates the communication among clients, and can choose dropouts maliciously based on it's knowledge of which pairs of clients exchanged masks with each other. We leave this improvement to future work.

IX. Conclusion

We have presented a practical protocol for securely aggregating data while ensuring that clients' inputs are only learned by the server in aggregate. The overhead of our protocol is very low, and it can tolerate large numbers of failing devices, making it ideal for mobile applications. We require only one service provider, which simplifies deployment. Our protocol has immediate applications to real-world federated learning, and we expect to deploy a full application in the near future.

References

[1] J. Paparrizos, R. W. White, and E. Horvitz, "Screening for pancreatic adenocarcinoma using signals from web search logs: Feasibility study and results," *Journal of Oncology Practice*, vol. 12, no. 8, pp. 737–744, 2016.

[2] V. Lampos, A. C. Miller, S. Crossan, and C. Stefansen, "Advances in nowcasting influenza-like illness rates using search query logs," *Scientific reports*, vol. 5, p. 12760, 2015.

[3] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[4] S. Advokat, "Publication of bork's video rentals raises privacy issue," *Chicago Tribune*, 1987.

[5] K. E. McCabe, "Just you and me and netflix makes three: Implications for allowing frictionless sharing of personally identifiable information under the video privacy protection act," *J. Intell. Prop. L.*, vol. 20, p. 413, 2012.

[6] A. Narayanan and V. Shmatikov, "Robust de-anonymization of large sparse datasets," in *2008 IEEE Symposium on Security and Privacy (sp 2008)*. IEEE, 2008, pp. 111–125.

[7] M. Barbaro, T. Zeller, and S. Hansell, "A face is exposed for aol searcher no. 4417749," *New York Times*, vol. 9, no. 2008, 2006.

[8] L. Sweeney and J. S. Yoo, "De-anonymizing south korean resident registration numbers shared in prescription data," *Technology Science*, 2015.

[9] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 1322–1333.

[10] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 1310–1321.

[11] R. Shokri, M. Stronati, and V. Shmatikov, "Membership inference attacks against machine learning models," *arXiv preprint arXiv:1610.05820*, 2016.

[12] C. Dwork, "Differential privacy," in *33rd International Colloquium on Automata, Languages and Programming, part II (ICALP 2006)*, vol. 4052. Venice, Italy: Springer Verlag, July 2006, pp. 1–12. [Online]. Available: https://www.microsoft.com/en-us/research/publication/differential-privacy/

[13] J. C. Duchi, M. I. Jordan, and M. J. Wainwright, "Local privacy and statistical minimax rates," in *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*. IEEE, 2013, pp. 429–438.

[14] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 308–318.

[15] G. Ács and C. Castelluccia, "I have a DREAM! (DiffeRentially privatE smArt Metering)," in *International Workshop on Information Hiding*. Springer, 2011, pp. 118–132.

[16] Ú. Erlingsson, V. Pihur, and A. Korolova, "Rappor: Randomized aggregatable privacy-preserving ordinal response," in *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*. ACM, 2014, pp. 1054–1067.

[17] M. J. Wainwright, M. I. Jordan, and J. C. Duchi, "Privacy aware learning," in *Advances in Neural Information Processing Systems*, 2012, pp. 1430–1438.

[18] S. Goryczka and L. Xiong, "A comprehensive comparison of multiparty secure additions with differential privacy," *IEEE Transactions on Dependable and Secure Computing*, 2015.

[19] J. Goodman, G. Venolia, K. Steury, and C. Parker, "Language modeling for soft keyboards," in *Proceedings of the 7th international conference on Intelligent user interfaces*. ACM, 2002, pp. 194–195.

[20] H. B. McMahan, E. Moore, D. Ramage, S. Hampson *et al.*, "Communication-efficient learning of deep networks from decentralized data," *arXiv preprint arXiv:1602.05629*, 2016.

[21] J. Chen, R. Monga, S. Bengio, and R. Jozefowicz, "Revisiting distributed synchronous sgd," in *ICLR Workshop Track*, 2016. [Online]. Available: https://arxiv.org/abs/1604.00981

[22] I. Goodfellow, Y. Bengio, and A. Courville, "Deep learning," 2016, book in preparation for MIT Press.

[23] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game," in *Proceedings of the nineteenth annual ACM symposium on Theory of computing*. ACM, 1987, pp. 218–229.

[24] 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*. ACM, 1988, pp. 1–10.

[25] Y. Lindell, E. Oxman, and B. Pinkas, "The ips compiler: Optimizations, variants and concrete efficiency," *Advances in Cryptology–CRYPTO 2011*, pp. 259–276, 2011.

[26] I. Damgård, V. Pastro, N. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Advances in Cryptology–CRYPTO 2012*. Springer, 2012, pp. 643–662.

[27] Y. Lindell, B. Pinkas, N. P. Smart, and A. Yanai, "Efficient constant round multi-party computation combining bmr and spdz," in *Annual Cryptology Conference*. Springer, 2015, pp. 319–338.

[28] M. Burkhart, M. Strasser, D. Many, and X. Dimitropoulos, "Sepia: Privacy-preserving aggregation of multi-domain network events and statistics," *Network*, vol. 1, p. 101101, 2010.

[29] D. Chaum, "The dining cryptographers problem: unconditional sender and recipient untraceability," *Journal of Cryptology*, vol. 1, no. 1, pp. 65–75, 1988.

[30] P. Golle and A. Juels, "Dining cryptographers revisited," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2004, pp. 456–473.

[31] H. Corrigan-Gibbs, D. I. Wolinsky, and B. Ford, "Proactively accountable anonymous messaging in verdict." in *USENIX Security*, 2013, pp. 147–162.

[32] Y. H. Kwon, "Riffle: An efficient communication system with strong anonymity," Ph.D. dissertation, Massachusetts Institute of Technology, 2015.

[33] T. Elahi, G. Danezis, and I. Goldberg, "Privex: Private collection of traffic statistics for anonymous communication networks," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014, pp. 1068–1079.

[34] R. Jansen and A. Johnson, "Safely measuring tor," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 1553–1567.

[35] V. Rastogi and S. Nath, "Differentially private aggregation of distributed time-series with transformation and encryption," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 2010, pp. 735–746.

[36] S. Halevi, Y. Lindell, and B. Pinkas, "Secure computation on the web: Computing without simultaneous interaction," in *Annual Cryptology Conference*. Springer, 2011, pp. 132–150.

[37] M. Jawurek, F. Kerschbaum, and C. Orlandi, "Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 955–966.

[38] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[39] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[40] M. Abdalla, M. Bellare, and P. Rogaway, "The oracle diffie-hellman assumptions and an analysis of dhies," in *Cryptographers' Track at the RSA Conference*. Springer, 2001, pp. 143–158.

[41] M. Bellare and C. Namprempre, "Authenticated encryption: Relations among notions and analysis of the generic composition paradigm," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2000, pp. 531–545.

[42] A. C. Yao, "Theory and application of trapdoor functions," in *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*. IEEE, 1982, pp. 80–91.

[43] M. Blum and S. Micali, "How to generate cryptographically strong sequences of pseudorandom bits," *SIAM journal on Computing*, vol. 13, no. 4, pp. 850–864, 1984.

[44] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in *Proceedings of the 1st ACM conference on Computer and communications security*. ACM, 1993, pp. 62–73.

[45] N. Koblitz and A. J. Menezes, "The random oracle model: a twenty-year retrospective," *Designs, Codes and Cryptography*, vol. 77, no. 2-3, pp. 587–610, 2015.

[46] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy," *Foundations and Trends® in Theoretical Computer Science*, vol. 9, no. 3–4, pp. 211–407, 2014.

APPENDIX A
CRYPTOGRAPHIC PRIMITIVES

In this section, we discuss the cryptographic primitives and assumptions needed for our construction.

A. Secret Sharing

We rely on Shamir's t-out-of-n Secret Sharing [38], which allows a user to split a secret s into n shares, such that any t shares can be used to reconstruct s, but any set of at most $t-1$ shares gives no information about s.

The scheme is parameterized over a finite field $\mathbb{F}$ of size at least $l > 2^k$ (where k is the security parameter of the scheme), e.g. $\mathbb{F} = \mathbb{Z}_p$ for some large public prime p. We note that such a large field size is needed because our scheme requires clients to secret share their secret keys (whose length must be proportional to the security parameter for the security proof to go through). We also assume that integers $1, \ldots, n$ (which will be used to denote users in the protocol) can be identified with distinct group elements in $\mathbb{F}$. Given these parameters, the scheme consists of two algorithms. The sharing algorithm $\textbf{SS.share}(s, t, \mathcal{U}) \rightarrow \{(u, s_u)\}_{u \in \mathcal{U}}$ takes as input a secret s, a set $\mathcal{U}$ of n field elements representing user IDs, and a threshold $t \leq |\mathcal{U}|$; it produces a set of shares s_u, each of which is associated with a different $u \in \mathcal{U}$. The reconstruction algorithm $\textbf{SS.reconstruct}(\{(u, s_u)\}_{u \in \mathcal{V}}, t) \rightarrow s$ takes as input the threshold t and the shares corresponding to a subset $\mathcal{V} \subseteq \mathcal{U}$ such that $|\mathcal{V}| \geq t$, and outputs a field element s.

Correctness requires that $\forall s \in \mathbb{F}, \forall t, n$ with $1 \leq t \leq n$, $\forall \mathcal{U} \subseteq \mathbb{F}$ where $|\mathcal{U}| = n$, if $\mathcal{V} \subseteq \mathcal{U}$ and $|\mathcal{V}| \geq t$, then $\textbf{SS.reconstruct}(\{(u, s_u)\}_{u \in \mathcal{V}}, t) = s$. Security requires that $\forall s, s' \in \mathbb{F}$ and any $\mathcal{V} \subseteq \mathcal{U}$ such that $|\mathcal{V}| < t$:

$$\{\{(u, s_u)\}_{u \in \mathcal{U}} \leftarrow \textbf{SS.share}(s, t, \mathcal{U}) : \{(u, s_u)\}_{u \in \mathcal{V}}\} = \{\{(u, s_u)\}_{u \in \mathcal{U}} \leftarrow \textbf{SS.share}(s', t, \mathcal{U}) : \{(u, s_u)\}_{u \in \mathcal{V}}\}$$

where "=" denotes that the two distributions are identical.

B. Key Agreement

Key Agreement consists of a tuple of algorithms $(\textbf{KA.paramGen}, \textbf{KA.gen}, \textbf{KA.agree})$. $\textbf{KA.paramGen}(k) \rightarrow pp$ produces some public parameters (over which our scheme will be parameterized); $\textbf{KA.gen}(pp) \rightarrow (s_u^{SK}, s_u^{PK})$ allows any party u to generate a private-public key pair, $\textbf{KA.agree}(s_u^{SK}, s_v^{PK}) \rightarrow s_{u,v}$ that allows any user u to combine their private key s_u^{SK} with the public key s_v^{PK} for any v (generated using the same pp), to obtain a private shared key $s_{u,v}$ between u and v.

The specific Key Agreement scheme we will use is Diffie-Hellman key agreement [39], composed with a hash function. More specifically, $\textbf{KA.paramGen}(k) \rightarrow (\mathbb{G}', g, q, H)$ samples group $\mathbb{G}'$ of prime order q, along with a generator g, and a hash function H[6]; $\textbf{KA.gen}(\mathbb{G}', g, q, H) \rightarrow (x, g^x)$ samples a random $x \leftarrow \mathbb{Z}_q$ as the secret key s_u^{SK}, and g^x as the public key s_u^{PK}; and $\textbf{KA.agree}(x_u, g^{x_v}) \rightarrow s_{u,v}$ outputs $s_{u,v} = H((g^{x_v})^{x_u})$.

[6]In practice, one can use SHA-256.

Correctness requires that, for any key pairs generated by users u and v (using **KA.gen** and the same parameters pp), $\textbf{KA.agree}(s_u^{SK}, s_v^{PK}) = \textbf{KA.agree}(s_v^{SK}, s_u^{PK})$. For security, in the honest but curious model, we want that for any adversary who is given two honestly generated public keys s_u^{PK} and s_v^{PK} (but neither of the corresponding secret keys s_u^{SK} or s_v^{SK}), the shared secret $s_{u,v}$ computed from those keys is indistinguishable from a uniformly random string. This exactly mirrors the Decisional Diffie-Hellman assumption, which we recall below:

Definition 1 (Decisional Diffie-Hellman assumption (DDH))**.** *Let $\mathcal{G}(k) \rightarrow (\mathbb{G}', g, q, H)$ be an efficient algorithm which samples a group $\mathbb{G}'$ of order q with generator g, as well as a function $H : \{0,1\}^* \rightarrow \{0,1\}^k$. Consider the following probabilistic experiment, parameterized by a PPT adversary M, a bit b and a security parameter k.*

$\mathbf{DDH} - \mathbf{Exp}^b_{\mathcal{G},M}(k)$*:*

1) $(\mathbb{G}', g, q, H) \leftarrow \mathcal{G}(k)$
2) $a \leftarrow \mathbb{Z}_q; A \leftarrow g^a$
3) $b \leftarrow \mathbb{Z}_q; B \leftarrow g^b$
4) *if* $b = 1$, $s \leftarrow H(g^{ab})$, *else* $s \xleftarrow{\$} \{0,1\}^k$
5) $M(\mathbb{G}', g, q, H, A, B, s) \rightarrow b'$
6) *Output 1 if* $b = b'$, *0 o/w.*

The advantage of the adversary is defined as

$$Adv^{DDH}_{\mathcal{G},M}(k) := |\Pr[\mathbf{DDH} - \mathbf{Exp}^1_{\mathcal{G},M}(k) = 1] - \Pr[\mathbf{DDH} - \mathbf{Exp}^0_{\mathcal{G},M}(k) = 1]|$$

We say that the Decisional Diffie-Hellman assumption holds for $\mathcal{G}$ if for all PPT adversaries M, there exists a negligible function ϵ such that $Adv^{DDH}_{\mathcal{G},M}(k) \leq \epsilon(k)$.

Note that, traditionally, the Diffie-Hellman assumption does not directly involve a hash function H (i.e. line step 4 is substituted with "if $b = 1$, $s \leftarrow g^{ab}$, else $s \xleftarrow{\$} \mathbb{G}'$"), and therefore to get from a random element of the group $\mathbb{G}'$ to a uniformly random string (which is necessary to be used as the seed for a **PRG**, or to sample secret keys for other primitives), one has to compose g^{ab} with a secure randomness extractor (which composes well with this specific key agreement operation). For simplicity, we choose to incorporate such an extractor function H in the assumption.

In order to prove security against malicious adversaries (Theorem 4), we need a somewhat stronger security guarantee for Key Agreement, namely that an adversary who is given two honestly generated public keys s_u^{PK} and s_v^{PK}, and also the ability to learn $\textbf{KA.agree}(s_u^{SK}, s^{PK})$ and $\textbf{KA.agree}(s_v^{SK}, s^{PK})$ for any s^{PK}s of its choice (but different from s_u^{PK} and s_v^{PK}), still cannot distinguish $s_{u,v}$ from a random string. In order to get this stronger property, we need to rely on a slight variant of the Oracle Diffie-Hellman assumption (ODH) [40], which we call Two Oracle Diffie-Hellman assumption (2ODH):

Definition 2 (Two Oracle Diffie-Hellman assumption (2ODH))**.** *Let $\mathcal{G}(k) \rightarrow (\mathbb{G}', g, q, H)$ be an efficient algorithm which samples a group $\mathbb{G}'$ of order q with generator g, as well*

as a function $H : \{0,1\}^* \to \{0,1\}^k$. Consider the following probabilistic experiment, parameterized by a PPT adversary M, a bit b and a security parameter k.

$\mathbf{2ODH} - \mathbf{Exp}^b_{\mathcal{G},M}(k)$*:*

1) $(\mathbb{G}', g, q, H) \leftarrow \mathcal{G}(k)$
2) $a \leftarrow \mathbb{Z}_q; A \leftarrow g^a$
3) $b \leftarrow \mathbb{Z}_q; B \leftarrow g^b$
4) *if* $b = 1$, $s \leftarrow H(g^{ab})$, *else* $s \xleftarrow{\$} \{0,1\}^k$
5) $M^{\mathcal{O}_a(\cdot),\mathcal{O}_b(\cdot)}(\mathbb{G}', g, q, H, A, B, s) \to b'$
6) *Output 1 if* $b = b'$, *0 o/w.*

where $\mathcal{O}_a(X)$ returns $H(X^a)$ on any $X \neq B$ (and an error on input B) and similarly $\mathcal{O}_b(X)$ returns $H(X^b)$ on any $X \neq A$. The advantage of the adversary is defined as

$$Adv^{2ODH}_{\mathcal{G},M}(k) := |\Pr[\mathbf{2ODH} - \mathbf{Exp}^1_{\mathcal{G},M}(k) = 1] - \Pr[\mathbf{2ODH} - \mathbf{Exp}^0_{\mathcal{G},M}(k) = 1]|$$

We say that the Two Oracle Diffie-Hellman assumption holds for $\mathcal{G}$ if for all PPT adversaries M, there exists a negligible function ϵ such that $Adv^{2ODH}_{\mathcal{G},M}(k) \leq \epsilon(k)$.

This assumption can be directly used to prove the security property we need for Key Agreement: the two oracles $\mathcal{O}_a(\cdot), \mathcal{O}_b(\cdot)$ formalize the ability of the adversary M to learn $\mathbf{KA.agree}(s^{SK}_u, s^{PK})$ and $\mathbf{KA.agree}(s^{SK}_v, s^{PK})$ for different s^{PK}, and the negligible advantage of M in the above game corresponds to an inability to distinguish between $s = s_{u,v} \leftarrow H(g^{ab})$, and $s \xleftarrow{\$} \{0,1\}^k$.

C. Authenticated Encryption

(Symmetric) Authenticated Encryption combines confidentiality and integrity guarantees for messages exchanged between two parties. It consists of a key generation algorithm which outputs a private key[7], an encryption algorithm **AE.enc** that given a key and a message produces a ciphertext, and a decryption algorithm **AE.dec** which given a ciphertext and a key returns either the original plaintext, or a special error symbol $\perp$. For correctness, we require that for all keys $c \in \{0,1\}^k$ and all messages x, $\mathbf{AE.dec}(c, \mathbf{AE.enc}(c, x)) = x$. For security, we require indistinguishability under a chosen plaintext attack (IND-CPA) and ciphertext integrity (IND-CTXT) as defined in [41]. Informally, the guarantee is that for any adversary M that is given encryptions of messages of its choice under a randomly sampled key c (where c is unknown to M), M cannot distinguish between fresh encryptions under c of two different messages, nor can M create new valid ciphertexts (different from the ones it received) with respect to c with better than negligible advantage.

D. Pseudorandom Generator

We require a secure Pseudorandom Generator [42], [43] **PRG** that takes in a uniformly random seed of some fixed length, and whose output space is $[0, R)^m$ (i.e. the input space for the protocol). Security for a Pseudorandom Generator guarantees that its output on a uniformly random seed is computationally indistinguishable from a uniformly sampled element of the output space, as long as the seed is hidden from the distinguisher.

[7]Without loss of generality, we make the simplifying assumption that the key generation algorithm samples keys as uniformly random strings.

E. Signature Scheme

The protocol relies on a standard UF-CMA secure signature scheme ($\mathbf{SIG.gen}, \mathbf{SIG.sign}, \mathbf{SIG.verify}$). The key generation algorithm $\mathbf{SIG.gen}(k) \to (d^{PK}, d^{SK})$ takes as input the security parameter and outputs a secret key d^{SK} and a public key d^{PK}; the signing algorithm $\mathbf{SIG.sign}(d^{SK}, m) \to \sigma$ takes as input the secret key and a message and outputs a signature σ; the verification algorithm $\mathbf{SIG.verify}(d^{PK}, m, \sigma) \to \{0,1\}$ takes as input a public key, a message and a signature, and returns a bit indicating whether the signature should be considered valid. For correctness, we require that $\forall m$,

$$\Pr[(d^{PK}, d^{SK}) \leftarrow \mathbf{SIG.gen}(k), \sigma \leftarrow \mathbf{SIG.sign}(d^{SK}, m) : \mathbf{SIG.verify}(d^{PK}, m, \sigma) = 1] = 1$$

Security demands that no PPT adversary, given a fresh honestly generated public key and access to an oracle producing signatures on arbitrary messages, should be able to produce a valid signature on a message on which the oracle was queried on with more than negligible probability.

F. Public Key Infrastructure

To prevent the server from simulating an arbitrary number of clients (in the malicious model), we require the support of a public key infrastructure that allows clients to register identities, and sign messages using their identity, such that other clients can verify this signature, but cannot forge them. We model such an infrastructure by assuming that a trusted party generates a public-private key-pair for a signature scheme for each client in the protocol (honest or malicious), and that, at the start of the protocol, this trusted party provides each client with its private signing key d^{SK}_u, together with the public verification key d^{PK}_v for every other user v.

APPENDIX B
TECHNICAL LEMMA

In our security arguments, we will make use of the following technical lemma. It says that if users' values have uniformly random pairwise masks added to them, then the resulting values look uniformly random, conditioned on their sum being equal to the sum of the users' values. In other words, the pairwise masks hide all information about users' individual inputs, except for their sum.

Lemma 1. *Fix n, m, R, $\mathcal{U}$ with $|\mathcal{U}| = n$, and $\{\boldsymbol{x}_u\}_{u\in\mathcal{U}}$ where $\forall u \in \mathcal{U}, \boldsymbol{x}_u \in \mathbb{Z}_R^m$. Then,*

$$\{\{\boldsymbol{p}_{u,v} \xleftarrow{\$} \mathbb{Z}_R^m\}_{u<v}, \qquad \boldsymbol{p}_{u,v} := -\boldsymbol{p}_{v,u} \forall u > v : \{\boldsymbol{x}_u + \sum_{v\in\mathcal{U}\setminus\{u\}} \boldsymbol{p}_{u,v} \pmod R\}_{u\in\mathcal{U}}\}$$

$$\equiv$$

$$\{\{\boldsymbol{w}_u \xleftarrow{\$} \mathbb{Z}_R^m\}_{u\in\mathcal{U}} \text{ s.t. } \sum_{u\in\mathcal{U}} \boldsymbol{w}_u = \sum_{u\in\mathcal{U}} \boldsymbol{x}_u \pmod R : \{\boldsymbol{w}_u\}_{u\in\mathcal{U}}\}$$

where "$\equiv$" denotes that the distributions are identical.

We omit the proof, noting that it can be proved easily by induction on n.

Appendix C
Privacy against Malicious Adversaries

In this section, we give formal arguments showing that our protocol preserves privacy against active adversaries.

Before we state our theorems, we introduce some additional definitions and notation for the setting of malicious and colluding users and servers. As is standard, we consider only computationally-bounded malicious parties, namely those whose strategies can be described by some probabilistic polynomial-time algorithm M. We note that our proofs will be performed in the so-called "Random Oracle" model [44]. A random oracle $\mathcal{O}$ can be thought of as a perfectly random function in the sky, which can be queried on any input x and bit length l, and which returns $\mathcal{O}(x)$, a binary string of length l, such that each $\mathcal{O}(x)$ is uniformly random and independent, and such that repeated queries on the same x and l give the same result. A random oracle can also be thought of as a "perfect PRG": whereas $\mathbf{PRG}(x)$ outputs a pseudorandom string, $\mathcal{O}(x)$ outputs a *truly* random string. While a random oracle can never actually be instantiated due to its provably exponential size, it is often replaced with a cryptographic hash function with little practical loss in security [45]. In our proofs in this section, we will assume that a common random oracle $\mathcal{O}$ is available to all the parties, who can each make arbitrarily many oracle queries to $\mathcal{O}$ during the course of their execution. Also, all honest parties will substitute all **PRG** calls with calls to $\mathcal{O}$, on the same input as they used for the **PRG**, and with the appropriate bit length l.

We also allow the malicious parties to adaptively choose the set of honest parties that (truly) abort in that round, which gives them more power than having the aborts be independently chosen, or predetermined fixed in advance.

For fixed n, t and k and a set $\mathcal{C}$ of corrupt parties, we let $M_{\mathcal{C}}$ indicate the polynomial-time algorithm that denotes the "next-message" function of parties in $\mathcal{C}$. That is, given a party identifier $c \in \mathcal{C}$, a round index i, a transcript T of all messages sent and received so far by all parties in $\mathcal{C}$, joint randomness $r_{\mathcal{C}}$ for the corrupt parties' execution, and access to random oracle $\mathcal{O}$, $M_{\mathcal{C}}(c, i, T, r_{\mathcal{C}})$ outputs the message for party c in round i (possibly making several queries to $\mathcal{O}$ along the way). Additionally, given a round index i, a transcript T of all messages sent and received so far by all parties in $\mathcal{C}$, and joint randomness $r_{\mathcal{C}}$ for the corrupt parties' execution, $M_{\mathcal{C}}(i, T, r_{\mathcal{C}})$ outputs the set of parties $\mathcal{U}_i$ that abort due to failure in that round (again, possibly making several queries to $\mathcal{O}$ along the way). [8] We note that $M_{\mathcal{C}}$ is thus effectively choosing the inputs for all malicious users.

Let $\texttt{REAL}_{\mathcal{C}}^{\mathcal{U},t,k}(M_{\mathcal{C}}, \boldsymbol{x}_{\mathcal{U}})$ be a random variable representing the combined views of all parties in $\mathcal{C}$ in the above protocol execution, where all corrupt parties' messages and the independent failures of the honest parties are chosen using $M_{\mathcal{C}}$, and all parties including $M_{\mathcal{C}}$ have access to $\mathcal{O}$. The random variable's distribution is over the random choices of honest parties' randomness and $r_{\mathcal{C}}$, over randomness of setup, and of the random oracle $\mathcal{O}$ which is provided to all parties.

As in the case of honest-but-curious users, we consider two separate cases: one where only a subset of users are malicious and colluding, and one where the server is additionally malicious and colluding with some subset of malicious users.

Theorem 3 (Privacy against actively-malicious users, with honest server)**.** *There exists a PPT simulator* SIM *such that for all PPT adversaries $M_{\mathcal{C}}$, all $k, t, \mathcal{U}, \boldsymbol{x}_{\mathcal{U}\setminus\mathcal{C}}, \mathcal{C} \subseteq \mathcal{U}$, the output of* SIM *is perfectly indistinguishable from the output of $\texttt{REAL}_{\mathcal{C}}^{\mathcal{U},t,k}$:*

$$\texttt{REAL}_{\mathcal{C}}^{\mathcal{U},t,k}(M_{\mathcal{C}}, \boldsymbol{x}_{\mathcal{U}\setminus\mathcal{C}}) \equiv \texttt{SIM}_{\mathcal{C}}^{\mathcal{U},t,k}(M_{\mathcal{C}})$$

Proof. The proof is identical to that for Theorem 1: even though the malicious users additionally get to send arbitrary messages and select the abort pattern of the honest users, the messages they receive from honest users never depend on the private input $\boldsymbol{x}_u$ of those users. Thus, the simulator can emulate the real view of the corrupted users $\mathcal{C}$ by using $M_{\mathcal{C}}$ for the malicious users, and running the honest users on dummy inputs.

Note: Though SIM does not get access to a random oracle $\mathcal{O}$, it can *simulate* $\mathcal{O}$ on the fly using standard techniques. Specifically, it first creates an internal table mapping x to $\mathcal{O}(x)$ that is initially empty. Now, whenever a party making an oracle request on input x, SIM checks if x is in its table, and if so, returns the associated $\mathcal{O}(x)$, and otherwise, generates a fresh, uniformly random string for $\mathcal{O}(x)$, puts $(x, \mathcal{O}(x))$ in its table, and sends $\mathcal{O}(x)$ as the response to the querying party. Since all parties run in polynomial time, the table can never grow to more than polynomial size, so SIM remains efficient, and additionally, all parties get exactly the same uniformly random distribution of responses to oracle queries as they would from a real $\mathcal{O}$. □

We now proceed to the proof of security when the malicious parties also include the server. Before we do so, we recall that, in contrast with Theorem 2, $M_{\mathcal{C}}$ is now allowed to *dynamically* choose which users abort in each round, rather than the aborts being statically fixed beforehand. Accordingly, the particular subset of honest users for which the server learns the sum is dynamically determined during the execution of the protocol,

[8] Note that additional honest parties may also abort in each round if the messages they receive from corrupt parties are malformed.

and in particular, we can no longer provide a sum $\boldsymbol{z}$ of some fixed subset of the users' inputs as input to the simulator SIM. Instead, we will allow SIM to make a single query to an ideal functionality that will allow it to learn the sum $\boldsymbol{z}$ of values for a single subset L of honest parties, chosen dynamically by SIM at run-time. More formally, we give SIM access to an oracle $\mathsf{Ideal}^{\delta}_{\{\boldsymbol{x}_u\}_{u\in\mathcal{U}\setminus\mathcal{C}}}$ for appropriately chosen δ, such that $\mathsf{Ideal}^{\delta}_{\{\boldsymbol{x}_u\}_{u\in\mathcal{U}\setminus\mathcal{C}}}$ can be queried only once, and, given a subset L, operates as follows:

$$\mathsf{Ideal}^{\delta}_{\{\boldsymbol{x}_u\}_{u\in\mathcal{U}\setminus\mathcal{C}}}(L) = \begin{cases} \sum_{u\in L} \boldsymbol{x}_u & \text{if } L \subseteq (\mathcal{U}\setminus\mathcal{C}) \text{ and } |L| \geq \delta \\ \perp & \text{otherwise} \end{cases}$$

We also stress that the following theorem relies, in order to leverage the security of the key agreement in a context where some of the clients might be malicious, on a slight variant of the Oracle Diffie-Hellman assumption (ODH) [40], which we call Two Oracle Diffie-Hellman assumption (2ODH) and detail in the appendix.

The theorem shows that the joint view of colluding malicious parties in a real execution of a protocol can be simulated given only a single sum of a (dynamically-chosen) subset of at least δ honest users, meaning intuitively that the malicious parties learn "nothing more" than a single sum of a subset of the honest parties' inputs.

Theorem 4 (Privacy against active adversaries, including the server)**.** *There exists a PPT simulator* SIM *such that for all* $k, t, \mathcal{U}, \mathcal{C} \subseteq \mathcal{U} \cup \{S\}$ *and* $\boldsymbol{x}_{\mathcal{U}\setminus\mathcal{C}}$*, letting* $n = |\mathcal{U}|$ *and* $n_{\mathcal{C}} = |\mathcal{C}\cap\mathcal{U}|$*, if* $2t > n+n_{\mathcal{C}}$*, then the output of* SIM *is computationally indistinguishable from the output of* $\texttt{REAL}^{\mathcal{U},t,k}$*:*

$$\texttt{REAL}^{\mathcal{U},t,k}_{\mathcal{C}}(M_{\mathcal{C}}, \boldsymbol{x}_{\mathcal{U}\setminus\mathcal{C}}) \approx_c \texttt{SIM}^{\mathcal{U},t,k,\mathsf{Ideal}^{\delta}_{\{\boldsymbol{x}_u\}_{u\in\mathcal{U}\setminus\mathcal{C}}}}_{\mathcal{C}}(M_{\mathcal{C}})$$

where $\delta = t - n_{\mathcal{C}}$*.*

Proof. We prove the theorem by a standard hybrid argument. We will define a simulator SIM through a series of (polynomially many) subsequent modifications to the real execution REAL, so that the views of $M_{\mathcal{C}}$ in any two subsequent executions are computationally indistinguishable. In each of the hybrids below, even though we do not explicitly mention it, SIM will cause honest parties to abort as they would during the real the protocol (e.g., if they receive a malformed message), and also if they are in a set $\mathcal{U}_i$ output by $M_{\mathcal{C}}$.

- Hyb_0 This random variable is distributed exactly as the view of $M_{\mathcal{C}}$ in REAL, the joint view of the parties $\mathcal{C}$ in a real execution of the protocol.
- Hyb_1 In this hybrid, the real execution is emulated by a simulator that knows all the inputs $\boldsymbol{x}_u$ of the honest parties, and runs a full execution of the protocol with $M_{\mathcal{C}}$, which includes simulating the random oracle "on the fly" (using a dynamically generated table), the PKI and the rest of the setup phase.
  The view of the adversary in this hybrid is the same as the previous one.
- Hyb_2 In this hybrid, the simulator additionally aborts if $M_{\mathcal{C}}$ provides any of the honest parties u (in round **AdvertiseKeys**) with a correct signature with respect to an honest v's public key, on $(c_v^{PK}||s_v^{PK})$ different from those sent by v. Since this amounts to breaking the security of the signature scheme, this hybrid is identical from the previous one.
- Hyb_3 This hybrid is identical to Hyb_2, except that, for any pair of honest users u, v, the messages among them are encrypted (in round **ShareKeys**, before being given to $M_{\mathcal{C}}$) and decrypted (in round **Unmasking**, after $M_{\mathcal{C}}$ has delivered them) using a uniformly random key (as opposed to the one obtained through the key agreement **KA.agree**(c_u^{SK}, c_v^{PK})).
  The 2ODH assumption guarantees that this hybrid is indistinguishable from the previous one. In particular, we can switch the encryption keys between one pair of honest users at a time (since n is polynomial in k, there are only polynomially many pairs of honest users), and argue that an adversary noticing the difference when one key is switched will also be able to break the 2ODH.
- Hyb_4 This hybrid is identical to Hyb_3, except additionally, SIM will abort if $M_{\mathcal{C}}$ succeeds to deliver, in round **ShareKeys**, a message to an honest client u on behalf of another honest client v, such that i) the message is different from the message SIM had given $M_{\mathcal{C}}$ in round **ShareKeys**, and ii) the message does not cause the decryption algorithm (using the proper key) to fail. Note that, as the encryption key that the two users were using in the previous hybrid was randomly selected, such a message would directly constitute a forgery against the INT-CTXT security of the encryption scheme.
- Hyb_5 In this hybrid, in addition, SIM substitutes all the encrypted shares sent between pairs of honest users with encryptions of 0. (It still returns the "real" shares in Round **Unmasking** as it did before).
  Note that, since the corresponding encryption keys were chosen uniformly at random, IND-CPA security of the encryption scheme guarantees this hybrid is indistinguishable from the previous one.
- Hyb_6 In this hybrid, in addition, SIM aborts if $M_{\mathcal{C}}$ provides any of the honest parties (in round **ConsistencyCheck**) with a signature on a set which correctly verifies w.r.t. the public key of an honest party, but such that the honest client never produced a signature on that set.
  Because of the security of the signature scheme, such forgeries can happen only with negligible probability, therefore this hybrid is indistinguishable from the previous one.
  We are now able to define the set $\mathcal{Q}$ to be the only set $\mathcal{Q} \subseteq \mathcal{U}$ such that there exists an honest user which received the set $\mathcal{Q}$ in round **ConsistencyCheck**, and later received at least t valid signatures on it in round **Unmasking** (where valid means that the signatures verify with respect to a set of distinct public signature

keys among those received by the client from the trusted party at the start of the protocol).

In case no such set $\mathcal{Q}$ exists (e.g. no set had enough signatures, or not enough honest users survived), we define $\mathcal{Q} = \emptyset$.

Note that this set is well defined: since the server cannot forge signatures on behalf of the honest clients, and each honest client will sign at most one set $\mathcal{Q}$, if there were two such sets this would imply that at least $t - n_{\mathcal{C}}$ distinct honest parties signed each of them, i.e. that $2(t - n_{\mathcal{C}}) \leq n - n_{\mathcal{C}}$, which directly contradicts $2t > n + n_{\mathcal{C}}$.

Hyb_7 In this hybrid, in addition, SIM aborts if $M_{\mathcal{C}}$ queries the random oracle/**PRG** on input b_u for some honest user u (i.e. the value sampled by SIM on behalf of u in round **ShareKeys**) either i) before the adversary received the responses from the honest players in round **Unmasking** or ii) after such responses have been received, but where $u \notin \mathcal{Q}$.

In both cases, because the value b_u is information theoretically hidden from $M_{\mathcal{C}}$, SIM will abort due to this new condition only if $M_{\mathcal{C}}$ is able to guess one of the b_u, which can only happen with negligible probability (as they are chosen from the exponentially large domain $\mathbb{F}$). To see why the view of $M_{\mathcal{C}}$ does not depend on b_u, let us analyze which of the view's components depend on any b_u. In case i), $M_{\mathcal{C}}$ only receives from SIM at most $n_{\mathcal{C}}$ shares of b (sent by u in round **ShareKeys**, one for each of the malicious clients). However. since $n_{\mathcal{C}} < t$, the distribution of any such shares is independent from b_u (because of the properties of secret sharing). Even in case ii), the view of $M_{\mathcal{C}}$ is still independent from b_u: since $u \notin \mathcal{Q}$, no honest user would send to the server any share of b_u, and therefore SIM does not have to send any to $M_{\mathcal{C}}$.

Hyb_8 In this hybrid, in addition, SIM aborts if $M_{\mathcal{C}}$ queries the random oracle/**PRG** on input $s_{u,v}$ for some honest users u, v either i) before the adversary received the responses from the honest players in round **Unmasking** or ii) after such responses have been received, but where $u, v \in \mathcal{Q}$.

To argue that this hybrid is indistinguishable from the previous one (except with negligible probability), we will reduce to the security of the 2ODH assumption. In particular, consider a distinguisher $\mathtt{SIM}'$ which receives a 2ODH challenge $(\mathbb{G}', g, q, A, B, z)$ and guesses at random two honest users u, v, hoping that the adversary's query which will cause the simulator to abort will be exactly $s_{u,v}$. $\mathtt{SIM}'$ acts exactly as SIM in the previous hybrid, except it sets up $s_u^{PK} = A$ and $s_v^{PK} = B$ as the public keys for those users and uses its two oracles to complete the simulation without having access to the corresponding secret keys. In particular, in round **AdvertiseKeys**, $\mathtt{SIM}'$ sends these modified public keys to $M_{\mathcal{C}}$ (as opposed to the fresh ones SIM would have sampled in the previous hybrid). In round **ShareKeys**, rather than generating shares of the secret keys s_u^{SK} and s_v^{SK} (which it does not know), it generates and sends to the malicious parties shares of 0. In round **MaskedInputCollection**, when generating $\boldsymbol{y}$ values for all the honest users (to be sent to $M_{\mathcal{C}}$), $\mathtt{SIM}'$ sets $s_{u,v} = z$, and uses its two oracles $\mathcal{O}_a$ and $\mathcal{O}_b$ to compute all other required s values for u and v and other users. Then, if $M_{\mathcal{C}}$ makes a random oracle query for z, $\mathtt{SIM}'$ will guess that $z = H(g^{ab})$ and abort the simulation; otherwise it will guess that z was chosen at random.

Let us now analyze the advantage of such $\mathtt{SIM}'$ in the $\mathbf{2ODH-Exp}$ game. Notice that, conditioned on the choice of u, v being correct, and until the point where the adversary makes a random oracle query for z, the view of the adversary in this simulated protocol execution is exactly the same as the one of Hyb_7. This is because, as in the previous argument, for both possible values of z, the adversary will obtain less than t shares of both s_u^{SK} and s_v^{SK}, which thus reveal "no information" about the actual values of s_u^{SK} and s_v^{SK}. Moreover, because we are modeling the **PRG** as a random oracle, $M_{\mathcal{C}}$ cannot extract any information about $s_{u,v}$ from $\boldsymbol{y}_u$ and $\boldsymbol{y}_v$ without querying the random oracle.

Therefore, if $M_{\mathcal{C}}$ can distinguish between Hyb_7 and Hyb_8 with more than negligible probability, then it must be triggering the abort condition with more than negligible probability and therefore (conditioned on the choice of u and v being correct) $M_{\mathcal{C}}$ must make to the random oracle/**PRG** a query of the form $H(g^{ab})$ with more than negligible probability. This implies that, when $z = H(g^{ab})$, $\mathtt{SIM}'$ will claim (correctly) that $z = H(g^{ab})$ with non negligible probability. On the other hand, when z is chosen uniformly random, it is information theoretically hidden from $M_{\mathcal{C}}$'s view, and therefore $M_{\mathcal{C}}$ can only make a query for it (which will cause $\mathtt{SIM}'$ to incorrectly claim that $z = H(g^{ab})$) with negligible probability. In other words, if $M_{\mathcal{C}}$ distinguishes between Hyb_7 and Hyb_8 with non-negligible probability p, then the algorithm $\mathtt{SIM}'$ described above also breaks 2ODH assumption probability at least $p/2n^2$, which is non-negligible, concluding the argument.

Hyb_9 This hybrid is defined exactly as the previous one, except that the values of $\boldsymbol{y}_u$ computed by the simulator on behalf of the honest clients and sent to $M_{\mathcal{C}}$ in round **MaskedInputCollection** are substituted with uniformly sampled values, and the output of some random oracle queries for the **PRG** is modified to ensure consistency/correctness for the result. More in detail, after the server delivers to honest clients the messages for round **ConsistencyCheck**, but before SIM sends their responses, these messages sent by $M_{\mathcal{C}}$ to the honest clients define a set $\mathcal{Q}$ (as defined

in hybrid Hyb_6). For all $u \in \mathcal{Q} \setminus \mathcal{C}$, SIM programs the random oracle to set $\mathbf{PRG}(b_u)$ as follows:

$$\mathbf{PRG}(b_u) \leftarrow \boldsymbol{y}_u - \boldsymbol{x}_u - \sum_{v \in \mathcal{F}_u} \mathbf{PRG}(s_{u,v})$$

where $v \in \mathcal{F}_u$ iff $v \notin \mathcal{Q} \setminus \mathcal{C}$ and $M_{\mathcal{C}}$ delivered a ciphertext to u from v in round **ShareKeys** (which captures the fact that in a real execution u would have included the joint noise $\boldsymbol{p}_{u,v}$ for v in its masked input vector $\boldsymbol{y}_u$). For all $u \notin \mathcal{Q} \setminus \mathcal{C}$, SIM sets $\mathbf{PRG}(b_u)$ arbitrarily.

We will argue that the view of $M_{\mathcal{C}}$ in this hybrid is statistically indistinguishable from the previous one. First, note that for honest clients $u \notin \mathcal{Q}$, since $M_{\mathcal{C}}$ cannot query the **PRG** on input b_u, in both hybrids the value $\boldsymbol{y}_u$ is distributed uniformly at random (and independent from the rest of the view).

Similarly, for honest clients $u \in \mathcal{Q}$, before Round **Unmasking**, $M_{\mathcal{C}}$ cannot query the **PRG** on input b_u, so y_u looks uniformly random as expected. After Round **Unmasking**, when $M_{\mathcal{C}}$ learns b_u, it has exactly the same distribution as in the previous hybrid, i.e. it satisfies

$$\boldsymbol{y}_u - \mathbf{PRG}(b_u) - \sum_{v \in \mathcal{F}_u} \mathbf{PRG}(s_{u,v}) = \boldsymbol{x}_u$$

Thus, this hybrid is indistinguishable from the previous one.

Hyb_{10} This hybrid is defined exactly as the previous one, except that for all $u \in \mathcal{Q} \setminus \mathcal{C}$, instead of programming the random oracle to set $\mathbf{PRG}(b_u)$ to[9]:

$$\begin{aligned} \mathbf{PRG}(b_u) \leftarrow \quad & \boldsymbol{y}_u - \boldsymbol{x}_u - \sum_{v \in \mathcal{F}_u} \mathbf{PRG}(s_{u,v}) \\ = \quad & \boldsymbol{y}_u - \boldsymbol{x}_u - \sum_{v \in \mathcal{Q}} \mathbf{PRG}(s_{u,v}) \\ & - \sum_{v \in \mathcal{F}_u \setminus \mathcal{Q}} \mathbf{PRG}(s_{u,v}) \end{aligned}$$

as in the previous hybrid, SIM instead sets

$$\mathbf{PRG}(b_u) \leftarrow \quad \boldsymbol{y}_u - \boldsymbol{w}_u - \sum_{v \in \mathcal{F}_u \setminus \mathcal{Q}} \mathbf{PRG}(s_{u,v})$$

where $\{\boldsymbol{w}_u\}_{u \in \mathcal{Q} \setminus \mathcal{C}}$ are chosen uniformly at random, subject to $\sum_{u \in \mathcal{Q} \setminus \mathcal{C}} \boldsymbol{w}_u = \sum_{u \in \mathcal{Q} \setminus \mathcal{C}} \boldsymbol{x}_u$. Since, as argued before, $s_{u,v}$'s for $u, v \in \mathcal{Q} \setminus \mathcal{C}$ are never queried by $M_{\mathcal{C}}$, by Lemma 1, in the view of $M_{\mathcal{C}}$, the above values are identically distributed as the previous hybrid.

Hyb_{11} This hybrid is defined as the previous one, with the only difference being that the simulator now does not receive the inputs of the honest parties, but instead, in round **Unmasking**, makes a query to the functionality Ideal for the set $\mathcal{Q} \setminus \mathcal{C}$ and uses the value to sample the required $\boldsymbol{w}_u$ values. Note that since by construction $|\mathcal{Q}| \geq t$, $|\mathcal{Q} \setminus \mathcal{C}| \geq t - n_{\mathcal{C}} = s$, and therefore the functionality Ideal will not return $\perp$.

It is easy to see that this change does not modify the view seen by the adversary, and therefore it is perfectly indistinguishable from the previous one. Moreover, this hybrid does not make use of the honest party's inputs, and this concludes the proof.

□

[9]Notice that $\mathcal{Q}$ must be a subset of $\mathcal{F}_u$, or else u aborts

APPENDIX D
DIFFERENTIAL PRIVACY AND SECURE AGGREGATION

Suppose that each of U users has a vector $\boldsymbol{x_i}$ with an ℓ^2-norm bounded by $\frac{\Delta}{2}$, such that the ℓ^2-sensitivity of $\sum_i \boldsymbol{x_i}$ is bounded by Δ. For $\epsilon \in (0, 1)$, we can achieve (ϵ, δ)-differential privacy for the sum via the Gaussian mechanism [46], by adding zero-mean multivariate Gaussian noise drawn from $\mathcal{N}(\mathbf{0}, \sigma^2 \boldsymbol{I})$, where $\sigma = \frac{\Delta}{\epsilon}\sqrt{2\ln(\frac{1.25}{\delta})}$.

In the local privacy setting, users distrust the aggregator, and so before any user submits her value to the aggregator, she adds noise $\boldsymbol{z_i} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \boldsymbol{I})$, achieving (ϵ, δ)-differential privacy for her own data in isolation. Summing contributions at the server yields $\sum_{i=1}^{U} \boldsymbol{x_i} + \sum_{i=1}^{U} \boldsymbol{z_i}$. Observe that the mean of k normally distributed random variables $\boldsymbol{r_i} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \boldsymbol{I})$ is $\bar{\boldsymbol{r}} \sim \mathcal{N}(\mathbf{0}, \frac{\sigma^2}{k} \boldsymbol{I})$; it follows that the server can form an unbiased estimator of $\bar{\boldsymbol{x}}$ from the user contributions as

$$\hat{\boldsymbol{x}}_{\boldsymbol{LDP}} = \frac{1}{U}\left(\sum_{i=1}^{U} \boldsymbol{x_i} + \sum_{i=1}^{U} \boldsymbol{z_i}\right) \sim \mathcal{N}(\bar{\boldsymbol{x}}, \frac{\sigma^2}{U} \boldsymbol{I}).$$

Now consider a setting wherein a trusted third party is available that can aggregate and privatize batches of n user inputs; for simplicity, assume that U is a multiple of n. The users deliver raw inputs $\boldsymbol{x_i}$ to the third party, who produces $\frac{U}{n}$ batch-sums, each with (ϵ, δ)-differential privacy for users in the batch, by adding $\boldsymbol{z_j} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \boldsymbol{I})$ noise to the batch-sum j before releasing it. Summing the released batch-sums at the server yields $\sum_{i=1}^{U} \boldsymbol{x_i} + \sum_{j=1}^{\frac{U}{n}} \boldsymbol{z_j}$. The server can once again form an unbiased estimator of $\bar{\boldsymbol{x}}$ as

$$\hat{\boldsymbol{x}}_{\boldsymbol{TTP}} = \frac{1}{U}\left(\sum_{i=1}^{U} \boldsymbol{x_i} + \sum_{j=1}^{\frac{U}{n}} \boldsymbol{z_j}\right) \sim \mathcal{N}(\bar{\boldsymbol{x}}, \frac{\sigma^2}{nU} \boldsymbol{I}).$$

Observe that the standard deviation of $\hat{\boldsymbol{x}}_{\boldsymbol{TTP}}$ is a factor of $\frac{1}{\sqrt{n}}$ smaller than that of $\hat{\boldsymbol{x}}_{\boldsymbol{LDP}}$.