DEVA: Decentralized, Verifiable Secure Aggregation for Privacy-Preserving Learning

Georgia Tsaloli[1], Bei Liang[2], Carlo Brunetta[1], Gustavo Banegas[3], and Aikaterini Mitrokotsa[1,4]

[1] Chalmers University of Technology, Gothenburg, Sweden
{tsaloli,brunetta}@chalmers.se
[2] Beijing Institute of Mathematical Sciences and Applications, Beijing, China
lbei@bimsa.cn
[3] Inria and Laboratoire d'Informatique de l'Ecole polytechnique, Institut Polytechnique de Paris, Palaiseau, France
gustavo@cryptme.in
[4] University of St. Gallen, School of Computer Science, St. Gallen, Switzerland
katerina.mitrokotsa@unisg.ch

Abstract Aggregating data from multiple sources is often required in multiple applications. In this paper, we introduce DEVA, a protocol that allows a distributed set of servers to perform secure and verifiable aggregation of multiple users' secret data, while no communication between the users occurs. DEVA computes the sum of the users' input and provides public verifiability, *i.e.,* anyone can be convinced about the correctness of the aggregated sum computed from a threshold amount of servers. A direct application of the DEVA protocol is its employment in the *machine learning* setting, where the aggregation of multiple users' parameters (used in the learning model), can be orchestrated by multiple servers, contrary to centralized solutions that rely on a single server. We prove the security and verifiability of the proposed protocol and evaluate its performance for the execution time and bandwidth, the verification execution, the communication cost, and the total bandwidth usage of the protocol. We compare our findings to the prior work, concluding that DEVA requires less communication cost for a big amount of users.

Keywords: secure aggregation, privacy, verifiability, decentralization

1 Introduction

Mobile phones, wearables, and other Internet-of-Things (IoT) devices are all connected to distributed network systems. These devices generate a significant amount of data, that often need to remain private. These data in many cases need to be aggregated to compute statistics, or even employed for user modeling and personalization via federated learning algorithms. Such an application scenario gives rise to the *secure data aggregation* problem, the goal of which is to compute

sums of local updated parameters from individual users' devices in a privacy-preserving manner, *i.e.*, any individual user's update is not revealed in the clear.

In the federated learning setting, each *user* maintains her private data on her mobile device, and shares local updated parameters (*e.g.*, gradients) to the server. The central *server* updates the training model using the aggregated updates and performs the appropriate testing of the model. An advantage of federated training is that it diminishes the risk of compromising the user's privacy, since it allows users (mobile devices or organizations) to collaboratively train learning models under the orchestration of a central server, while the data remain located on the sources (*i.e.,* mobile devices or data centers of organizations).

The secure aggregation problem has received significant attention in the literature. Bonawitz *et al.* [13] proposed a practical and secure aggregation protocol for federated learning, which enables a central server to compute the sum of multiple users' parameters and guarantees robustness in a dynamic environment where users may drop out. Even though Bonawitz *et al.* [13] addressed the problem of maintaining user's privacy (i.e., local gradients) in the learning process, Xu *et al.* [19] considered another fundamental issue of data integrity in federated learning, *i.e.*, how to assure the correctness of the aggregated results returned from the central server, since a malicious server might modify the aggregation process [11], bias the final result and cause inferences according to its preferences [6,11,16,20]. To this end, Xu *et al.* provided a privacy-preserving and verifiable aggregation protocol, VerifyNet [19]. The latter enables the users to verify the correctness of the computed sum, while guaranteeing the users' privacy in the training process. In our work, we focus on the verifiability as considered in [19], *i.e.,* guaranteeing the correctness of the aggregated result. Bonawitz *et al.*'s [13] and Xu *et al.*'s [19] solutions adopt a centralized architecture since a *single* central server is responsible for the aggregation of the users' parameters and orchestrates the federated learning process. Even though a central server is an important component of the federated learning process, a single server might attempt to bias the model and cause inferences. For instance, the server may tamper with the learning model so that it always misclassifies a certain pattern in an image recognition system, or allows access to unauthorized users in a biometric authentication system [5]. Decentralized systems have raised considerable interest, since they distribute the storage and the computation among *multiple* servers, thus allowing different organizations to collaboratively perform computations and diminish the security threats incurred by centralized systems.

In this paper, we propose DEVA, a decentralized, verifiable and privacy-preserving aggregation protocol, which enables multiple servers to *jointly* compute the sum of the parameters of multiple users, and further to train and evaluate a global learning model. We stress that although VerifyNet [19] achieves data integrity in the process of training neural networks, it employs a *single central* server for both the aggregation and for returning the verification results. In contrary, our DEVA protocol performs federated learning collaboratively by employing multiple servers for the aggregation process. A single server (hosted by a single organization) might not be trusted by different organizations with similar

objectives (*e.g.,* hospitals, banks) that want to **collaboratively** train learning models [7] and thus, multiple cloud servers can resolve this issue. The involvement of multiple servers is challenging, since we need to find a way to obtain the aggregated result from partial outputs, but also need to ensure the correctness of the computed result. In this work, we make the following **contributions**: *(i)* We propose DEVA, a protocol for securely computing the sum (aggregation) of n inputs from multiple users, by employing multiple servers. Our DEVA has a constant number of rounds, low communication cost for each server, and tolerates up to $n-(t_{key}+1)m$ users dropping out during the protocol execution, for t_{key} being a threshold value. Contrary to the setting of only one central server that requires limited trust, in DEVA no server has to be individually trusted and a fraction of the servers can collude. DEVA also handles possible servers' failure as it requires $t+1$ servers to compute the sum. *(ii)* DEVA guarantees the individual user's privacy, *i.e.*, the servers learn only the aggregated result of all users' inputs without knowing any user's input itself. *(iii)* DEVA ensures the correctness of the computed sum by requiring the employed servers to provide a proof about the correctness of their aggregated results. We prove that it is infeasible for any adversary to deceive the users by altering the aggregated results with a valid proof. *(iv)* DEVA is practical and we present experimental results from our prototype implementation. DEVA provides less communication cost for each user participating in the protocol. DEVA also allows to maintain bandwidth cost since increased amount of users can be leveraged by having more servers.

Related Work. To solve the security, accuracy and privacy challenges in learning, some works have been proposed recently [12,13,15,16]. Phong *et al.* [12] proposed a secure deep learning system based on additively homomorphic encryption, Shokri *et al.* [16] proposed a privacy-preserving deep learning protocol focusing on the trade-off between private and accurate learning. Bonawitz *et al.* [13] proposed a secure aggregation protocol tailored for the federated learning process that attempts to achieve a good balance between security, privacy and efficiency, being robust to users dropping out. However, these solutions have multiple limitations: *(i)* they assume a **single** server which is not suitable when different organizations collaboratively train a model; and *(ii)* they provide **no verifiability** guarantees of the learning model. We stress that Bonawitz *et al.* [13] discuss how to address the input verifiability, *i.e.,* verifying that the inputs are in the correct range; however, they **do not** deal with the issue of output verifiability, *i.e.,* verifying that the aggregated result is correct. Some works [4,5,10,17,19] attempted to address the problem of *verifiability* (output correctness), but all of them require a central server and additionally, either they ignore users dropping out [4,5,10] and privacy leakages [5] or require special hardware [17] (thus, placing trust to the hardware manufacturer) or costly computations for verification (low efficiency) [19]. They consider a centralized system, while our goal is to avoid placing the trust to a single server and allow different organizations (hosted by different cloud servers) to collaboratively perform the learning process. Thus, we employ multiple servers and achieve decentralized aggregation.

2 Preliminaries

In this section, we show definitions and assumptions used throughout the paper.
Hash functions. We employ a collision-resistant homomorphic hash function [21] satisfying additive homomorphism [9], *i.e.,* $H : x \mapsto \mathsf{g}^x$ where g is a generator of the group $\mathbb{G}$ of prime order p.
Key Agreement. Let $\mathbb{G}$ be a cyclic group of order p prime with generator g, *e.g.,* groups based on elliptic curves [8]. Let us report the definition of the Diffie-Hellman key agreement [3] and the related assumptions.

Assumption 1 (Discrete Logarithm Problem) *Consider a cyclic group $\mathbb{G}$ of order p prime with generator g. Given $y \in \mathbb{G}$, the* ***discrete logarithm problem (dLog)*** *requires to find the value $x \in [0, p-1]$ such that $g^x = y$. We assume the advantage of solving the dLog problem to be negligible,* i.e., $\epsilon_{dLog} < \mathsf{negl}$.

Assumption 2 (Diffie-Hellman Assumptions) *Consider a cyclic group $\mathbb{G}$ of prime order p with generator g and $a, b \in [0, p-1]$. Given elements $(A, B) = (g^a, g^b)$, the* ***computation Diffie-Hellman problem (CDH)*** *requires to compute the element $g^{ab} \in \mathbb{G}$. The* ***distinguishing Diffie-Hellman problem (DDH)*** *requires to correctly distinguish between (g, A, B, g^{ab}) and (g, A, B, g^c) for some random $c \in [0, p-1]$. We assume the advantage of solving the CDH and the DDH problems to be negligible,* i.e., $\epsilon_{CDH} < \mathsf{negl}$ *and* $\epsilon_{DDH} < \mathsf{negl}$.

Definition 1 (Diffie-Hellman Key Exchange). *Consider a Diffie-Hellman key agreement scheme with algorithms ($\mathsf{Ksetup}, \mathsf{Kgen}, \mathsf{Kagree}$) to be defined as:*

- $\mathsf{Ksetup}(1^\lambda) \to \mathsf{pp}$*: the setup algorithm takes as input the security parameter and outputs the public parameters pp which contain a prime p, the description of a cyclic group $\mathbb{G}$ of order p and a generator g for the group $\mathbb{G}$.*
- $\mathsf{Kgen}(\mathsf{pp}, U_i) \to (\mathsf{sk}_i, \mathsf{pk}_i)$*: the user U_i samples a value $\mathsf{sk}_i \in [0, p-1]$ and computes $\mathsf{pk}_i = g^a$. The key generation algorithm outputs $(\mathsf{sk}_i, \mathsf{pk}_i) = \left(\mathsf{sk}_i, g^{\mathsf{sk}_i}\right)$.*
- $\mathsf{Kagree}(\mathsf{sk}_i, \mathsf{pk}_j) \to s_{ij}$*: the user U_i runs the key agreement algorithm with its own secret sk_i and U_j's public key $\mathsf{pk}_j = g^{\mathsf{sk}_j}$ to obtain the agreed secret key $s_{ij} = \mathsf{pk}_j^{\mathsf{sk}_i} = g^{\mathsf{sk}_j \cdot \mathsf{sk}_i}$ between the users U_i and U_j.*

The key agreement is said to be ***correct*** *if for any $\mathsf{pp} \leftarrow \mathsf{Ksetup}(1^\lambda)$, $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{Kgen}(\mathsf{pp}, U_i)$, and $(\mathsf{sk}_j, \mathsf{pk}_j) \leftarrow \mathsf{Kgen}(\mathsf{pp}, U_j)$, it holds that $s_{ij} = s_{ji}$. The key agreement scheme is said to be* ***secure*** *if for any $\mathsf{pp} \leftarrow \mathsf{Ksetup}(1^\lambda)$, $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{Kgen}(\mathsf{pp}, U_i)$, as well as $(\mathsf{sk}_j, \mathsf{pk}_j) \leftarrow \mathsf{Kgen}(\mathsf{pp}, U_j)$, it holds that any PPT adversary $\mathcal{A}$ has negligible probability to compute s_{ij} from $(\mathsf{pk}_i, \mathsf{pk}_j)$. The key agreement's security reduces to the CDH and dLog assumptions.*

Secret Sharing. We provide the definition of a (t, m)-threshold secret sharing scheme in order to achieve additive homomorphism in our protocols. Precisely:

Definition 2. *A (t, m)-threshold secret sharing scheme allows a user U_i to split a secret $x_i \in \mathbb{F}$, where $\mathbb{F}$ is the input domain, into m shares, such that any $t+1$ shares can be used to reconstruct x_i, while any set of at most t shares gives no information about x_i. Let $\mathcal{S}$ be the set such that $|\mathcal{S}| = m$ and $\mathcal{T} \subseteq \mathcal{S}$ with $|\mathcal{T}| > t$. Then we consider two algorithms ($\mathsf{SS.share}, \mathsf{SS.recon}$):*

– $\mathsf{SS.share}(t, x_i, j, \mathcal{S}) \rightarrow \{x_{ij}\}_{j\in\mathcal{S}}$*: for a given threshold* t*, a secret input* $x_i \in \mathbb{F}$*, an index* j *which corresponds to the receiver of the share and the set* $\mathcal{S}$*, the algorithm outputs a list of shares, namely,* $\{x_{i1}, \ldots, x_{im}\}$.
– $\mathsf{SS.recon}(t, \{x_{ij}\}_{j\in\mathcal{T}}, \mathcal{T}) \rightarrow x_i$*: given a threshold* t*,* $|\mathcal{T}| > t$ *amount of shares* x_{ij} *and the set* $\mathcal{T}$*, the algorithm gives* x_i.

Shamir's threshold secret sharing [14], as well as other secret sharing schemes [**?**,**?**,18] have an homomorphic property, as described by Benaloh [1]. More precisely, these schemes allow to combine multiple secrets by performing computations directly on shares. For linear functions, a (t, m) threshold scheme has the additive homomorphic property if the sum of the shares are shares of the sum [1]. Thus, with our notation, if we consider n secret inputs $x_1, \ldots, x_n$ and denote the sum of shares of each $j \in \mathcal{T}$ by y_j, then $\mathsf{SS.recon}(t, \{y_j\}_{j\in\mathcal{T}}, \mathcal{T}) \rightarrow y$, where $y = x_1 + \ldots + x_n$ (1).

In fact, Shamir's scheme is an additive homomorphic secret sharing scheme and, therefore, we use it in the implementation of our protocol.

Zero-Knowledge Proofs of Discrete Logarithm Knowledge. We will need a zero-knowledge proof of knowledge of a value $\alpha \in [0, p-1]$ such that $A = \mathsf{g}^\alpha$ and $B = \mathsf{h}^\alpha$ given the group generators g, h and the corresponding values A, B. We denote the protocol which generates this proof by $\mathsf{DLEQ}(\mathsf{g}, \mathsf{h}, A, B, \alpha)$. Chaum and Pedersen proposed a sigma protocol to perform this proof in [2]. Precisely, the zero knowledge protocol we use is specified as follows: $\mathsf{DLEQ}(\mathsf{g}, \mathsf{h}, A, B, \alpha)$:

– **Proof**.$\mathsf{DLEQ}(\mathsf{g}, \mathsf{h}, A, B, \alpha)$: (i) for the given g, h, compute $s_1 = \mathsf{g}^s, s_2 = \mathsf{h}^s$ where s is a field element, chosen uniformly at random; (ii) for a hash function Ha such that $\mathsf{Ha}(\cdot) \in \{0, 1\}$, compute $c = \mathsf{Ha}(\mathsf{g}, \mathsf{h}, A, B, s_1, s_2)$, (iii) compute $r = s + c \cdot \alpha$, and (iv) output the proof (s_1, s_2, r).
– **Verify**.$\mathsf{DLEQ}(\mathsf{g}, \mathsf{h}, A, B, (s_1, s_2, r))$: (i) for the aforementioned hash function, compute $c = \mathsf{Ha}(\mathsf{g}, \mathsf{h}, A, B, s_1, s_2)$, (ii) check if both $\mathsf{g}^r \stackrel{?}{=} s_1 \cdot A^c$ and $\mathsf{h}^r \stackrel{?}{=} s_2 \cdot B^c$ are satisfied, and (iii) if they are satisfied, accept the proof, otherwise abort.

3 Framework of a DECENTA Problem

In this chapter, we describe the DECENTA problem as well as the required properties that a solution to DECENTA must satisfy.

Problem Statement. Consider n users $\mathsf{U}_1, \ldots, \mathsf{U}_n$, each with a secret input x_i, and m servers $\mathsf{S}_1, \ldots, \mathsf{S}_m$. A DECENTA problem aims to **securely** compute the sum of the users' secret inputs, *i.e.,* $y = \sum_{i=1}^{n} x_i$, by aggregating more than a certain amount of partial results; which are computed by the servers. Moreover, the aggregated final result y can be publicly verified, *i.e.*, anyone is able to check if y is the correct sum of all users' inputs without revealing their input itself.

In the setting of a DECENTA problem, no communication is allowed between the users; thus rendering it suitable for application settings where an immense number of users are participating, *e.g.,* this is the case for the federated learning setting, where a very large number of users participate via their mobile devices and thus, cannot establish direct communications channels with other mobile devices (need to rely

on a server to play the intermediate communication role). Furthermore, DECENTA supports a dynamic setting, where the participating users (mobile devices) may drop out during the execution of the protocol and the correct aggregation of the values of the remaining users (devices) is still possible. The DECENTA problem captures both features of *decentralization*, since multiple servers are involved in the system instead of a single centralized server, thus, allowing a subset of the servers to be corrupted while still securely computing the sum value; and *verifiability* since it allows the participating users to verify the correctness of the computed result. A protocol solving the DECENTA problem involves the following phases:

Setup: generation of all key pairs that are used during the protocol execution.

Shares and Public Values Generation: each user U_i hides its secret data x_i by splitting it into different shares that are sent to the servers instead of the actual secret users' data. Additionally, each user computes and publishes some values that are used by a verifier to fulfill, later on, the verification process.

Aggregation: it consists of all the steps that are needed to output partial values by each server, which are appropriately used for the generation of the final result y, and the proof (that y is indeed the correct sum), denoted by σ.

Verification: ultimately, combining suitably the result y and the proof σ, this phase performed by a verifier gives out either 1, implying that y is the actual correct sum of all users' secret data x_i, or 0 implying that y is incorrect.

Threat Model and Design Goal. We adopt the threat model proposed by Xu *et al.*, which is used to define the security of VerifyNet [19], a recently proposed privacy-preserving and verifiable federated learning framework. In contrast to the single server (*i.e.* centralized) setting used in VerifyNet, we adjust the threat model to a decentralized multiple-server setting. Precisely, we consider that both the cloud servers and the users follow the protocol's execution as agreed, but they may also try to infer information about other users' data. Additionally to this, in our protocol, we employ multiple servers with the following abilities: *(i)* a threshold of the servers may collude to discover the users' private inputs, and *(ii)* they can modify their computed results and forge proofs in order to provide an incorrect sum to be accepted.

Properties. We require a solution to the DECENTA problem to be *correct*, *secure*, and *verifiable*. Below, we provide the corresponding definitions.

Definition 3 (Correctness). *For all n users $\mathsf{U}_1, \mathsf{U}_2, \ldots, \mathsf{U}_n$ with inputs $x_1, \ldots, x_n$, for all m servers $\mathsf{S}_1, \ldots, \mathsf{S}_m$, where all U_i and S_j honestly execute the protocol, and for all the partial values output by the servers S_j, the protocol is correct if it satisfies the following requirement:*

$$\Pr\left[\textbf{\textit{Verification}}(\textit{pub\_pars}, \sigma, y) = 1 \wedge y = \sum_{i=1}^{n} x_i\right] = 1.$$

where pub_pars denotes all public parameters necessary for the protocol (if any), y denotes the aggregated final result, which comes from the partial values output by the servers during the protocol, and σ denotes the corresponding proof of y.

Definition 4 (Verifiability). *For n users $\mathsf{U}_1, \ldots, \mathsf{U}_n$ with inputs $x_1, \ldots, x_n$, that honestly execute the protocol, and any set of corrupted servers $T = \{\mathsf{S}_{j_1}, \ldots, \mathsf{S}_{j_{|T|}}\}$*

with $|T| < m$ that are controlled by a PPT adversary $\mathcal{A}$, i.e., *$\forall j \in [j_1, j_{|T|}]$ such that $S_j \in T$, S_j gives $\{x_{1j}, \ldots, x_{nj}\}$ to $\mathcal{A}$ where x_{ij} is the share given to the server S_j from the user U_i. $\mathcal{A}$ outputs the malicious partial results on behalf of the corrupted servers $S_j \in T$, while the honest servers $S_j \notin T$ output correct partial results. Then, if $\mathcal{A}$ outputs an aggregated result y' together with the corresponding proof σ' such that $y' \neq \sum_{i=1}^{n} x_i$, we require that $\mathcal{A}$ can pass the verification phase with negligible probability. More precisely, for any PPT adversary $\mathcal{A}$, it holds:*

$$\Pr\left[\textbf{\textit{Verification}}(\textit{pub\_pars}, \sigma', y') = 1\right] \leq \varepsilon,$$

for some negligible ε; pub_pars are the public parameters of the protocol.

Definition 5 (Security). *Let $T = \{S_{j_1}, \ldots, S_{j_{|T|}}\}$ be the set of the corrupted servers with $|T| \leq t$ which are controlled by the adversary $\mathcal{A}$. The goal of the adversary $\mathcal{A}$ is to infer sensitive information about the users' data. We consider security in the setting where all the servers (including the corrupted servers) correctly execute the protocol. A protocol is t-secure if there is no leak of information about the users' data besides what can be derived from publicly available information.*

4 A DECENTA Solution: DEVA

In this section, we present DEVA, an interactive multi-round protocol, inspired by Segal *et al.* [13] work, designed to solve the DECENTA problem.

DEVA Construction. At any point during the protocol, users may drop out, *i.e.,* a user U_i after sending the round-k messages, may not send the consecutive round-$(k+1)$ messages, where $k \in \{1, 2, 3\}$. By the end of the last round, at least $t+1$ servers together, where $t \leq m-1$, will be able to produce an outcome y and a proof σ, which are used to allow anyone to verify if y is indeed the sum of all the inputs of the *"involved"* (active) users.

Briefly, our idea is to split the secret input x_i of each user U_i among m servers via Shamir's threshold secret sharing as described in Sec. 2, and provide x_{ij} to server S_j. Given the property of Shamir's secret sharing scheme to be additive homomorphic, any subset of $t+1$ servers will be enough to reconstruct y (*i.e.,* the sum of the inputs of the active users) from the given shares x_{ij}. Our main concern is how to prove that the resulted sum y is correct without revealing each user's secret input. A naive way is that each user publishes a value g^{x_i}, and the verification is to check if $\prod_i g^{x_i} = g^y$. We should note that the public value g^{x_i}, probably reveals some information of x_i, but not all x_i (due to the dLog assumption), so we need to randomize g^{x_i} with some random value Ran_i that belongs in the employed group such that $\prod_i \mathsf{Ran}_i = 1$, which implies $\prod_i (g^{x_i} \cdot \mathsf{Ran}_i) = \prod_i g^{x_i} = g^y$. More precisely, the trick is to generate a randomness Ran_i for each user U_i, and looking ahead, Ran_i consists of a sequence of agreed keys between U_i and each other user $U_{i'}$.

Each user needs to execute a key agreement with the other participating users. Thus, we assign groups of participating users to a unique server to reduce the computational and communication costs. More precisely, we sort n users into m groups, each of which consists of n/m amount of users. Here to simplify the explanation,

we assume $m \mid n$, for the general case $m \nmid n$ please refer to our protocol in detail. Next, each group of n/m users generates their own randomness, via their corresponding server, following the trick proposed by Bonawitz *et al.* [13] in which the server plays the role of a bulletin board and coordinates the communications in each group. Later, we address the possible dropouts by suitably adapting the approach in [13] to our case. We assume that, by the end of the last round, there are at least $t_{key}+1$ users which have not dropped out, in each group of n/m users. Our DEVA protocol is described below:

Setup: all parties are given the security parameter λ, the numbers of users n and servers m, thresholds $t < m$ and $t_{key} < \lceil \frac{n}{m} \rceil$, honestly generated $\mathsf{pp} \leftarrow \mathsf{Ksetup}(1^\lambda)$, parameter q such that $\mathbb{Z}_q$ is the space from which inputs are sampled, and a group $\mathbb{G}$ of prime order p to be used for key agreement. All n users are partitioned into m disjoint subsets, *i.e.,* $\Gamma_1, \ldots, \Gamma_m$ where for any $j \in [1,m]$, $\Gamma_i \cap \Gamma_j = \emptyset$. Here, we assume n is divided by m, and $\mid \Gamma_j \mid = \frac{n}{m}$ for $j \in [1,m]$.[5]

Round 1 - KeyGeneration for user U_i associated with S_j: U_i generates key pairs $(\mathsf{sk}_i^{\mathsf{KA}}, \mathsf{pk}_i^{\mathsf{KA}}) \leftarrow \mathsf{KA.Kgen}(\mathsf{pp}, \mathsf{U}_i)$ along with the pairs $(\mathsf{sk}_i^{\mathsf{PKE}}, \mathsf{pk}_i^{\mathsf{PKE}}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$; and publish $(\mathsf{pk}_i^{\mathsf{KA}}, \mathsf{pk}_i^{\mathsf{PKE}})$ before moving to the next round;

Round 1 - KeyGeneration for server S_j associated with Γ_j: S_j collects users' public keys (We denote this set of users by Γ_j^1); broadcasts to all users belonging to Γ_j^1 the list of keys $\{(\mathsf{pk}_i^{\mathsf{KA}}, \mathsf{pk}_i^{\mathsf{PKE}})\}_{\mathsf{U}_i \in \Gamma_j^1}$, and goes to next round;

Round 2 - ShareKeys for user U_i associated with S_j: U_i receives the list $\{(\mathsf{pk}_i^{\mathsf{KA}}, \mathsf{pk}_i^{\mathsf{PKE}})\}_{\mathsf{U}_i \in \Gamma_j^1}$ broadcasted by the server S_j and proceeds to *sharing keys*:

- using a t_{key}-out-of-$\left|\Gamma_j^1\right|$, with $t_{key} < \left|\Gamma_j^1\right|$, secret sharing scheme, it generates shares of $\mathsf{sk}_i^{\mathsf{KA}}$ for each $\mathsf{U}_{i'} \in \Gamma_j^1$. More precisely, user U_i generates $\mathsf{sk}_{i,i'}^{\mathsf{KA}} \leftarrow \mathsf{SS.share}(t_{key}, \mathsf{sk}_i^{\mathsf{KA}}, \mathsf{U}_{i'}, \Gamma_j^1)$;
- uses PKE to encrypt shares $\mathsf{sk}_{i,i'}^{\mathsf{KA}}$ under the public key $\mathsf{pk}_{i'}^{\mathsf{PKE}}$ of each other user $\mathsf{U}_{i'} \in \Gamma_j^1$. More precisely, U_i computes $c_{i,i'} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_{i'}^{\mathsf{PKE}}, \mathsf{sk}_{i,i'}^{\mathsf{KA}})$;

U_i sends ciphertexts $\{c_{i,i'}\}_{\mathsf{U}_{i'} \in \Gamma_j^1}$ to the server S_j, and goes to the next round;

Round 2 - ShareKeys for server S_j associated with Γ_j^1: S_j collects the list of users U_i which have sent $c_{i,i'}$ (we denote this set of users by Γ_j^2); and sends to each user $\mathsf{U}_{i'} \in \Gamma_j^2$ all ciphertexts under his public key $\mathsf{pk}_{i'}^{\mathsf{PKE}}$, *i.e.,* $\{c_{i,i'}\}_{\mathsf{U}_{i'} \in \Gamma_j^2}$;

Round 3 - ShareInputs for user U_i associated with S_j: U_i receives the list of ciphertexts $\{c_{i',i}\}_{\mathsf{U}_{i'} \in \Gamma_j^2}$ broadcasted by S_j and proceeds to *sharing its input*:

- with the list $\{\mathsf{pk}_i^{\mathsf{KA}}\}_{\mathsf{U}_i \in \Gamma_j^2}$ broadcasted by the server S_j, uses the key agreement scheme to compute the agreed key between any two users $\mathsf{U}_i, \mathsf{U}_{i'} \in \Gamma_j^2$, *i.e.,* $\mathsf{s}_{ii'} \leftarrow \mathsf{KA.Kagree}(\mathsf{sk}_{i'}^{\mathsf{KA}}, \mathsf{pk}_i^{\mathsf{KA}})$;
- uses a t-out-of-m secret sharing scheme to generate shares of the input x_i for each server $\mathsf{S}_{j'}$ for $j' \in [1,m]$, *i.e.,* $x_{ij'} \leftarrow \mathsf{SS.share}(t, x_i, \mathsf{S}_{j'}, \{\mathsf{S}_{j'}\}_{j' \in [1,m]})$;

[5] If $m \nmid n$, then $\mid \Gamma_j \mid = \lceil \frac{n}{m} \rceil$ for $j \in [1, m-1]$ and $\mid \Gamma_m \mid = n - (m-1)\lceil \frac{n}{m} \rceil$.

- randomly selects $R_i{}'$ and computes $R_i{}''$ such that $R_i{}' + R_i{}'' = |\mathbb{G}| \cdot \mathsf{Int}$ (2) where Int denotes any positive integer, and computes the values

$$\tau_i := \mathsf{g}^{x_i} \cdot \mathsf{g}^{R_i{}'}, \qquad \rho_i := \mathsf{g}^{R_i{}''} \cdot \prod_{i' \in \Gamma_j^2 : i < i'} \mathsf{s}_{ii'} \cdot \prod_{i' \in \Gamma_j^2 : i > i'} \mathsf{s}_{i'i}{}^{-1}.$$

U_i publishes and sends (τ_i, ρ_i) to the specified server S_j and, additionally, sends $x_{ij'}$ to each server $\mathsf{S}_{j'}$ where $j' \in [1, m]$, and goes to the next round;

Round 3 - ShareInputs for server S_j associated with Γ_j^2: S_j collects the list of users U_i which have sent (τ_i, ρ_i) to S_j (denoted by Γ_j^3); then, S_j *collects the shared inputs* x_{ij} of all $\mathsf{U}_i \in \bigcup_{j=1}^m \Gamma_j^3$, *i.e.,* $\{x_{ij}\}_{\mathsf{U}_i \in \Omega}$ where $\Omega := \bigcup_{j=1}^m \Gamma_j^3$;

Round 4 - Aggregation for user U_i associated with S_j: on receiving the ciphertexts $\{c_{i',i}\}_{\mathsf{U}_{i'} \in \Gamma_j^2}$ of each user $\mathsf{U}_{i'}$, with the decryption key $\mathsf{sk}_i^{\mathsf{PKE}}$, U_i decrypts $\{c_{i',i}\}_{\mathsf{U}_{i'} \in \Gamma_j^2}$. More precisely, U_i gets $\mathsf{sk}_{i',i}^{\mathsf{KA}} \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}_i^{\mathsf{PKE}}, c_{i',i})$, and sends a list of shares $\{\mathsf{sk}_{i',i}^{\mathsf{KA}}\}_{\mathsf{U}_{i'} \in \Gamma_j^2 \setminus \Gamma_j^3}$ to the server S_j;

Round 4 - Aggregation for server S_j associated with Γ_j: S_j collects the list of shares $\{\mathsf{sk}_{i',i}^{\mathsf{KA}}\}_{\mathsf{U}_{i'} \in \Gamma_j^2 \setminus \Gamma_j^3}$ from the users U_i (denote this set of users by Γ_j^4) such that $|\Gamma_j^4| \geq t_{key}$; Consecutively, for each user $\mathsf{U}_{i'} \in \Gamma_j^2 \setminus \Gamma_j^3$, the server S_j:

- *evaluates the shared keys* $\mathsf{sk}_{i'}^{\mathsf{KA}}$ by running the $\mathsf{SS.recon}(t_{key}, \{\mathsf{sk}_{i',i}^{\mathsf{KA}}\}_{i \in \Gamma_j^4}, \Gamma_j^4)$ reconstruction algorithm, and computes $\mathsf{s}_{ii'} \leftarrow \mathsf{KA.Kagree}(\mathsf{sk}_{i'}^{\mathsf{KA}}, \mathsf{pk}_i^{\mathsf{KA}})$, *i.e.,* the agreed keys $\mathsf{s}_{ii'}$;
- *evaluates the missing values* $z_{i'} := \prod_{i \in \Gamma_j^3 : i < i'} \mathsf{s}_{ii'}^{-1} \cdot \prod_{i \in \Gamma_j^3 : i > i'} \mathsf{s}_{i'i}$, $\forall \mathsf{U}_i \in \Gamma_j^3$;
- computes $\omega_{i'} := \prod_{i \in \Gamma_j^3 : i < i'} (\mathsf{pk}_i^{\mathsf{KA}})^{-1} \prod_{i \in \Gamma_j^3 : i > i'} \mathsf{pk}_i^{\mathsf{KA}}$ for all users $\mathsf{U}_i \in \Gamma_j^3$, and a proof $\mathbf{Proof}.\mathsf{DLEQ}(\mathsf{g}, \omega_{i'}, \mathsf{pk}_{i'}^{\mathsf{KA}}, z_{i'}, \mathsf{sk}_{i'}^{\mathsf{KA}})$ with witness $\mathsf{sk}_{i'}^{\mathsf{KA}}$ using the ZK protocol in [2], described in detail in Sec. 2;
- computes the partial value $y_j := \sum_{\mathsf{U}_i \in \Omega} x_{ij}$;

The list $\left(\{\mathsf{pk}_i^{\mathsf{KA}}\}_{\mathsf{U}_i \in \Gamma_j^3}, y_j, \{z_{i'}, \mathbf{Proof}.\mathsf{DLEQ}(\mathsf{g}, \omega_{i'}, \mathsf{pk}_{i'}^{\mathsf{KA}}, z_{i'}, \mathsf{sk}_{i'}^{\mathsf{KA}})\}_{\mathsf{U}_{i'} \in \Gamma_j^2 \setminus \Gamma_j^3}\right)$ is finally given as the output by the server S_j;

Public Verification: given a set of servers $\mathcal{T}$ where $|\mathcal{T}| > t$, any verifier:

- gets from each server S_j the set of active users $\{\mathsf{pk}_i^{\mathsf{KA}}\}_{\mathsf{U}_i \in \Gamma_j^3}$, and computes $\hat{\omega}_{i'} := \prod_{i \in \Gamma_j^3 : i < i'} (\mathsf{pk}_i^{\mathsf{KA}})^{-1} \cdot \prod_{i \in \Gamma_j^3 : i > i'} \mathsf{pk}_i^{\mathsf{KA}}$ for each user $\mathsf{U}_{i'} \in \Gamma_j^2 \setminus \Gamma_j^3$;
- executes $\mathbf{Verify}.\mathsf{DLEQ}(\mathsf{g}, \hat{\omega}_{i'}, \mathsf{pk}_{i'}^{\mathsf{KA}}, z_{i'}, \mathbf{Proof}.\mathsf{DLEQ}(\mathsf{g}, \omega_{i'}, \mathsf{pk}_{i'}^{\mathsf{KA}}, z_{i'}, \mathsf{sk}_{i'}^{\mathsf{KA}}))$ to check if it satisfies $\mathsf{g}^{\mathsf{sk}_{i'}^{\mathsf{KA}}} = \mathsf{pk}_{i'}^{\mathsf{KA}}$ and $(\hat{\omega}_{i'})^{\mathsf{sk}_{i'}^{\mathsf{KA}}} = z_{i'}$, for each user $\mathsf{U}_{i'} \in \Gamma_j^2 \setminus \Gamma_j^3$. If it fails, abort and output 0.
- computes the final result $y := \mathsf{SS.recon}(t, \{y_j\}_{j \in \mathcal{T}}, \mathcal{T})$ given $|\mathcal{T}|$ servers, the value σ as $\sigma := \prod_{j=1}^m \left(\prod_{\mathsf{U}_i \in \Gamma_j^3} \tau_i \cdot \prod_{\mathsf{U}_i \in \Gamma_j^3} \rho_i \cdot \prod_{\mathsf{U}_{i'} \in \Gamma_j^2 \setminus \Gamma_j^3} z_{i'} \right)$ and checks if $\sigma \stackrel{?}{=} H(y)$, for H defined to be the hash function described in Sec. 2. If true, output $(y, 1)$. Otherwise output 0.

Below, we state the DEVA's satisfied properties.

Theorem 1 (DEVA Correctness). *The DEVA protocol is correct,* i.e., *it holds* $Pr\left[\textbf{\textit{Verification}}(\sigma, y) = (y, 1)\right] = 1$, *where* σ *and* y *are the outputs of the protocol, honestly executed by all users and servers.*

We present and prove the following lemma which is necessary to prove DEVA's properties. We abuse notation by equivalently denoting $\mathsf{U}_i \in \Gamma_j^3$ as $i \in \Gamma_j^3$.

Lemma 1. *It holds that*

$$\prod_{i\in\Gamma_j^3}\left(\prod_{i'\in\Gamma_j^2:i<i'} \mathsf{s}_{ii'} \prod_{i'\in\Gamma_j^2:i>i'} \mathsf{s}_{i'i}^{-1}\right)\cdot \prod_{i'\in\Gamma_j^2\setminus\Gamma_j^3}\left(\prod_{i\in\Gamma_j^3:i<i'} \mathsf{s}_{ii'}^{-1} \prod_{i\in\Gamma_j^3:i>i'} \mathsf{s}_{i'i}\right) = \\ = \prod_{i\in\Gamma_j^3}\hat{\rho}_i \cdot \prod_{i'\in\Gamma_j^2\setminus\Gamma_j^3} z_{i'} = 1 \tag{3}$$

Proof (DEVA's Lem. 1). Since $\Gamma_j^2 \equiv \Gamma_j^3 \cup (\Gamma_j^2 \setminus \Gamma_j^3)$, for all $i \in \Gamma_j^3$, it holds

$$\hat{\rho}_i = \left(\prod_{i'\in\Gamma_j^3:i<i'} \mathsf{s}_{ii'} \prod_{i'\in\Gamma_j^2\setminus\Gamma_j^3:i<i'} \mathsf{s}_{ii'}\right)\cdot\left(\prod_{i'\in\Gamma_j^3:i>i'} \mathsf{s}_{i'i}^{-1} \prod_{i'\in\Gamma_j^2\setminus\Gamma_j^3:i>i'} \mathsf{s}_{i'i}^{-1}\right)$$

Observe that $\prod_{i\in\Gamma_j^3}\left(\prod_{i'\in\Gamma_j^3:i<i'} \mathsf{s}_{ii'}\cdot\prod_{i'\in\Gamma_j^3:i>i'} \mathsf{s}_{i'i}^{-1}\right) = 1$, thus implying,

$$\prod_{i\in\Gamma_j^3}\hat{\rho}_i = \prod_{i\in\Gamma_j^3}\left(\prod_{i'\in\Gamma_j^2\setminus\Gamma_j^3:i<i'} \mathsf{s}_{ii'} \prod_{i'\in\Gamma_j^2\setminus\Gamma_j^3:i>i'} \mathsf{s}_{i'i}^{-1}\right) \\ = \prod_{i'\in\Gamma_j^2\setminus\Gamma_j^3}\left(\prod_{i\in\Gamma_j^3:i<i'} \mathsf{s}_{ii'} \prod_{i\in\Gamma_j^3:i>i'} \mathsf{s}_{i'i}^{-1}\right) = \prod_{i'\in\Gamma_j^2\setminus\Gamma_j^3} z_{i'}^{-1}$$

□

Proof (DEVA's Correctness - Thm. 1). Let $\Omega = \bigcup_{j=1}^{m}\Gamma_j^3$ be the set of all users that have sent shared inputs x_{ij} to their corresponding servers. For any $\mathcal{T}$ set of servers with $|\mathcal{T}| > t$, it holds:

$$y = \mathsf{SS.recon}(t, \{y_j\}_{j\in\mathcal{T}}, \mathcal{T}) \overset{see\ eq.(1)}{=} \sum_{i\in\Omega} x_i \tag{4}$$

By construction, we get the following relation that is needed later on:

$$\prod_{\mathsf{U}_i\in\Gamma_j^3}\rho_i = \prod_{i\in\Gamma_j^3}\mathsf{g}^{R_i''}\cdot\prod_{i'\in\Gamma_j^2,i'<i}\mathsf{s}_{ii'}\cdot\prod_{i'\in\Gamma_j^2,i'>i}\mathsf{s}_{ii'}^{-1} \overset{Eq.\ (3)}{=} \prod_{i\in\Gamma_j^3}\mathsf{g}^{R_i''}\cdot\prod_{i\in\Gamma_j^3}\hat{\rho}_i \tag{5}$$

Therefore, we can expand σ as follows:

$$
\begin{aligned}
\sigma &= \prod_{j=1}^{m}\left[\prod_{\mathsf{U}_i\in\Gamma_j^3}\tau_i\prod_{\mathsf{U}_i\in\Gamma_j^3}\rho_i\prod_{\mathsf{U}_{i'}\in\Gamma_j^2\setminus\Gamma_j^3}z_{i'}\right]= \\
&\overset{Eq.\ (5)}{=}\prod_{j=1}^{m}\left[\prod_{i\in\Gamma_j^3}\mathsf{g}^{x_i+R_i'}\left(\prod_{i\in\Gamma_j^3}\mathsf{g}^{R_i''}\prod_{i\in\Gamma_j^3}\hat{\rho}_i\right)\prod_{i'\in\Gamma_j^2\setminus\Gamma_j^3}z_{i'}\right] \\
&=\prod_{j=1}^{m}\left[\left(\prod_{i\in\Gamma_j^3}\mathsf{g}^{x_i+R_i'}\prod_{i\in\Gamma_j^3}\mathsf{g}^{R_i''}\right)\cdot\left(\prod_{i\in\Gamma_j^3}\hat{\rho}_i\prod_{i'\in\Gamma_j^2\setminus\Gamma_j^3}z_{i'}\right)\right] \qquad (6)\\
&\overset{Lem.\ 1}{=}\prod_{j=1}^{m}\left[\prod_{i\in\Gamma_j^3}\mathsf{g}^{x_i+R_i'}\prod_{i\in\Gamma_j^3}\mathsf{g}^{R_i''}\right]=\prod_{i\in\Omega}\mathsf{g}^{x_i+R_i'+R_i''} \\
&\overset{Eq.\ (2)}{=}\mathsf{g}^{\sum_{i\in\Omega}x_i}\overset{Eq.\ (4)}{=}\mathsf{g}^{y}
\end{aligned}
$$

Thus, we get that $\sigma = \mathsf{g}^y = H(y)$ which shows that the verification will give 1 with probability 1, *i.e.,* $\Pr\left[\textbf{Verification}(\sigma, y) = (y, 1)\right] = 1$. □

Theorem 2 (DEVA Verifiability). *For n users $\{U_i\}_{i\in[n]}$ with inputs $\{x_i\}_{i\in[n]}$ such that $y = \sum_{i=1}^{n} x_i$, which honestly execute the protocol, consider any set of corrupted servers $T = \{S_{j_1}, \ldots, S_{j_{|T|}}\}$ with $|T| < m$ which are controlled by a PPT adversary $\mathcal{A}$. The verifiability requirement of DEVA follows Def. 4 and it is specified as follows:*

1. *Users and servers run the protocol's setup round 1 and round 2.*
2. *Execute round 3 and, $\forall j \in [j_1, j_{|T|}]$ such that $S_j \in T$, the server S_j gives $\{x_{1j}, \ldots, x_{nj}\}$ to $\mathcal{A}$ where x_{ij} is the share given to S_j from the user U_i.*
3. *Given the tuples output by the corrupted servers $S_j \in T$ at the end of round 4, $\mathcal{A}$ outputs $\left(y_j{}^*, \{z_{i'}^*, \textbf{Proof}.\mathsf{DLEQ}(g, \omega_{i'}^*, pk_{i'}^{\mathsf{KA}}, z_{i'}^*, sk_{i'}^{\mathsf{KA}})\}_{U_{i'}\in\Gamma_j^2\setminus\Gamma_j^3}\right)$ as a malicious tuple. For honest servers $S_j \notin T$, it honestly computes and publishes $\left(y_j, \{z_{i'}, \textbf{Proof}.\mathsf{DLEQ}(g, \omega_{i'}, pk_{i'}^{\mathsf{KA}}, z_{i'}, sk_{i'}^{\mathsf{KA}})\}_{U_{i'}\in\Gamma_j^2\setminus\Gamma_j^3}\right)$.*
4. *$\mathcal{A}$ outputs the aggregated result y' and the corresponding proof σ' such that $y' \neq y$.*

For any PPT adversary $\mathcal{A}$, DEVA satisfies $\Pr\left[\textbf{\textit{Verification}}(\sigma', y') = 1\right] \leq \mathsf{negl}$.

Proof (DEVA's Verifiability - Thm. 2). Assume **Verification**$(\sigma', y') = 1$, where $y' = y + \Delta$ with $\Delta \neq 0$. Due to the property of proof of knowledge, with overwhelming probability $\mathcal{A}$ knows the secret keys (witnesses) $\mathsf{sk}_{i'}^{\mathsf{KA}}$ of all users that dropout at the end of round 2 and before round 3, such that the proof $\textbf{Proof}.\mathsf{DLEQ}(\mathsf{g}, \omega_{i'}^*, \mathsf{pk}_{i'}^{\mathsf{KA}}, z_{i'}^*, \mathsf{sk}_{i'}^{\mathsf{KA}})$ is valid, *i.e.,* $\mathsf{g}^{\mathsf{sk}_{i'}^{\mathsf{KA}}} = \mathsf{pk}_{i'}^{\mathsf{KA}}$ and $(\omega_{i'}^*)^{\mathsf{sk}_{i'}^{\mathsf{KA}}} = z_{i'}^*$. Let $\Gamma_j^2\setminus\Gamma_j^3$ be the list of honestly dropped users at the end of round 2 and before round 3. Let us consider the two possible cases:

– $\mathcal{A}$ reports an active user as dropped. *W.l.o.g.*, denote this user as U_{fd} and let z_{fd} denote the related missing value computed[6]. Then, we get:

$$\begin{aligned}
&\textbf{Verification}(\sigma', y') = 1 \iff \sigma' = H(y') \\
&\iff \prod_{j=1}^{m} \left[\left(\prod_{\mathsf{U}_i \in \Gamma_j^3} \tau_i \prod_{\mathsf{U}_i \in \Gamma_j^3} \rho_i \prod_{\mathsf{U}_{i'} \in \Gamma_j^2 \setminus \Gamma_j^3} z_{i'} \right) z_{\mathsf{fd}} \right] = \mathsf{g}^{y'} \\
&\iff \prod_{j=1}^{m} \left[\prod_{\mathsf{U}_i \in \Gamma_j^3} \tau_i \prod_{\mathsf{U}_i \in \Gamma_j^3} \rho_i \prod_{\mathsf{U}_{i'} \in \Gamma_j^2 \setminus \Gamma_j^3} z_{i'} \right] \prod_{j=1}^{m} z_{\mathsf{fd}} = \mathsf{g}^{y+\Delta} \\
&\overset{Eq.\ (6)}{\iff} \mathsf{g}^{y} \prod_{j=1}^{m} z_{\mathsf{fd}} = \mathsf{g}^{y} \mathsf{g}^{\Delta} \iff \prod_{j=1}^{m} z_{\mathsf{fd}} = \mathsf{g}^{\Delta}
\end{aligned}$$

– $\mathcal{A}$ reports a dropped out user as active. *W.l.o.g.*, denote this user as U_{fa} and let z_{fa} denote the value computed for this user. Then, we get:

$$\textbf{Verification}(\sigma', y') = 1 \iff \sigma' = H(y') \tag{7}$$

and expanding σ' we have:

$$\begin{aligned}
&\sigma' = \prod_{j=1}^{m} \left[\prod_{\mathsf{U}_i \in \Gamma_j^3} \tau_i \prod_{\mathsf{U}_i \in \Gamma_j^3} \rho_i \prod_{\mathsf{U}_{i'} \in \Gamma_j^2 \setminus (\Gamma_j^3 \cup \mathsf{U}_{\mathsf{fa}})} z_{i'} \right] \\
&\iff \sigma' = \prod_{j=1}^{m} \left[\left(\prod_{\mathsf{U}_i \in \Gamma_j^3} \tau_i \prod_{\mathsf{U}_i \in \Gamma_j^3} \rho_i \prod_{\mathsf{U}_{i'} \in \Gamma_j^2 \setminus (\Gamma_j^3 \cup \mathsf{U}_{\mathsf{fa}})} z_{i'} \right) (z_{\mathsf{fa}} z_{\mathsf{fa}}^{-1}) \right] \\
&\iff \sigma' = \prod_{j=1}^{m} \left[\left(\prod_{\mathsf{U}_i \in \Gamma_j^3} \tau_i \prod_{\mathsf{U}_i \in \Gamma_j^3} \rho_i \prod_{\mathsf{U}_{i'} \in \Gamma_j^2 \setminus \Gamma_j^3} z_{i'} \right) z_{\mathsf{fa}}^{-1} \right] \\
&\iff \sigma' = \prod_{j=1}^{m} \left[\prod_{\mathsf{U}_i \in \Gamma_j^3} \tau_i \prod_{\mathsf{U}_i \in \Gamma_j^3} \rho_i \prod_{\mathsf{U}_{i'} \in \Gamma_j^2 \setminus \Gamma_j^3} z_{i'} \right] \prod_{j=1}^{m} z_{\mathsf{fa}}^{-1} \\
&\overset{Eq.\ (6)}{\iff} \sigma' = \mathsf{g}^{y} \prod_{j=1}^{m} z_{\mathsf{fa}}^{-1}
\end{aligned}$$

Then, Eq. (7) becomes, equivalently:

$$\begin{aligned}
&Eq.\ (7) \iff \mathsf{g}^{y} \prod_{j=1}^{m} z_{\mathsf{fa}}^{-1} = \mathsf{g}^{y'} \iff \mathsf{g}^{y} \prod_{j=1}^{m} z_{\mathsf{fa}}^{-1} = \mathsf{g}^{y+\Delta} \\
&\overset{Eq.\ (6)}{\iff} \mathsf{g}^{y} \prod_{j=1}^{m} z_{\mathsf{fa}}^{-1} = \mathsf{g}^{y} \mathsf{g}^{\Delta} \iff \prod_{j=1}^{m} z_{\mathsf{fa}}^{-1} = \mathsf{g}^{\Delta}
\end{aligned}$$

[6] $\mathcal{A}$ must know the secret key by either breaking the key agreement security **or** by maliciously corrupting the user, *e.g.,* by personally creating it.

In both cases, finding Δ requires to solve a dLog problem which is assumed to be hard. Thus, the only two cases that exist are not feasible. Therefore, it holds $\Pr[\textbf{Verification}(\sigma', y') = 1] \leq \mathsf{negl}$. □

We consider security in the setting where at most t servers are corrupted by the adversary $\mathcal{A}$, namely, assume $T = \{\mathsf{S}_{j_1}, \ldots, \mathsf{S}_{j_{|T|}}\}$ be the set of the corrupted servers such that $|T| \leq t$. All those $|T|$ servers are controlled by $\mathcal{A}$ and all users and servers correctly execute the protocol. $\mathcal{A}$ has the knowledge of at most t_{key} corrupted users' secret inputs. $\mathcal{A}$ attempts to infer the remaining non-corrupted users' secret inputs. We show that the joint view of any set of less than $(t+1)$ corrupted servers and any set of less than $(t_{key}+1)$ corrupted users can be simulated, given the inputs of the corrupted users and only the sum of the inputs of the remaining users. Intuitively, this means that those users and servers learn nothing more than their own inputs, and the sum of the other users' inputs. Consider n users $\mathcal{U} = \{\mathsf{U}_i\}_{i\in[n]}$ along with m servers $\mathcal{S} = \{\mathsf{S}_j\}_{j\in[m]}$, and $\mathcal{U}$ is partitioned into m disjoint subsets, *i.e.,* $\mathcal{U} = \Gamma_1, \ldots, \Gamma_m$ where for any $j, j' \in [1,m]$, $\Gamma_j \cap \Gamma_{j'} = \emptyset$. Let the input of each user U_i be x_i. For simplicity, we assume m divides n, and $|\Gamma_j| = \frac{n}{m}$ for $j \in [1,m]$. Assume that the group of users Γ_j corresponds to server S_j. Denote by $\Gamma_j^1, \Gamma_j^2, \Gamma_j^3, \Gamma_j^4$ the subsets of users in Γ_j that successfully sent their messages to the corresponding server S_j at round 1, 2, 3 and 4 respectively, such that $\Gamma_j \supseteq \Gamma_j^1 \supseteq \Gamma_j^2 \supseteq \Gamma_j^3 \supseteq \Gamma_j^4$. For example, users in $\Gamma_j^1 \setminus \Gamma_j^2$ are those that abort after completing the execution of round 1 but before sending the message to S_j in round 2. Let $\mathcal{S}'$ be the corrupted servers such that $|\mathcal{S}'| \leq t$, and $\mathcal{U}'$ the corrupted users such that $|\mathcal{U}'| \leq t_{key}$. Let $\mathsf{Real}_{\mathcal{U},\mathcal{S}}^{\mathcal{U}',\mathcal{S}',t,t_{key}}(\{x_i\}_{\mathsf{U}_i\in\mathcal{U}}, \{\Gamma_j^1, \Gamma_j^2, \Gamma_j^3, \Gamma_j^4\}_{j\in[1,m]})$ be a random variable representing the views of all corrupted users in $\mathcal{U}'$ and all corrupted servers in $\mathcal{S}'$ after executing the above instantiated protocol, where the randomness is over their internal randomness and the ones in the setup phase.

Theorem 3 (DEVA Security). *There exists a PPT simulator* Sim *such that for all* $t < m$ *and* $t_{key} < \lceil \frac{n}{m} \rceil$, $\mathcal{U}$, $\mathcal{S}$, $\mathcal{U}'$, $\mathcal{S}'$, $\{x_i\}_{\mathsf{U}_i\in\mathcal{U}'}$, *and* $\{\Gamma_j^1, \Gamma_j^2, \Gamma_j^3, \Gamma_j^4\}_{j\in[1,m]}$, *such that* $|\mathcal{S}'| \leq t$, $|\mathcal{U}'| \leq t_{key}$, $\mathcal{U}' \subseteq \mathcal{U}$, $\mathcal{S}' \subseteq \mathcal{S}$, $\Gamma_j^1 \supseteq \Gamma_j^2 \supseteq \Gamma_j^3 \supseteq \Gamma_j^4$ *for* $j \in [1,m]$, *and* $\mathcal{U}' \subset (\bigcup_{j=1}^m \Gamma_j^4)$, *the output of* Sim *is computationally indistinguishable from the output of* $\mathsf{Real}_{\mathcal{U},\mathcal{S}}^{\mathcal{U}',\mathcal{S}',t,t_{key}}$, *or:*

$$\begin{aligned}&\mathsf{Real}_{\mathcal{U},\mathcal{S}}^{\mathcal{U}',\mathcal{S}',t,t_{key}}\left(\{x_i\}_{\mathsf{U}_i\in\mathcal{U}}, \{\Gamma_j^1, \Gamma_j^2, \Gamma_j^3, \Gamma_j^4\}_{j\in[1,m]}\right)\\ &\quad\overset{c}{\approx}\ \mathsf{Sim}_{\mathcal{U},\mathcal{S}}^{\mathcal{U}',\mathcal{S}',t,t_{key}}\left(\{x_i\}_{\mathsf{U}_i\in\mathcal{U}'}, \mathsf{aux}, \{\Gamma_j^1, \Gamma_j^2, \Gamma_j^3, \Gamma_j^4\}_{j\in[1,m]}\right)\end{aligned}$$

where, by considering $\Omega \coloneqq \bigcup_{j\in[1,m]} \Gamma_j^3$, *and* $\mathsf{aux} \coloneqq \sum_{\mathsf{U}_i\in\Omega\setminus\mathcal{U}'} x_i$ *if* $|\Gamma_j^4| > t_{key}$ *for* $\forall j \in [1,m]$; *otherwise* $\mathsf{aux} \coloneqq \perp$.

Proof (DEVA's Security - Thm. 3). Let us construct the simulator Sim by doing a sequence of games from the initial view of the real execution $\mathsf{Real}_{\mathcal{U},\mathcal{S}}^{\mathcal{U}',\mathcal{S}',t,t_{key}}$ such that any two consecutive games are computationally indistinguishable.

Game$_0$: Real is exactly the joint view of the set of corrupted servers $\mathcal{S}'$ and corrupted users $\mathcal{U}'$ in a real execution of the above instantiated protocol.

Game$_1$: given the set of corrupted users $\mathcal{U}'$, let $\Upsilon_j^2 := \mathcal{U}' \cap \Gamma_j^2$ for all $j \in [1,m]$. In Game_1, for all $j \in [1,m]$, the ciphertexts that are received by honest users $\mathsf{U}_{i'} \in \Gamma_j^2 \setminus \Upsilon_j^2$ and sent from honest users $\mathsf{U}_i \in \Gamma_j^2 \setminus \Upsilon_j^2$, are replaced with encryptions of 0 instead of $\mathsf{sk}_{i,i'}^{\mathsf{KA}}$, *i.e.,* computing $c_{i,i'} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_{i'}^{\mathsf{PKE}}, 0)$ instead of $c_{i,i'} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}_{i'}^{\mathsf{PKE}}, \mathsf{sk}_{i,i'}^{\mathsf{KA}})$. The IND-CPA security of the PKE encryption scheme guarantees that this game is indistinguishable from the previous one.

Game$_2$: for all $j \in [1,m]$, when the user $\mathsf{U}_i \in (\Gamma_j^2 \setminus \Gamma_j^3) \setminus \Upsilon_j^2$ generates shares of $\mathsf{sk}_i^{\mathsf{KA}}$, we substitute all shares of $\mathsf{sk}_i^{\mathsf{KA}}$ with shares of 0 (every user U_i in the set $(\Gamma_j^2 \setminus \Gamma_j^3) \setminus \Upsilon_j^2$ uses a different sharing of 0), and give those shares to the corrupted users in set Υ_j^2 in Round ShareKeys, *i.e.,* computing $\mathsf{sk}_{i,i'}^{\mathsf{KA}} \leftarrow \mathsf{SS.share}(t_{key}, 0, \mathsf{U}_{i'}, \Upsilon_j^2)$ for $\mathsf{U}_{i'} \in \Upsilon_j^2$ instead of computing $\mathsf{sk}_{i,i'}^{\mathsf{KA}} \leftarrow \mathsf{SS.share}(t_{key}, \mathsf{sk}_i^{\mathsf{KA}}, \mathsf{U}_{i'}, \Upsilon_j^2)$. The properties of Shamir's secret sharing guarantee that the distribution of any $|\mathcal{U}'|$ shares of 0 is identical to the distribution of an equivalent number of shares of $\mathsf{sk}_i^{\mathsf{KA}}$, making this game and the previous one identically distributed.

Game$_3$: for all $j \in [1,m]$, for each user $\mathsf{U}_i \in (\Gamma_j^2 \setminus \Gamma_j^3) \setminus \Upsilon_j^2$, instead of computing $\rho_i := \mathsf{g}^{R_i''} \cdot \prod_{i' \in \Gamma_j^2 : i < i'} \mathsf{s}_{ii'} \cdot \prod_{i' \in \Gamma_j^2 : i > i'} {\mathsf{s}_{i'i}}^{-1}$ and $\tau_i := \mathsf{g}^{x_i} \cdot \mathsf{g}^{R_i'}$, we compute $\rho_i := \mathsf{g}^{\zeta_i} \cdot \prod_{i' \in \Gamma_j^2 : i < i'} \mathsf{s}_{ii'} \cdot \prod_{i' \in \Gamma_j^2 : i > i'} {\mathsf{s}_{i'i}}^{-1}$ and $\tau_i := \mathsf{g}^{\eta_i}$, where $\zeta_i := -\eta_i$ and η_i is sampled uniformly at random. Since R_i', R_i'' are uniformly random values, this game and the previous one are identically distributed.

Game$_4$: given the set of corrupted users $\mathcal{U}'$, let $\Upsilon_j^3 := \mathcal{U}' \cap \Gamma_j^3$ for all $j \in [1,m]$. In Game_4, for all $j \in [1,m]$, when user $\mathsf{U}_i \in \Gamma_j^3 \setminus \Upsilon_j^3$ generates shares of $\mathsf{sk}_i^{\mathsf{KA}}$, we substitute all shares of $\mathsf{sk}_i^{\mathsf{KA}}$ with shares of 0 (every $\mathsf{U}_i \in \Gamma_j^3 \setminus \Upsilon_j^3$ uses a different sharing of 0), and give those shares to the corrupted users in set Υ_j^3 in Round 2 - ShareKeys for user U_i, *i.e.,* computing $\mathsf{sk}_{i,i'}^{\mathsf{KA}} \leftarrow \mathsf{SS.share}(t_{key}, 0, \mathsf{U}_{i'}, \Upsilon_j^3)$ for $\mathsf{U}_{i'} \in \Upsilon_j^3$. The security of the threshold secret sharing scheme guarantee that Game_4 is identically distributed as Game_3.

Game$_5$: for a fixed user $\mathsf{U}_{i^*} \in \Gamma_j^3 \setminus \Upsilon_j^3$ as well as for other users $\mathsf{U}_i \in (\Gamma_j^3 \setminus \Upsilon_j^3) \setminus \{\mathsf{U}_{i^*}\}$, we substitute $\mathsf{s}_{i^*i} = \mathsf{s}_{ii^*}$ with a uniformly random value, instead of computing the value $\mathsf{s}_{i^*i} = \mathsf{s}_{ii^*} \leftarrow \mathsf{KA.Kagree}(\mathsf{sk}_{i^*}^{\mathsf{KA}}, \mathsf{pk}_i^{\mathsf{KA}})$. More precisely, Sim computes, for any user $\mathsf{U}_i \in (\Gamma_j^3 \setminus \Upsilon_j^3) \setminus \{\mathsf{U}_{i^*}\}$:

$$\rho_i \coloneqq \mathsf{g}^{\zeta_i} \underbrace{\left(\prod_{i' \in \Gamma_j^2 \setminus \{\mathsf{U}_{i^*}\}}^{i<i'} \mathsf{s}_{ii'} \prod_{i' \in \Gamma_j^2 \setminus \{\mathsf{U}_{i^*}\}}^{i>i'} {\mathsf{s}_{i'i}}^{-1} \right) \widetilde{\mathsf{s}_{ii^*}}}_{\vartheta_i},$$

$$\text{where} \quad \widetilde{\mathsf{s}_{ii^*}} \coloneqq \begin{cases} \mathsf{s}_{ii^*} & \text{if } i^* > i \\ \mathsf{s}_{ii^*}^{-1} & \text{if } i^* < i \end{cases} \quad \text{and} \quad \mathsf{s}_{ii^*} = \mathsf{s}_{i^*i}$$

is a random element of $\mathbb{G}$, $z_i := \vartheta_i \cdot \widetilde{\mathsf{s}_{ii^*}}$, and

$$\omega_i := \left(\prod_{i' \in \Gamma_j^2 \setminus \{\mathsf{U}_{i^*}\}}^{i<i'} \mathsf{pk}_{i'}^{\mathsf{KA}} \prod_{i' \in \Gamma_j^2 \setminus \{\mathsf{U}_{i^*}\}}^{i>i'} (\mathsf{pk}_{i'}^{\mathsf{KA}})^{-1} \right) \widetilde{\mathsf{pk}_{i^*}^{\mathsf{KA}}},$$

$$\text{where} \quad \widetilde{\mathsf{pk}_{i^*}^{\mathsf{KA}}} := \begin{cases} \mathsf{pk}_{i^*}^{\mathsf{KA}} & \text{if } i^* > i \\ (\mathsf{pk}_{i^*}^{\mathsf{KA}})^{-1} & \text{if } i^* < i \end{cases}$$

and generates $\textbf{Proof}.\mathsf{DLEQ}(\mathsf{g}, \omega_i, \mathsf{pk}_i^{\mathsf{KA}}, z_i, \mathsf{sk}_i^{\mathsf{KA}})$ using the simulator of the ZK proof. For the fixed user $\mathsf{U}_{i^*} \in \Gamma_j^3 \setminus \Upsilon_j^3$, Sim computes,

$$\rho_{i^*} := \mathsf{g}^{\zeta_{i^*}} \underbrace{\left(\prod_{i' \in \Gamma_j^2}^{i^*<i'} \mathsf{s}_{i^*i'} \prod_{i' \in \Gamma_j^2}^{i^*>i'} {\mathsf{s}_{i^*i'}}^{-1} \right)}_{\vartheta_{i^*}}, \; z_{i^*} := \vartheta_{i^*}, \text{ and}$$

$$\omega_{i^*} := \prod_{i' \in \Gamma_j^2}^{i^*<i'} \mathsf{pk}_{i^*}^{\mathsf{KA}} \prod_{i' \in \Gamma_j^2}^{i^*>i'} (\mathsf{pk}_{i^*}^{\mathsf{KA}})^{-1}$$

and generates $\textbf{Proof}.\mathsf{DLEQ}(\mathsf{g}, \omega_{i^*}, \mathsf{pk}_{i^*}^{\mathsf{KA}}, z_{i^*}, \mathsf{sk}_{i^*}^{\mathsf{KA}})$ using the ZK proof's simulator. The DDH assumption and ZK property assure Game_5 to be indistinguishable from Game_4.

$\textbf{Game}_6$ or Sim: for all users $\mathsf{U}_i \in \Gamma_j^3 \setminus \Upsilon_j^3$, instead of computing

$$\tau_i := \mathsf{g}^{x_i} \mathsf{g}^{{R_i}'}, \quad \rho_i := \mathsf{g}^{{R_i}''} \left(\prod_{i' \in \Gamma_j^2}^{i<i'} \mathsf{s}_{ii'} \prod_{i' \in \Gamma_j^2}^{i>i'} {\mathsf{s}_{i'i}}^{-1} \right) = \mathsf{g}^{{R_i}''} \cdot \tag{8}$$

$$\cdot \left(\prod_{i' \in \Gamma_j^2 \setminus (\Gamma_j^3 \setminus \Upsilon_j^3)}^{i<i'} \mathsf{s}_{ii'} \prod_{i' \in \Gamma_j^2 \setminus (\Gamma_j^3 \setminus \Upsilon_j^3)}^{i>i'} {\mathsf{s}_{i'i}}^{-1} \right) \left(\prod_{i' \in \Gamma_j^3 \setminus \Upsilon_j^3}^{i<i'} \mathsf{s}_{ii'} \prod_{i' \in \Gamma_j^3 \setminus \Upsilon_j^3}^{i>i'} {\mathsf{s}_{i'i}}^{-1} \right)$$

we compute

$$\tau_i := \mathsf{g}^{\eta_i}, \qquad \rho_i := \mathsf{g}^{\zeta_i} \prod_{i' \in \Gamma_j^2 \setminus (\Gamma_j^3 \setminus \Upsilon_j^3) : i<i'} \mathsf{s}_{ii'} \prod_{i' \in \Gamma_j^2 \setminus (\Gamma_j^3 \setminus \Upsilon_j^3) : i>i'} {\mathsf{s}_{i'i}}^{-1} \tag{9}$$

where η_i and ζ_i are sampled uniformly at random and are subject to

$$\sum_{i \in \bigcup_{j=1}^m (\Gamma_j^3 \setminus \Upsilon_j^3)} (\eta_i + \zeta_i) = \mathsf{aux} = \sum_{i \in \left(\bigcup_{j=1}^m \Gamma_j^3 \right) \setminus \mathcal{U}'} x_i$$

To generate the shares of an input for each user $\mathsf{U}_i \in \bigcup_{j=1}^m (\Gamma_j^3 \setminus \Upsilon_j^3)$, given aux, the simulator Sim randomly chooses x_i' such that $\sum_{i \in \bigcup_{j=1}^m (\Gamma_j^3 \setminus \Upsilon_j^3)} x_i' = \mathsf{aux}$, and shares x_i' among m servers using t-out-of-m secret sharing scheme, *i.e.,* for each server S_j for $j \in [1, m]$, $x_{ij} \leftarrow \mathsf{SS.share}(t, x_i', \mathsf{S}_j, \{\mathsf{S}_j\}_{j \in [1,m]})$.

For τ_i and ρ_i generated as in Eq. (9), it implies that, for $\Xi_j := \bigcup_{j=1}^{m}(\Gamma_j^3 \setminus \Upsilon_j^3)$,

$$\begin{aligned}
\prod_{i\in\Xi_j} \tau_i \cdot \rho_i &= \prod_{i\in\Xi_j} \left[\mathsf{g}^{\eta_i} \cdot \mathsf{g}^{\zeta_i} \cdot \left(\prod_{i'\in\Gamma_j^2\setminus(\Gamma_j^3\setminus\Upsilon_j^3)}^{i<i'} \mathsf{s}_{ii'} \prod_{i'\in\Gamma_j^2\setminus(\Gamma_j^3\setminus\Upsilon_j^3)}^{i>i'} \mathsf{s}_{i'i}{}^{-1} \right) \right] \\
&= \mathsf{g}^{\sum_{i\in\Xi_j}(\eta_i+\zeta_i)} \cdot \left[\prod_{i\in\Xi_j} \left(\prod_{i'\in\Gamma_j^2\setminus(\Gamma_j^3\setminus\Upsilon_j^3)}^{i<i'} \mathsf{s}_{ii'} \prod_{i'\in\Gamma_j^2\setminus(\Gamma_j^3\setminus\Upsilon_j^3)}^{i>i'} \mathsf{s}_{i'i}{}^{-1} \right) \right] \\
&= \mathsf{g}^{\mathsf{aux}} \cdot \left[\prod_{i\in\Xi_j} \left(\prod_{i'\in\Gamma_j^2\setminus(\Gamma_j^3\setminus\Upsilon_j^3)}^{i<i'} \mathsf{s}_{ii'} \prod_{i'\in\Gamma_j^2\setminus(\Gamma_j^3\setminus\Upsilon_j^3)}^{i>i'} \mathsf{s}_{i'i}{}^{-1} \right) \right]
\end{aligned}$$

while for honestly generated τ_i and ρ_i as Eq. (8),

$$\begin{aligned}
&\text{it holds that} \quad \prod_{i\in\Xi_j} \tau_i \cdot \rho_i = \\
&= \prod_{i\in\Xi_j} \left[\mathsf{g}^{x_i + R_i{}' + R_i{}''} \cdot \left(\prod_{i'\in\Gamma_j^2\setminus(\Gamma_j^3\setminus\Upsilon_j^3)}^{i<i'} \mathsf{s}_{ii'} \prod_{i'\in\Gamma_j^2\setminus(\Gamma_j^3\setminus\Upsilon_j^3)}^{i>i'} \mathsf{s}_{i'i}{}^{-1} \right) \cdot \right. \\
&\qquad \left. \cdot \left(\prod_{i'\in\Gamma_j^3\setminus\Upsilon_j^3}^{i<i'} \mathsf{s}_{ii'} \prod_{i'\in\Gamma_j^3\setminus\Upsilon_j^3}^{i>i'} \mathsf{s}_{i'i}{}^{-1} \right) \right] \\
&= \mathsf{g}^{\sum_{i\in\Xi_j} x_i} \cdot \left[\prod_{i\in\Xi_j} \left(\prod_{i'\in\Gamma_j^3\setminus\Upsilon_j^3}^{i<i'} \mathsf{s}_{ii'} \prod_{i'\in\Gamma_j^3\setminus\Upsilon_j^3}^{i>i'} \mathsf{s}_{i'i}{}^{-1} \right) \right] \cdot \\
&\qquad \cdot \left[\prod_{i\in\Xi_j} \left(\prod_{i'\in\Gamma_j^2\setminus(\Gamma_j^3\setminus\Upsilon_j^3)}^{i<i'} \mathsf{s}_{ii'} \prod_{i'\in\Gamma_j^2\setminus(\Gamma_j^3\setminus\Upsilon_j^3)}^{i>i'} \mathsf{s}_{i'i}{}^{-1} \right) \right] \\
&= \mathsf{g}^{\mathsf{aux}} \cdot \left[\prod_{i\in\Xi_j} \left(\prod_{i'\in\Gamma_j^2\setminus(\Gamma_j^3\setminus\Upsilon_j^3)}^{i<i'} \mathsf{s}_{ii'} \prod_{i'\in\Gamma_j^2\setminus(\Gamma_j^3\setminus\Upsilon_j^3)}^{i>i'} \mathsf{s}_{i'i}{}^{-1} \right) \right]
\end{aligned}$$

This implies that, choosing η_i and ζ_i uniformly at random to compute τ_i and ρ_i as in Eq. (9) is identically distributed with computing τ_i and ρ_i as in Eq. (8). Since for all $\mathsf{U}_i \in \Gamma_j^3 \setminus \Upsilon_j^3$, η_i and ζ_i are sampled uniformly at random, to generate τ_i and ρ_i, the simulator Sim does not need the knowledge of individual x_i for $\mathsf{U}_i \in \Xi_j$ but, instead, their sum $\sum_{i\in\Xi_j} x_i = \mathsf{aux}$ is sufficient for the simulation. This implies the indistinguishability between Game_5 and Game_6. □

5 Evaluation

This section describes several experimental results from the implementation of our DEVA protocol. We explain the different findings of DEVA, and provide comparison

to prior work of VerifyNet by Xu *et al.* [19]. We got our protocol's experimental results, by implementing a prototype in Python 3.8.3. The execution of the tests was on MacOS 10.14.6 over a MacBookPro (2017) with processor Intel i7-7820HQ CPU @ 2.9GHz, with 16GB LPDDR3 2133MHz RAM, 1MB L2 cache and 8MB L3 cache. We used Diffie-Hellman over the elliptic curve secp256k1 for the key agreement, the Shamir's secret sharing scheme as an additive homomorphic secret sharing scheme, and RSA-2048 as a public key encryption scheme. The execution time provided is expressed in milliseconds (ms), while the bandwidth is presented in kilobytes (kB). The source code of our protocol is publicly released[7].

5.1 Implementation analysis

In this subsection, we explore how our DEVA protocol performs when considering different parameters, *e.g.,* number of users, number of servers or the amount of dropout users and how this affects the communication bandwidth of the protocol and the execution time required.

We are interested in *(i)* each user's execution time and the output data size in relation to the amount of employed servers but also to a different percent of dropout users; *(ii)* each server's execution time and input data size *w.r.t.* the amount of users and the percentage of dropout users; *(iii)* the verification's execution time and the data input size in relation to the amount of users, servers and the number of dropout users considered; and lastly, *(iv)* the total communication bandwidth in relation to the amount of users, servers and number of users that have dropped out.

We describe how our DEVA protocol performs and explain its behavior in each case. The results for the different costs considered per user or per server include all the rounds of the protocol (excluding **Round 1**). Specifically for the server execution time the results contain the cost just from **Round 4** where the aggregation takes place, since no other computation is performed elsewhere by the server.

Execution and communication cost analysis per user. Our decentralized protocol employs multiple servers for the computation to achieve less computation time per user which is shown to be the case in Fig. 2. In fact, in this figure, it is clear that

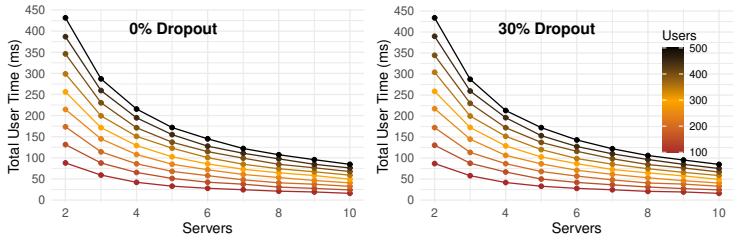

Figure 1. User execution time for 0% and 30% of dropout users.

when the amount of servers is increased, the required execution time for each user

[7] All code will be released publicly after publication, but is already available to reviewers upon request through the program committee.

decreases. We also observe that when we consider more users, the execution time increases, which is expected since, in that case, each user belongs to a bigger disjoint subset Γ_j; therefore, needs to exchange information within a bigger set of users. Lastly, comparing the two scenarios of dropout, 0% and 30% respectively, we notice minor differences.This happens because the dropout of the users, in the experiments, occurs in **Round 3**, where the computational costly operations that the user performs are already made. We should clarify here that, in our implementation, dropout takes place at that point of the DEVA protocol with the aim to illustrate the maximum computation time from the user side. Regarding the communication bandwidth that each user has in our DEVA protocol, we expect that the employment of multiple servers results in smaller communication cost for each user. This is because when the protocol uses more servers, less amount of users are connecting to a single server; thus, for *e.g.,* a single user exchanges shares of keys with less users. This expectation is represented in the Fig. 3. Additionally, the figure shows that when dropouts of users occur, less output data are given by each user; which is reasonable since less users are active in that case. Finally, when more users participate in the DEVA protocol, more data communication is required from each user because of the exchange of keys between the users.

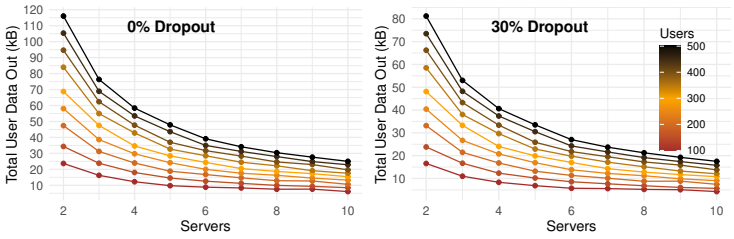

Figure 2. User output data for 0% and 30% of dropout users.

Execution and communication cost analysis per server. The execution time required during the DEVA protocol per server depends on the number of servers that participate in the protocol. More precisely, a big amount of servers participating, offloads the execution time required for each server. On the other hand, the amount of users can affect the time cost of the server in two ways. Firstly, more users require more execution time for the server since each of them handles more computations (since each server handles $\frac{n}{m}$ users when it comes to key sharing (**Round 2**)). Secondly, when there is a user dropout, servers need to compute, among other values, the missing keys from the dropout users as well as the proof **Proof**.DLEQ$(\mathsf{g}, \omega_{i'}, \mathsf{pk}_{i'}^{\mathsf{KA}}, z_{i'}, \mathsf{sk}_{i'}^{\mathsf{KA}})$ for each of them; thus, requiring more execution time. The expected behavior of DEVA is illustrated in Fig. 4. The bandwidth cost of each server is easily explained. Less data are received when dropouts of users occur (less users send data to each server). Our experiments show a small difference due to when the dropout happens in the implementation, as we have previously mentioned. Similarly, when more servers are employed, each server receives less data because it handles less users. Our expectations are clearly depicted in Fig. 5.

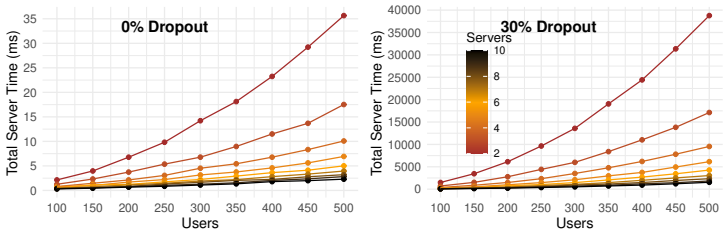

Figure 3. Server execution time for 0% and 30% of dropout users.

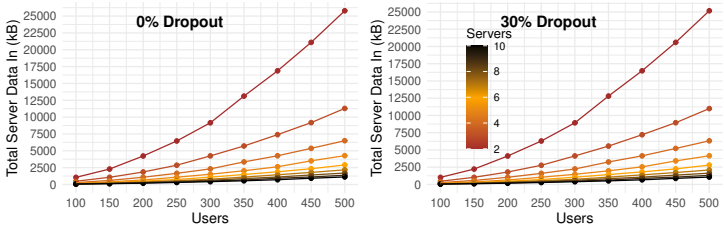

Figure 4. Server input data for 0% and 30% of dropout users.

DEVA verification time and communication cost. The verification execution time depends on several parameters that can affect the timing. Surely, a bigger amount of servers should not influence the verification execution time, while *w.r.t.* bigger amount of users or percent of dropout users, the verification time is expected to increase. Fig. 6 illustrates the expected behavior of our protocol, considering 500 users and 10 servers for the presented plots, respectively. Regarding the input

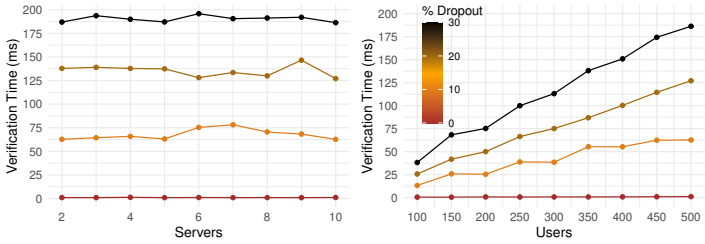

Figure 5. Verification time of DEVA

data needed for the verification, the amount of servers does not affect the input data needed, while bigger dropouts of users require more data. This is because for a smaller number of active users, less public keys are received but more zero knowledge proofs need to be checked. In fact, observe our experimental results depicted in Fig. 7.

DEVA total communication cost (bandwidth). Finally, the total bandwidth of the DEVA protocol is shown in Fig. 8 and shows that when multiple servers are employed the total bandwidth of DEVA decreases. Therefore, using more servers results in less communication cost which reports precisely our expectation. Moreover,

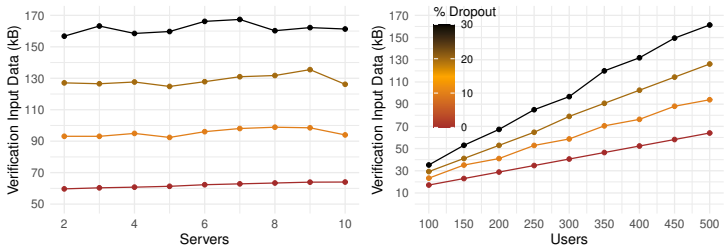

Figure 6. Verification input data of DEVA.

we observe that DEVA requires smaller communication cost for more dropout users, demonstrating that our protocol handles dropouts very well.

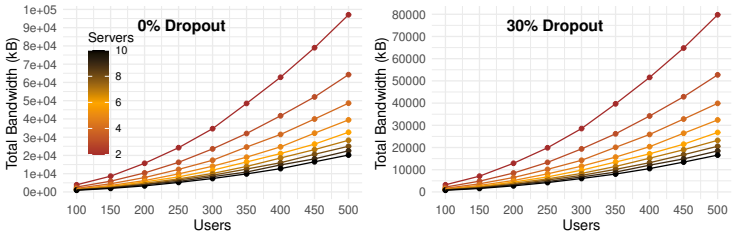

Figure 7. Total bandwidth of DEVA.

5.2 Comparison

In this subsection, we compare DEVA and the protocol provided by Xu *et al.* VerifyNet [19]. VerifyNet's experiments are conducted on a Intel Xeon E5-2620 CPU @ 2.10GHz, 16GB RAM running Ubuntu 18.04. To the best of our knowledge, the authors did not publicly release their source code and, as an additional complication, the CPUs used for running the experiment are hard to compare since Xu *et al.*'s machines are server-CPUs while DEVA's experiments are obtained from a laptop-CPU. For these reasons, we limit our comparison on just the amount of data transmitted by the user. VerifyNet's users have secret *vectors* of length $K = 1000$ as input to the aggregation protocol. To fairly compare, we repeatedly execute our DEVA protocol K times in order to achieve the same amount of aggregated bytes. We execute our experiments in a reasonably distributed setting of $m = 10$ servers, threshold $t = 1$ and key threshold $t_{key} = 1$. In Fig. 9, we compare the amount of data transmitted for each user in executing DEVA or VerifyNet with respect to different amounts of users n or vector sizes K. DEVA is linearly dependent both in the amount of user n and vector size K, while VerifyNet is linear in the vector size but *quadratic* in the amount of users. This different increase factor implies that there will always be, for a fixed vector size K, an amount of users from which our DEVA protocol is more efficient than VerifyNet. As previously discussed, this is due to the fact that in DEVA, the higher the amount of servers, the smaller the amount of data transmitted

by each user because it belongs to a smaller subset Γ_j, while the size of this subset depends on the amount of servers. On the other hand, DEVA is clearly not optimal when considering large vector-inputs. It must be observed that VerifyNet is *designed* to work with vectors, key aspect of the specific comparison. DEVA is penalized since multiple executions must be made, thus, posing the DEVA's extension, that allows the usage of vectors as input, an interesting future development.

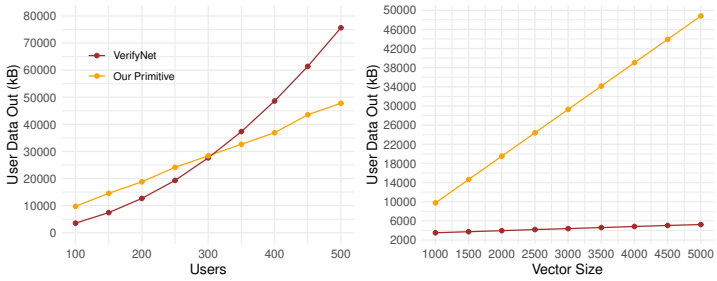

Figure 8. User's data out comparison for fixed $K = 1000$ and $n = 100$.

6 Conclusion

We proposed DEVA, a secure and practical protocol that allows organizations to collaboratively train their model by employing multiple cloud servers. It protects users' privacy, handles users' dropouts that occur at any round, and provides public output verifiability allowing anyone to check the correctness of the aggregated parameters and thus, it provides greater transparency in the learning process. Servers are *independent* in DEVA and only a threshold amount of them is required to compute the sum. We provided the execution time and bandwidth cost analysis of DEVA for different cases. DEVA is designed to deal well with a large number of users compared to the state of the art, while a future direction would be to extend our work integrating vector size inputs.

Acknowledgement. This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

References

1. J. C. Benaloh. Secret sharing homomorphisms: Keeping shares of a secret secret. In *Conference on the Theory and Application of Cryptographic Techniques*, Berlin, 1987. Springer-Verlag.
2. D. Chaum and T. P. Pedersen. Wallet databases with observers. In E. F. Brickell, editor, *Advances in Cryptology — CRYPTO' 92*, pages 89–105, Berlin, Heidelberg, 1993. Springer.
3. W. Diffie and M. E. Hellman. New Directions in Cryptography. *IEEE Trans. Inf. Theory*, 22(6):644–654, 1976.

4. K. Emura. Privacy-preserving aggregation of time-series data with public verifiability from simple assumptions. In J. Pieprzyk and S. Suriadi, editors, *Information Security and Privacy*, pages 193–213, Cham, 2017. Springer International Publishing.
5. Z. Ghodsi, T. Gu, and S. Garg. Safetynets: Verifiable execution of deep neural networks on an untrusted cloud. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*, pages 4672–4681, 2017.
6. B. Hitaj, G. Ateniese, and F. Pérez-Cruz. Deep models under the GAN: information leakage from collaborative deep learning. In *Proc. of CCS*, pages 603–618, 2017.
7. P. Kairouz, H. B. McMahan, B. Avent, and A. B. *et al*. Advances and open problems in federated learning. *CoRR*, abs/1912.04977, 2019.
8. N. Koblitz. Elliptic curve cryptosystems. *Mathematics of computation*, 48(177):203–209, Jan. 1987.
9. M. Krohn, M. Freedman, and D. Mazieres. On-the-fly verification of rateless erasure codes for efficient content distribution. In *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004*, pages 226–240, Berkeley, CA, USA, 2004.
10. I. Leontiadis, K. Elkhiyaoui, M. Önen, and R. Molva. Puda – privacy and unforgeability for data aggregation. In M. Reiter and D. Naccache, editors, *Cryptology and Network Security*, pages 3–18, Cham, 2015. Springer International Publishing.
11. Y. Liu, S. Ma, Y. Aafer, W. Lee, J. Zhai, W. Wang, and X. Zhang. Trojaning attack on neural networks. In *25th Annual Network and Distributed System Security Symposium, NDSS*. The Internet Society, 2018.
12. L. T. Phong, Y. Aono, T. Hayashi, L. Wang, and S. Moriai. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Trans. Information Forensics and Security*, 13(5):1333–1345, 2018.
13. A. Segal, A. Marcedone, B. Kreuter, D. Ramage, H. B. McMahan, K. Seth, K. Bonawitz, S. Patel, and V. Ivanov. Practical secure aggregation for privacy-preserving machine learning. In *CCS*, 2017.
14. A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.
15. E. Shi, T.-H. Chan, E. Rieffel, R. Chow, and D. Song. Privacy-preserving aggregation of time-series data. volume 2, 01 2011.
16. R. Shokri and V. Shmatikov. Privacy-preserving deep learning. In I. Ray, N. Li, and C. Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1310–1321. ACM, 2015.
17. F. Tramèr and D. Boneh. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. In *Proceedings of ICLR*, 2019.
18. G. Tsaloli and A. Mitrokotsa. Sum it up: Verifiable additive homomorphic secret sharing. In J. H. Seo, editor, *Information Security and Cryptology – ICISC 2019*, pages 115–132, Cham, 2020. Springer International Publishing.
19. G. Xu, H. Li, S. Liu, K. Yang, and X. Lin. Verifynet: Secure and verifiable federated learning. *IEEE Trans. Information Forensics and Security*, 15:911–926, 2020.
20. W. Xu, D. Evans, and Y. Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society, 2018.
21. H. Yao, C. Wang, B. Hai, and S. Zhu. Homomorphic Hash and Blockchain Based Authentication Key Exchange Protocol for Strangers. In *International Conference on Advanced Cloud and Big Data (CBD)*, pages 243–248, Lanzhou, 2018.