Secure Single-Server Aggregation with (Poly)Logarithmic Overhead

James Henry Bell
The Alan Turing Institute
London, UK
jbell@turing.ac.uk

Kallista A. Bonawitz
Google
New York, US
bonawitz@google.com

Adrià Gascón
Google
London, UK
adriag@google.com

Tancrède Lepoint
Google
New York, US
tancrede@google.com

Mariana Raykova
Google
New York, US
marianar@google.com

ABSTRACT

Secure aggregation is a cryptographic primitive that enables a server to learn the sum of the vector inputs of many clients. Bonawitz et al. (CCS 2017) presented a construction that incurs computation and communication for each client linear in the number of parties. While this functionality enables a broad range of privacy preserving computational tasks, scaling concerns limit its scope of use.

We present the first constructions for secure aggregation that achieve polylogarithmic communication and computation per client. Our constructions provide security in the semi-honest and the semi-malicious settings where the adversary controls the server and a γ-fraction of the clients, and correctness with up to δ-fraction dropouts among the clients. Our constructions show how to replace the complete communication graph of Bonawitz et al., which entails the linear overheads, with a k-regular graph of logarithmic degree while maintaining the security guarantees.

Beyond improving the known asymptotics for secure aggregation, our constructions also achieve very efficient concrete parameters. The semi-honest secure aggregation can handle a billion clients at the per-client cost of the protocol of Bonawitz et al. for a thousand clients. In the semi-malicious setting with 10^4 clients, each client needs to communicate only with 3% of the clients to have a guarantee that its input has been added together with the inputs of at least 5000 other clients, while withstanding up to 5% corrupt clients and 5% dropouts.

We also show an application of secure aggregation to the task of secure shuffling which enables the first cryptographically secure instantiation of the shuffle model of differential privacy.

CCS CONCEPTS

• **Security and privacy** → **Cryptography**; **Distributed systems security**; **Privacy-preserving protocols**.

KEYWORDS

multi-party computation; secure aggregation; secure shuffling;

CCS '20, November 9–13, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-7089-9/20/11…$15.00
https://doi.org/10.1145/3372297.3417885

ACM Reference Format: James Henry Bell, Kallista A. Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. 2020. Secure Single-Server Aggregation with (Poly)Logarithmic Overhead In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS'20), November 9–13, 2020, Virtual Event,* ACM, New York, NY, USA, 17 pages.

1 INTRODUCTION

Once considered a purely theoretical tool, cryptographic secure multiparty computation has become a tool that underlies several technological solutions [1, 7, 8, 12, 23, 27, 29]. In this context, constructions for two parties (or a small number of parties) still have dominant presence. One reason for this is the increased complexity that many party solutions bring, which could be a challenge for adoption among a large number of participants. Another reason is the fact that many multiparty solutions require communication channels between all participants, which is not always viable. Further, real scenarios with many parties need to account for the fact that a fraction of the parties may drop out during the execution.

All of these concerns apply to the setting where a service provider collects aggregate statistics from a large population in a privacy preserving way. This includes basic statistical tasks such as computing mean, variance, and histograms, as well as large scale distributed training of machine learning models as in federated learning [7, 25]. In such settings there is a powerful central server and a large number of clients with constrained resources, a single communication channel only to the server, and intermittent network connectivity that results in a significant probability of dropping out during the protocol execution, a common problem in production systems [7].

While there have been both theoretical and applied works proposing secure computation solutions for settings with restricted communication [10, 14, 22, 26, 30, 33], Bonawitz et al. [8] introduced the first practical secure computation construction whose implementation scaled to a thousand clients, a larger number of parties than any existing system. That work presents a secure aggregation protocol that enables a central server to learn the summation of the input vectors of many clients securely, i.e. without obtaining any information beyond the sum. The protocol is also robust in the presence of a fraction of clients dropping out. That paper and subsequent work [7] showed that secure vector summation enables powerful privacy-preserving functionalities such as federated learning [25].

In this work we focus on two aspects of secure vector aggregation. First, we construct two new protocols with semi-honest and malicious security, which provide better efficiency both in terms of asymptotics as well as concrete costs. Second, we present a new application of secure aggregation for construction of secure shuffling

protocols. This enables anonymous data collection in the single-server setting, and in particular provides the first cryptographically secure instantiation of the shuffle model of differential privacy [6].

Efficiency of Secure Aggregation. While the existing secure aggregation construction of Bonawitz et al. [8] is sufficiently efficient to run in production systems [7], scaling concerns limit its scope of use. Its limitations stem from the computation and communication complexity of the protocol. For adding n length l vectors (one provided by each client) their protocol requires $O(n^2 + ln)$ computation and $O(n + l)$ communication per device, and $O(ln^2)$ computation and $O(n^2 + ln)$ communication for the server. This introduces *linear overhead* over the computation in the clear where every client sends a vector and the server adds up all n vectors. Although recently a variant with polylog overhead has been proposed [34], it requires $n/\log n$ rounds (as opposed to four for Bonawitz et al.) and, contrary to Bonawitz et al.'s approach, relies on revealing a partial sum of $\log n$ input values to a coalition of the server and a client.

Reducing client compute time is significant: the more compute time is required at each device, the more likely that device is to drop out. This not only results in wasted computation, but also induces bias as powerful devices with fast connection will be overrepresented in the collected statistics. In practice, these costs limit the use of secure aggregation to settings of no more than approximately a thousand devices for large values of l, e.g., larger than 10^6. This prevents computing large histograms or training neural networks that require large client batches to achieve good quality [31].

Looking beyond concrete efficiency, current theoretical constructions do not provide constant round solutions with sublinear communication (in the number of clients) per client. This is the case even in the semi-honest setting when we need to account for the key distribution phase as well as support dropouts [33]. Note that there exist relevant solutions based on homomorphic encryption (HE) [17, 18, 32], where the server computes the sum of encrypted inputs under a key shared among the clients. However, the generation of shared HE parameters among all parties with sublinear communication and in a way that is robust to dropouts remains a challenge. Boyle et al. [9] present efficient large-scale secure computation but do use a broadcast channel per party.

Amplification by Shuffling. Differential privacy (DP) [13] has become the de facto notion of individual privacy in data analysis. Until recently there have been two main threat models for DP: the central model, where a curator is trusted with all private inputs and the task of outputting privatized aggregates, and the local DP setting where individuals release DP versions of their data. While the second model has minimal trust assumptions it also comes with significant limitations in terms of accuracy.

The recently introduced shuffle model of DP [6, 11] assumes only a trusted shuffler (a party that applies a random permutation to input data before publishing it) rather than a trusted curator computing arbitrary functionalities. The shuffle model matches exactly the setting of a single powerful server and a large number of devices in a star network. Several recent works [2, 3, 15, 16, 19] have shown that this model offers a fruitful middle ground (in the terms of tradeoffs between trust distribution and accuracy) between the local and curator models. Implementing efficient shufflers in practice has either required reliance on trusted computing hardware or onion-routing/mixnet constructions, which require strong non-collusion assumptions and significantly increased communication. While we can implement the shuffling step with general multiparty computation to achieve local DP privacy, any practical deployment would require an efficient shuffling construction.

1.1 Our contributions

Our paper has three main contributions: two new constructions for secure aggregation which provide security in the semi-honest and semi-malicious setting, and a new construction for secure shuffling based on secure aggregation.

For our constructions we consider an aggregation server and n clients with input vectors of length l. The goal of the protocol is to provide the server with a summation of the inputs of all clients that complete the protocol execution. We require that correctness holds with all but $2^{-\eta}$ probability, where $\eta > 0$ is the correctness parameter. The protocols are secure in the presence of an adversary controlling the server and up to an arbitrary γ-fraction of the clients that are corrupted independently of the protocol execution. In other words, client corruptions happen before the protocol execution starts. Note that the only assumption is that the adversary does not have the ability to compromise new devices adaptively as the protocol progresses. Our protocols are robust to an arbitrary δ-fraction of clients dropping out during the protocol execution. Moreover, a number of dropouts that exceeds that threshold can only lead to an aborted execution, and does not affect the security of the protocol. Let us remark that the set of clients that dropout during the protocol execution are considered public knowledge, i.e. we do not aim to provide full anonymity to the set of dropout users. Our constructions achieve information theoretic security with a security parameter σ, except for the use of a key agreement protocol for randomness generation, and encryption and signature for secure communication.

Semi-honest Construction. The construction of Bonawitz et al. [8] uses the server as a relay that forwards encrypted and authenticated messages between clients. Their solution requires that every pair of clients are able to communicate. Intuitively, the complete communication graph serves both security and dropout robustness. Roughly speaking, every client (a) negotiates shared randomness with every other client to mask their submitted value, and (b) shares (with threshold t) their random seeds with every other client. While (a) ensures security, (b) guarantees that the protocol can recover from dropouts without compromising security as long as t is set appropriately.

The main insight to our efficiency improvement is that a complete graph is not necessary: it is enough to consider a k-regular communication graph, i.e., each client speaks to $k < n - 1$ other clients, where $k = O(\log n)$. We obtain this result by using a randomized communication graph construction, and then leveraging its properties with respect to the distributions of corrupt clients and dropouts.

Our semi-honest construction requires $O(\log^2 n + l \log n)$ computation and $O(\log n + l)$ communication per client, and $O(n \log^2 n + nl \log n)$ computation and $O(n \log n + nl)$ communication for the server. It requires three rounds of interactions between the server and clients. We characterize the properties of the communication

graph that suffice for the security and correctness of the resulting protocol, and present a graph generation construction with concrete parameters. For example, with $\sigma = 40$ and $\eta = 30$, we need only $k = 150$ neighbors per client in order to run the protocol with $n = 10^8$ clients and provide security for up to $\gamma = 1/5$ corrupt nodes and $\delta = 1/20$ dropouts. In fact, we can run our protocol with a billion clients, while incurring roughly the same costs per client as the protocol of Bonawitz et al. [8] when run on a thousand clients (see Section 5 for more details).

Semi-malicious Construction. Our semi-malicious setting assumes that the server behaves honestly in the first step of the protocol when it commits to the public keys of all clients. After this point, it can deviate arbitrarily from the protocol (as in the usual malicious security notion) and our construction provides security. This is analogous to the assumption in [8], and weaker than assuming a public key infrastructure for key distribution.

The security definition for the malicious case is a bit more involved, and is discussed in detail in Section 4. Roughly speaking, this is due to the fact that a malicious server can disrupt communication between parties at any round, and thus can simulate dropouts inconsistently across clients. As it is impossible for clients to distinguish real from simulated dropouts, a malicious server cannot be prevented from excluding $(\gamma + \delta)$ clients from the final sum *by definition* of the summation functionality itself. Instead of requiring that a malicious server cannot learn the sum of the inputs of less than $(1 - \gamma - \delta)n$ clients, as in the semi-honest case, we formalize and prove that our protocol ensures that the server can only learn sums including at least a constant fraction α of the clients' inputs. In other words, every honest client is guaranteed that their input will be added with *at least* $\alpha(1-\gamma)n$ other inputs from honest clients even when the malicious server is controlling γn other clients.

Bonawitz et al. [8] also propose a semi-malicious version of their protocol. The main idea there is to add their semi-honest variant a round in which clients verify that the server reported consistent views of dropouts to all of them. This extension incurs additional linear communication and computation. Extending our semi-honest protocol while maintaining sublinear overhead is more challenging. First, the server cannot be trusted to generate the communication graph honestly, and thus we propose a protocol where clients choose their $k = O(\log n)$ neighbors in a distributed verifiable way. Second, we find an alternative approach to ensuring global consistency of reported dropouts by having each client perform only a local verification on their neighborhood. We then prove that this corresponds to a global property of the communication graph thanks to the connectivity properties of random graphs.

Our semi-malicious construction requires $O(\log^2 n + l \log n)$ computation and $O(\log^2 n + l)$ communication per client, and $O(l \log^2 n)$ computation and $O(\log^2 n + ln)$ communication for the server. It runs in five and a half rounds of interactions. Our protocol also achieves very efficient concrete costs. For example, with $\sigma = 40$ and $\eta = 30$, if we run the protocol with 10^4 clients and corrupt and dropout rates $\gamma = \delta = 0.05$ we need only 300 neighbors to guarantee that every client's input is aggregated with the inputs of at least 5000 clients (see Section 5 for more details).

Secure Shuffle Construction. We provide an instantiation of the shuffle model of differential privacy by showing a reduction of shuffling to vector summation. Our solution leverages a randomized data structure called an invertible Bloom lookup table (IBLT) [21]. To shuffle m messages distributed among n clients, it suffices to run a single execution of secure vector summation with vectors of length $\sim 2m$. This covers the case where each user has multiple messages to send, as in the multi-message shuffle model [4, 11, 19], as well as the case where most users do not have any input, which models submissions of error reports.

2 PRELIMINARIES AND NOTATION

Hypergeometric distribution. We recall that the Hypergeometric distribution $\mathtt{HyperGeom}(n, m, k)$ is a discrete probability distribution that describes the probability of s successes in k draws, without replacement, from a finite population of size n that contains exactly m objects with that feature. We use the following two tail bounds for $X \sim \mathtt{HyperGeom}(n, m, k)$: (i) $\forall d > 0 : \Pr[X \leq (m/n - d)k] \leq e^{-2d^2k}$, and (ii) $\forall d > 0 : \Pr[X \geq (m/n + d)k] \leq e^{-2d^2k}$. Moreover, by choosing $d = 1 - m/n$, we get that $\Pr[X \geq k] = \Pr[X \leq k] \leq e^{-2(1-m/n)^2k}$

Graphs. We denote a graph with a vertex set V and edge set E as $\mathsf{G}(\mathsf{V}, \mathsf{E})$, where $(i, j) \in \mathsf{E}$ if there is an edge between vertices i and j. The set of all nodes connected to the i-th node is its neighbors $\mathsf{N}_\mathsf{G}(i) = \{j \in \mathsf{V} : (i, j) \in \mathsf{E}\}$. A graph $\mathsf{G}'(\mathsf{V}', \mathsf{E}')$ is a subgraph of $\mathsf{G}(\mathsf{V}, \mathsf{E})$ if $\mathsf{V}' \subseteq \mathsf{V}$ and $\mathsf{E}' \subseteq \mathsf{E}$. The subgraph of G induced by a subset of the vertices $\mathsf{V}' \subset \mathsf{V}$ and the edges between E' where $(i, j) \in \mathsf{E}'$ if and only if $(i, j) \in \mathsf{E})$ and $i, j \in \mathsf{V}'$ is denoted $\mathsf{G}[\mathsf{V}']$.

Parameters. We provide in Table 1, Appendix A, the parameters we will use throughout the paper. In particular, σ will denote an *information-theoretic* security parameter bounding the probability of bad events happening and η will denote the correctness parameter. We denote by λ a security parameter associated with standard cryptographic primitives (such as Shamir secret sharing, pseudorandom generator, and authenticated encryption).

We says that two distributions $\mathcal{D}, \mathcal{D}'$ are computationally indistinguishable with respect to σ and λ, denoted $\mathcal{D} \approx_{\sigma,\lambda} \mathcal{D}'$, if the statistical distance between $\mathcal{D}$ and $\mathcal{D}'$ is bounded by the sum of a negligible function in λ and of a negligible function in σ.

Throughout the paper, we denote $\mathbb{X} = \mathbb{Z}/R\mathbb{Z}$ the domain on which the summation protocol is performed, and we assume the representation of elements of $\mathbb{X}$ (resp. computational cost of operations in $\mathbb{X}$) is $\tilde{O}(1)$ in n (resp. $\log(n)$) so as to enable additions of n elements in $\mathbb{X}$ without overflow.

Cryptographic primitives. In our protocols, we will use the following cryptographic primitives for randomness generation and secure communication. A signature scheme scheme that is existentially unforgeable under chosen message attacks (EUF-CMA); for example, it can be instantiated with ECDSA in practice. A cryptographically secure pseudorandom generator $F\colon \{0,1\}^\lambda \to \mathbb{X}^l$; for example, it can be instantiated with AES-CTR in practice [7, 8]. An authenticated encryption scheme with associated data (AEAD), which is semantically secure under a chosen plaintext attack (IND-CPA) and provides integrity of ciphertext (INT-CTXT), which means that it is computationally infeasible to produce a ciphertext not previously

produced by the sender regardless of whether or not the underlying plaintext is "new"; for example, it can be instantiated with ChaCha20+Poly1305 [24] in practice. A λ-secure key-agreement protocol, i.e., a key-agreement protocol such that there exists a simulator $\mathsf{Sim}_{\mathcal{KA}}$, which takes as input an output key sampled uniformly at random and the public key of the other party, and simulates the messages of the key agreement execution so that the statistical distance is negligible in λ; for example, it can be instantiated with a Diffie–Hellman key agreement protocol followed by a key derivation function in practice.

3 THE SEMI-HONEST PROTOCOL

In this section, we present our semi-honest summation protocol. Our construction is parametrized by a (possibly random) undirected regular graph G with n nodes and degree k. Intuitively the graph G will determine the direct communication channels that will be used in the protocol in the following sense: clients that are connected in G will exchange private messages in the protocol via the server which, however, will not be able to see the message content. We will prove the correctness and the security of our protocol assuming a set of properties of the graph G. Next we will describe a *randomized* algorithm called GenerateGraph, which generates graphs for which these properties hold with high probability. Since we are in the semi-honest setting this algorithm can be generated by the server (in the malicious setting protocol of Section 4, we will describe a distributed graph generation protocol).

3.1 An *abstract* summation protocol

We present our protocol in Algorithm 2 which can be found in the appendix. It runs among n clients with identifiers $1, \ldots, n$ and the server. All parties have access to the following primitives: a pseudorandom generator (PRG) F, which is used to expand short random keys, a secure key agreement protocol $\mathcal{KA}$ to create shared random keys, and an authenticated encryption scheme for private communication $\mathcal{E}_{auth}$.

Construction Overview. The main idea of our construction is a generalization of the secure aggregation protocol of Bonawitz et al. [8], which only works with *complete* graphs (i.e., all the vertices are connected between each other), that works with any graph sampled from a larger set of sparser graphs. Our construction enables significant efficiency improvements.

As we discussed above, the first step of the protocol will be to generate a k-regular graph G and a threshold $1 \leq t \leq k$, where the n vertices are the clients participating to the protocol. To do this the server runs a *randomized* graph generation algoritmhm GenerateGraph that takes the number n of clients/nodes and samples output (G, t) from a distribution $\mathcal{D}$. Below, we will define which properties of this distribution suffice for the proofs of correctness and security.

The edges of the graph determine pairs of clients each of which run a key agreement protocol to share a random key, which later will be used by each party to derive a mask for her input. More precisely, each client i generates key pairs (sk_i^1, pk_i^1), (sk_i^2, pk_i^2) and sends (pk_i^1, pk_i^2) to all of her neighbors. Then, each pair (i, j) of connected parties G runs a key agreement protocol $s_{i,j} = \mathcal{KA}.Agree(sk_i^1, pk_j^1)$, which uses the keys exchange in the previous step to derive a shared random key $s_{i,j}$.

Each client i derives pairwise masks for her input $m_{i,j} = F(s_{i,j})$ derived from shared keys with each of her neighbors $j \in \mathsf{N}_\mathsf{G}(i)$, which she adds to her input as follows

$$\vec{x_i} - \sum_{j \in \mathsf{N}_\mathsf{G}(i), j<i} \vec{m_{i,j}} + \sum_{j \in \mathsf{N}_\mathsf{G}(i), j>i} \vec{m_{i,j}}.$$

In the setting where all parties submit their masked inputs, all pairwise masks cancel in the final sum. However, to support execution when dropouts occur, the protocol needs to enable removal of the pairwise masks of dropout clients (who never submitted their masked inputs). For this purpose, each client i shares her key sk_i^1 to her neighbors j's by sending a ciphertext containing the share produced using the public keys pk_j^2's. Later, if client i drops out, her neighbors can send the decrypted shares to the server. Armed with those shares, the server can reconstruct the secret key sk_i^1 and use it together with the public keys of i's neighbors to compute $s_{i,j}$. Finally, the server can recover the corresponding pairwise masks $\vec{m_{i,j}}$ and remove them from the final output sum.

The above approach has a shortcoming that if the server announces dropouts and later some masked inputs of the claimed dropouts arrive, the server will be able to recover those inputs in the clear. To prevent this possibility the protocol introduced another level of masks, called self masks, that each client generates locally $\vec{r_i} = F(b_i)$ from a randomly sampled seed b_i. This mask is also added to the input

$$\vec{x_i} + \vec{r_i} - \sum_{j \in \mathsf{N}_\mathsf{G}(i), j<i} \vec{m_{i,j}} + \sum_{j \in \mathsf{N}_\mathsf{G}(i), j>i} \vec{m_{i,j}}.$$

Now, client i also shares b_i to her neighbors. Later, if i submitted her masked input, the server will instead request shares of b_i from the client's neighbors in order to reconstruct and remove $\vec{r_i}$ from the sum. In other words, either client i has submitted her masked input and the server will obtain shares from the mask b_i, or client i has dropped out and the server will obtain shares of sk_i^1. Crucially, we require each client to provide to the server only one share for each if her neighbors. This guarantees that the masked inputs of clients that are not included in the final sum cannot be revealed in the clear to the server.

Dropouts may happen throughout the steps of the protocol. We denote by A_1 the set of parties that send their secret shares to their neighbors, $A_2 \subseteq A_1$ is the set of parties that send their masked inputs with their self mask and the pairwise masks generated from the shared keys with her neighbors in A_1, $A_3 \subseteq A_2$ is the set of clients that send shares to the server to be used in the reconstruction of the output. At each of these steps the server will only wait a set time for these messages, A_i' denotes the subset of A_i whose messages arrive on time. If the complements of these prime sets becomes larger than the threshold δn for dropouts, the server aborts. Also if a client has less than t neighbors in A_3', the server aborts since it cannot reconstruct at least one mask needed to obtain the output.

The construction of Bonawitz et al. [8] uses a complete graph where each client shares a mask with every other client in the system. While a single random mask hides perfectly a private value, the intuition of why we need more masks is the presence of corrupt

clients, who will share their masks with the server, and of dropouts whose masks will be removed. However, we will show that $n-1$ masks per input may be more than what is needed for security. In particular, the insight in our construction is that the number of such masks can be significantly reduced to $O(\log n)$, in a setting where we can assume that the pairs of parties sharing common randomness used to derive masks are chosen at random and independent of the set of corrupted parties and the set of dropouts. In particular we model this by using a random k-regular graph that determines the node neighbors with whom masks are shared. In our security proofs, we will argue that, when $k = O(\log n)$, for each honest client there is a sufficient number of honest non-dropout neighbors to protect the client's input.

Graph Properties. Let $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ be a k-regular undirected graph with n nodes, and let $0 < t < k$ be an integer. Recall that $\mathsf{N}_{\mathsf{G}}(i) = \{j \in \mathsf{V} : (i,j) \in \mathsf{E}\}$ is the set of neighbors of i.

The first property that we require from any graph output by GenerateGraph is that, for every set of corrupt clients C, with all but negligible probability, no honest client i has t neighbors in C. Note that this happening would immediately break security, as the adversary would be able to recover the secrets of i by combining shares from i's corrupted neighbors. Formally, we define the event E_1 as a predicate on a set C and a pair (G, t) that is 1 iff the "good" property holds.[1]

Definition 3.1 (Not too many corrupt neighbors). Let $C \subset [n]$ be such that $|C| \leq \gamma n$. We define event E_1 as

$$E_1(C, \mathsf{G}, t) = 1 \text{ iff } \forall i \in [n] \setminus C : |\mathsf{N}_{\mathsf{G}}(i) \cap C| < t$$

Event E_1 is not the only event related to the communication graph G that could break security: consider the situation where a set of D clients, with $|D| \leq \delta n$, drop out right before sending their masked input in such a way that removing clients/nodes in D disconnects the communication graph G, i.e. $\mathsf{G}[[n] \setminus D]$ is not connected. Now, as discussed above, edges in the graph correspond to shared masks $\vec{m_{i,j}}$ that are crucial to ensure security, as these masks are used to mask values $\vec{y_i}$ that involve private inputs. However, if the graph gets disconnected by D, and $|D| \leq \delta n$, then by definition of the functionality (δ-robustness) the server would be able to recover the masks $m_{i,j}$ involving clients in D, i.e. the edges in the cut induced by D. This implies that the server would receive at least two disjoint sets S_1, S_2 of masked inputs (one for each connected component in $\mathsf{G}[[n] \setminus D]$) whose masks cancel independently of the other set, resulting in the server learning (at least) two partial sums, which breaks security. We state our "good" event E_2 where the graph remains connected. Note that although we excluded corrupted nodes in the above description for simplicity, they need to be taken into account. As before, we define E_2 as a predicate on sets C, D of appropriate size, and a graph G.

Definition 3.2 (Connectivity after dropouts). Let $C \subset [n]$ and $D \subset [n]$ be such that $|C| \leq \gamma n$ and $|D| \leq \delta n$. We define event E_2 as

$$E_2(C, D, \mathsf{G}) = 1 \text{ iff } \mathsf{G}[[n] \setminus (C \cup D)] \text{ is connected}$$

[1] Note that for a complete graph, the event E_1 is 1 for every set C trivially, as long as $t > |C|$, where $|C| \leq \gamma n$. However this implies that $k \geq t = O(n)$, and results in linear overhead.

Note that the above event trivially holds for a complete graph G (and reasonable parameters γ and δ) and any sets C, D, as in that case $k = n - 1 > (\gamma + \delta)n$.

Perhaps surprisingly, we will prove that events E_1 and E_2 capture all possible ways in which security could be broken (assuming perfect cryptographic primitives) due to the choice of communication network. More concretely, we will show the following: Consider a graph generation algorithm GenerateGraph such that for any sets C, D of appropriate sizes, a pair (G, t) generated using GenerateGraph will satisfy $E_1(C, \mathsf{G}, t) \wedge E_2(C, D, \mathsf{G}) = 1$ except for negligible probability. Then, GenerateGraph can be used in Algorithm 2, and the result will be a secure protocol.

Finally, we still need one more property that GenerateGraph must satisfy to ensure *correctness*. Note that if after removing dropouts some client does not have at least t neighbors then the server can't recover the final sum.

Definition 3.3 (Enough surviving neighbors for reconstruction). Let $D \subset [n]$ such that $|D| \leq \delta n$. We define event E_3 as

$$E_3(D, \mathsf{G}, t) = 1 \text{ iff } \forall i \in [n] : |\mathsf{N}_{\mathsf{G}}(i) \cap ([n] \setminus D)| \geq t$$

3.2 Generating "Good" Graphs

This section characterizes "good" graph generation algorithms as those that generate graphs for which events E_1, E_2, E_3 hold with probability parameterized by a statistical security parameter σ (for E_1 and E_2) and a correctness parameter η (for E_3).

Definition 3.4. Let $\mathcal{D}$ be a distribution over pairs (G, t). We say that $\mathcal{D}$ is (σ, η)-good if, for all sets $C \subset [n]$ and $D \subset [n]$ such that $|C| \leq \gamma n$ and $|D| \leq \delta n$, we have that

(1) $\Pr[E_1(C, \mathsf{G}', t') \wedge E_2(C, D, \mathsf{G}', t') = 1 \mid (\mathsf{G}', t') \leftarrow \mathcal{D}] > 1 - 2^{-\sigma}$
(2) $\Pr[E_3(D, \mathsf{G}', t') = 1 \mid (\mathsf{G}', t') \leftarrow \mathcal{D}] > 1 - 2^{-\eta}$

Analogously, we say that a graph generation algorithm is (σ, η)-good if it implements a (σ, η)-good distribution.

In Section 3.5, we describe a concrete (randomized) (σ, η)-good graph generation algorithm.

3.3 Correctness and Security

In this section, we state our correctness and security theorems, whose proofs are provided in Appendix B of our full paper [5]. We note that the proof of security uses a standard simulation-based approach [20, 28] similar to the one by Bonawitz et al. [8]. It is important to remark that this formulation does not in general imply security, in the formal sense of [20, 28], in the weaker threat model where the adversary only corrupts a set of clients and the server is honest. This is however easy to see for our protocol: note that the messages sent to the clients are all functions of the other clients' randomness alone, and in particular do not depend at all on any inputs. We discuss further the honest server case in the context of the semi-malicious threat model in Section 4.6.

Theorem 3.5 (Correctness). *Let Π be Algorithm 2 instantiated with a (σ, η)-good graph generation algorithm GenerateGraph. Consider an execution of Π with inputs $\mathcal{X} = (\vec{x_i})_{i \in [n]}$. If $|A_3'| \geq (1-\delta)n$, i.e., less than a fraction δ of the clients dropout, then the server does not abort and obtains $\vec{z} = \sum_{A_2'} \vec{x_i}$ with probability $1 - 2^{-\eta}$.*

Theorem 3.6 (Security). *Let σ, η, λ be integer parameters. Let Π be an instantiation of Algorithm 2 with a (σ, η)-good graph generation algorithm GenerateGraph, a IND-CPA secure authenticated encryption scheme, and a λ-secure key agreement protocol. There exists a PPT simulator* Sim *such that for all k, all sets of surviving clients A_1, A_2, A_2', A_3 as defined in Algorithm 2, all inputs $\mathcal{X} = (\vec{x}_i)_{i\in[n]}$, and all sets of corrupted clients C with $|C| \leq \gamma n$, denote $\vec{z} = \sum_{i\in A_2'} \vec{x}_i$, the output of* Sim *is perfectly indistinguishable from the joint view of the server and the corrupted clients* $\mathsf{Real}_C(A_1, A_2, A_2', A_3)$ *in that execution, i.e.,* $\mathsf{Real}_C(A_1, A_2, A_2', A_3) \approx_{\sigma,\lambda} \mathsf{Sim}(\vec{z}, C, A_1, A_2, A_2', A_3)$.

3.4 Performance Analysis

We report the communication and computation costs for the client and server when $k = O(\log n)$. We recall that we assume that basic operations and representation of elements in $\mathbb{X}$ are $O(1)$.

Client computation: $O(\log^2 n + l \log n)$. Each client computation can be broken up as $2k$ key agreements and k encryptions ($O(k)$ complexity), creating twice t-out-of-k Shamir secret shares ($O(k^2)$ complexity), generating $\vec{m_{i,j}}$ for all neighbors j ($O(kl)$ complexity).

Client communication: $O(\log n + l)$. Each client performs $2k$ key agreements ($O(k)$ messages), sends $2k$ encrypted shares ($O(k)$ messages), sends a masked input ($O(l)$ complexity), reveals up to $2k$ shares ($O(k)$ messages).

Server computation: $O(n(\log^2 n + l \log n))$. The server computation can be broken up as reconstructing t-out-of-k Shamir secrets for each client ($O(n \cdot k^2)$ complexity), generating values $\vec{m_{i,j}}$ for all (dropped out) neighbors j of each client i ($O(nkl)$ complexity).

Server communication: $O(n(\log n + l))$. The server receives or sends $O(\log n + l)$ to each client.

3.5 Our Random Graph Constructions

Since Bonawitz et al. [8] uses a complete graph, all of the events $E_1(C, \mathrm{G}, t)$, $E_2(C, D, \mathrm{G})$ and $E_3(D, \mathrm{G}, t)$ are deterministically equal to 1. That is to say that the complete graph is (σ, η)-good for any σ and η. In this section we will describe how to construct a much sparser random graph, which is still (σ, η)-good for reasonable σ and η.

Our randomized construction is shown is Algorithm 1, and consists of uniformly renaming the nodes of a graph known as a *Harary graph* with n nodes and degree k. This graph, which we denote Harary(n, k), has vertices $\mathsf{V} = [n]$ and an edge between two distinct vertices i and j if and only if $j - i$ modulo n is $\leq (k+1)/2$ or $\geq n - k/2$. Roughly speaking, you can think of this as writing the nodes of the graph in a circle and putting edges between those within distance $k/2$ of each other.

Our whole problem now reduces to defining exactly the function ComputeDegreeAndThreshold such that the values of k and t it returns result in GenerateGraph being (σ, η)-good. This in turn leads to a secure protocol, as we saw in the previous section. More concretely, we will see in this section that choosing $k \geq O(\log n + \sigma + \eta)$ is enough to achieve the (σ, η)-goodness property.

Consider any graph generation algorithm G constructed by sampling k neighbors uniformly and without replacement from the set of remaining $n - 1$ clients (as done in Algorithm 1). This general property is all we need to argue about events E_1, E_3 in the definitions of (σ, η)-good, so we don't need to get into the specifics of Harary graphs yet.

Algorithm 1: GenerateGraph

Public Parameters: Max. fraction of corrupt nodes γ, max. fraction of dropout nodes δ.
Input: Number of nodes n, statistical security parameter σ, correctness parameter η.
Output: A triple (G, t, k)
(k, t) = ComputeDegreeAndThreshold$(n, \gamma, \delta, \sigma, \eta)$
Let H = Harary(n, k)
Sample a random permutation $\pi : [n] \mapsto [n]$
Let G be the set of edges $\{(\pi(i), \pi(j)) \mid (i, j) \in H\}$
return (G, t, k)

E_1: Not too many corrupt neighbors. Let us first focus on the event $E_1(C, \mathrm{G}, t)$, which holds if every client i has fewer than t corrupt neighbors in $\mathsf{N}_\mathrm{G}(i)$ (Definition 3.1). Let $\mathsf{X_i}$ be the random variable counting the number of malicious neighbors of a user i, and note that $\mathsf{X_i} \sim \mathtt{HyperGeom}(n-1, \gamma n, k)$, i.e. $\mathsf{X_i}$ is hypergeometrically distributed. Thus by a union bound across all clients we have that $\Pr[E_1(C, \mathrm{G}, t) = 0] \leq nPr[\mathsf{X_i} \geq t] = n(1 - \mathrm{cdf}_{X_i}(t-1))$.

E_3: Not too many neighbors drop out. Let us now turn our attention towards correctness: if we set t too large then the server will fail to recover enough shares of a required mask and abort, and that would result in a wasted computation. The intuition behind this event for G is analogous to the case of E_1, as if $\mathsf{Y_i}$ is the number of surviving (not dropped out) neighbors of the ith user we have that $\mathsf{Y_i} \sim \mathtt{HyperGeom}(n-1, (1-\delta)n, k)$, thanks again to the fact that G is such that the k neighbors of i are randomly sampled from $[n] \setminus \{i\}$. Hence, again by a union bound across clients, we have that $\Pr[E_3(C, \mathrm{G}, t) = 0] \leq nPr[\mathsf{Y_i} \leq t] = n(\mathrm{cdf}_{Y_i}(t))$.

Hypergeometrics (like Binomials) are concentrated around their mean and have sub-gaussian tails. This means that $\Pr[\mathsf{X_i} \geq t]$ decreases exponentially fast as t gets away from $E[\mathsf{X_i}] = \gamma n/(n-1)$; thus it is possible to make both of the above probabilities very small.

The Security/Correctness tradeoff. To gain additional intuition, let us now discuss the interaction between E_1 and E_3, as they correspond to the tension between correctness and security in out protocol: For a fixed k, one should be setting $t \in (0, k)$ according to $\mathsf{X_i}$ for security, while simultaneously satisfying correctness with respect to $\mathsf{Y_i}$. Large t achieves results in better security (because the probability of E_1 not holding decreases, while smaller t helps correctness by reducing the failure probability associated to E_3). Figure 1 visually illustrates the situation by showing the probability mass function of both $\mathsf{X_i}$ and $\mathsf{Y_i}$ for $n = 10^4$, $k = 200$, $\gamma = 1/5$, $\delta = 1/10$, and a choice of threshold $t = 100$ that gives $\Pr[X_i \geq t] < 2^{-40}$ and $\Pr[Y_i < t] < 2^{-30}$. Note that as the probability mass function of hypergeometric distributions has a closed form we can numerically compute these values accurately. We exploit this fact to get tighter bounds than the analytical ones in Section 5.

E_2: Connectivity after dropouts. Up to this point, we only used the fact that the neighbors of i in G are random subsamples of $[n] \setminus \{i\}$.

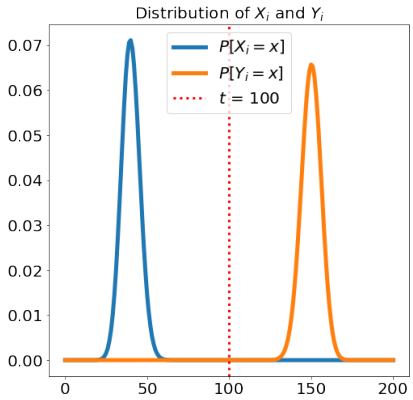

Figure 1: Probability mass functions of $\mathsf{X_i} \sim \mathtt{HyperGeom}(n-1, \gamma n, k)$ and $\mathsf{Y_i} \sim \mathtt{HyperGeom}(n-1, (1-\delta)n, k)$.

To argue about E_2 we will leverage the connectivity properties of Harary graphs. Consider a graph $H = \mathtt{Harary}(n, k)$ and assume that k is even w.l.o.g. Recall that these graphs can be easily constructed by putting nodes $1, \ldots, n$ in a circle and connecting each node to the $k/2$ nodes to its left and the $k/2$ nodes to its right (around the circle). The property that we will use in our argument is that to disconnect H one needs to remove at least $k/2$ successive nodes. To see this consider any way of removing nodes, assume without loss of generality that node 1 is not removed and assume that some node is not connected to 1 (and has not been removed). Let m be such a node of smallest index. Consider nodes $m - k/2, \ldots, m-1$. None of these can be 1 otherwise m would be directly connected to 1. Furthermore none of them can be connected to 1 and present as otherwise m would be connected to 1 via them, but as m was minimal none of them can be disconnected from 1 and present. Therefore they must all be missing, as required.

This property implies that, as G from Algorithm 1, is simply H but with nodes randomly renamed, we have that $\Pr[E_2(C, D, \mathrm{G}') = 0] \leq n(\gamma+\delta)^{k/2}$, by a union bound across clients and the fact that $(\gamma+\delta)^{k/2}$ is an upper bound on the probability that $k/2$ "successive" nodes following a particular node in G are in $C \cup D$ (recall that $|C| \leq \gamma n$ and $D \leq \delta n$).

The following lemma captures the three points we have made so far. Let us remark that it does not tell us immediately how large k should be. Instead, it states sufficient (efficiently checkable) conditions that would imply that a given k was secure given the rest of the parameters, and thus it will become central in Section 5.

Lemma 3.7. *Let $n > 0$ be a set of clients, let σ, η be security and correctness let $\gamma, \delta \in [0,1]$ be the maximum fraction of corrupt and dropout clients, respectively, and let k, t be natural numbers such that $t \in (0, k)$. Let*

$$\mathsf{X_i} \sim \mathtt{HyperGeom}(n-1, \gamma n, k), \ \mathsf{Y_i} \sim \mathtt{HyperGeom}(n-1, (1-\delta)n, k)$$

be random variables. If the following two constraints hold then the distribution $\mathcal{D}$ over pairs (G, t) implemented by Algorithm 1 is (σ, η)-good:

(1) $1 - \mathrm{cdf}_{X_i}(t-1) + (\delta+\gamma)^{k/2} < 2^{-\sigma}/n$
(2) $\mathrm{cdf}_{Y_i}(t) < 2^{-\eta}/n$

Equipped with the above observation, in the rest of this section we show that in fact $k \geq O(\log n + \sigma + \eta)$ suffices, while giving some evidence that the hidden constant is in fact small (that point will be addressed fully in Section 5). The following lemmas and theorem follow from the tail bounds on the hypergeometric distribution stated in the preliminaries section. Their detailed proofs can be found in Appendix A of our full paper [5].

Lemma 3.8. *Let G be such that, for all $i \in [n]$, $\mathrm{G}(i)$ is a uniform sample of size k from $[n] \setminus \{i\}$. Let $C \subset [n]$ be such that $|C| \leq \gamma n$, and $t = \beta k$. If $k \geq c(\sigma_1 \log 2 + \log n)$ and $c > \frac{1}{2(\beta - \frac{n\gamma}{n-1})^2}$ then* $\Pr[E_1(C, \mathrm{G}, t) = 0] \leq 2^{-\sigma_1}$.

Let us now turn our attention towards correctness. Maybe not surprisingly at this point, it turns out that $k > O(\eta + \log n)$, with a small constant depending on the dropout rate δ is enough to ensure a failure probability bounded by $2^{-\eta}$, as we show in the next lemma.

Lemma 3.9. *Let G be such that, for all $i \in [n]$, $\mathrm{G}(i)$ is a uniform sample of size k from $[n] \setminus \{i\}$. Let $D \subset [n]$ such that $|D| \leq \delta n$ and let $t = \beta k$. If $k \geq c(\eta \log 2 + \log n)$ and $c > \frac{1}{2\left(\frac{n(1-\delta)}{n-1} - \beta\right)^2}$ then* $\Pr[E_3(D, \mathrm{G}, t) = 0] \leq 2^{-\eta}$.

It is important to remark that our previous two lemmas did not rely entirely on our Harary graph construction. In fact any algorithm that results in $\mathsf{X_i}$ and $\mathsf{Y_i}$ being concentrated would work. These include Erdős-Rényi graphs, as well as a distributed construction where every client samples k neighbors at random (as done in the malicious version of our protocol presented in the next section). However, as discussed above to address our remaining property $E_2(C, D, \mathrm{G})$ we will heavily leverage the Harary graph based construction, as it results in an efficient and simple solution. This is done in the proof of the following theorem that ties this section together.

Theorem 3.10. *Let $\gamma, \delta \in [0,1]$ be such that $\frac{\gamma n}{n-1} + \delta < 1$. The distribution $\mathcal{D}$ over pairs (G, t) implemented by Algorithm 1 is (σ, η)-good, as long as $\beta = t/k$ satisfies $\frac{\gamma n}{n-1} < \beta < (1-\delta)$, and*

$$k \geq \max\left(\frac{((\sigma+1)\log 2 + \log n)}{c} + 1, \frac{\eta \log 2 + \log n}{2\left(\frac{n(1-\delta)}{n-1} - \beta\right)^2}\right)$$

with $c = \min\left(2(\beta - 2\gamma)^2, -\log(\gamma+\delta)\right)$.

As an example consider the situation in which $\gamma = \delta = 1/5$ and take $\beta = 1/2$ then we get security and correctness with $n = 10^6$, $\sigma = 40$ and $\eta = 30$, so long as $k \geq 385$. Whilst this already saves a factor of 2500 over the complete graph, even lower requirements are shown to suffice in Section 5.

4 THE MALICIOUS PROTOCOL

In this section we show how to extend the ideas behind our semi-honest protocol to withstand an adversary that controls the server and a fraction γ of the n clients, as before, but where the adversary can deviate from the protocol execution. This includes, for example, sending incorrect messages, dropping out, or ignoring certain messages. Crucially, our protocol retains the computational benefits of the semi-honest variant: sublinear (polylog) client computation and communication in n.

Powerful adversary. To illustrate the power of such an adversary, let us describe a simple attack on the protocol of the previous section that can be run by a malicious server, by simply giving inconsistent views of which users dropped out to different clients. The goal of the server in this attack is to recover the private vector $\vec{x_u}$ from a target client u. Let $N = \mathsf{N}_\mathsf{G}(u)$ be the set of neighbors chosen by u in an execution without drop-outs. After collecting all masked inputs, the server tells all clients in $\overline{N} = [n] \setminus N$ that the immediate neighbors of u, i.e., every client in N, have dropped out. In other words, the server requests shares that are sufficient to recompute the pairwise masks of everyone in N. Note that these masks include values that cancel with all of u's pairwise masks. Hence, to obtain u's private vector, the server can announce to clients in N that u did not drop out to also recover u's self mask. Note that the server's malicious behavior here is only in notifying everyone in $\overline{N}$ that clients in N have dropped out, while simultaneously requesting shares of u's self mask from all clients in N. For this reason, this attack succeeds against any variant of the abstract protocol from the previous section, regardless of the choice of graph G.

What can the server legitimely learn in a robust protocol? While clearly the above attack is a problem as the server can learn a single client's data, formalizing what constitutes an attack against a protocol that aims at being robust against dropouts has some subtleties. Note that a malicious server can always wait for an execution where there are no dropouts, and simulate them by ignoring certain messages. Concretely, if a protocol is robust against a fraction δ of the clients dropping out, and the adversary controls a fraction γ of the clients, we cannot hope to prevent the server from learning the sum of any subset H of honest clients of their choice, as long as $|H| \geq (1 - \delta - \gamma)n$.

Bonawitz et al. [8] show how to modify their protocol so that it is secure in the presence of such a server by adding a "consistency check" round at the end of the protocol. This additional round prevents the server from learning the sum of any subset H of size $|H| \leq \alpha \cdot n$, by ensuring that at least $\alpha \cdot n$ clients, with $\alpha = \Omega(1)$, are given a consistent view of who dropped out. Unfortunately, this consistency check requires to transmit such a set of size $\alpha \cdot n$ to each client, yielding a communication overhead linear in n. Achieving the analogous goal (ensuring that α is a constant fraction of the secret sharing degree) in our $O(\log n)$-degree regular graphs from Section 3 would give $\alpha = O(\log n/n)$, which is unsatisfactory from a security standpoint: the number of values among which an honest client's value is hidden is too small, e.g., about 9 for $n = 10^4$. Thus, we require completely new ideas to make α independent of n.

We show that α can in fact be made a constant fraction independent of n while retaining polylog communication in n. More concretely, we show that the server cannot learn the sum of any subset H of honest clients such that $|H| < \alpha n$ for $\alpha = \Omega(1)$.

4.1 Security Definition

Intuitively, we want to define a summation protocol as being α-secure, for $\alpha \in [0, 1]$, if honest clients are guaranteed that their private value will be aggregated at most *once* with *at least* αn other values from honest clients. Formalizing this requires care. We will use a simulation-based proof, where we show that any attacker's view of the execution can be simulated in a setting where the attacker (which controls the server and a fraction of the clients) does not interact with the honest clients but a simulator that does not have the honest parties' inputs. Instead, we assume that the simulator can query *once* an oracle computing an "ideal" functionality that captures the leakage that we are willing to tolerate. We denote the functionality by $F_{\mathcal{X},\alpha}$ as it is parameterized by the set $\mathcal{X}$ of the honest parties' inputs and $\alpha \in [0, 1]$. It takes as input a partition of the honest clients (a collection of pairwise disjoint subsets $\{N_1, ..., N_{k+1}\}$) and, for each subset N_i it either returns $\sum_{i \in N_i} \vec{x_i}$ if the subset is "large enough", namely $|N_i| \geq \alpha \cdot |\mathcal{X}|$, or answers $\perp$.

Definition 4.1 (α-Summation Ideal Functionality). Let n, R, ℓ be integers, and $\alpha \in [0, 1]$. Let $H \subseteq [n]$ and $\mathcal{X}_H = \{\vec{x_i}\}_{i \in H}$ where $\vec{x_i} \in \mathbb{Z}_R^\ell$. Let $\mathcal{P}_H$ be the set of partitions of H.

The α-summation ideal functionality over $\mathcal{X}_H$, denoted by $F_{\mathcal{X}_H,\alpha}$, is defined for all partition $\{H_1, \ldots, H_\kappa\} \in \mathcal{P}_H$ as

$$F_{\vec{x},\alpha}(\{H_1, \ldots, H_\kappa\}) = \{S_1, \ldots, S_\kappa\},$$

where

$$\forall k \in [1, \kappa], \quad S_k = \begin{cases} \sum_{i \in H_k} \vec{x_i} & \text{if } |H_k| \geq \alpha \cdot |H|, \\ \perp & \text{otherwise.} \end{cases}$$

Note that the above definition's only goal is to characterize an "upper bound" on what an adversary controlling the server could learn from the protocol, and is unrelated to correctness guarantees. Let us remark that the prescribed output for the server, as in the semi-honest case, is the sum of the inputs $\vec{x_i}$ of surviving clients. Along with our security theorem, we provide a correctness result that states a guarantee for semi-honest executions in Theorem 4.8, and discuss the security of our protocol in the (weaker) threat model where the server is honest in Section 4.6.

4.2 The Malicious Protocol

Our precise protocol is Algorithm 3 in the appendix. Here we discuss the intuition behind it.

Similarly to Bonawitz et al. [8], we need the assumption that, roughly speaking, the clients participating in the execution are "real" clients, and not simulated by the server as part of a Sybil attack. This could be achieved assuming a Public Key Infrastructure (PKI) external to the clients and server. It also suffices to assume that the server behaves semi-honestly during the key collection phase. Thus below we assume: the server behaves semi-honestly during Part I of the protocol and commits the public keys of all "real" clients in a Merkle tree. This limits the power of a malicious server to interrupting the communication between parties in subsequent rounds, which is equivalent to making it appear to certain parties that certain other parties have dropped out.

A first hurdle in extending our efficient semi-honest protocol from Section 3 to the malicious setting, which does not apply to the protocol of Bonawitz et al., is that we cannot rely on the server to generate the communication graph G anymore, as it may deviate from the prescribed way and assign many malicious neighbors to an honest client. Hence, the first difference will be that in the protocol from this section the graph will be generated in a distributed way (Part II of Algorithm 3). $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ with $\mathsf{V} = [n]$, will now be a directed graph, and $(i, j) \in \mathsf{E}$ will mean that client i chose to trust client j with shares of its secrets; this relationship will

not be symmetric. We denote by $\mathsf{N}_{\bullet\rightarrow}(i) = \{j \in \mathsf{V} : (i,j) \in \mathsf{E}\}$ the outgoing neighbors of i. This graph generation algorithm will enable to prove that no honest client has too many corrupt neighbors with overwhelming probability. In particular, we define the following event, and will show in Lemma 4.7 that the event holds with overwhelming probability for the (random) graph generated in Part II.

Definition 4.2 (Not too many corrupt neighbors (malicious case)). Let k, t be integers such that $k < n$ and $t \in (k/2, k)$, and let $C \subset [n]$ be such that $|C| \leq \gamma n$. We define event E_4 as

$$E_4(C, \mathsf{G}, k, t) = 1 \text{ iff } \forall i \in [n] \setminus C : |\mathsf{N}_{\bullet\rightarrow}(i) \cap C| < 2t - k\,.$$

Another hurdle, emphasized by the simple attack described at the beginning of the section, comes as the adversary can give inconsistent views to each honest client about which clients dropout. The first issue is that the adversary must never learn both the shares of the self-mask and the secret key of a user i that submitted its masked value. However, even if what precedes holds, this does not mean that we are satisfying our security definition. As discussed above, we also want to provide a K-anonymity-style guarantee that a client input revealed to the server has been combined with $K = \alpha \cdot n$ clients where $\alpha = \Omega(1)$. We show that it suffices to have a logarithmic number of neighbors and a *local* consistency check.

Our first issue is solved by the bound $2t - k$ (instead of t) in Definition 4.2, and by the fact that a neighbor of i reveals at most one share from i. Indeed, if m_i is the number of malicious neighbors of i, the adversary needs to learn $2t - 2m_i$ shares from the $k - m_i$ honest neighbors of i to recover both b_i and sk_i^1; when the event in Definition 4.2 holds, i.e. $m_i < 2t - k$, we know the adversary cannot learn both secret values of i.

As for the second issue, let us describe a way to fix the simple attack we described, which as we will see leads to a general solution. Recall that in the attacks, all clients believe that neighbors $c_1, \ldots, c_k$ of u (the target client) have dropped out, while the c_i's are in fact alive and report shares from u. Instead, we will make sure each c_i refuses to release any secrets (about u or anybody else) unless the server can convince them that "enough" of their neighbors *know that they are alive*. In particular, the server will have to provide the c_i's with $p = k - t + 1$ signed messages (assuming that all clients are honest) from c_i's neighbors stating that they have been informed that c_i is alive, and thus will not release shares of c_i's secret key. To extend this idea to the setting with corrupted clients it is enough to note that if we knew that every honest client has no more than m corrupted neighbors then setting $p = k - (t - m) + 1$ would suffice, as we could conservatively assume that the server already possesses m shares from each client via the corrupted clients. Finally, to find a value for m that works with overwhelming probability we will rely on the fact that the number X of corrupted neighbors is distributed as $\mathtt{HyperGeom}(n-1, \gamma n, k)$ (as in Section 3), and thus an $m \approx k\gamma + \sqrt{k + \log n}$ suffices.

Lemma 4.3 (Informal). *No honest client i reveals a share and has more than t shares of their secret key sk_i^1 revealed.*

The previous modification prevents the secrets of the c_i from being revealed, but does not yet prevent the attack from going through. Indeed, if the server told u that all its neighbors have dropped out, u will mask its input vector only with its self-mask, which can be recovered from the c_i's by telling them u dropped out. Henceforth, we will additionally have c_i not reveal a secret about u *unless she received a signed message from u* that the pairwise mask between u and c_i was included in u's masked input.

The final challenge therefore consists in proving that the two countermeasures above prevent the adversary from learning the sum of the inputs of a "small clique" of clients. Instead, we want to show that the server needs to aggregate at least $\alpha \cdot n$ clients to hope to learn anything about their inputs with $\alpha = \Omega(1)$. Denote by S a set of honest clients and assume that every honest client has no more than m corrupted neighbors; in order to learn the self-masks of the clients in S, the server needs all of them to have at least $t - m$ honest clients revealing shares of their self-masks. However, these honest clients reveal such a share only if they know the pairwise mask has been included in the sum. Therefore, the server will also need to include those neighbors in the set S. Now the server needs that each client in S chooses a fraction $\approx t/k - \gamma$ of their neighbors from S, where γ is the fraction of compromised clients, and hence we obtain that the server will not learn anything about a set S unless $|S|/n$ is at least $\approx t/k - \gamma$, which is independent of n when t is a fraction of k. We define the following event, and show in Lemma 4.7 that it holds with overwhelming probability for the random graph generated in Part II.

Definition 4.4 (No small near cliques). Let $C \subset [n]$. We define the event E_5 as $E_5(C, \mathsf{G}, t, \alpha) = 1$ iff

$$\forall S \subset [n] \setminus C, |S| < \alpha n, \exists i \in S, |\mathsf{N}_{\bullet\rightarrow}(i) \cap (C \cup S)| < t\,.$$

Finally, for the protocol to be correct in the presence of up to $\delta \cdot n$ dropouts, we define the following event and will show in Lemma 4.7 that the event holds with overwhelming probability for the graph generated in Part II.

Definition 4.5 (Enough shares are available). Let $D \subset [n]$. We define the event E_6 as

$$E_6(D, \mathsf{G}, t) = 1 \text{ iff } \forall i \in [n] : |\mathsf{N}_{\bullet\rightarrow}(i) \cap ([n] \setminus D)| \geq t\,.$$

(In)consistent shares. Note that malicious clients may deliver inconsistent shares to their neighbors, e.g., in a way that results in different secrets being reconstructed if different sets of shares are used for reconstruction. This means that we allow malicious parties to tailor their inputs to which of their neighbors survive until the end of the protocol. This is a consequence of the fact that we do not model the fact that a client dropped out as private information.

One could limit that above ability with a simple modification in the protocol: when sharing (encrypted) secret shares, the clients also send hashes of their secrets (in the clear) to the server (note that the secrets have high entropy, hence this does not leak information to the server). At the end of the protocol, when reconstructing either the self-mask seed b_i or the pairwise masks keys sk_i^1, the server checks that the reconstructed secret matches the earlier hashes, and otherwise aborts. This accomplishes that dishonest clients cannot change their input based on which of their neighbors dropout *after* they send their masked input. This, however, does not prevent clients from changing their input based on which clients dropout *before* that happens. In particular, a client i chosen as a neighbor by a malicious client j could dropout immediately after

the protocol begins, and nothing prevents j from changing their input in that event.

The above observation is also relevant to Bonawitz et al. [8], and the protocol extension would also apply there.

4.3 Generating "Nice" Graphs

As explained above, we would like to show that Part II of Algorithm 3 generates "nice" graphs, i.e., that the events E_4, E_5, and E_6 happen with overwhelming probability on graphs generated according to Part II of Algorithm 3. Below, we define formally what a nice graph generation algorithm is, and state in Lemma 4.7 that Part II of Algorithm 3 satisfies the definition. A detailed proof of Lemma 4.7 can be found in Appendix C of our full paper [5].

Definition 4.6. Let k, σ, η be integers and let $\alpha, \delta \in [0,1]$. Let $C \subseteq [n]$. Let $\mathcal{D}$ be a distribution over pairs (G, t). We say that $\mathcal{D}$ is $(\sigma, \eta, C, \alpha)$-nice if, for all sets $D \subset [n]$ such that $|D| \leq \delta n$, we have that

(1) $Pr[E_4(C, \mathrm{G}', t') \wedge E_5(C, \mathrm{G}', t', \alpha) = 1 \mid (\mathrm{G}', t') \leftarrow \mathcal{D}] > 1 - 2^{-\sigma}$
(2) $Pr[E_6(D, \mathrm{G}', t') = 1 \mid (\mathrm{G}', t') \leftarrow \mathcal{D}] > 1 - 2^{-\eta - 1}$

Analogously, we say that a graph generation algorithm is $(\sigma, \eta, C, \alpha)$-nice if it implements a $(\sigma, \eta, C, \alpha)$-nice distribution.

Lemma 4.7. *Let $\gamma \geq 0$ and $\delta \geq 0$ such that $\gamma + 2\delta < 1$. Then there exists a constant c making the following statement true for all sufficiently large n. Let k be such that*

$$c(1 + \log n + \eta + \sigma) \leq k < (n-1)/4$$

$t = \lceil (3 + \gamma - 2\delta)k/4 \rceil$ and $\alpha = (1 - \gamma - 2\delta)/12$. Let $C \subset [n]$, such that $|C| \leq \gamma n$, be the set of corrupted clients. Then for sufficiently large n, the distribution $\mathcal{D}$ over pairs (G, t) implemented by Part II of Algorithm 3 is $(\sigma, \eta, C, \alpha)$-nice.

4.4 Correctness and Security

In this section, we state our correctness and security theorems; we formally prove them in Appendix C of our full paper [5].

Theorem 4.8 (Correctness). *Let Π be the protocol of Algorithm 3 with the parameters of Lemma 4.7. Consider an execution of Π with inputs $\mathcal{X} = (\vec{x}_i)_{i \in [n]}$, in which all parties follow the protocol. If $A'_4 \geq (1 - \delta)n$, i.e. less than a fraction δ of the clients dropout, then the server does not abort and obtains $\vec{z} = \sum_{A'_2} \vec{x}_i$ with probability $1 - 2^{-\eta}$.*

Theorem 4.9 (Security). *Let σ, η, λ be integer parameters. Let Π be the protocol of Algorithm 3 with the parameters of Lemma 4.7,*

$$p = k - (t - \frac{k\gamma n}{n-1} + \sqrt{\frac{k}{2}((\sigma+1)\log(2) + \log n)}) + 1,$$

and instantiated with a IND-CPA and INT-CTXT authenticated encryption scheme, a EUF-CMA signature scheme, and a λ-secure key agreement protocol. There exists a PPT simulator Sim such that, for all $C \subset [n]$ such that $|C| \leq \gamma n$, inputs $\mathcal{X} = (\vec{x}_i)_{i \in [n] \setminus C}$, and for all malicious adversary $\mathcal{A}$ controlling the server and the set of corrupted clients C behaving semi-honestly in Part I, the output of Sim is computationally indistinguishable from the joint view of the server and the corrupted clients Real_C, i.e., $\mathsf{Real}_C \approx_{\sigma,\lambda} \mathsf{Sim}^{F_{\mathcal{X}',\alpha}}(C)$, where the simulator can query once the ideal functionality $F_{\mathcal{X},\alpha}$.

4.5 Performance Analysis

We report the communication and computation costs for the client and server when $k = O(\log n)$. We recall that we assumed that basic operations and representation of elements in $\mathbb{X}$ are $O(1)$.

Client computation: $O(\log^2 n + l \log n)$. Each client computation can be broken up as receiving $\leq 4k \log n$ hashes ($O(k \log n)$ complexity), $\leq 5k$ key agreements and k encryptions ($O(k)$ complexity), $\leq 5k$ signatures signing and verifications ($O(k)$ complexity), creating twice t-out-of-k Shamir secret shares ($O(k^2)$ complexity), generating values $\vec{m_{i,j}}$ for all neighbors j ($O(kl)$ complexity).

Client communication: $O(\log^2 n + l)$. Each client performs $\leq 5k$ key agreements ($O(k)$ messages), send $2k$ encrypted shares ($O(k)$ messages), send a masked input ($O(l)$ complexity), send $\leq 5k$ signatures, and reveal up to $2k$ shares ($O(k)$ messages).

Server computation: $O(n(\log^2 n + l \log n))$. The server computation can be broken up as reconstructing t-out-of-k Shamir secrets for each client ($O(n \cdot k^2)$ complexity), generating values $\vec{m_{i,j}}$ for all (dropped out) neighbors j of each client i ($O(nkl)$ complexity).

Server communication: $O(n(\log^2 n + l))$. The server receives or sends $O(\log^2 n + l)$ to each client.

4.6 Guarantees with an honest server

We now discuss the setting where an adversary controls a subset of γn actively corrupted clients, but the rest of the clients and the server are honest. It is easy to see that the view of the adversary in this case does not include any information whatsoever about honest inputs, because clients only receive shares of a PRG seed (for self masks) and shares of a public key (for pairwise masks). The only message that contains the input is the masked input $\vec{y}_i$, which is sent to the server. Note that malicious clients could lie about their randomness $\vec{r}_i$ and pairwise masks when sharing seeds for it, but this boils down to sending an arbitrary $\vec{y}_i$ maliciously, which is in turn equivalent to lying about their input $\vec{x}_i$. Let us remark that we do not prevent malicious clients to choose their input adaptively depending on which of their neighbors dropped out (analogously to [8]). As mentioned in Section 4.2 one could limit this flexibility to choosing input based on which neighbors dropped out *before* the masked input $\vec{y}_i$ is sent with a cheap modification in the protocol.

To make the above security claim formal in the standard ideal vs. real simulation paradigm, one needs to show that there is a simulator in the ideal world that can produce the view of the malicious clients, along with the output of the honest server. Moreover, this should be done for any setting D of honest dropped out clients (recall that the adversary controls if/when malicious clients dropout). The simulator has one-time access to an ideal functionality returning the sum of the values of the clients that did not dropout (the prescribed output for the server), and can evaluate an attacker A that controls γn malicious clients. The simulator just needs to run A with an arbitrary *fake* input for honest clients, say all zeroes, and dropping out honest clients according to D . This allows to extract the sum z_{mal} of the inputs of malicious clients provided by the adversary. Note that z_{mal} is all that is needed to query the ideal functionality and obtain the server's prescribed output (including the *true* sum of the honest clients in this case).

5 NUMERICAL BOUNDS AND CONCRETE EFFICIENCY RESULTS

In Theorems 3.6 and 4.9, we established that a number of neighbors $k = O(\log n + \eta + \sigma)$ suffices to obtain secure and correct protocols in the semi-honest and malicious variants. These theorems are derived from tail bounds on the hypergeometric distribution. While the same tail bounds could be used in practice to set the operating parameters k, t (i.e. by direct evaluation of the expression for k in Theorem 3.10), more efficient choices are found below by directly evaluating the hypergeometric CDF.

5.1 Semi-honest Variant

The results in this Section follow from Lemma 3.7, and the fact that the CDF of the hypergeometric distribution can be evaluated directly. More concretely, note that Lemma 3.7 gives sufficient efficiently checkable conditions that k, t can satisfy (given the rest of the parameters) implying that our protocol is secure and correct (Lemmas 3.6 and 3.5). This gives a numerical approach to obtain secure parameters k, t given $n, \sigma, \eta, \gamma, \delta$. The naive algorithm iterates over all possible values of k, t in lexicographic order and stops as soon as it finds one that satisfies both conditions in Lemma 3.7 (assuming our intent is to minimize computation and communication for the required security σ). We implemented a more efficient and stable variant of this approach, using binary search and checks in the log domain, avoiding numerical under(over)flows. Our implementation consists of less than 100 lines of Python code leveraging the Scipy scientific computing library [35].

Fig. 2 (a) shows secure values of k for several settings of parameters corresponding to all combinations of γ, δ taking values in $\{1/3, 1/20\}$, $\sigma = 40$, and $\eta = 30$, as n grows from 10^3 to 10^8. Fig. 2 (b) shows how k scales with γ, everything else being the same. Note that less than 150 neighbors are enough to provide security up to $n = 10^8$ clients where at most 1 in 5 clients are corrupted by and 1 in 20 clients dropout (or vice versa). Moreover, $k = 100$ suffices for $n = 10^4$, which immediately translates into a 100x *concrete* improvement in client computation and communication with respect to the protocol by Bonawitz et al [8], with roughly the same server computation cost. These gains increase linearly with n, as our protocol retains roughly the same client runtime and communication costs for values of n for which the protocol by Bonawitz et al [8] becomes highly impractical.

Benchmarking. In the semi-honest protocol, each client performs (a) $2k$ key agreements, (b) secret sharing 2 secrets into k shares (which takes $O(k^2)$ time), and (c) generating and stretching $k + 1$ seeds using the PRG F (which we implement using AES) to the length of the input vector l (which is $O(kl)$). Thus our runtime and communication improvements with respect to the semi-honest version of [8] are roughly a factor $n/k = O(n/\log n)$. We benchmarked Shamir share generation and PRG expansion using AES to confirm that the PRG expansion is the bottleneck (which is consistent with the running times reported in [8]). We report running times for a particular setting in Table 2 in the appendix. The setting was chosen for ease of comparison with the running times reported to the semi-honest version of the protocol of Bonawitz et al. (see Figure 8 in [8]). One can observe that our approach is roughly an order of magnitude faster for $n = 1000$ (which is the the only value of n for which Bonawitz et al. report runtimes). Crucially, our costs remain almost the same as n increase, which the complete-graph construction of in [8] quickly becomes impractical. This is consistent with the factor of n/k improvement mentioned above, as $k = O(\log n)$ stays within $(80, 120)$ as n grows within $(10^3, \ldots, 10^5)$.

5.2 Malicious Variant

Our approach to compute numerical bounds for the malicious variant is analogous to the one from the semi-honest variant: Definition 4.6 states sufficient conditions that a triple (k, t, α) must satisfy to get security and correctness. As in the semi-honest case these checks involve only simple calculations and querying the CDF of a hypergeometric random variable. Our implementation consists of roughly 100 lines of Python code leveraging the Scipy scientific computing library [35].

Fig. 2 (c) shows secure values of k and α for several settings corresponding to all combinations of γ, δ taking values in $\{1/5, 1/20\}$, $\sigma = 40$, and $\eta = 30$, as n grows from 10^3 to 10^8. For example, with 10^4 clients and $\gamma = \delta = 1/5$, less than 600 neighbors per client are enough for security, and every honest client is guaranteed that if the (possibly malicious) server gets a sum z in the clear involving their value, then z is the result of aggregating *at least* $0.39n = 3900$ *honest clients*. The value of k in this plot is the minimum value that guarantees security σ, thus minimizing client computation and communication. Hence, the resulting α, which grows with k, is not as large as it could be for each setting. This explains the counter-intuitive fact that settings with smaller γ also have smaller α (as they allow a smaller k). To clarify this point Fig. 2 (d) shows how α scales with k: α converges towards $(1 - \gamma - \delta)n$, which is the best one can hope for (as in a round without dropouts a malicious server can drop any set of $(\delta)n$ honest parties from the sum). Hence, by increasing k, our analysis covers the full spectrum of possible α, and allows one to fine tune parameters to trade-off between security (captured by α, σ) and the main computational costs (captured by k, t). Finally, to illustrate this kind of fine-tuning, Figure 3 shows how α grows for small numbers of neighbors in some parameters. For example, the plot shows that, for $n = 10^4, \gamma = \delta = 1/20$, 300 neighbors are enough so that every honest client is certain that their value will be aggregated with at least 5000 other clients values!

6 SHUFFLING FROM SUMMATION

In this section we consider the following shuffling primitive between many clients with inputs and a server who should obtain the shuffled clients' inputs. Each client i has a message $x_i \in M$ with $|M| = m$, and the goal is for the server to receive the multiset of messages with no information on which message came from whom. This is equivalent to the messages being sent to a trusted third party and shuffled (or sorted) before being handed to the server. This primitive is the basis of the shuffle model of differential privacy [3, 4, 11, 19] and could be used for anonymized submissions [6].

We show a reduction from secure shuffling to secure summation. Our reduction makes a single call to secure summation, and thus the security and drop-out robustness properties of our protocol directly translate to the shuffling functionality. In particular, in the

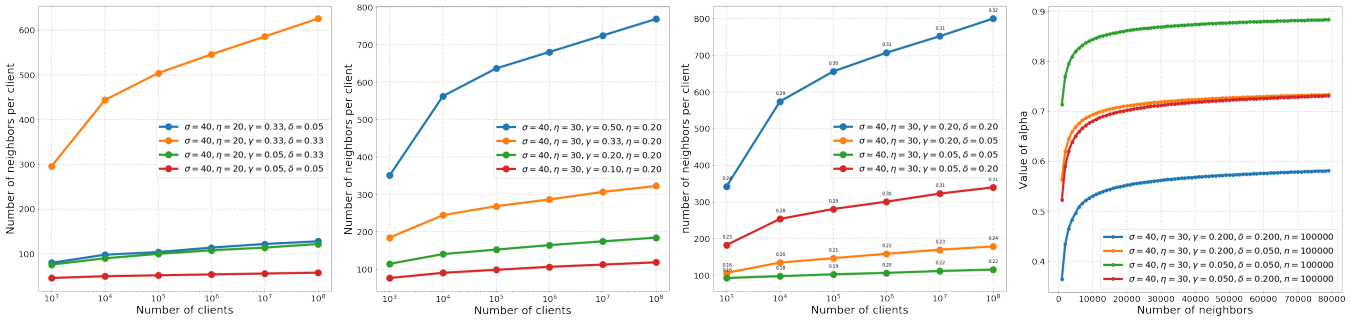

Figure 2: (From left to right, a) Secure values of k for several settings (Semi-honest). b) Secure values of k for several settings, as γ grows (Semi-honest). c) Required number of neighbors for several settings, and the value of α guaranteed in each case (Malicious). d) Values of α as the number of neighbors increases, for $n = 10^5$ (Malicious).

semi-honest variant, honest clients are guaranteed to have their value shuffled with at least $(1 - \gamma - \delta)n$ other values from honest clients. In the malicious setting, one gets the analogous guarantee with αn, as explained in Section 4.

We first discuss a simple baseline solution that only works for small m. A histogram of how many times each message appears can be considered as a vector of length m. Thus, each client can locally build a histogram of their input (which would be a one-hot vector) and then a vector summation protocol can be used to add these local histograms together. The server then learns only the aggregate histogram, as desired. The problem with this solution is that it is impractical for large m, e.g., $m = 2^{32}$. More generally, this solution is wasteful for scenarios where we know that the result histogram is going to be sparse.

We address the issues in the above protocol, i.e. for settings where the size of the message domain m is much larger than the number of client n by leveraging a probabilistic data structure called an invertible Bloom lookup table (IBLT) [21]. An IBLT is a linear sketch of a key-value store, such that if the vector representations for two IBLTs are added together, the result is a new IBLT that encodes the union of the key-value stores for the original IBLTs. IBLTs support the following operations (among others):

- Insert(p, x): insert the key-value pair (p, x).
- ListEntries(): list every key-value pair in the data structure.

Though the ListEntries operation may fail, we can choose parameters so that this failure happens with very small probability.

Using this data structure, the shuffle primitive can be achieved as follows. Every client first creates an empty local IBLT of length ℓ, all with the same parameters. Then each client i chooses a pseudonym π_i uniformly at random from a set P that is sufficiently large to avoid collisions (e.g. 64 bit strings would work well). They then insert the pair (π_i, x_i) into their IBLT. A vector summation protocol is then used to combine the IBLTs, and the server recovers the messages using the ListEntries IBLT functionality.

We provide the details of the above construction and the exact implementations of the local vector preparation algorithm run by each client and the message recovery algorithm run by the server in Appendix D of the full version [5]. There we also discuss the exact parameters, which tell us that for $n > 100$, the bit length of the vectors used for the IBLTs can be be taken to be less than $2n\lceil \log_2(|P|) + \log_2(m) + \log_2(n)\rceil$. For example if $n = 10{,}000$ and the clients' inputs are 32 bits long, the construction requires $2 \cdot 10{,}000 \cdot (64 + 32 + 14) = 2{,}200{,}000$ bit, i.e. 269kB, vectors. As a final remark, note that this protocol is very easily adapted to the case where clients have different numbers of messages to send. This covers the case where each user has multiple messages to send, as in the multi-message shuffle model [4, 11, 19], and the case where most users don't have any message to send, which might be useful for submitting error reports.

7 CONCLUSION

We presented new constructions for secure aggregation that achieve both better asymptotic computation and communication costs than previous solutions as well as very efficient concrete parameters, which enable much better scalability with the number of clients. The efficiency cost of the construction of Bonawitz et al. [8] limited its use to a thousand clients. Our semi-honest construction supports billions of clients and our semi-malicious construction supports tens of thousands of clients for the same per client cost. Last but not least we presented a construction for secure shuffling using secure vector aggregation, which is the first cryptographically secure instantiation of the shuffle model of differential privacy. This construction requires each client to have an input vector of size linear in the total number of submitted messages.

We leave as future work a system implementation of our protocols, and leave as an intriguing open question how to achieve secure shuffling with sublinear complexity in the single-server setting.

Acknowledgments. JB was supported by the UK Government's Defence & Security Programme in support of the Alan Turing Institute. We thank Aurélien Bellet, Matt Kusner, Kobbi Nissim, and Brooks Paige for useful discussions in the early stages of this work. We thank Cindy Lin, Sarvar Patel, Aaron Segal, and Karn Seth for their feedback, as well as insightful discussions about the semi-malicious variant of the problem.

REFERENCES

[1] David W. Archer, Dan Bogdanov, Yehuda Lindell, Liina Kamm, Kurt Nielsen, Jakob Illeborg Pagter, Nigel P. Smart, and Rebecca N. Wright. 2018. From Keys to Databases - Real-World Applications of Secure Multi-Party Computation. *Comput. J.* 61, 12 (2018), 1749–1771.

[2] Borja Balle, James Bell, Adrià Gascón, and Kobbi Nissim. 2019. Improved Summation from Shuffling. *arXiv: 1909.11225* (2019).

[3] Borja Balle, James Bell, Adrià Gascón, and Kobbi Nissim. 2019. The Privacy Blanket of the Shuffle Model. In *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 11693)*, Alexandra Boldyreva and Daniele Micciancio (Eds.). Springer, 638–667. https://doi.org/10.1007/978-3-030-26951-7_22

[4] Borja Balle, James Bell, Adria Gascon, and Kobbi Nissim. 2020. Private Summation in the Multi-Message Shuffle Model. arXiv:2002.00817 [cs.CR]

[5] James Bell, Keith Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. 2020. Secure Single-Server Aggregation with (Poly)Logarithmic Overhead. *IACR Cryptol. ePrint Arch.* 2020 (2020), 704.

[6] Andrea Bittau, Úlfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Ushasree Kode, Julien Tinnes, and Bernhard Seefeld. 2017. Prochlo: Strong Privacy for Analytics in the Crowd. In *Proceedings of the 26th Symposium on Operating Systems Principles* (Shanghai, China) *(SOSP '17)*. ACM, New York, NY, USA, 441–459. https://doi.org/10.1145/3132747.3132769

[7] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé M Kiddon, Jakub Konečný, Stefano Mazzocchi, Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. 2019. Towards Federated Learning at Scale: System Design. In *SysML 2019*. https://arxiv.org/abs/1902.01046

[8] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical Secure Aggregation for Privacy-Preserving Machine Learning. In *ACM Conference on Computer and Communications Security*. ACM, 1175–1191.

[9] Elette Boyle, Kai-Min Chung, and Rafael Pass. 2015. Large-Scale Secure Computation: Multi-party Computation for (Parallel) RAM Programs. In *Advances in Cryptology – CRYPTO 2015*, Rosario Gennaro and Matthew Robshaw (Eds.).

[10] Elette Boyle, Ran Cohen, Deepesh Data, and Pavel Hubáček. 2018. Must the Communication Graph of MPC Protocols be an Expander?. In *Advances in Cryptology – CRYPTO 2018*, Hovav Shacham and Alexandra Boldyreva (Eds.).

[11] Albert Cheu, Adam D. Smith, Jonathan Ullman, David Zeber, and Maxim Zhilyaev. 2019. Distributed Differential Privacy via Mixnets. In *EUROCRYPT*. 375–403.

[12] Henry Corrigan-Gibbs and Dan Boneh. 2017. Prio: Private, Robust, and Scalable Computation of Aggregate Statistics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*.

[13] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating Noise to Sensitivity in Private Data Analysis. In *Proceedings of the Third Conference on Theory of Cryptography*.

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

[15] Úlfar Erlingsson, Vitaly Feldman, Ilya Mironov, Ananth Raghunathan, Shuang Song, Kunal Talwar, and Abhradeep Thakurta. 2020. Encode, Shuffle, Analyze Privacy Revisited: Formalizations and Empirical Evaluation. *arXiv preprint arXiv:2001.03618* (2020).

[16] Úlfar Erlingsson, Vitaly Feldman, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Abhradeep Thakurta. 2019. Amplification by Shuffling: From Local to Central Differential Privacy via Anonymity. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '19)*.

[17] Taher El Gamal. 1985. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Information Theory* 31, 4 (1985), 469–472.

[18] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In *In Proc. STOC*. 169–178.

[19] Badih Ghazi, Rasmus Pagh, and Ameya Velingker. 2019. Scalable and Differentially Private Distributed Aggregation in the Shuffled Model. *arXiv preprint arXiv:1906.08320* (2019).

[20] Oded Goldreich. 2004. *The Foundations of Cryptography - Volume 2: Basic Applications*. Cambridge University Press.

[21] Michael T. Goodrich and Michael Mitzenmacher. 2011. Invertible bloom lookup tables. In *49th Annual Allerton Conference on Communication, Control, and Computing, Allerton 2011, Allerton Park & Retreat Center, Monticello, IL, USA, 28-30 September, 2011*. IEEE, 792–799. https://doi.org/10.1109/Allerton.2011.6120248

[22] Shai Halevi, Yehuda Lindell, and Benny Pinkas. 2011. Secure Computation on the Web: Computing without Simultaneous Interaction. In *Proceedings of the 31st Annual Conference on Advances in Cryptology*.

[23] Mihaela Ion, Ben Kreuter, Ahmet Erhan Nergiz, Sarvar Patel, Mariana Raykova, Shobhit Saxena, Karn Seth, David Shanahan, and Moti Yung. 2020. On Deploying Secure Computing Commercially: Private Intersection-Sum Protocols and their Business Applications. In *5th IEEE European Symposium on Security and Privacy*.

[24] Internet Research Task Force (IRTF). 2018. ChaCha20 and Poly1305 for IETF Protocols. https://datatracker.ietf.org/doc/rfc8439/; accessed 2020-05-12.

[25] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. 2019. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977* (2019).

[26] Iraklis Leontiadis, Kaoutar Elkhiyaoui, and Refik Molva. 2014. Private and Dynamic Time-Series Data Aggregation with Trust Relaxation. In *Cryptology and Network Security*, Dimitris Gritzalis, Aggelos Kiayias, and Ioannis Askoxylakis (Eds.).

[27] KU Leuven. 2019. SCALE-MAMBA Software. https://homes.esat.kuleuven.be/~nsmart/SCALE/. (2019).

[28] Yehuda Lindell. 2017. How to Simulate It - A Tutorial on the Simulation Proof Technique. In *Tutorials on the Foundations of Cryptography*. Springer International Publishing, 277–346.

[29] Yehuda Lindell and Ariel Nof. 2018. Fast Secure Multiparty ECDSA with Practical Distributed Key Generation and Applications to Cryptocurrency Custody. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*. 1837–1854.

[30] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. 2012. On-the-Fly Multiparty Computation on the Cloud via Multikey Fully Homomorphic Encryption. In *Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing* (New York, New York, USA) *(STOC '12)*. Association for Computing Machinery, New York, NY, USA, 1219?1234. https://doi.org/10.1145/2213977.2214086

[31] H Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. 2018. Learning Differentially Private Recurrent Language Models. In *International Conference on Learning Representations (ICLR)*.

[32] Pascal Paillier. 1999. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *EUROCRYPT (Lecture Notes in Computer Science, Vol. 1592)*. Springer, 223–238.

[33] Leonid Reyzin, Adam Smith, and Sophia Yakoubov. 2018. Turning HATE Into LOVE: Homomorphic Ad Hoc Threshold Encryption for Scalable MPC. Cryptology ePrint Archive, Report 2018/997. https://eprint.iacr.org/2018/997.

[34] Jinhyun So, Basak Guler, and Amir Salman Avestimehr. 2020. Turbo-Aggregate: Breaking the Quadratic Aggregation Barrier in Secure Federated Learning. *IACR Cryptol. ePrint Arch.* 2020 (2020), 167.

[35] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake Vand erPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1. 0 Contributors. 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* 17 (2020), 261–272. https://doi.org/10.1038/s41592-019-0686-2

A APPENDIX

Table 1: Summary of parameters used throughout the paper.

Parameter	Description
n	Number of clients.
k	Number of neighbors of each client $k < n$.
t	Secret Sharing reconstruction threshold $t \leq k$.
σ	Information-theoretic security parameter (bounding the probability of bad events).
η	Correctness parameter (bounding the failure probability).
λ	Cryptographic security parameter (for cryptographic primitives).
δ	Maximum fraction of dropout clients.
γ	Maximum fraction of corrupted clients.
$\mathbb{X}$	Domain of the summation protocol.
l	Size of the clients' vector input.

n	l	Client Cost		Server Cost per client			
		Sharing	PRG Evaluation	Reconstruction	PRG Evaluation		
					0%	10%	30%
10^3	10^5	0.0002	0.1711	1.8e-06	0.0019	0.01679	0.039
10^4	10^5	0.0003	0.2058	2.24e-06	0.0020	0.0200	0.0467
10^5	10^5	0.0003	0.3453	2.44e-06	0.0031	0.0342	0.0756

Table 2: Client and server running time (in seconds) for Shamir secret sharing/reconstruction and PRG expansion (AES in counter mode), for different rates of dropouts. The setting is $\gamma = 1/20, \delta = 1/3, \sigma = 40, \eta = 30$, and a semi-honest server. In this setting the number of neighbors that provides security is is roughly 100 for all the values of n considered. We do not assume any parallelization at the client (and thus all pairwise masks are computed sequentially). For the server we return the cost per client. In particular, note that the cost per client of the server is two orders of magnitude smaller than the client cost. This is because the client has to expand $k \sim 100$ seeds in this setting, while the server only needs to obtain one self-mask per client. We fixed $l = 10^5$ to ease the comparison with the experimental results in Bonawitz et al. [8]. It is easy to extrapolate costs w.r.t. l, as the relationship is linear.

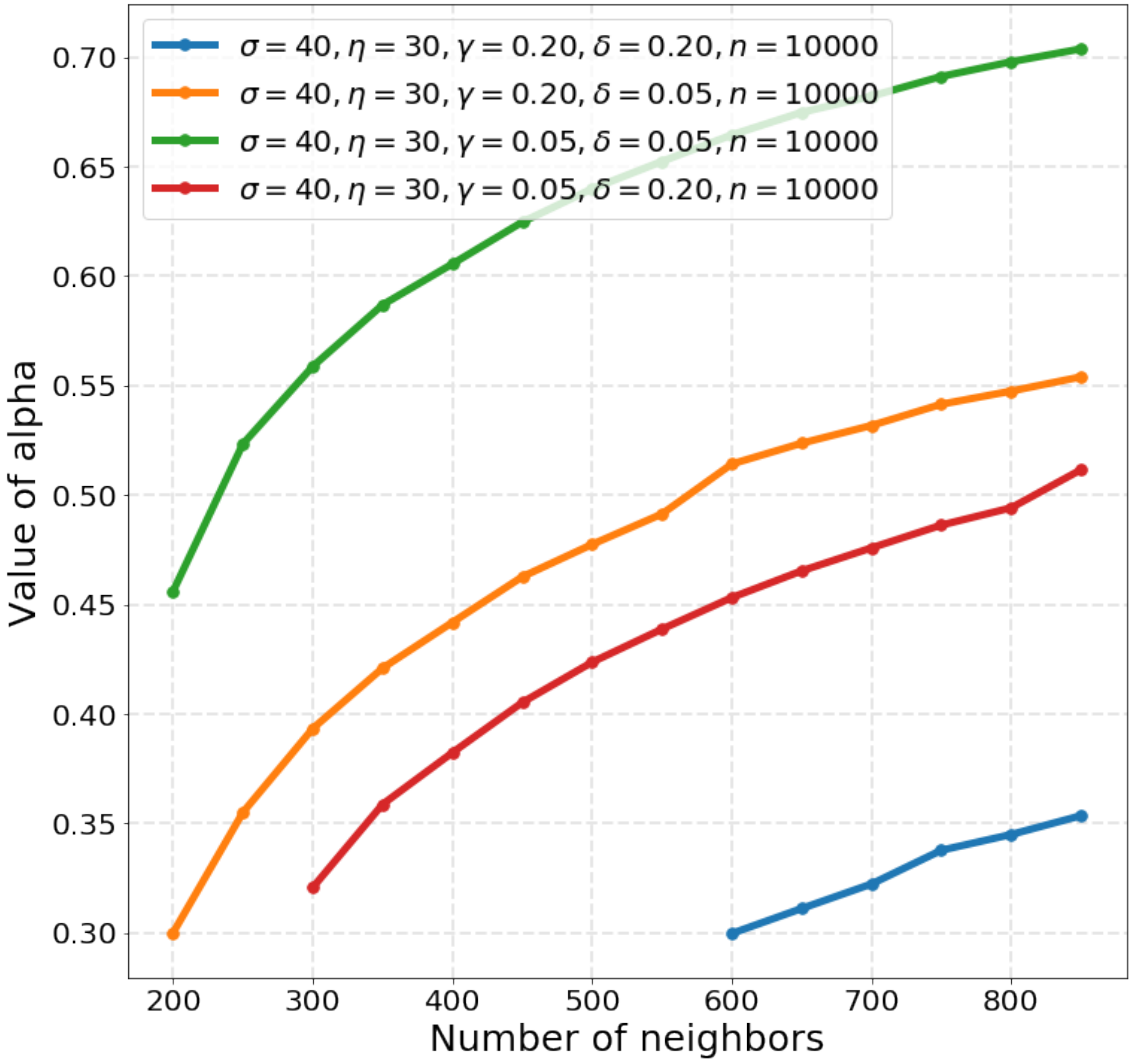

Figure 3: Values of α as the number of neighbors increases, for $n = 10^4$.

Algorithm 2: Abstract summation protocol.

Parties: Clients $1, \ldots, n$, and Server.
Public Parameters: Vector length l, input domain $\mathbb{X}^l$, and PRG $F\colon \{0,1\}^\lambda \mapsto \mathbb{X}^l$
Input: $\vec{x_i} \in \mathbb{X}^l$ (by each client i).
Output: $z \in \mathbb{X}$ (for the server).

We denote by A_1, A_2, A'_2, A_3 the sets of clients that reach certain points without dropping out. Specifically A_1 consists of the clients who finish step 3, A_2 those who finish step 5, and A_3 those who finish step 7. For each A_i, A'_i is the set of clients for which the server sees they have completed that step on time. Thus, $A_1 \supseteq A'_1 \supseteq A_2 \supseteq A'_2 \supseteq A_3 \supseteq A'_3$.

(1) The server runs $(\mathrm{G}, t, k) = \textsc{GenerateGraph}(n)$, where G is a regular degree-k undirected graph with n nodes. By $\mathsf{N}_\mathrm{G}(i)$ we denote the set of k nodes adjacent to i (its neighbors).
(2) Each client i generates key pairs (sk_i^1, pk_i^1), (sk_i^2, pk_i^2) and sends (pk_i^1, pk_i^2) to the server who forwards the message to $\mathsf{N}_\mathrm{G}(i)$.
(3) Each client i:
- Generates a random PRG seed b_i.
- Computes two sets of shares:
$$H_i^b = \{h_{i,1}^b, \ldots, h_{i,k}^b\} = \mathtt{ShamirSS}(t, k, b_i)$$
$$H_i^s = \{h_{i,1}^s, \ldots, h_{i,k}^s\} = \mathtt{ShamirSS}(t, k, sk_i^1)$$
- Sends to the server a message $m = (j, c_{i,j})$, where $c_{i,j} = \mathcal{E}_{auth}.\mathrm{Enc}(k_{i,j}, (i||j||h_{i,j}^b||h_{i,J}^s))$ for each $j \in \mathsf{N}_\mathrm{G}(i)$, where $c_{i,j}$ is a ciphertext encrypted under $k_{i,j} = \mathcal{KA}.Agree(sk_i^2, pk_j^2)$.

(4) The server aborts if $|A'_1| < (1-\delta)n$ and otherwise forwards all messages $(j, c_{i,j})$ to client j, who deduces $A'_1 \cap N_\mathrm{G}(j)$.
(5) Each client i:
- Computes a shared random PRG seed $s_{i,j}$ as $s_{i,j} = \mathcal{KA}.Agree(sk_i^1, pk_j^1)$.
- Sends to the server their *masked input*
$$\vec{y_i} = \vec{x_i} + \vec{r_i} - \sum_{\substack{j \in A_1 \cap \mathsf{N}_\mathrm{G}(i) \\ 0<j<i}} \vec{m_{i,j}} + \sum_{\substack{j \in A_1 \cap \mathsf{N}_\mathrm{G}(i) \\ i<j \le n}} \vec{m_{i,j}}$$
- where $\vec{r_i} = F(b_i)$ and $\vec{m_{i,j}} = F(s_{i,j})$.

(6) The server collects masked inputs for a determined time period. It aborts if $|A'_2| < (1-\delta)n$ and otherwise sends $(A'_2 \cap \mathsf{N}_\mathrm{G}(i), (A_1 \setminus A'_2) \cap \mathsf{N}_\mathrm{G}(i))$ to every client $i \in A'_2$.
(7) Each client j receives (R_1, R_2) from the server and sends $\{(i, h_{i,j}^b)\}_{i \in R_1} \cup \{(i, h_{i,j}^s)\}_{i \in R_2}$, obtained by decrypting the ciphertext $c_{i,j}$ received in Step 3.
(8) The server aborts if $|A'_3| < (1-\delta)n$ and otherwise:
- Collects, for each client $i \in A'_2$, the set B_i of all shares in H_i^b sent by clients in A_3. Then aborts if $|B_i| < t$ and recovers b_i and $\vec{r_i}$ otherwise using the t shares received which came from the lowest client IDs.
- Collects, for each client $i \in (A_1 \setminus A'_2)$, the set S_i of all shares in H_i^s sent by clients in A_3. Then aborts if $|S_i| < t$ and recovers sk_i^1 and $\vec{m_{i,j}}$ otherwise.

Outputs $\sum_{i \in A'_2} \vec{x_i}$ as
$$\sum_{i \in A'_2} \left(\vec{y_i} - \vec{r_i} + \sum_{\substack{j \in \mathsf{N}_\mathrm{G}(i) \cap (A'_1 \setminus A'_2) \\ 0<j<i}} \vec{m_{i,j}} - \sum_{\substack{j \in \mathsf{N}_\mathrm{G}(i) \cap (A'_1 \setminus A'_2) \\ i<j \le n}} \vec{m_{i,j}} \right).$$

Algorithm 3: Summation protocol in the malicious setting.

Parties: Clients $1, \ldots, n$, and Server.
Public Parameters: Vector length l, input domain $\mathbb{X}^l$, and PRG $F\colon \{0,1\}^\lambda \mapsto \mathbb{X}^l$.
Input: $\vec{x_i} \in \mathbb{X}^l$ (by each client i).
Output: $z \in \mathbb{X}$ (for the server).

We denote by A_1, A_2, A_3 and A_4 the sets of clients that send messages at the end of steps 6, 8, 11 and 13 of the protocol respectively. Then A'_i is the set of clients whose messages reach the server on time. Note that $[n] \supseteq A_1$, $A_i \supseteq A'_i$, $A'_i \supseteq A_{i+1}$ and A'_2 is the set of clients who will be included in the final sum.

Part I: public key commitments. *In this part only, we assume the server to behave semi-honestly.*

(1) Each client i generates key pairs $\mathcal{K}_i^1 = (sk_i^1, pk_i^1)$, $\mathcal{K}_i^2 = (sk_i^2, pk_i^2)$ and sends (pk_i^1, pk_i^2) to the server.
(2) The server commits to both vectors of public keys $pk^1 = (pk_i^1)_i$ and $pk^2 = (pk_i^2)_i$ by means of a Merkle tree.

Part II: distributed graph generation. *In these steps, the clients and server will jointly generate a directed graph* $\mathsf{G}([n], E)$.

(3) Each client i selects k neighbors randomly by sampling without replacement k times from the set of all clients $[n]$, and sends the resulting set $\mathsf{N}_{\bullet\rightarrow}(i)$ to the server. This set represents the "outgoing" neighbors of client i. *We note that the choices made by all clients implicitly define a set of "ingoing" neighbors for client i, denoted as* $\mathsf{N}_{\bullet\leftarrow}(i) \subseteq \{i \in \mathsf{N}_{\bullet\rightarrow}(j) : j \in [n]\}$ *Denote* $\mathsf{N}(i) = \mathsf{N}_{\bullet\leftarrow}(i) \cup \mathsf{N}_{\bullet\rightarrow}(i)$.
(4) The server sends $\left(\mathsf{N}_{\bullet\leftarrow}(i), (j, pk_j^1, pk_j^2)_{j\in\mathsf{N}(i)}\right)$ to client $i \in [n]$, together with $|\mathsf{N}(i)|\log_2(n)$ hashes for the Merkle tree verification.
(5) Each client i aborts if the server is sending her more than $3k + k$ public keys. Otherwise, she verifies that the public keys sent by the server are consistent with the Merkle tree root and that she has been given the public keys of everyone in $\mathsf{N}_{\bullet\rightarrow}(i)$, and aborts otherwise.

Part III: Masks generation and secret sharing.

(6) Each client i:
- Generates a random PRG seed b_i.
- Computes two sets of shares $H_i^b = \{h_{i,1}^b, \ldots, h_{i,k}^b\} = \mathtt{ShamirSS}(t, k, b_i)$ and $H_i^s = \{h_{i,1}^s, \ldots, h_{i,k}^s\} = \mathtt{ShamirSS}(t, k, sk_i^1)$.
- Sends to the server messages $m_j = (j, c_{i,j})$, where $c_{i,j} = \mathcal{E}_{auth}.\mathsf{Enc}(k_{i,j}, (i \,||\, j \,||\, h_{i,j}^b \,||\, h_{i,j}^s))$ for each $j \in \mathsf{N}_{\bullet\rightarrow}(i)$, where $c_{i,j}$ is a ciphertext encrypted under $k_{i,j} = \mathcal{KA}.Agree(sk_i^2, pk_j^2)$.

(7) The server aborts if $|A'_1| < (1-\delta)n$, and otherwise forwards all messages $(j, c_{i,j})$ to client j. *We note that this essentially defines a set $A_{2,j} \subseteq \mathsf{N}(j)$ of the clients i from which client j received $(j, c_{i,j})$.*
(8) Each client i:
- Decrypts all the ciphertexts received, and aborts if decryption fails.
- Computes a shared random PRG seed $s_{i,j}$ as $s_{i,j} = \mathcal{KA}.Agree(sk_i^1, pk_j^1)$ with every $j \in A_{2,i}$.
- Computes $\vec{r_i} = F(b_i)$ and $\vec{m}_{i,j} = F(s_{i,j})$ and computes their masked input $\vec{y_i} = \vec{x_i} + \vec{r_i} - \sum_{\substack{j\in A_{2,i}\\ 0<j<i}} \vec{m_{i,j}} + \sum_{\substack{j\in A_{2,i}\\ i<j\le n}} \vec{m_{i,j}}$.
- Signs the message $m_{i,j} = (\text{"}\mathtt{included}\text{"} \,||\, i \,||\, j)$ with sk_i^2 to obtain a signature $\sigma_{i,j}^{\text{incl}}$ for all $j \in A_{2,j}$.
- Sends $(\vec{y_i}, (m_{i,j}, \sigma_{i,j}^{\text{incl}})_{j\in A_{2,i}})$ to the server.

Part IV: Unmasking.

(9) If $|A'_2| < (1-\delta)n$, it aborts, and otherwise sends $(A'_2 \cap \mathsf{N}_{\bullet\leftarrow}(i), (A_1 \setminus A'_2) \cap \mathsf{N}_{\bullet\leftarrow}(i))$ and all messages/signatures $(m_{j,i}, \sigma_{j,i}^{\text{incl}})$ to every $i \in A'_2$. *We note that this essentially defines two sets $A_{3,i}^b$, $A_{3,i}^s$ of the clients j for every client i that received the message sent by the server.*
(10) Each client checks that $A_{3,i}^b \cap A_{3,i}^s = \emptyset$, $A_{3,i}^b$, $A_{3,i}^s \subseteq \mathsf{N}_{\bullet\leftarrow}(i) \cap A_{2,i}$, and that all signatures $\sigma_{j,i}^{\text{incl}}$ are valid for $j \in A_{3,i}^b$, and aborts otherwise.
(11) Each client i, for every $j \in A_{3,i}^b \subseteq \mathsf{N}_{\bullet\leftarrow}(i)$, signs a message $m_{i,j} = (\text{"}\mathtt{ack}\text{"} \,||\, i \,||\, j)$ using sk_i^2, and send the signature $(m_{i,j}, \sigma_{i,j}^{\text{ack}})$ to the server.
(12) The server aborts if $|A'_3| < (1-\delta)n$, and otherwise forwards all messages $(j, m_{i,j}, \sigma_{i,j})$ to client j.
(13) Each client collects all messages and signatures, and checks that all the signatures are valid using pk_i^2 (abort otherwise). Once client j receives p such valid signatures from parties in $\mathsf{N}_{\bullet\rightarrow}(j)$, she sends $\{(i, h_{i,j}^b)\}_{i\in A_{3,j}^b} \cup \{(i, h_{i,j}^s)\}_{i\in A_{3,j}^s}$.
(14) The server aborts if $|A'_4| < (1-\delta)n$, and otherwise:
- Collects, for each $i \in A'_2$, the set B_i of all shares in H_i^b sent by clients in A'_4. It aborts if $|B_i| < t$ and otherwise recovers b_i and $\vec{r_i}$ using the t shares received which came from the lowest client IDs.
- Collects, for each $i \in A_1 \setminus A'_2$, the set S_i of all shares in H_i^s sent by clients in A'_4. It aborts if $|S_i| < t$ and recovers sk_i^1 and $\vec{m}_{i,j}$ otherwise.

Outputs

$$\sum_{i\in A'_2}\left(\vec{y_i} - \vec{r_i} + \sum_{\substack{j\in\mathsf{N}(i)\cap(A_1\setminus A'_2)\\ 0<j<i}} \vec{m_{i,j}} - \sum_{\substack{j\in\mathsf{N}(i)\cap(A_1\setminus A'_2)\\ i<j\le n}} \vec{m_{i,j}}\right).$$