A Lightweight Secure Aggregation Protocol for Federated Learning Applications

Abstract—**Federated learning (FL) enables a server and a set of users to jointly train a model without the need to disclose training data. To guarantee strong privacy for users, FL must be combined with a secure aggregation (SA) protocol ensuring that individual gradients remain confidential. Pioneering SA protocols for FL applications exhibit a per-user overhead that grows logarithmically with the number of users and require multiple rounds of communication.**

In this paper we present LISA, a lightweight SA protocol that leverages public randomness to reduce both per-user overhead and the number of rounds needed. LISA requires only two rounds of interaction for most users, offering a communication overhead asymptotically equivalent to that of a non-private protocol where the server obtains user inputs in the clear. More concretely, LISA uses public randomness to select a small committee of users tasked with aiding the server in the aggregation process. Users blind their individual inputs with random masks shared with each committee member, and after sending their blinded input to the server they can go offline. Committee members provide the server with an aggregation of the shared masks over all users, so that the server can remove all masks and obtain the aggregated input. Hence, as long as one committee member is honest, the server can only obtain the aggregation of threshold-many inputs without learning any individual input. We compare LISA with existing SA protocols through both theoretical analysis and simulations. We further integrate LISA in an FL pipeline and compare its accuracy and time to converge with those of a non-private FL application.

Index Terms—**Federated Learning, Secure Aggregation**

I. Introduction

Secure Aggregation (SA) allows a set of parties to compute a linear function (e.g., the sum) of their inputs while keeping each party's input private. Recently, SA protocols have been proposed as a privacy-preserving mechanism for Federated Learning (FL), where a large number of users and a central server train a joint model by leveraging user-private gradients [1], [2].

Most SA protocols require users to provide noisy version of their inputs to the server, so that when all noisy inputs are aggregated, the noise cancels out and the server learns the aggregated gradient. Security of SA protocols is defined against static corruptions of the server and a fraction of the users, and are robust to user drop-outs throughout the protocol execution. Among available SA protocols, the one proposed by Bell *et al.* [2] has communication overhead of $\mathcal{O}(\log^2 n + m)$ for users and $\mathcal{O}(n(\log^2 n + m))$ for the server, given n users, each with an input of size m. Further, the protocol in [2] requires 5.5 rounds of interaction in case of malicious adversaries. Note that a non-private aggregation protocol has a communication overhead of $\mathcal{O}(m)$ for a user, $\mathcal{O}(nm)$ for the server, and requires a single round.

Our contribution. We study how to design SA protocols for federated learning, assuming the availability of a random beacon service, i.e., a source of public randomness. Random beacons can be instantiated with financial data [3] or crypto-currencies [4]. Alternatively, one can use a trusted source of randomness (e.g., https://beacon.nist.gov/home) or distributed protocols to generate public randomness [5], [6], [7]. Random beacons have been used in other application scenarios, such as secure messaging [8], [9], [10], sortition [11] or e-voting [12].

In this paper we present *LIghtweight Secure Aggregation* (LISA), an SA protocol that leverages a public random beacon to minimize overhead. In particular, LISA protects user inputs with random noise. We use the random beacon at each learning epoch to select a small subset of the users, a *committee*, that hold the seeds necessary to remove the noise from the inputs. The technique we employ to add and remove noise from the inputs ensures that, as long as one committee member is honest, a compromised server cannot learn individual user inputs but only the sum of threshold-many inputs. Intuitively, the random choice of committee members guarantees that at least one committee member is honest (with high probability), despite a fraction of the users may be compromised.

By using a randomly-selected committee, LISA does not require each user to secret-share the randomness used to add noise to her input with other peers like in [1], [2]. Users of LISA add noise to their inputs with randomness derived from a non-interactive key agreement with each of the committee members. Hence, committee members must provide the server with the same randomness so that inputs can be de-noised. In order to protect individual user inputs, honest committee members do not share with the server the noise used to blind an individual input, but rather provide the server with aggregated noise, so that the server can de-noise only aggregated inputs.

LISA exhibits a communication overhead of $\mathcal{O}(m)$ for most of the users (e.g., 99.9% of 100k users) and $\mathcal{O}(nm)$ for the server. Further, most users in LISA are required to be online for only 2 rounds. We theoretically compare LISA with previous work in terms of overhead. Further, we use a simulator to compare the per-user communication overhead of our protocol with the one of [2] and show that LISA has a lower average number of messages per user. We also integrate LISA in an FL pipeline and compare its accuracy and time to converge with those of a non-private FL application.

	Communication complexity		Computation complexity		Rounds (#)	Thresholds
	User	Server	User	Server		
Bonawitz *et al.* [1]	$n+m$	n^2+nm	n^2+nm	mm^2	5	$\delta < 1/3$, $\gamma < 1/3$
Bell *et al.* [2]	$\log^2 n + m$	$n\log^2 n + nm$	$\log^2 n + m\log n$	$n\log^2 n + nm\log n$	5.5	$2\delta+\gamma<1$
Stevens *et al.* [13]	$n+m$	nm	$m+n\log n$	$nm+n\log n$	3	honest majority
MicroFedML [14]	$m\log n$	$nm\log n$	nm	nm	3	$2\delta+\gamma<1$
Flamingo [15]	Regular: $m+A$ Decryptors: $L+An$	$n(m+A+L)$	Regular: $Am+L^2+n\log n$ Decryptors: $L+n$	$nm+n^2$	1 3	$2\delta_D+\gamma_D<1/3$
LISA (this work)	Regular: m Committee: $n+m$ Backup: 1	nm	Regular: m Committee: nm Backup: 1	nm	2 3 5	$2\delta+\gamma<1$
Non-private protocol	m	nm	1	nm	1	N/A

TABLE I: Comparison between prior SA protocols (with malicious security) and LISA. Communication and computation complexity are expressed using "Big O" notation. We use n to denote the number of users and m for the size of the user's input. Flamingo considers a set of decryptors of size L; moreover, users distribute their masks with a set of neighbors of size A (similarly to Bell *et al.*). According to [15], L is $\mathcal{O}(1)$ and A is $\mathcal{O}(\log n)$; δ_D denotes the fraction of decryptors that dropout and γ_D denotes the fraction of honest decryptors.

II. RELATED WORK

Single-server SA was introduced by Bonawitz *et al.* [1]. Their protocol requires each user to blind her input with two blindings, one private and one shared with another peer. Both blindings are secret-shared with all other users. A reconstruction protocol allows the server to remove the blindings from the sum of individual blinded inputs. The resulting protocol has an overhead that is linear in the number of users, tolerates user dropouts, and can withstand collusion between a malicious server and a fraction of the users. To the best of our knowledge, only other few SA protocols consider malicious security [2], [13], [14], [15].

Bell *et al.* [2] propose an extension of [1] that reduces the communication overhead by partitioning users in groups of $\log n$ size and restricting communication among members of the same group. Stevens *et al.* [13] use differential privacy techniques but their communication overhead is not on par with [2].

MicroFedML [14] carries out aggregation "at the exponent" of a cyclic group and requires discrete-log computation. As such, MicroFedML is only suitable when the bit-length of the sum of the inputs ℓ is "small" (e.g., 20 bits)—therefore, the input of a single users must be smaller than $2^\ell/n$. In contrast, LISA can work with large-domain inputs. MicroFedML has a setup phase of three rounds, and aggregation takes three rounds for all users. Storage requirement due to setup is $\mathcal{O}(n)$. Aggregation in LISA takes 2 rounds for most users and has no setup phase. If we added a setup phase to LISA where users exchange keys similarly to MicroFedML, then aggregation in LISA would take only one round for most users, but storage requirement would become $\mathcal{O}(n)$.

Similarly to LISA, Flamingo leverages a trusted source of randomness to select a subset of users (called *decryptors* in [15]) that aid the server in computing the aggregation from the masked inputs submitted by users. Flamingo and LISA, however, have a few significant differences. First, Flamingo uses linearly homomorphic encryption whereas LISA leverages zero-sum random shares. Users in Flamingo must be active for one round whereas users in LISA stay online for two rounds. Communication for regular users is $\mathcal{O}(m+A)$ in Flamingo—where A is the size of the neighborhood of a user and should be $\mathcal{O}(\log n)$—while LISA has communication complexity $\mathcal{O}(m)$ for regular users. Flamingo requires a setup phase of 7 rounds, and a periodic share-transfer phase where a new set of decryptors is selected; the share-transfer phase takes 4 rounds. Changing the set of committee members in LISA requires no communication among protocol participants. Finally, and most importantly. Flamingo tolerates less corruptions and drop-outs than LISA. In particular, during the setup phase of Flamingo, at least $\frac{2}{3}$ of the decryptors must be honest and alive; differently, LISA remains secure as long as a single committee member is honest and alive.

Another SA protocol that uses random masks to preserve privacy and leverages a set of "assisting nodes" for the server to recover the aggregate is e-SeaFL [16]. This protocol, however, requires stronger assumptions compared to LISA or the SA protocols described above. In particular, the set of assisting nodes in e-SeaFL is known upfront and does not change—hence an adversary may eventually compromise all of them, thereby breaking privacy. Further, e-SealFL is not robust, i.e., does not tolerate dropouts of assisting nodes. Finally, the authors of [16] present an extension of e-SealFL that provides integrity, but it only works if all nodes (assisting and regular) are honest, i.e., the adversary can corrupt only the server. Differently, LISA does not rely on a fixed set of committee members and can tolerate committee members that dropout throughout the protocol execution. Further, the extension to LISA that provides integrity protection (see the Appendix), tolerates corruptions of the server and a fraction of the nodes, including committee members.

Other SA protocols have been proposed [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], however, they can only withstand a semi-honest adversary and, in some cases, they do not consider collusion between server and users.

We summarize closely related work and compare it to LISA in Table I. The table also reports overhead and number of rounds of a non-private protocol where users sends their inputs to the server in the clear.

III. Background

Notations. For $m, n \in \mathbb{N}$, we denote by $[m..n]$ the set of integers $\{i : m \leq i \leq n\}$, and we use the shorthand $[n] := [1..n]$. We write $x \leftarrow v$ for the assignment of value v to variable x.

Cryptographic primitives. We now briefly describe the cryptographic primitives used in LiSA. We omit their complete definitions because they have been used (and defined) in previous work on secure aggregation [1], [2].

A secret-sharing scheme SS with algorithms SS.Share and SS.Rec. The sharing algorithm $\{s_i\}_{i\in[n]} \leftarrow \mathsf{SS.Share}(s, t, n)$ takes as input a secret s, a reconstruction threshold t and a target number of shares n and outputs a set of shares $\{s_i\}_{i\in[n]}$. The reconstruction algorithm $s \leftarrow \mathsf{SS.Rec}(s_{i_1}, \ldots, s_{i_l})$ takes as input a set of shares $s_{i_1}, \ldots, s_{i_l}$ such that $l \geq t$ and outputs a secret s. In practice, SS can be instantiated with Shamir secret-sharing [27].

A non-interactive key agreement scheme KA with algorithm $(sk_i, pk_i) \leftarrow \mathsf{KA.Gen}(1^\lambda)$ to generate a key-pair and algorithm $k_{i,j} \leftarrow \mathsf{KA}(sk_i, pk_j)$ to generate a shared key between party i and party j. We add a "context" string CTX to the key agreement protocol and write $\mathsf{KA}(\text{"CTX"}; sk_i, pk_j)$ so to allow party i and party j to derive multiple keys, by using different context strings. KA can be instantiated with DHKE [28] followed by a key-derivation function.

A pseudorandom number generator (PRG) F that can be instantiated with AES-CTR [1].

An authenticated encryption scheme AE. We denote by $c \leftarrow \mathsf{AE.Enc}(k, m)$ the encryption of message m under key k, and $m \leftarrow \mathsf{AE.Dec}(k, c)$ to denote the decryption of ciphertext c under key k. Authenticated encryption can be instantiated with AES-GCM [29].

A digital signature scheme DS with key-generation algorithm $(sk, pk) \leftarrow \mathsf{DS.Gen}(1^\lambda)$, signing algorithm $\sigma \leftarrow \mathsf{DS.Sign}(sk, m)$, and verification algorithm $\{0, 1\} \leftarrow \mathsf{DS.Verif}(pk, \sigma, m)$. In practice, DS can be instantiated with ECDSA [30].

Hypergeometric distribution. The hypergeometric distribution $\mathcal{HG}(N, C, n)$, where N is the overall number of elements in a set, $0 \leq C \leq N$ is the number of "special" elements, and $0 \leq n \leq N$ is the number of extracted elements, without repetition, among all elements, defines the probability of having a certain number of special elements in the extracted sample. For a random variable $X \sim \mathcal{HG}(N, C, n)$, we have:

$$\Pr[X = x] = \frac{\binom{C}{x}\binom{N-C}{n-x}}{\binom{N}{n}} \quad \text{and} \quad \mathbb{E}[X] = n\frac{C}{N}. \tag{1}$$

For $0 \leq t \leq n\frac{C}{N}$, the following tail bounds hold [31]:

$$\Pr[X \geq \mathbb{E}[X] + tn] \leq e^{-2t^2 n}, \tag{2}$$

$$\Pr[X \leq \mathbb{E}[X] - tn] \leq e^{-2t^2 n}. \tag{3}$$

IV. LiSA

System and threat model. We assume a single server S and set of users $\mathcal{U} = [n]$. Each user $i \in \mathcal{U}$ holds private input x_i, and the protocol allows the server to learn $y = \sum_{i\in\mathcal{U}} x_i$. We seek to design a *robust* protocol, which provides the aggregation output despite up to a δ fraction of users going offline at any time during the protocol execution.

We consider the exact same threat model of [1], [2]. Users have no direct communication channels as all messages are routed through the server. In line with previous work, we assume a Public Key Infrastructure (PKI) to distribute genuine user public keys. Thus, any pair of users can establish a confidential and authenticated channel. Further, each user has a confidential and authenticated channel with the server. In particular, we account for *semi-honest* as well as *malicious* settings. In the semi-honest setting, corrupted parties follow the protocol as expected; in the malicious setting, corrupted parties can act arbitrarily. In both cases, we allow the adversary to corrupt the server and a γ fraction of the users, statically. The security of our protocols guarantees that the input of an honest user is aggregated with at least an $\alpha-$fraction of all other inputs before it is revealed to the server in the clear. In other words, the server learns $y = \sum_{i\in\mathcal{U}'} x_i$ only if $\mathcal{U}' \subseteq \mathcal{U}$ and $|\mathcal{U}'| \geq \alpha n$.

In line with previous work on SA [1], [2], we consider denial of services attacks or attacks to the integrity of the computation as out of scope.

Semi-honest Protocol. We describe the semi-honest protocol in Figure 1 (only text in black). In LiSA, each user can assume up to three roles: committee member, backup neighbor, and regular user (i.e., those neither selected as committee members nor as backup neighbors). We assume each user $i \in \mathcal{U}$ to have two key-pairs (sk_i, pk_i) (SK_i, PK_i) used for non-interactive key agreement. Key pair (SK_i, PK_i) is used by user i when she acts as a committee member, whereas key-pair (sk_i, pk_i) is used for all other purposes.

During Round 1, all users and the server use the random beacon to select a *committee* of users, denoted as $\mathcal{K}$, at random. In particular, we use $\mathcal{K} \leftarrow \mathsf{Select}(Q, \mathcal{U}, k)$ to denote a function that selects k users from $\mathcal{U}$ by means of a PRG seeded with randomness Q.

During Round 2, each user i adds noise to her input x_i by using k random masks, each shared with one of the committee members. In particular, user i can agree on a shared key with committee member j by running $k^*_{i,j} = \mathsf{KA}(\text{"PRG"}; sk_i, PK_j)$ and then obtain a mask as $F(k^*_{i,j})$, where F is a PRG. The noisy input, denoted as c_i, is sent to the server.

The server aggregates the noisy inputs as c_{agg}, and asks each of the committee members for the aggregated randomness required to de-noise c_{agg} (Round 3). Let $\mathcal{U}'_2$ be the set of users from who the server receives a noisy input. Each committee member j runs a non-interactive key agreement with each user $i \in \mathcal{U}'_2$ to obtain $\{k^*_{j,i} = \mathsf{KA}(SK_j, pk_i)\}_{i\in\mathcal{U}'_2}$. Next, committee member j uses each of the agreed keys to obtain the shared masks by means of a PRF, aggregates all the shared masks as

Parties: Server and users $\mathcal{U} = [n]$.
Public parameters: input domain $\mathbb{X}$, fraction of drop-outs δ, fraction of corruptions γ, security parameter for cryptographic primitives λ, committee size k, backup-neighborhood size ℓ, secret sharing reconstruction threshold t, minimum fraction of aggregated inputs α, maximum number of corrupt committee members $\tilde{c}$.
Prerequisites: Each user $i \in \mathcal{U}$ has key-pairs computed as $(sk_i, pk_i) \leftarrow \mathsf{KA.Gen}(1^\lambda)$, $(SK_i, PK_i) \leftarrow \mathsf{KA.Gen}(1^\lambda)$ and $(sk_i^s, pk_i^s) \leftarrow \mathsf{DS.Gen}(1^\lambda)$; public keys (pk_i, PK_i, pk_i^s) are registered with the PKI.
For $\mathcal{U}$ and $r \in [1..6]$ and, we denote by $\mathcal{U}_r$ the set of users that complete the execution of round r without dropping out, and we denote by $\mathcal{U}'_r$ the set of users the server knows have completed round r. It holds $\mathcal{U}'_r \subseteq \mathcal{U}_r \subseteq \mathcal{U}_{r-1}$ for all $r \in [1..6]$. We also set $\mathcal{K}_{alive} := \mathcal{K} \cap \mathcal{U}'_3$, $\mathcal{K}_{drop} := \mathcal{K} \setminus \mathcal{K}_{alive}$, and $\mathcal{L} = \bigcup_{j \in \mathcal{K}} \mathcal{L}_j$.

```
Round 1
  Each party
  1) receives random seed $Q$
  2) selects committee $\mathcal{K} \leftarrow \mathsf{Select}(Q, \mathcal{U}, k)$
Round 2
  Committee member $j \in \mathcal{K}$
  1) selects backup neighbors
         $\mathcal{L}_j \leftarrow \mathsf{Select}(Q||j, \mathcal{U} \setminus \{j\}, \ell)$
  2) fetches their public keys from PKI
         $\{pk_i\}_{i \in \mathcal{L}_j} \leftarrow PKI$
  3) derives symmetric keys
         $\{k^e_{j,i} \leftarrow \mathsf{KA}(\text{"ENC"}; SK_j, pk_i)\}_{i \in \mathcal{L}_j}$
  4) secret shares key $SK_j$
         $\{S_{j,i}\}_{i \in \mathcal{L}_j} \leftarrow \mathsf{SS.Share}(SK_j, t, \ell)$
  5) encrypts shares of $SK_j$
         $\{E_{j,i} \leftarrow \mathsf{AE.Enc}(k^e_{j,i}, S_{j,i})\}_{i \in \mathcal{L}_j}$
  6) sends $\{(j; i; E_{j,i})\}_{i \in \mathcal{L}_j}$ to the server
  User $i \in \mathcal{U}$
  1) fetches public keys from PKI
         $\{PK_j\}_{j \in \mathcal{K}} \leftarrow PKI$
  2) derives symmetric keys
         $\{k^*_{i,j} \leftarrow \mathsf{KA}(\text{"PRG"}; sk_i, PK_j)\}_{j \in \mathcal{K}}$
  3) computes blinded input
         $c_i \leftarrow x_i + \sum_{j \in \mathcal{K}} F(k^*_{i,j})$
  4) sends $c_i$ to the server
Round 3
  Server
  1) receives encrypted key shares $\{(j; i; E_{j,i})\}_{i \in \mathcal{L}_j}$
         from $j \in \mathcal{K} \cap \mathcal{U}'_2$ and sends each of them
         to corresponding backup neighbor
  2) receive blinded inputs $c_i$ from users $i \in \mathcal{U}'_2$
  3) aggregates input $c_{agg} \leftarrow \sum_{i \in \mathcal{U}'_2} c_i$
  4) sends $\mathcal{U}'_2$ to committee members $j \in \mathcal{K} \cap \mathcal{U}'_2$.
  Committee member $j \in \mathcal{K} \cap \mathcal{U}'_2$
  1) receives $\mathcal{U}'_2$ from the server
  2) if $|\mathcal{U}'_2| < \alpha n$, aborts
  3) fetches public keys from PKI
         $\{pk_i\}_{i \in \mathcal{U}'_2} \leftarrow PKI$
  4) derives symmetric keys
         $\{k^*_{j,i} \leftarrow \mathsf{KA}(\text{"PRG"}; SK_j, pk_i)\}_{i \in \mathcal{U}'_2}$
  5) computes partial blinding
         $\partial_j \leftarrow \sum_{i \in \mathcal{U}'_2} F(k^*_{j,i})$
  6) sends $\partial_j$ to the server
Round 4
  Server
  1) receives $\{\partial_j\}_{j \in \mathcal{K}_{alive}}$
  2) if $|\mathcal{K}_{alive}| = k$ go-to step (2) of Round 6
  3) sends $\mathcal{K}_{drop}$ to all users in $\mathcal{L}$
  Backup neighbor $i \in \mathcal{L}$
  1) receives $\mathcal{K}_{drop}$ from the server
  2) if $|\mathcal{K}_{drop}| \geq k - \tilde{c}$ then aborts
  3) computes signature $\sigma_i \leftarrow \mathsf{DS.Sign}(sk^s_i, \mathcal{K}_{drop})$
  4) sends $\sigma_i$ to the server
Round 5
  Server
  1) receives $\{\sigma_i\}_{i \in \mathcal{L} \cap \mathcal{U}'_4}$ from users $\mathcal{L} \cap \mathcal{U}'_4$
         and forwards them to all users in $\mathcal{L} \cap \mathcal{U}'_4$
  Backup neighbor $i \in \mathcal{L} \cap \mathcal{U}'_4$
  1) receives $\{\sigma_i\}_{i \in \mathcal{L} \cap \mathcal{U}'_4}$
  2) fetches public keys from PKI
         $\{pk^s_i\}_{i \in \mathcal{L} \cap \mathcal{U}'_4} \leftarrow PKI$
  3) computes set
         $\mathcal{L}_{ack} = \{l \in \mathcal{L} \cap \mathcal{U}'_4 : \mathsf{DS.Verif}(pk^s_l; \sigma_l; \mathcal{K}_{drop}) = 1\}$
  4) if $|\mathcal{L}_j \cap \mathcal{L}_{ack}| < t$ for any $j \in \mathcal{K}$ then aborts
  5) for any $j \in \mathcal{K}_{drop}$ such that $i \in \mathcal{L}_j$,
     a) fetches $pk_j$ from the PKI
     b) derives symmetric key
            $(k^e_{i,j}) \leftarrow \mathsf{KA}(\text{"ENC"}; sk_i, pk_j)$
     c) decrypts share
            $S_{j,i} \leftarrow \mathsf{AE.Dec}(k^e_{i,j}; E_{j,i})$
     d) sends $S_{j,i}$ to the server
Round 6
  Server
  1) For each $j \in \mathcal{K}_{drop}$
     a) collects shares $\{S_{j,i}\}_{i \in \mathcal{L}_j \cap \mathcal{U}'_5}$
            and aborts if receives less than $t$ shares
     b) reconstructs secret key
            $SK_j \leftarrow \mathsf{SS.Rec}(\{S_{j,i}\}_{i \in \mathcal{L}_i \cap \mathcal{U}'_5})$
     c) derives symmetric keys
            $\{k^*_{j,i} \leftarrow \mathsf{KA}(\text{"PRG"}; SK_j; pk_i)\}_{i \in \mathcal{U}'_2}$
     d) computes missing partial blinding
            $\partial'_j \leftarrow \sum_{i \in \mathcal{U}'_2} F(k^*_{i,j})$
  2) given $\{\partial_j\}_{j \in \mathcal{K}} = \{\partial_j\}_{j \in \mathcal{K}_{alive}} \cup \{\partial_j\}_{j \in \mathcal{K}_{drop}}$,
         computes the output $y \leftarrow c_{agg} - \sum_{j \in \mathcal{K}} \partial_j$
```

Fig. 1: LISA secure aggregation protocol. Instructions highlighted in red are executed only in the malicious version.

∂_j and sends it to the server. Finally, the server obtains the aggregated masks from each committee member and subtracts those to c_{agg} to remove the noise and obtain the aggregated output.

The above protocol is not robust against dropouts by committee members. If a user $j \in \mathcal{K}$ goes offline before she sends her aggregated noise share ∂_j to the server (Round 3, step 6), then the aggregated noisy input can no longer be de-noised. We add robustness by selecting a set $\mathcal{L}_j$ of ℓ *backup neighbors* for each committee member j, and by asking j to secret-share her secret key SK_j among them, with reconstruction threshold $t \leq \ell$ (Round 2).

Similar to the committee selection, the selection of a backup neighborhood uses the randomness provided by the random beacon service. In this case, each committee member j runs $\mathcal{L}_j \leftarrow \mathsf{Select}(Q||j, \mathcal{U}, l)$ to select l backup neighbors.[1]

In case a committee member j goes offline before she can send the aggregated noise share, the server can recover j's secret key by asking for her shares to backup neighbors in $\mathcal{L}_j$ (Round 4). If t-many users in $\mathcal{L}_j$ send a share of SK_j to the server (Round 5), then the server can recover the key and compute the aggregated noise share (Round 6), despite j being offline.

Backup neighbors aid the server in recovering the secret key of a committee member only if the number of dropped-out committee members $|\mathcal{K}_{drop}|$ is below $k - \tilde{c}$, where $\tilde{c}$ is the expected number of committee members that are corrupted. This is to make sure that even by corrupting some committee members and by recovering some secret keys via backup neighbors, the server is still missing at least one secret key of a committee member and, therefore, cannot de-noise individual noisy inputs of victim users.

Note that regular users need to be online only until Round 2, then they can go offline. Committee members and backup neighbors are required to remain online until Round 3 and until Round 5, respectively.

Also note that Figure 1 assumes each user to encrypt a single message to ease readability. In case users encrypt vectors of size m, each element is encrypted separately, and aggregation is carried-out element-wise.

Malicious Protocol. The malicious protocol follows the same blueprint of the semi-honest one. Differences between the two protocol stems from the ability of a corrupted server to act arbitrarily. Red text in Figure 1 shows the changes we make to the semi-honest protocol so to withstand a malicious server.

A malicious server may declare different sets of dropped-out committee members, each of size $< k - \tilde{c}$, to different backup neighborhoods, so to obtain their key-shares. If the server manages to obtain the secret keys of each honest committee member, it can remove noise from any single noisy input, thereby breaking security. Hence, we use Round 4 and Round 5 to run a consistency check among backup neighbors, so to agree on the set of dropped-out committee members; if the size of this set is bigger than threshold $k - \tilde{c}$, then no honest backup neighbor will help the server in recovering the missing keys.

[1]We seed the PRG of the Select function with $Q||j$—the random beacon Q concatenated with the index of the committee member j—so that different committee members obtain independent sets of backup neighbors.

V. Analysis

In this section we prove that LiSA is correct, i.e., the server obtains the aggregation of users provided inputs, as long as all parties follow the protocol and at most δn users dropout. We also prove that LiSA is secure, i.e., if the protocol execution completes then the server learns the aggregation of no less than αn inputs, despite a malicious server and up to γn compromised users.

Overview. We will analyze the probability of certain events occurring during an execution of the secure aggregation protocol, and relate them to the probability of violating correctness or security.

Let $\mathcal{C}$ be the set of corrupted users and let $\mathcal{D}$ be the set of users that drop out. Further, let d, d_j, c, and c_j denote the number of dropped committee members, of dropped backup neighbors of committee member j, of corrupt committee members, and of corrupt backup neighbors of committee member j, respectively. That is:

$$d := |\mathcal{K} \cap \mathcal{D}|, \quad d_j := |\mathcal{L}_j \cap \mathcal{D}|, \quad c := |\mathcal{K} \cap \mathcal{C}|, \quad c_j := |\mathcal{L}_j \cap \mathcal{C}|.$$

Recall that LiSA (pseudo-)randomly selects committee members and backup neighbors, based on a public random seed. As users in the committee, as well as in each backup neighborhood, must be distinct, we can think of selecting them, without replacement, from a population of n users. Then, in the worst case with all δn users dropping out and all γn users being corrupt, i.e., $|\mathcal{D}| = \delta n$ and $|\mathcal{C}| = \gamma n$, random variables d, d_j, c and c_j are distributed according the hypergeometric distribution (cf. Section III):

$$d \sim \mathcal{HG}(n, \delta n, k), \quad d_j \sim \mathcal{HG}(n-1, \delta n, \ell), \tag{4}$$

$$c \sim \mathcal{HG}(n, \gamma n, k), \quad c_j \sim \mathcal{HG}(n-1, \gamma n, \ell). \tag{5}$$

Correctness. In addition to the correctness of the cryptographic building blocks of LiSA, we need to define some requirements to guarantee robustness. This means limiting the number of drop-outs in the committee and in the backup neighborhoods, as specified through the following events.

Event C_1: all committee members stay alive

$$|\mathcal{K} \setminus \mathcal{D}| = k. \tag{6}$$

Event C_2: sufficiently many committee members stay alive:

$$|\mathcal{K} \setminus \mathcal{D}| > \tilde{c}. \tag{7}$$

Event C_3: sufficiently many backup neighbors of dropped committee members stay alive:

$$\forall j \in \mathcal{K} \cap \mathcal{D} \ : \ |\mathcal{L}_j \setminus \mathcal{D}| \geq t. \tag{8}$$

Correctness of LiSA requires that either all committee members complete the protocol (event C_1), or for each committee member j that dropped out, the server can recover enough shares from neighbors in $\mathcal{L}_j$ to compute the secret key(s) of j (events C_2 and C_3). Therefore, it is sufficient to select parameters k, ℓ, and t (based on n and δ) so that the event $C_1 \vee (C_2 \wedge C_3)$ occurs with overwhelming probability. We can ensure these requirements are met with high probability by selecting parameters k, ℓ, and t such that

$$\Pr\left[C_1 \vee (C_2 \wedge C_3)\right] \geq 1 - 2^{-\eta}. \tag{9}$$

To this end, we consider the complement event:

$$\Pr\left[C_1 \vee (C_2 \wedge C_3)\right] \geq 1-2^{-\eta} \iff \Pr\left[\neg C_1 \wedge \neg(C_2 \wedge C_3)\right] \leq 2^{-\eta} \tag{10}$$

where

$$\begin{aligned}\Pr\left[\neg C_1 \wedge \neg(C_2 \wedge C_3)\right] &= \Pr\left[\neg C_1 \wedge (\neg C_2 \vee \neg C_3)\right] &(11)\\ &\leq \Pr\left[\neg C_1 \wedge \neg C_2\right] + \Pr\left[\neg C_1 \wedge \neg C_3\right]. &(12)\end{aligned}$$

By observing that

$$\begin{aligned}\neg C_1 &: |\mathcal{K} \setminus \mathcal{D}| < k \iff |\mathcal{K} \cap \mathcal{D}| > 0 &(13)\\ \neg C_2 &: |\mathcal{K} \setminus \mathcal{D}| \leq \tilde{c} \iff |\mathcal{K} \cap \mathcal{D}| \geq k - \tilde{c} &(14)\\ \neg C_3 &: \exists j \in \mathcal{K} \cap \mathcal{D} \ : \ |\mathcal{L}_j \setminus \mathcal{D}| < t &(15)\\ &\iff \exists j \in \mathcal{K} \cap \mathcal{D} \ : \ |\mathcal{L}_j \cap \mathcal{D}| \geq \ell - t &(16)\end{aligned}$$

we obtain

$$\begin{aligned}\Pr\left[\neg C_1 \wedge \neg C_2\right] &= \Pr\left[(d > 0) \wedge (d \geq k - \tilde{c})\right] &(17)\\ &= \Pr\left[d \geq k - \tilde{c}\right] &(18)\end{aligned}$$

and

$$\begin{aligned}\Pr\left[\neg C_2 \wedge \neg C_3\right] &= \Pr\left[(d > 0) \wedge \exists j \in |\mathcal{K} \cap \mathcal{D}| : d_j \geq \ell - t\right] &(19)\\ &\leq \Pr\left[\exists j \in \mathcal{K} : d_j \geq \ell - t\right] &(20)\\ &\leq \sum_{j \in \mathcal{K}} \Pr\left[d_j \geq \ell - t\right] &(21)\end{aligned}$$

Putting it all together, we require:

$$\Pr\left[d \geq k - \tilde{c}\right] + \sum_{j \in \mathcal{K}} \Pr\left[d_j \geq \ell - t\right] \leq 2^{-\eta}. \tag{22}$$

Let $X \sim \mathcal{HG}(n, \delta n, k)$ and $Y \sim \mathcal{HG}(n-1, \delta n, \ell)$. By Equation (4), we can express the requirement above as:

$$\Pr\left[X \geq k - \tilde{c}\right] + k \ \Pr\left[Y \geq \ell - t\right] \leq 2^{-\eta}. \tag{23}$$

The following two conditions imply equation (23):

$$\begin{aligned}\Pr\left[X \geq k - \tilde{c}\right] &\leq 2^{-(\eta+1)} &(24)\\ k \ \Pr\left[Y \geq \ell - t\right] &\leq 2^{-(\eta+1)} &(25)\end{aligned}$$

Let $\tilde{c} = \tilde{\gamma} k$, with $\gamma < \tilde{\gamma} < 1 - \delta$. Using the tail bounds of the hypergeometric distribution (cf. Section III), we have

$$\begin{aligned}\Pr\left[X \geq k - \tilde{c}\right] &= \Pr\left[X > \delta k + (1-\delta)k - \tilde{\gamma}k\right] &(26)\\ &= \Pr\left[X > \mathbb{E}[X] + (1 - \delta - \tilde{\gamma})k\right] &(27)\\ &\leq e^{-2(1-\delta-\tilde{\gamma})^2 k}, &(28)\end{aligned}$$

and similarly

$$\begin{aligned}\Pr\left[Y \geq \ell - t\right] &= \Pr\left[Y \geq \frac{\delta n}{n-1}\ell + \left(1 - \frac{\delta n}{n-1}\right)\ell - \frac{t}{\ell}\ell\right] &(29)\\ &= \Pr\left[Y \geq \mathbb{E}[Y] + \left(1 - \frac{\delta n}{n-1} - \beta\right)\ell\right] &(30)\\ &\leq e^{-2(1-\frac{\delta n}{n-1}-\beta)^2 \ell}. &(31)\end{aligned}$$

with $t = \beta\ell$ and $\beta < 1 - \delta$. We therefore require

$$e^{-2(1-\delta-\tilde{\gamma})^2 k} < 2^{-(\eta+1)} \implies k > \frac{1}{2\log(e)} \frac{\eta + 1}{(1 - \delta - \tilde{\gamma})^2} \tag{32}$$

and

$$k \ e^{-2(1-\delta-\frac{t}{\ell})^2 \ell} < 2^{-(\eta+1)} \implies \ell > \frac{\log(k) + \eta + 1}{2\log(e)(1 - \frac{\delta n}{n-1} - \beta)^2} \tag{33}$$

We finally obtain

$$\begin{aligned}k &> \frac{1}{2\log(e)} \frac{\eta + 1}{(1 - \delta - \tilde{\gamma})^2} &(34)\\ \ell &> \frac{\log(k) + \eta + 1}{2\log(e)(1 - \frac{\delta n}{n-1} - \beta)^2} &(35)\end{aligned}$$

All cryptographic primitives invoked by LiSA are assumed to be correct. Therefore, we can state the following result.

Theorem 1 (Correctness)**.** *Consider an execution of* LiSA *with user inputs $\{x_i\}_{i \in \mathcal{U}}$. If no more than δn users dropout throughout the execution, i.e., $|\mathcal{U}'_5| \geq (1-\delta)n$, then the protocol completes and the server obtains output $y = \sum_{i \in \mathcal{U}'_1} x_i$ with overwhelming probability.*

Security. Similarly to the case of correctness, we identify relevant events that establish a limit on the number of corrupt users in the committee and in the backup neighborhoods.

Intuitively, if the protocol completes, at least one committee member must be honest and alive. Otherwise, the server would obtain the decryption keys of the c committee members it corrupts, as well as the decryption keys of the $k - c$ dropped-out committee members from the corrupted backup neighbors. Once the server has the secret keys of all committee members, it can decrypt any individual ciphertext, thereby breaking security. Hence, we must ensure the following:

$$|\mathcal{K} \cap \mathcal{C}| + |\mathcal{K} \cap \mathcal{D}| < |\mathcal{K}|. \tag{36}$$

To meet this requirement, our protocol instructs backup neighbors to release shares of committee members' decryption keys only if not too many of them have dropped out, where "too many" is set as

$$|\mathcal{K} \cap \mathcal{D}| < k - \tilde{c}. \tag{37}$$

Combining this with the requirement in equation (36), we have:

$$|\mathcal{K} \cap \mathcal{C}| < |\mathcal{K}| - |\mathcal{K} \cap \mathcal{D}| = k - (k - \tilde{c}) = \tilde{c} \tag{38}$$

Moreover, the protocol must ensure that backup neighbors do not release (too many) shares of honest and alive committee members. The reason is that the malicious server could equivocate during the consistency check, and try to obtain the decryption keys of more than $k - \tilde{c} - 1$ committee members by presenting different sets of dropped committee members to different backup neighbors.

For a victim $j \in \mathcal{K}$, consider two subsets of her backup neighbors $S_1, S_2 \subseteq \mathcal{L}_j$, of size $|S_1| = |S_2| = t$, that overlap only on corrupt members, i.e., $S_1 \cap S_2 \subseteq \mathcal{C}$. Then, the server could present to all users in S_1 a set of dropped users $\mathcal{K}_{drop,1}$ such that $j \in \mathcal{K}_{drop,1}$, and to all users in S_2 a set $\mathcal{K}_{drop,2}$ such that $j \notin \mathcal{K}_{drop,2}$. In this way, the server would obtain t shares $s_{j,i}$ from the backup neighbors $i \in S_1$ and reconstruct key sk_j, and would also obtain t signatures acknowledging that j is alive from the backup neighbors in S_2, which would allow the server to reconstruct more decryption keys than we want (i.e., more than $k - \tilde{c} - 1$).

To prevent the server from equivocating as in the example above, we therefore require that any two sets $S_1, S_2 \subseteq \mathcal{L}_j$ authorized to release shares overlap in at least an honest member:

$$S_1 \cap S_2 \setminus \mathcal{C} \neq \emptyset. \tag{39}$$

Using the expression $|S_1 \cap S_2| = |S_1| + |S_2| - |S_1 \cup S_2|$, and considering the worst-case scenario where $\mathcal{C} \cap \mathcal{L}_j \subseteq S_1 \cap S_2$, we have

$$|S_1 \cap S_2 \setminus \mathcal{C}| \geq 0 \iff |S_1| + |S_2| - |S_1 \cup S_2| - c_j \geq 0. \tag{40}$$

Therefore, we derive the minimum size $t = |S_1| = |S_2|$ of backup sets that are authorized to release shares by observing that, as $|S_1 \cup S_2| \leq \ell$, it suffices to require that

$$2t - \ell - c_j > 0. \tag{41}$$

Putting it all together, we establish the following security requirements.

Event S_1: Not too many corrupt committee members:

$$|\mathcal{K} \cap \mathcal{C}| \leq \tilde{c}. \tag{42}$$

Event S_2: Not too many corrupt backup neighbors:

$$\forall j \in \mathcal{K} : |\mathcal{L}_j \cap \mathcal{C}| \leq 2t - \ell. \tag{43}$$

The security of LISA requires $S1 \wedge S2$. By selecting parameters k, ℓ, and t (based on n and γ) we can ensure that this event occurs with overwhelming probability.

Again considering the complementary event, we have:

$$\Pr[S_1 \wedge S_2] \geq 1 - 2^{-\lambda} \iff \Pr[\neg S_1 \vee \neg S_2] \leq 2^{-\lambda} \tag{44}$$

We thus have

$$\begin{aligned}\Pr[\neg S_1 \vee \neg S_2] &\leq \Pr[\neg S_1] + \Pr[\neg S_2] \\ &\leq \Pr[c > \tilde{c}] + \Pr[\cup_{j \in \mathcal{K}} c_j > 2t - \ell] \\ &\leq \Pr[c > \tilde{c}] + \sum_{j \in \mathcal{K}} \Pr[c_j > 2t - \ell]\end{aligned}$$

Now, let $Z \sim \mathcal{HG}(n, \gamma n, k)$ and $W \sim \mathcal{HG}(n-1, \gamma n, \ell)$. Then $c \sim Z$ and $c_j \sim W$ for all $j \in \mathcal{K}$, and we can express the above requirement as

$$\Pr[Z > \tilde{c}] + k \ \Pr[W > 2t - \ell] \leq 2^{-\lambda} \tag{45}$$

The following two conditions imply equation (45):

$$\Pr[Z \geq \tilde{c}] \leq 2^{-(\lambda+1)} \tag{46}$$

$$k \ \Pr[W \geq 2t - \ell] \leq 2^{-(\lambda+1)} \tag{47}$$

Again using the tail bounds of the hypergeometric distribution, and setting $\tilde{c} = \tilde{\gamma} k$, we find:

$$\begin{aligned}\Pr[Z \geq \tilde{c}] &= \Pr[Z \geq \gamma k - \gamma k + \tilde{\gamma} k] &(48)\\ &= \Pr[Z \geq \mathbb{E}[Z] + (\tilde{\gamma} - \gamma)k] &(49)\\ &\leq e^{-2(\tilde{\gamma}-\gamma)^2 k} &(50)\end{aligned}$$

and by requiring $e^{-2(\tilde{\gamma}-\gamma)^2 k} < 2^{-(\lambda+1)}$ we finally obtain

$$k > \frac{\lambda + 1}{2\log(e)(\tilde{\gamma} - \gamma)^2}. \tag{51}$$

Similarly, we require:

$$\Pr[W \geq 2t - \ell] = \Pr\left[W \geq \frac{\gamma n}{n-1}\ell - \frac{\gamma n}{n-1}\ell + 2(\beta - 1)\ell\right] \tag{52}$$

$$= \Pr\left[W \geq \mathbb{E}[W] - (2\beta - \frac{\gamma n}{n-1} - 1)\ell\right] \tag{53}$$

$$\leq e^{-2(2\beta - \frac{\gamma n}{n-1} - 1)^2 \ell} \tag{54}$$

with $t = \beta\ell$ and $\frac{1}{2}(1 + \frac{\gamma n}{n-1}) < \beta$. By requiring $k \ e^{-2(2\beta - \frac{\gamma n}{n-1} - 1)^2 \ell} < 2^{-(\lambda+1)}$ we finally obtain

$$\ell > \frac{\log(k) + \lambda + 1}{2\log(e)(2\beta - \frac{\gamma n}{n-1} - 1)^2} \tag{55}$$

As $\frac{1}{2}(1 + \frac{\gamma n}{n-1}) < \beta < 1 - \delta$, we require that $1 - 2\delta - \frac{\gamma n}{n-1} > 0$.

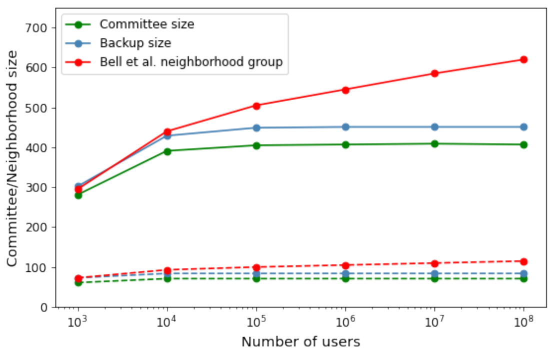
Putting it all together, we can select parameters k, ℓ and t so that LISA guarantees the security requirements are fulfilled.

To formally prove that our protocol (instantiated with k, ℓ and t as above) provides privacy of users' inputs against a malicious adversary controlling the server and a subset $\mathcal{C}$ of users, we show how to construct a simulator whose output is computationally indistinguishable from the output of any such malicious adversary. The simulator is given as input only the protocol leakage, e.g., the output of the server, and simulates a faithful protocol execution for the adversary. The security proof is available in the appendix.

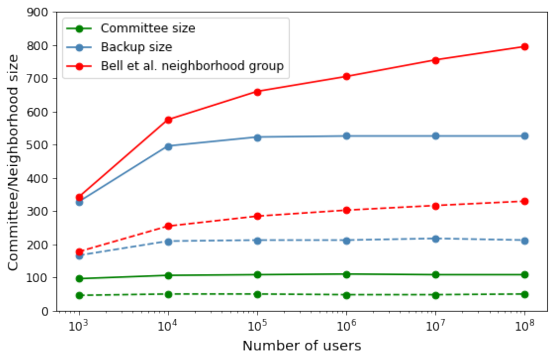
VI. Performance Evaluation

Theoretical Evaluation. We theoretically evaluate the communication of users and the server for the malicious version of our protocol.

Regular users: $\mathcal{O}(m + k)$. Must be online only during the first two rounds. They fetch public keys of k committee members from the PKI and send m noisy inputs.



(a) Semi-honest setting



(b) Malicious setting

Fig. 2: Size of committee and backup neighborhood for LISA, and of neighborhood for Bell. Semi-honest setting (Fig 2a): solid lines assume $\gamma = 0.33, \delta = 0.33$; dashed lines assume $\gamma = 0.05, \delta = 0.33$; we set $\lambda = 40, \eta = 20$. Malicious settings (Fig 2b): solid lines assume $\gamma = 0.2, \delta = 0.2$; dashed lines assume $\gamma = 0.05, \delta = 0.2$; we set $\lambda = 40, \eta = 30$.

Committee members: $\mathcal{O}(n+m+\ell)$. Fetches ℓ public keys from the PKI and sends ℓ ciphertexts Round 2. Next, in Round 3, receives up to n public keys and sends m random values.
Backup neighbor: $\mathcal{O}(k\ell)$. Receives receives up to k ciphertexts during Round 3. In Round 4, each backup neighbor receives one messages from the server and sends one signature. Next, during Round 5 it receives up to $k\ell$ signatures and sends up to k shares.
Server: $\mathcal{O}(nm + k^2\ell^2)$. Receives $k\ell$ shares and nm blinded inputs during Round 3, and sends $k\ell$ shares. During Round 4, the server receives up to km partial blindings and sends one message to up to $k\ell$ users. During Round 5, it receives up to $k\ell$ signatures and sends the same number of signatures to each member of each backup neighborhood. Finally, during round 6 it receives up to $k\ell$ shares. The server aggregates up to m partial blindings. In case committee members drop out, the server makes up to k calls to SS.Rec, up to nm calls to KA, and up to nm partial blindings recoveries.

	Semi-honest	**Malicious**	**# Rounds**
User	m	m	2
Committee	$n+m$	$n+m$	3
Backup	1	1	5
Server	nm	nm	6

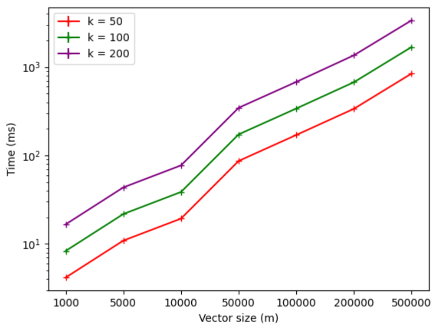
TABLE II: Theoretical communication complexity (expressed using "Big O" notation) and number of rounds in LISA.

Table II summarizes asymptotic communication overhead for each type of party. For completeness, the table also includes the overhead for the semi-honest protocol and the number of rounds. As shown in Section V, k and ℓ are $\mathcal{O}(1)$. The overhead for a user is consequently reduced from $\mathcal{O}(k+m)$ to $\mathcal{O}(m)$.

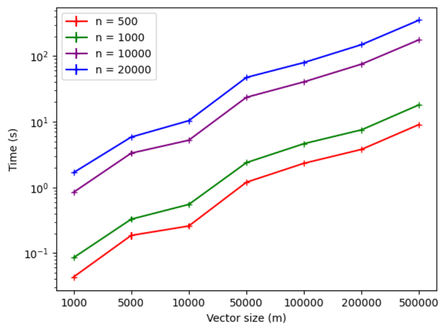
Numerical parameter optimization. LISA takes only 2 rounds for regular users and 3 rounds for committee members. Backup neighbors are required to be online for 5 out of 6 rounds, but only in case any committee member drops out before the third round is over. In the following section, we show how many committee members (k) and how many backup neighbors (ℓ) are needed to satisfy the security and correctness requirements, given n users, corruption rate γ, and dropout rate δ. We will show that the vast majority of users act as regular users and thus can complete the protocol in 2 rounds with little overhead. We also compare our committee and neighborhood sizes with the neighborhood size of Bell *et al.* [2] ("Bell" from now on). To this end, we wrote a program in Julia [32] to obtain the minimum sizes k and ℓ for LISA to satisfy security and correctness conditions, both in the semi-honest and malicious cases. Reported figures for Bell are estimated from the results presented in [2]. Figure 2a shows committee size k and backup neighborhood size ℓ in the semi-honest settings. In the same figure, we also report the neighborhood size for Bell. As n grows, both k and ℓ parameters for LISA remain almost constant; the neighborhood size for Bell grows but the slope is less steep for large number of users (10^5 and above). For $n = 10^6$ users, corruption rate $\gamma = 0.33$ and dropout rate $\delta = 0.33$, LISA requires only $k = 407$ committee members, each having $\ell = 451$ backup neighbors. In other words, more than 81% of the users are regular users and must be online only for 2 rounds. When $\gamma = 0.05$ and $\delta = 0.33$, this number increases to 99%. Using the same parameters, Bell requires each user to have a backup neighborhood of more than 550 and 100 users respectively. All users are required to be online for the whole protocol execution (5.5 rounds).

Figure 2b shows the sizes of relevant sets in the malicious settings. In this scenario, the backup neighborhood size ℓ increases at first, until reaching a constant value, and is always smaller than the size of a neighborhood in Bell. For $n = 10^6$ and $\gamma = \delta = 0.2$, LISA uses 111 committee members, each with 526 backup neighbors. Only 0.01% of the users are committee members, and less than 5.4% are backup neighbors.

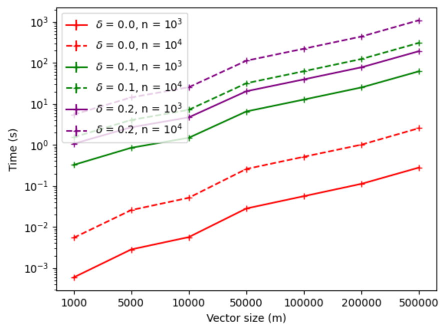
In our protocol, the security and correctness requirements for the committee are the same in both the honest-but-curious and the malicious model (i.e., same committee size). The security requirements for the backup neighbors are relaxed in the honest-but-curious protocol, allowing fewer backup neighbors per committee member than in the malicious case.



(a) Regular user



(b) Committee member



(c) Server

Fig. 3: Average runtime per participant, as the input size increases. We consider as participant's role: regular users, for different k (Fig. 3a); committee member, for different n (Fig 3b); and server (Fig 3c), for different n and δ. Solid and dashed lines represent 1k and 10k users respectively.

Benchmarks. To assess the performance of LISA we implemented and tested a prototype written in Golang 1.20. All experiments were run on a machine with Intel(R) Core(TM) i5-9600K CPU(3.70GHz) and 62 GB of RAM, running Ubuntu 18.04 OS. Our results are summarized in Figure 3.

Figure 3a shows the time for a user to compute a blinded input (Round 2) as the size of the input vector increases, for different committee sizes. For example, the overhead for a vector of size $m = 100K$ ranges between 170ms ($k = 50$) and 340ms ($k = 100$). Note that user overhead is only dependent on m and k. The user overhead in Bonawitz *et al.* [1] is dependent on m and n. For example, Bonawitz *et al.* report $694ms$ to compute the noisy input for a vector of $100K$ elements with $n = 500$, and $1357ms$ for $n = 1000$.

Figure 3b shows the time for a committee member to compute the aggregated randomness in Round 3. This time depends only on the number of elements m in the input vector and the number of users n. For example, the overhead for an input size $m = 100K$ ranges between 1s ($n = 1K$) and 42s ($n = 10K$).

The server runtime for the entire aggregation process is shown in Figure 3c. It depends on the dropout rate δ, as in Round 6 the server needs to reconstruct the missing partial binding for the offline committee members. The server's computation cost increases as the number of users grows. For example, for an input with $m = 100K$ entries and $n = 10K$ users, the server requires approximately 80s and 210s for 10% and 20% dropouts, respectively. For the same input size ($m = 100K$) but with only $n = 500$ users and a dropout rate of 10%, the server overhead in [1] is approximately 62 seconds. We note that [1] does not consider larger values of n.

Integration with FL applications. To demonstrate the usefulness of our proposal for realistic federated-learning applications, we integrated LISA into a concrete FL training algorithm and evaluated its effectiveness. We compare the performance of LISA with the default (non-private) aggregation. Recall that standard aggregation generates the aggregated model-update by averaging the local updates provided by users, while secure aggregation enhances it by providing cryptographic protection to the user updates (i.e., LISA adds a pseudorandom mask to each input component).

Dataset. We use the Bank Marketing Data Set [33], a dataset containing real data related to direct marketing campaigns based on phone calls to users. The dataset contains 41,188 instances of bank user data, including 20 features such as age, job, marital status, education, and previous marketing campaigns, and the contact outcome ("success" or "failure") as the label. We normalized the dataset to ensure better performance and accuracy of the model. The dataset is unbalanced, as only 4,640 out of the 41,188 are successful. We divided the total dataset into two, 80% training data and 20% testing data, with 32,950 records and 8,238 records respectively.

Setup. We implemented a federated learning process using Golang 1.20, based on an open-source machine learning library [34]. Each training round consists of a local training phase run by each user, with a fixed learning rate of 0.01 and a pre-specified number of local training epochs set to 10, and an aggregation phase to combine the local models. Users train their local model using a multi-layer perceptron with three layers: an input layer with 62 neurons, a hidden layer with 40 neurons, and an output layer with 2 neurons [35]. We used the normal sigmoid function as the activation function. To emulate a realistic FL deployment, we horizontally split the training set and conducted multiple rounds of training and aggregation until convergence, i.e., until the model stabilizes. We ran various executions of the aforementioned FL process, varying the number of users $n \in \{100, 500, 1000\}$. We repeated each experiment 5 times.

With this set of experiments, we aim to demonstrate the feasibility of embedding LISA into FL deployments, hence we set the dropout and corruption rates to $\delta = \gamma = 0$ for the sake of simplicity. To evaluate the effectiveness of our proposal, we compare the cases where the aggregation phase uses (i) standard aggregation (i.e., without privacy) vs. (ii) LISA, and we measure in each case: minimum number of rounds needed to converge, overall training time, and prediction accuracy of the resulting model.

Results. The results of our experiments are summarized in

	n	Rounds	Time(s)	Accuracy
Baseline	100	7	72.11	98.45%
LISA		7	74.66	98.23%
Baseline	500	8	92.44	98.23%
LISA		9	133.76	97.13%
Baseline	1000	7	59.79	97.12%
LISA		11	158.05	97.34%

TABLE III: Performance of standard aggregation vs. LISA in a FL application. Rounds is the minimum number of training rounds until convergence; Time is the overall training time.

Table III. Our results show that using our protocol slightly increases the minimum number of training rounds required for the model to converge when the number of users exceed 100 (precisely, from 8 to 9 rounds for $n = 500$, and from 7 to 11 rounds for $n = 1000$), compared to the default aggregation protocol. This behavior can be attributed to precision loss introduced by LISA. Besides, the total training time is higher for LISA compared to the baseline (requiring in addition $\approx 2s$, $41s$, and $98s$ respectively for $n = 100$, 500, and 1000 respectively), due to the extra blinding and unblinding steps in the aggregation phase. Nevertheless, LISA has a negligible impact on the accuracy and F1 score of the model, demonstrating that it can improve the privacy without sacrificing accuracy.

Simulated Results. In case of federated learning applications that use SA, the application runs across several rounds and invokes the aggregation protocol at every round. If all users have the same per-round overhead, it is easy to compute the overhead across several rounds. In LISA, however, a user overhead depends on the role that user takes in a round (e.g., regular user, committee member, backup neighborhood). Thus, it is fair to consider user overhead across multiple rounds. To this end, we wrote a python program that simulates several rounds of LISA and collects statistics for each of the users.

n	Committee	Backup
10^3	8.9 ± 3	2919.2 ± 54
10^4	1.0 ± 1	493.0 ± 22
10^5	0.1 ± 0.3	52.1 ± 7
10^6	0.01 ± 0.1	5.3 ± 2

TABLE IV: Average number of times a user is selected as a committee member/backup neighbor across 100 rounds, for $n \in [10^3 - 10^6]$.

In this set of experiments, we set $n \in [10^3 - 10^6]$, $\gamma = \delta = 0.2$, and run the secure aggregation protocol for 100 consecutive rounds. Table IV shows the measured average number of times a user was selected as a committee member or as a backup neighbor during the 100 executions of the protocol. For example, with $n = 10^5$, the maximum number of times one user was selected as a committee member was 3, and 90.4% of users where never selected as part of the committee. The average number of times a user was selected as a backup neighbor was 52 ± 7. The maximum number of times a user was selected as a backup neighbor was 88 (over 100 rounds). Note that a user can be selected as a backup neighbor by multiple committee members in a given round.

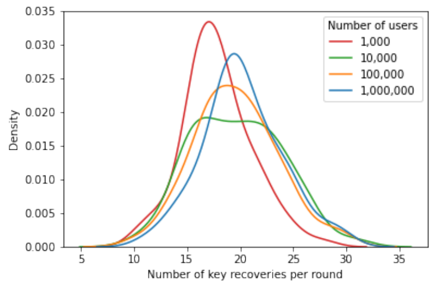


Fig. 4: Number of key recoveries per round across 100 rounds. $\gamma = 0.2$, $\delta = 0.2$

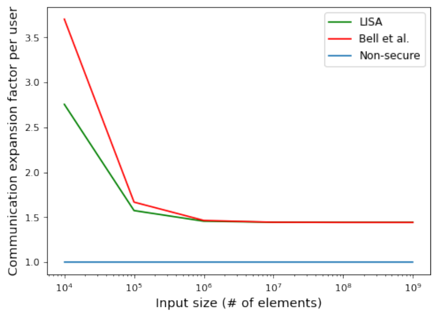


Fig. 5: Average communication expansion factor per user per round with $n = 100,000$ in the semi-honest case. $\gamma = 0.05$, and $\delta = 0.33$

Dropout of committee members has a direct impact on the overhead of LISA. This is because when committee member j drops out, the server must run a key-recovery protocol with the backup neighbors of j. Figure 4 shows the empirical CDF of the number of key-recoveries necessary across 100 iterations of the secure aggregation protocol. The average number of key-recoveries for $n = 10^5$ was 19 ± 4, with a maximum of 31 during one iteration. As shown in Figure 4, increasing the number of users has little impact on the average number of key recoveries. This is because committee size stays almost constant, even if we increase the number of users.

Table V shows statistics on the messages sent and received by each user across 100 iterations, for both LISA and Bell, in the semi-honest version of both protocols. Here we set $n = 10^4$, $m = 1$, $\gamma = 0.05$ and $\delta = 0.33$. The average number of messages sent and received in LISA was $10,085 \pm 5,780$. On average, a user in Bell sends and receives $25,351 \pm 1,778$ messages.

In Figure 5 we compare the average communication expansion factor per user during one round of secure aggregation with the one estimated from Bell *et al.* [2]. The non-secure protocol, i.e., the one where users send their input in plaintext, is used as baseline. As Figure 5 shows, the bandwidth required by LISA is lower than the one needed in [2], until $m = 10^6$. For larger input sizes, the expansion factor is similar for both protocols since the number of bytes sent and received per user is very high due to the large input vector. This was also observed in Bonawitz *et al.* [1], where the communication expansion factor is small for large inputs.

	Bell	**LISA**
Average	$25,351$	$10,085$
Standard deviation	$1,778$	$5,780$
Minimum	$18,891$	$3,607$
Maximum	$31,482$	$39,838$
Per-round average	254	101

TABLE V: Messages sent/received per user across 100 rounds; $n = 10^4$.

VII. Conclusions

In this paper we have presented LISA, a LIghtweight Secure Aggregation protocol that leverages a public source of randomness. LISA conceals individual user inputs with noise, and leverages public randomness to select a (small) subset of users designated to remove the noise from the sum of individual inputs and reveal the aggregation to the server. Our protocol completes in 6 rounds but most of the users are required to be online only during the first two rounds. Further, their communication overhead is asymptotically equal to the communication overhead of a non-private protocol where users send their inputs in the clear to the server. We have evaluated LISA both theoretically and by means of software prototype. Results show that LISA outperforms state-of-the-art secure aggregation protocols in terms of per-user communication complexity.

References

[1] K. A. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *CCS*, 2017, pp. 1175–1191.

[2] J. H. Bell, K. A. Bonawitz, A. Gascón, T. Lepoint, and M. Raykova, "Secure single-server aggregation with (poly)logarithmic overhead," in *CCS*, 2020, pp. 1253–1269.

[3] J. Clark and U. Hengartner, "On the use of financial data as a random beacon," in *USENIX Electronic Voting Technology Workshop*, 2010.

[4] J. Bonneau, J. Clark, and S. Goldfeder, "On bitcoin as a public randomness source," *IACR Cryptol. ePrint Arch.*, p. 1015, 2015.

[5] E. Syta, P. Jovanovic, E. Kokoris-Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer, and B. Ford, "Scalable bias-resistant distributed randomness," in *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE Computer Society, 2017, pp. 444–460.

[6] I. Cascudo and B. David, "SCRAPE: scalable randomness attested by public entities," in *Applied Cryptography and Network Security - 15th International Conference, ACNS 2017, Kanazawa, Japan, July 10-12, 2017, Proceedings*, ser. Lecture Notes in Computer Science, vol. 10355. Springer, 2017, pp. 537–556.

[7] S. Das, V. Krishnan, I. M. Isaac, and L. Ren, "Spurt: Scalable distributed randomness beacon with transparent setup," in *IEEE Symposium on Security and Privacy (SP)*, 2022, pp. 2502–2517.

[8] A. Kwon, D. Lu, and S. Devadas, "XRD: scalable messaging system with cryptographic privacy," in *17th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2020, Santa Clara, CA, USA, February 25-27, 2020*. USENIX Association, 2020, pp. 759–776.

[9] A. Kwon, H. Corrigan-Gibbs, S. Devadas, and B. Ford, "Atom: Horizontally scaling strong anonymity," in *ACM Symposium on Operating Systems Principles (SOSP)*, 2017, pp. 406–422.

[10] N. Tyagi, Y. Gilad, D. Leung, M. Zaharia, and N. Zeldovich, "Stadium: A distributed metadata-private messaging system," in *ACM Symposium on Operating Systems Principles (SOSP)*, 2017, pp. 423–440.

[11] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*. ACM, 2017, pp. 51–68.

[12] B. Adida, "Helios: Web-based open-audit voting," in *USENIX Security Symposium*.

[13] T. Stevens, C. Skalka, C. Vincent, J. Ring, S. Clark, and J. P. Near, "Efficient differentially private secure aggregation for federated learning via hardness of learning with errors," in *USENIX Security Symposium*, 2022, pp. 1379–1395.

[14] Y. Guo, A. Polychroniadou, E. Shi, D. Byrd, and T. Balch, "Microfedml: Privacy preserving federated learning for small weights," *IACR Cryptol. ePrint Arch.*, p. 714, 2022.

[15] Y. Ma, J. Woods, S. Angel, A. Polychroniadou, and T. Rabin, "Flamingo: Multi-round single-server secure aggregation with applications to private federated learning," in *SP*. IEEE, 2023, pp. 477–496.

[16] R. Behnia, M. Ebrahimi, A. Riasi, B. Padmanabhan, and T. Hoang, "Efficient secure aggregation for privacy-preserving federated machine learning," *CoRR*, vol. abs/2304.03841, 2023.

[17] B. Choi, J. Sohn, D. Han, and J. Moon, "Communication-computation efficient secure aggregation for federated learning," *CoRR*, vol. abs/2012.05433, 2020.

[18] J. So, B. Güler, and A. S. Avestimehr, "Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning," *IEEE J. Sel. Areas Inf. Theory*, vol. 2, no. 1, pp. 479–489, 2021.

[19] T. Jahani-Nezhad, M. A. Maddah-Ali, S. Li, and G. Caire, "Swiftagg+: Achieving asymptotically optimal communication load in secure aggregation for federated learning," *CoRR*, vol. abs/2203.13060, 2022.

[20] S. Kadhe, N. Rajaraman, O. O. Koyluoglu, and K. Ramchandran, "Fastsecagg: Scalable secure aggregation for privacy-preserving federated learning," *CoRR*, vol. abs/2009.11248, 2020.

[21] I. Ergun, H. U. Sami, and B. Guler, "Sparsified secure aggregation for privacy-preserving federated learning," *CoRR*, vol. abs/2112.12872, 2021.

[22] A. R. Elkordy and A. S. Avestimehr, "Heterosag: Secure aggregation with heterogeneous quantization in federated learning," *IEEE Trans. Commun.*, vol. 70, no. 4, pp. 2372–2386, 2022.

[23] S. Truex, N. Baracaldo, A. Anwar, T. Steinke, H. Ludwig, R. Zhang, and Y. Zhou, "A hybrid approach to privacy-preserving federated learning," in *AISec@CCS*. ACM, 2019, pp. 1–11.

[24] J. Ma, S. Naas, S. Sigg, and X. Lyu, "Privacy-preserving federated learning based on multi-key homomorphic encryption," *Int. J. Intell. Syst.*, vol. 37, no. 9, pp. 5880–5901, 2022.

[25] H. Fereidooni, S. Marchal, M. Miettinen, A. Mirhoseini, H. Möllering, T. D. Nguyen, P. Rieger, A. Sadeghi, T. Schneider, H. Yalame, and S. Zeitouni, "Safelearn: Secure aggregation for private federated learning," in *SP (Workshops)*. IEEE, 2021, pp. 56–62.

[26] F. Karakoç, M. Önen, and Z. Bilgin, "Secure aggregation against malicious users," in *SACMAT*. ACM, 2021, pp. 115–124.

[27] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[28] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Trans. Inf. Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[29] M. Dworkin, "Recommendation for block cipher modes of operation: Galois/counter mode (GCM) and GMAC," NIST Special Publication 800-38D, November 2007.

[30] D. Johnson, A. Menezes, and S. A. Vanstone, "The elliptic curve digital signature algorithm (ECDSA)," *Int. J. Inf. Sec.*, vol. 1, no. 1, pp. 36–63, 2001.

[31] M. Skala, "Hypergeometric tail inequalities: ending the insanity," 2013.

[32] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM Rev.*, vol. 59, no. 1, pp. 65–98, 2017.

[33] S. Moro, P. Cortez, and P. Rita, "A data-driven approach to predict the success of bank telemarketing," *Decis. Support Syst.*, vol. 62, pp. 22–31, 2014.

[34] Anastasios, "Banking-ml," https://github.com/anaktas/bp7, 2021.

[35] A. Nouri, "Banking-ml," https://github.com/dynamic11/Banking-ML, 2021.

Appendix A
Correctness of LISA

In this section we show formally that LISA allows the Server to obtain the aggregated sum of the user's inputs despite a limited number of users dropping out during the execution.

Recall that in Round 2, after the committee members and their backup neighbors have been selected, each user i derives two shared secrets with each committee member j: a PRG seed $k^*_{i,j} \leftarrow \mathsf{KA}(\text{"PRG"}; sk_i, PK_j)$ to derive the input mask, and, if $i \in \mathcal{L}_j$, a symmetric key $k^e_{i,j} \leftarrow \mathsf{KA}(\text{"ENC"}; sk_j, pk_i)$ to retrieve encrypted shares of j's secret key. User i then blinds her input by computing $c_i \leftarrow x_i + \sum_{j\in\mathcal{K}} F(k^*_{i,j})$. Similarly to regular users, a committee member j masks her input with k blindings, where $k^*_{j,j} \leftarrow \mathsf{KA}(\text{"PRG"}; SK_j, pk_j)$ is a secret "shared with herself".

In Round 3, committee member j computes the partial blinding as $\partial_j \leftarrow \sum_{i\in\mathcal{U}'_1} F(k^*_{j,i})$, where $k^*_{j,i} = \mathsf{KA}(\text{"PRG"}; SK_j, pk_i) = k^*_{i,j}$.

After receiving the blinded inputs from users and the partial blindings from committee members, the server proceeds as follows. If $\mathcal{K}_{drop} = \emptyset$, it computes the aggregated input as:

$$y = \sum_{i\in\mathcal{U}'_1} c_i - \sum_{j\in\mathcal{K}} \partial_j \tag{56}$$

$$= \sum_{i\in\mathcal{U}'_1}\left(x_i + \sum_{j\in\mathcal{K}} F(k^*_{i,j})\right) - \sum_{j\in\mathcal{K}}\sum_{i\in\mathcal{U}'_1} F(k^*_{i,j}) \tag{57}$$

$$= \sum_{i\in\mathcal{U}'_1} x_i \tag{58}$$

In case $\mathcal{K}_{drop} \neq \emptyset$, the server reconstructs the secret keys of dropped committee members using the shares released by the backup neighbors (Rounds 4 and 5), and for each $j \in \mathcal{K}_{drop}$ it computes the partial blinding ∂_j as the committee member would do. Let $\mathcal{K}_{alive} = \mathcal{K} \setminus \mathcal{K}_{drop}$. The server computes the aggregate as:

$$y = \sum_{i\in\mathcal{U}'_1} c_i - \sum_{j\in\mathcal{K}_{alive}} \partial_j - \sum_{j\in\mathcal{K}_{drop}} \partial_j \tag{59}$$

$$= \sum_{i\in\mathcal{U}'_1} c_i - \sum_{j\in\mathcal{K}} \partial_j \tag{60}$$

$$= \sum_{i\in\mathcal{U}'_1} x_i. \tag{61}$$

Appendix B
Security of LiSA

To prove security of LiSA, we follow the standard simulation-based paradigm and show that the view of any attacker against the protocol can be simulated using only the input of the corrupted parties and the protocol output. Intuitively, this means that corrupted parties learn nothing more than their own inputs and the intended protocol leakage (i.e., the aggregated input in our case).

We show security for the malicious version of LiSA (security in the honest-but-curious setting follows directly). Recall that in the malicious setting, the adversary can corrupt the server and a subset of users $\mathcal{C} \subset \mathcal{U}$ of size $|\mathcal{C}| = \gamma n$. Simulating the view of such an adversary requires accounting for the intended leakage of the protocol. In particular, LiSA allow the server to learn the aggregated input of any subset $U \subset \mathcal{U}'_1$ as long as $|U| > \alpha n$. Thus, we give the simulator one-time access to an oracle providing the sum of the inputs by honest users in any subset U of size at least αn. We note that the first functionality also appears in [1], [2].

For a subset $U \subseteq \mathcal{U}$, let x_i denote the input of user $i \in U$. Below we use the shorthand $x_{\mathcal{U}\setminus\mathcal{C}}$ for the set of inputs $\{x_i\}_{i\in\mathcal{U}\setminus\mathcal{C}}$. The α–sum functionality $\mathcal{F}^{SUM}$ over $x_{\mathcal{U}\setminus\mathcal{C}}$ is parameterized by α, takes as input a subset $U \subseteq \mathcal{U}\setminus\mathcal{C}$, and outputs the aggregation of inputs in U, if U is sufficiently large, else it returns $\perp$:

$$\mathcal{F}^{SUM}_{x_{\mathcal{U}\setminus\mathcal{C}},\alpha}(U) = \begin{cases} \sum_{i\in U} x_i & \text{if } |U| \geq \alpha|\mathcal{U}|, \\ \perp & \text{otherwise.} \end{cases} \tag{62}$$

We will provide a security proof in the random oracle model. In particular we model PRG and key derivation functions (F and KA) with a random oracle $\mathcal{H}\colon \{0,1\}^* \rightarrow \{0,1\}^*$ that the simulator will build on the fly.

Given n, k, t, l and a set of corrupted parties $\mathcal{C}$, we denote by A the (poly-time) algorithm that defines the strategy of the adversary. This includes logic to decide which messages are sent to and received by corrupted parties and if messages sent by honest parties are ever delivered. Thus, A can also decide to make an honest user abort (e.g, by sending her a malformed message) or make an honest user look as if she dropped out (e.g., by not delivering her messages).

Let $\mathrm{REAL}_{n,t,k,l,\gamma,\delta,\mathcal{C}}(A)$ be the random variable defining the combined view of all corrupted parties during an execution of LiSA, distributed over the random choices of the honest parties, randomness of the adversary, randomness used to setup cryptographic primitives, and the random oracle. We will now present a simulator SIM that can emulate the real view of corrupted parties by running A and by simulating honest parties on dummy inputs.

We prove the following result.

Theorem 2 (Privacy against malicious adversaries)**.** *For every PPT adversary A, all $n, t, k, l \in \mathbb{N}$ with $k < n$ and $t < l$, every set $\mathcal{C} \subset \mathcal{U}$, all corruption and dropout rates $\gamma, \delta \in (0,1)$, and all set of inputs $x_{\mathcal{U}\setminus\mathcal{C}}$, if the events S_1 and S_2 hold (cf. Section V), then there exists a simulator* SIM *with access to $\mathcal{F}^{SUM}$ such that the output of the simulator is computationally indistinguishable from $\mathrm{REAL}_{n,t,k,l,\gamma,\delta,\mathcal{C}}(A)$. In other words:*

$$\mathrm{REAL}_{n,t,k,l,\gamma,\delta,\mathcal{C}}(A, x_{\mathcal{U}\setminus\mathcal{C}}) \equiv \mathrm{SIM}^{\mathcal{F}^{SUM}}_{n,t,k,l,\gamma,\delta,\mathcal{C}}(A)$$

Proof. We show a sequence of hybrids, starting from the real protocol execution and terminating with a simulated execution that does not use any of the honest parties'inputs.

Hyb$_0$ The real execution of the protocol.

Hyb$_1$ A simulator who knows the inputs $\{x_i\}_{i\in\mathcal{U}\setminus\mathcal{C}}$ of honest users emulates the real execution by running the protocol with the adversary. This includes simulating the PKI, the randomness beacon, and the random oracle $\mathcal{H}$ on the fly.

Hyb$_2$ For every pair of honest users i, j with $i \in \mathcal{U}$ and $j \in \mathcal{K}$, the simulator replaces the shared symmetric key $k^e_{j,i} \leftarrow \mathsf{KA}(\text{"ENC"}; sk_j, pk_i)$ derived by j in Round 2(3) with a uniformly chosen value $r_{j,i}$ and, to ensure consistency, it also replaces the corresponding key $k^e_{i,j}$ derived by user i in Round 5(5b) with the same value $r_{j,i}$. Indistinguishability from the previous hybrid follows from the properties of KA.

Hyb$_3$ The simulator aborts if, for every pair of honest users i, j with $i \in \mathcal{U}$ and $j \in \mathcal{K}$, the adversary makes i deliver a different ciphertexts than the encryption $E_{j,i}$ generated in Round 2(5), and decryption in Round 5(5c) does not fail. Indistinguishability from the previous hybrid follows from the INT-CTXT security of the AE scheme.

Hyb$_4$ For every pair of honest users i, j with $i \in \mathcal{U}$ and $j \in \mathcal{K}$, the simulator replaces all values $E_{j,i}$ in Round 2(5) with encryptions of zero. Notice that the simulator still returns the real shares $S_{j,i}$ to the Server in Round 5(5d). Indistinguishability from the previous hybrid follows from the IND-CPA security of the AE scheme.

Hyb$_5$ The simulator aborts its execution if the adversary forges a valid signature σ_i', on behalf of any honest user i, on a set $\mathcal{K}_{drop}'$ different from the set $\mathcal{K}_{drop}$ signed by i, and yet the signature verification in Round 5(3) succeeds. Indistinguishability from the previous hybrid follows from the unforgeability of the digital signature scheme. From this hybrid on, we are sure that all honest backup neighbors agree on the (now well-defined) set $\mathcal{K}_{drop}$ of committee members the server declares to have dropped out.

Hyb$_6$ The simulator aborts if, for every pair of honest users i, j with $i \in \mathcal{U}$ and $j \in \mathcal{K}$, the adversary queries the random oracle $\mathcal{H}$ on input $k_{j,i}^*$ before Round 6. Indistinguishability from the previous hybrid follows from the properties of the secret sharing scheme and event S_2 (cf. Section V), preventing the adversary from recovering $\{S_{j,i}\}_{j \in \mathcal{K} \setminus \mathcal{C}}$.

Hyb$_7$ For every pair of honest users i, j with $i \in \mathcal{U}$ and $j \in \mathcal{K}$, the simulator replaces the shared PRG seeds $k_{i,j}^*$ and $k_{j,i}^*$, computed by i as $k_{i,j}^* \leftarrow \mathsf{KA}(\text{"PRG"}; sk_i, PK_j)$ in Round 2(2), and by j as $k_{j,i}^* \leftarrow \mathsf{KA}(\text{"PRG"}; SK_j, pk_i)$ in Round 3(4), with a value $r_{i,j}$ chosen uniformly at random, and programs the random oracle so that $F(k_{i,j}^*) = F(r_{i,j})$ for every $j \in \mathcal{K}_{drop}$; this ensures that the adversary obtains consistent values when computing $F(k_{i,j}^*)$ in Round 6(1c). Indistinguishability from the previous hybrid follows from the key indistinguishability of KA.

Hyb$_8$ For all honest users $i \in \mathcal{U}_1'$, the simulator replaces the blinded value $c_i \leftarrow x_i + \sum_{j \in \mathcal{K}} F(k_{ij}^*)$ with a randomly chosen value, and programs the random oracle (by changing the values of the masks $\{F(k_{ij}^*)\}_{j \in \mathcal{K}_{alive} \setminus \mathcal{C}}$ each user i shares with honest, alive committee members) to ensure consistency of the final output, so that:

$$\sum_{j \in \mathcal{K}_{alive} \setminus \mathcal{C}} F(k_{ij}^*) = c_i - x_i - \sum_{\substack{j \in \mathcal{K}_{drop} \cup \\ (\mathcal{K}_{alive} \cap \mathcal{C})}} F(k_{ij}^*). \quad (63)$$

Notice that this step is possible as long as the set $\mathcal{K}_{alive} \setminus \mathcal{C}$ of honest and alive committee members is non-empty. This condition is guaranteed by event S_1 (cf. Section V). Concretely, we let the simulator choose a user $\hat{j} \in \mathcal{K}_{alive} \setminus \mathcal{C}$ and program the random oracle F so that:

$$F(k_{i\hat{j}}^*) = c_i - x_i - \sum_{\mathcal{K} \setminus \{\hat{j}\}} F(k_{ij}^*). \quad (64)$$

Indistinguishability from the previous hybrid follows from the properties of the random oracle.

Hyb$_9$ For every pair of honest users i, j with $i \in \mathcal{U}$ and $j \in \mathcal{K}$, the simulator programs the random oracle so that every honest users $i \in \mathcal{U}_1' \setminus \mathcal{C}$ computes the mask by replacing the real input x_i with a randomly chosen value, i.e., instead of

$$F(k_{ij}^*) = c_i - x_i - \sum_{\mathcal{K}} F(k_{ij}^*), \quad (65)$$

it sets

$$F(k_{ij}^*) = c_i - w_i - \sum_{\mathcal{K}} F(k_{ij}^*), \quad (66)$$

where all w_i's are chosen at random subject to $\sum_{i \in \mathcal{U}_1' \setminus \mathcal{C}} w_i = \sum_{i \in \mathcal{U}_1' \setminus \mathcal{C}} x_i$. Indistinguishability from the previous hybrid follows from the properties of the random oracle.

Hyb$_{10}$ Notice that in the last hybrid, the simulator no longer needs the individual inputs x_i of honest users in $\mathcal{U}_1'$, it only needs their sum. We can therefore let SIM emulate the protocol by querying the summation functionality $\mathcal{F}^{SUM}$, on input the set $U = \mathcal{U}_1' \setminus \mathcal{C}$, and then choose the values w_i so that they sum up to the output $\mathcal{F}^{SUM}(U)$ returned by the functionality. This last hybrid is perfectly indistinguishable from the previous hybrid. As the simulator does not use the inputs of the honest parties, this concludes the proof.

□