# Non-Interactive, Secure Verifiable Aggregation for Decentralized, Privacy-Preserving Learning

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

[1] Chalmers University of Technology, Gothenburg, Sweden
{brunetta, tsaloli}@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** We propose a novel primitive called NIVA that allows the distributed aggregation of multiple users' secret inputs by multiple untrusted servers. The returned aggregation result can be publicly verified in a non-interactive way, *i.e.* the users are not required to participate in the aggregation except for providing their secret inputs. NIVA allows the secure computation of the sum of a large amount of users' data and can be employed, for example, in the federated learning setting in order to aggregate the model updates for a deep neural network. We implement NIVA and evaluate its communication and execution performance and compare it with the current state-of-the-art, *i.e.* Segal *et al.* protocol (CCS 2017) and Xu *et al.* VerifyNet protocol (IEEE TIFS 2020), resulting in better user's communicated data and execution time.

**Keywords:** secure aggregation, privacy, verifiability, decentralization

## 1 Introduction

Smartphones, wearables and other Internet-of-Things (IoT) devices are all interconnected generating a lot of data, that often need to be aggregated to compute statistics in order to improve services. These improvements are often achieved by relying on *machine learning* (ML) algorithms, that simplify the prediction and/or inference of patterns from massive users' data. Given the high volume of data required, the ML paradigm creates serious privacy and security concerns [15,20] that require a careful security analysis in order to guarantee the minimization of private information leakage while, concurrently, allowing the aggregation of the collected users' data. The growing storage and computational power of mobile devices as well as the increased privacy concerns associated with

sharing private information, has led to a new distributed learning paradigm, *federated learning* (FL) [21]. FL allows multiple users to collaboratively train learning models under the orchestration of a central server, while providing strong privacy guarantees by keeping the users' data stored on the source, *i.e.* the user's devices. More precisely, the central server collects and aggregates the local parameters from multiple users' and uses the aggregated value in order to train a global training model. The server plays the role of a central trusted **aggregator** that facilitates the communication between multiple users and guarantees the correct execution of the model update which, often, in current FL frameworks, is obtained by **summing** the individual users' parameters.

The shared model must be kept confidential since it might be employed to infer secret user information or disrupt the correct model update, *e.g.* a malicious server might bias the final result according to its preferences [15,20,27,35,20,13]. Furthermore, when the aggregation process is orchestrated by a single central server, this may lead to single *points-of-failure*. Our aim is to maximise the distributed nature of the learning process by: *(i) decentralizing* the aggregation process between multiple servers; *(ii)* providing the ability to *verify* the correctness of the *computed aggregation*; and *(iii)* guaranteeing the *confidentiality* of the users' inputs. Fig. 1 depicts the described scenario.

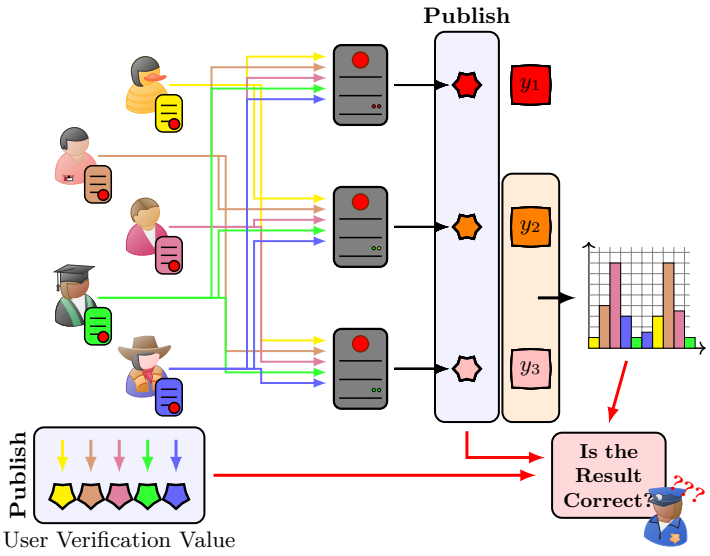



Figure 1: Several users delegate the secure aggregation of their inputs to independent servers. A threshold amount of server's outputs is necessary to publicly reconstruct and verify the resulting aggregated value.

### 1.1 Our Contributions

We define NIVA: a *Non-Interactive*, *decentralized* and publicly *Verifiable* secure *Aggregation* primitive inspired by the *verifiable homomorphic secret sharing* primitive introduced by Tsaloli *et al.* [33] **but** differs in both the construction and hypothesis. NIVA achieves decentralization by allowing the users to split their secret inputs and distribute the shares to multiple servers; while only a subset (*threshold*) of these servers need to collaborate in order to correctly reconstruct the output. Furthermore, NIVA allows the public verification of the computed aggregated value and contrary to existing work [25,34], NIVA is **non-interactive**, *i.e.* the users participate in the aggregation by releasing the appropriate messages and their participation is not required for the rest of the aggregation process. This allows NIVA to simplify the handling of *users' dropping out* from the aggregation process, which is a complex problem to handle in the case of interactive protocols. We further discuss possible optimizations to the verification algorithm as well as extensions useful for realistic applications, *e.g.* verification of users' shares, multiple executions and how to introduce a differentially private [10] mechanism. We implement NIVA, evaluate the communication costs, execution time, and perform a detailed experimental analysis. Furthermore, we compare our primitive with the current state-of-the-art, *i.e.* the secure aggregation protocols PPML and VerifyNet proposed by Segal *et al.* [25] and Xu *et al.* [34] correspondingly. NIVA optimizes the users' output and execution time making it multiple orders of magnitude more suitable than PPML and VerifyNet for the FL setting that requires a big amount of users, *i.e.* more than $10^5$ users.

### 1.2 Related Work

This work addresses a general problem that lies in the intersection of *"decentralized aggregation"* and *"verifiable delegation of computations"*.

*Secret Sharing.* A *threshold* secret sharing (SS) scheme allows a user to split a secret $x$ into multiple shares $(x_1, x_2, \ldots, x_m)$ that are distributed to different servers. Whenever at least a *threshold* number $t$ of servers collaborates by exchanging their shares, they are able to reconstruct the original secret. If any malicious adversary controls less than this threshold, it is impossible to reconstruct $x$. The first instantiation was provided by Shamir [26]. In the following decades, several publications [1,4,14,17] expanded Shamir's concept by providing schemes with additional properties such as *verification* and *homomorphism*.

An *additive homomorphic secret sharing* (additive HSS) allows the server to aggregate several shares coming from different users into a single one which, when correctly reconstructed, will allow the reconstruction of the *sum* of the original secrets. Besides Shamir's, the first instance of such a scheme was proposed by Benealoh [2] and many other variations can be found in the literature [3,11,18].

Generally, the *verifiability* property describes the possibility to verify that some specific value is *"correctly evaluated"*. Whenever considering this property in the context of SS, it must be specified if *(a)* the server wants to verify the

user's received shares; or *(b)* anyone wants to check if the servers' reconstructed secret is indeed the correct one. Chor *et al.* [8] provided the first SS scheme that is able to identify the existence of a *"cheating"* user, while Stadler [28] extended it in order to detect both cheating users and servers. Tsaloli *et al.* [32] proposed a *verifiable homomorphic secret sharing* (VHSS) scheme in which the *verifiability* property holds by *assuming the user's honesty* in generating the shares **but** allows the verification of the server's aggregation correctness. In this paper, we consider the properties of *verifiability* and *homomorphic secret sharing* as considered by Tsaloli *et al.* [32,31]. Our primitive NIVA is *inspired* by Tsaloli *et al.*'s primitive [33], *however*, it is based on a completely different construction.

*Federated Learning and Cryptography.* The setting posed by federated learning (FL) is similar to the aggregation problems we consider. Concretely, every time the FL model must be updated, the users send their parameters to the server that must provide the final aggregated model back. The work in Bonawitz *et al.* [25] proposes a secure aggregation protocol, called PPML, that achieves security and privacy with a major focus on maintaining high efficiency. This solution provides a procedure to correctly handle *users' drop-outs*, *i.e.* users that are unable to correctly terminate the protocol. In the same spirit, Xu *et al.* [34] introduced VerifyNet, an (conceptually) extended version of PPML that introduces a *public verification* procedure to check the correctness of the aggregation process. However, these solutions are based on a single central server, and they are therefore susceptible to *single points-of-failure*, *i.e.* if the central server crashes, the whole protocol aborts. To avoid this, it is required to *distribute/decentralise* the role of the central server, *e.g.* by either introducing *threshold* cryptographic primitives between multiple aggregators [29] or by completely decentralising the aggregation using a blockchain [5]. Recently, privacy-preserving aggregation problems have gained substantial attention in the past few years [30,13,34,5,29,27,23,25]. The solutions presented achieve different properties related to security, privacy, and verifiability by considering specific cryptographic assumptions, security models, and/or application requirements. Our primitive allows to *publicly verify* the correctness of the final output, handles the users' drop-outs **as well as** possible servers' failure by distributing the aggregation computation among several *independent* servers.

### 1.3 Paper Organisation

Sec. 2 contains the necessary preliminaries used throughout the paper. Sec. 3 introduces our primitive NIVA, its security and verifiability properties, further discusses additional properties and compares to the related work. Sec. 4 describes NIVA's implementation details and showcases relevant performance statistics, *e.g.* execution timing and bandwidth usage in relation to scaling the amount of users and servers. Furthermore, we compare our implementation with Segal *et al.* [25] and Xu *et al.* [34] for similar evaluation parameters.

## 2 Preliminaries

In this section, we show the definitions used throughout the paper.

Denote with $\mathsf{Pr}[E]$ the probability that the event $E$ occurs. Let the natural number be denoted by $\mathbb{N}$, the integer number ring with $\mathbb{Z}$, the real number field with $\mathbb{R}$ and the positive ones with $\mathbb{R}_+$. Let $[a, b]$ denote intervals between $a$ and $b$. Let $|X| \in \mathbb{N}$ indicate the cardinality of the set $X$ and $\mathsf{rk}(A)$ the rank of the matrix $A$. Let $\sum_{x\in X}^{y\in Y}$ be the sum $\sum_{x\in X, y\in Y}$, respectively $\prod_{x\in X}^{y\in Y}$ is $\prod_{x\in X, y\in Y}$.

**Theorem 1 (Rouché-Capelli [6]).** *An $n$-variable linear equation system $Ax = b$ has a solution $\Leftrightarrow$ $\mathsf{rk}(A) = \mathsf{rk}(A|b)$ where $A|b$ is the augmented matrix,* i.e. *$A$ with appended the column $b$.*

***Key Agreement.*** Let $\mathbb{G}$ be a cyclic group of order $p$ prime with generator $\mathsf{g}$, *e.g.* groups based on elliptic curves [16]. Let us report the Diffie-Hellman key agreement [9] and the related assumptions.

**Assumption 1 (Diffie-Hellman Assumptions)** *Consider a cyclic group $\mathbb{G}$ of prime order $p$ with generator $\mathsf{g}$ and $a, b \in [0, p-1]$. Given elements $(A, B) = (\mathsf{g}^a, \mathsf{g}^b)$, the **computational Diffie-Hellman problem (CDH)** requires to compute the element $\mathsf{g}^{ab} \in \mathbb{G}$. The **decisional Diffie-Hellman problem (DDH)** requires to correctly distinguish between $(\mathsf{g}, A, B, \mathsf{g}^{ab})$ and $(\mathsf{g}, A, B, \mathsf{g}^c)$ 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).** *The Diffie-Hellman key agreement scheme is defined with the following algorithms:*

- *$\mathsf{KSetup}(\lambda) \to pp$: the setup algorithm takes as input the security parameter and outputs the public parameters $\mathsf{pp}$ which contains a prime $p$, the description of a cyclic group $\mathbb{G}$ of order $p$ and a generator $\mathsf{g}$ for the group $\mathbb{G}$.*
- *$\mathsf{KGen}(pp) \to (\mathsf{sk}, \mathsf{pk})$: the key generation algorithm samples the secret key $\mathsf{sk} \in [0, p-1]$ and computes the public key $\mathsf{pk} = \mathsf{g}^{\mathsf{sk}}$. It outputs $(\mathsf{sk}, \mathsf{pk}) = (\mathsf{sk}, \mathsf{g}^{\mathsf{sk}})$.*
- *$\mathsf{KAgree}(\mathsf{sk}_i, \mathsf{pk}_j) \to \mathsf{s}_{ij}$: the key agreement algorithm takes as input a secret $\mathsf{sk}_i$ and public key $\mathsf{pk}_j = \mathsf{g}^{\mathsf{sk}_j}$ and outputs the shared secret $\mathsf{s}_{ij} = \mathsf{pk}_j^{\mathsf{sk}_i} = \mathsf{g}^{\mathsf{sk}_j \cdot \mathsf{sk}_i}$.*

*The scheme is said to be **correct** if for any $pp \leftarrow \mathsf{KSetup}(\lambda)$, $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KGen}(pp)$ and $(\mathsf{sk}_j, \mathsf{pk}_j) \leftarrow \mathsf{KGen}(pp)$, it holds that $\mathsf{KAgree}(\mathsf{sk}_i, \mathsf{pk}_j) = \mathsf{s}_{ij} = \mathsf{s}_{ji} = \mathsf{KAgree}(\mathsf{sk}_j, \mathsf{pk}_i)$. The scheme is said to be **secure** if for any $pp \leftarrow \mathsf{KSetup}(\lambda)$, and keys $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KGen}(\mathsf{pp}, U_i)$, $(\mathsf{sk}_j, \mathsf{pk}_j) \leftarrow \mathsf{KGen}(\mathsf{pp}, U_j)$, it holds that any PPT adversary $\mathcal{A}$ has negligible probability to compute $\mathsf{s}_{ij}$ from $(\mathsf{pk}_i, \mathsf{pk}_j)$ which reduces to the CDH Assumption 1.*

For our primitive, we use the shared secret $\mathsf{s}_{ij}$ as a pseudorandom integer despite being an element of the group $\mathbb{G}$. This is possible by considering a generic hash function $\mathsf{H}$ mapping the group $\mathbb{G}$ to the integers $\mathbb{Z}$, which translates $\mathsf{s}_{ij}$ into a *pseudorandom* integer. To avoid heavy notation, we denote this output as $\mathsf{s}_{ij}$.

Additionally, consider the *discrete logarithm problem* for a subset $I$, *i.e.* the $\mathsf{dLog}$ problem where the solution is contained in a subset $I \subseteq [0, p-1]$.

**Assumption 2 (Discrete Logarithm in Subset $I$ Problem)** *Consider $\mathbb{G}$ a cyclic group of prime order $p$ with generator $g$ and a subset $I \subseteq [0, p-1]$. Given $y \in \mathbb{G}$, the **discrete logarithm problem for the subset $I$** ($\mathsf{dLog}_I$) requires to find the value $x \in I$ such that $g^x = y$.*

In order to assume the $\mathsf{dLog}_I$ problem to be computationally hard, the cardinality of $I$ needs to be *"big enough",i.e.* if $|I| > 2^{160}$ then the kangaroo Pollard's rho algorithm [24] has complexity $\sim 2^{80}$ which we consider to be infeasible.

***Secret Sharing.*** We report the additive homomorphic SS scheme's definition.

**Definition 2 (Additive Homomorphic SS Scheme).** *Let $n, m, t \in \mathbb{N}$ such that $0 < t < m$. For each $i \in [1, n]$, let $x_i \in \mathbb{F}$ be the secret input of the user $\mathsf{U}_i$ for some input space $\mathbb{F}$. Consider the set of servers $\mathcal{S} = \{\mathsf{S}_j\}_{j\in[1,m]}$. Define $(t, m)$-**threshold additive homomorphic secret sharing** scheme as:*

- $\mathsf{SS.Share}(x_i, t, \mathcal{S}) \rightarrow \{x_{ij}\}_{j\in[1,m]}$ *: given the secret input $x_i$, the threshold value $t$ and the list of servers $\mathcal{S}$, the share generation algorithm outputs a list of $m$ shares $x_{ij}$ for $j \in [1, m]$, one for each server $\mathsf{S}_j$.*
- $\mathsf{SS.Eval}(\{x_{ij}\}_{i\in[1,n]}) \rightarrow y_j$ *: given as input a set of shares $x_{ij}$ for the same server $\mathsf{S}_j$, the evaluation algorithm outputs an aggregated share $y_j$.*
- $\mathsf{SS.Recon}(t, \{y_j\}_{j\in\mathcal{T}}) \rightarrow y$ *: given as input the threshold value $t$ and a list of shares $y_j$ for a subset of servers $\mathsf{S}_j \in \mathcal{T} \subseteq \mathcal{S}$ such that $|\mathcal{T}| > t$, the reconstruction algorithm outputs the reconstructed secret $y$.*

*A $(t, m)$ additive homomorphic secret sharing scheme is said to be **correct** if for all $i \in [1, n]$, any choice of secrets $x_i \in \mathbb{F}$, for all the shares $\mathsf{SS.Share}(x_i, t, \mathcal{S}) \rightarrow \{x_{ij}\}_{j\in[1,m]}$, aggregated shares $\mathsf{SS.Eval}(\{x_{ij}\}_{i\in[1,n]}) \rightarrow y_j$, for all the servers' reconstruction subset $\mathcal{T}$ such that $|\mathcal{T}| > t$, it holds that the reconstructed value $\mathsf{SS.Recon}(t, \{y_j\}_{j\in\mathcal{T}}) \rightarrow y$ is equal to $y = \sum_{i=1}^{n} x_i$.*

*A $(t, m)$ additive homomorphic secret sharing scheme is **secure** if for all $i \in [1, n]$, any secrets $x_i \in \mathbb{F}$, for all the shares $\mathsf{SS.Share}(x_i, t, \mathcal{S}) \rightarrow \{x_{ij}\}_{j\in[1,m]}$, aggregated shares $\mathsf{SS.Eval}(\{x_{ij}\}_{i\in[1,n]}) \rightarrow y_j$, an adversary $\mathcal{A}$ that controls a servers' subset $\mathcal{T} \subseteq \mathcal{S}$, such that $|\mathcal{T}| \leq t$, is unable to obtain the reconstructed value $y$.*

## 3 NIVA

In this section, we describe the decentralised aggregation problem's setting as well as the security and privacy requirements and how they must guarantee public verifiability of the aggregated computations. We instantiate NIVA and define the security and verifiability properties.

Consider $n$ users $\mathsf{U}_i$, each owns a secret input $x_i$, and $m$ servers $\mathsf{S}_j$. The goal is to *distribute the computation* of the sum of the users' secret inputs' $\sum_{i=1}^{n} x_i$ between the $m$ servers of which only a *designed threshold* amount $t + 1 \leq m$ of servers is required to obtain the aggregated value. Formally:

**Definition 3.** *Let the algorithms $(\mathsf{Setup}, \mathsf{SGen}, \mathsf{Agg}, \mathsf{Ver})$ defined as:*

- $\mathsf{Setup}(\lambda) \rightarrow (\mathsf{sk}_I, \mathsf{pk}_I)$ *: given the security parameter* $\lambda$*, the setup algorithm provides a keypair* $(\mathsf{sk}_I, \mathsf{pk}_I)$ *associated to the user/server* $I$*.*
- $\mathsf{SGen}\big(x_i, \mathsf{sk}_{U_i}, t, \{\mathsf{pk}_{S_j}\}_{j=1}^m\big) \rightarrow \big(\mathsf{pk}_{U_i}, \{\widehat{x_{ij}}\}_{j=1}^m, R_i, \{\tau_{ij}\}_{j=1}^m\big)$ *: given a secret input* $x_i \in I$ *and the user's* $\mathsf{U}_i$ *secret key* $\mathsf{sk}_{U_i}$*, the designed threshold amount* $0 < t < m-1$ *and the list of servers' public keys* $\{\mathsf{pk}_{S_j}\}_{j=1}^m$ *from which we obtain the list of servers' identities* $\{\mathsf{S}_j\}_{j=1}^m$*, the share generation algorithm outputs the shares* $\widehat{x_{ij}}$*, additional information* $R_i$ *and the verification coefficients* $\tau_{ij}$ *to be either shared with the server* $\mathsf{S}_j$ *or publicly released.*
- $\mathsf{Agg}\big(\{(\mathsf{pk}_{U_i}, \widehat{x_{ij}}, R_i)\}_{i \in N}, \mathsf{sk}_{S_j}\big) \rightarrow (y_j, \pi_j, R_{S_j}, \rho_j)$ *: given a set of public keys, shares and additional information* $(\widehat{x_{ij}}, R_i, \mathsf{pk}_i)$ *for a list of users* $\mathsf{U}_i$ *in the subset* $N \subseteq [1, n]$*, the aggregation algorithm outputs the partial evaluation* $y_j$*, a partial verification proof* $\pi_j$ *and additional information* $(R_{S_j}, \rho_j)$*.*
- $\mathsf{Ver}\Big(t, \{\tau_{ij}\}_{i \in N}^{j \in M}, \big\{(y_j, \pi_j, R_{S_j}, \rho_j)\big\}_{j \in M}\Big) \rightarrow \{y, \perp\}$ *: given the threshold* $t$*, a set of servers* $M$ *with* $t{+}1 \leq |M| \leq m$*, given partial evaluations, proofs and additional information* $(y_j, \pi_j, R_{S_j}, \rho_j)$ *and a set of verification coefficients* $\{\tau_{ij}\}_{i \in N}^{j \in M}$ *for a subset of users* $N$*, the verification algorithm outputs the aggregated value* $y = \sum_{i \in N} x_i$ *if the servers correctly computed the aggregation of their shares. Otherwise, it outputs* $\perp$*.*

The primitive must be **correct**, *i.e.* the verification always outputs $y = \sum_{i \in N} x_i$ whenever using correctly aggregated outputs computed from correctly generated shares of the secrets $\{x_i\}_{i \in N}$. Additionally, the users' input must be **secure**. The security experiment describes a realistic scenario in which the adversary $\mathcal{A}$ must recover the secret inputs $x_i$, which are randomly sampled by the challenger $\mathcal{C}$. The amount of servers that $\mathcal{A}$ is able to compromise is at most $t$ since this servers' subset is not enough for using the secret share's reconstruction algorithm $\mathsf{SS.Recon}$. Our experiment includes the *single-user input privacy*, *i.e.* whenever $\mathcal{A}$ requests a challenge for $n = 1$, the property holds for the input $x_i$.

**Definition 4 (Security).** *Consider the primitive of Def. 3 to be defined between* $n$ *users and* $m$ *servers and threshold* $t$*. Let* $\mathcal{A}$ *be a PPT adversary that maliciously controls* $t$ *servers,* w.l.o.g. $\{\mathsf{S}_j\}_{j=1}^t$*. Consider the security experiment* $\mathsf{Exp}^{\mathsf{sec}}(\mathcal{A})$*:*

1. *For every* $j \in [1, t]$*, the challenger* $\mathcal{C}$ *executes* $\mathsf{Setup}(\lambda)$ *and sends to* $\mathcal{A}$ *all the corrupted servers' key-pairs* $(\mathsf{sk}_{S_j}, \mathsf{pk}_{S_j})$*, while for the remaining* $j \in [t{+}1, m]$ *servers, it returns only the non-corrupted server's public key* $\mathsf{pk}_{S_j}$*.*
2. $\mathcal{A}$ *outputs to* $\mathcal{C}$ *the number of users* $n$ *to be challenged on.*
3. $\mathcal{C}$ *executes* $\mathsf{Setup}(\lambda)$ *and generates the key pairs* $(\mathsf{sk}_{U_i}, \mathsf{pk}_{U_i})$ *and randomly samples an input* $x_i \in I$ *for each user* $\mathsf{U}_i$*.*
4. $\mathcal{C}$ *computes the shares* $\mathsf{SGen}\big(x_i, \mathsf{sk}_{U_i}, t, \{\mathsf{pk}_{S_j}\}_{j=1}^m\big)$ *and outputs to* $\mathcal{A}$ *the compromised servers' shares* $\big(\mathsf{pk}_{U_i}, \{\widehat{x_{ij}}\}_{j=1}^t, R_i\big)$ *plus all the verification values* $\{\tau_{ij}\}_{j=1}^m$ *for each* $i \in [1, n]$*.*
5. $\mathcal{A}$ *outputs the aggregated secret* $y^*$*.*
6. *If* $y^* = \sum_{i=1}^n x_i$*, the experiment outputs 1, otherwise 0.*

*The primitive is said to be* ***secure*** *if* $\mathsf{Pr}[\mathsf{Exp}^{\mathsf{sec}}(\mathcal{A}) = 1] < \mathsf{negl}$*.*

Finally, we require to **publicly verify** the computations of the servers, *i.e.* the servers must provide a proof of the correct computation. In other words, the verifiability property requires the impossibility for an adversary $\mathcal{A}$ to force the correct verification of a wrong aggregated value. This property holds whenever there exists at least one honestly computed partial evaluation, regardless of the number of servers that $\mathcal{A}$ compromises. On the other hand, whenever $\mathcal{A}$ controls more than $t$ servers, the security property does not hold, thus obtaining a potentially verifiable primitive but definitely not secure. For this reason, we design the verifiability experiment in which, before obtaining the correct partial evaluations, $\mathcal{A}$ is allowed to select the subset of inputs $N^*$ to be aggregated and, after receiving the non-compromised partial evaluations, $\mathcal{A}$ outputs tampered partial evaluations for the compromised servers and selects a set $M^*$ of evaluations to be used in the verification challenge. The adversarial set $M^*$ must contain at least one honestly generated partial evaluation and it is used to describe the realistic attack scenario in which the adversary denies the verifier to obtain all the partial evaluations **but** at least an honest one is present.

**Definition 5 (Verifiability).** *Consider the primitive of Def. 3 to be defined between $n$ users, $m$ servers and threshold $t$. Let $\mathcal{A}$ be a PPT adversary that maliciously controls $k < m$ servers,* w.l.o.g. $\{\mathsf{S}_j\}_{j=1}^{k}$. *Consider the experiment* $\mathsf{Exp}^{\mathsf{ver}}(\mathcal{A})$*:*

1. *For every $j \in [1,k]$, the challenger $\mathcal{C}$ executes* $\mathsf{Setup}(\lambda)$ *and sends to $\mathcal{A}$ all the corrupted servers' key-pairs* $(\mathsf{sk}_{\mathsf{S}_j}, \mathsf{pk}_{\mathsf{S}_j})$*, while for the remaining $j \in [k+1,m]$ servers it returns only the non-corrupted server's public key* $\mathsf{pk}_{\mathsf{S}_j}$.
2. *$\mathcal{A}$ outputs to $\mathcal{C}$ the number of users $n$ to be challenged on.*
3. *$\mathcal{C}$ executes* $\mathsf{Setup}(\lambda)$ *and generates the key pairs* $(\mathsf{sk}_{\mathsf{U}_i}, \mathsf{pk}_{\mathsf{U}_i})$ *and randomly samples an input $x_i \in I$ for each user* $\mathsf{U}_i$.
4. *$\mathcal{C}$ computes the shares* $\mathsf{SGen}\big(x_i, \mathsf{sk}_{\mathsf{U}_i}, t, \{\mathsf{pk}_{\mathsf{S}_j}\}_{j=1}^{m}\big)$ *and outputs to $\mathcal{A}$ the compromised servers' shares* $\big(\mathsf{pk}_{\mathsf{U}_i}, \{\widehat{x_{ij}}\}_{j=1}^{k}, R_i\big)$ *plus all the verification values* $\{\tau_{ij}\}_{j=1}^{m}$ *for each $i \in [1,n]$.*
5. *$\mathcal{A}$ provides to $\mathcal{C}$ the list of inputs $N^*$ to be challenged.*
6. *For each non compromised server $\mathsf{S}_j$ where $j \in [k+1,m]$, $\mathcal{C}$ returns to $\mathcal{A}$ the $\mathsf{S}_j$'s partial evaluations* $(y_j, \pi_j, R_{\mathsf{S}_j}, \rho_j) \leftarrow \mathsf{Agg}\big(\{(\mathsf{pk}_{\mathsf{U}_i}, \widehat{x_{ij}}, R_i)\}_{i\in N^*}, \mathsf{sk}_{\mathsf{S}_j}\big)$.
7. *$\mathcal{A}$ outputs tampered evaluations* $\{{y_j}^*, {\pi_j}^*, {R_{\mathsf{S}_j}}^*, {\rho_j}^*\}_{j=1}^{k}$.
8. *$\mathcal{A}$ provides to $\mathcal{C}$ the list of verifying servers $M^*$ in which there exists a non-compromised server $\mathsf{S}_l \in M^*$ with $l \in [k+1,m]$.*
9. *The experiment computes the verification algorithm*

$$\mathsf{Ver}\Big(t, \{\tau_{ij}\}_{i\in N^*}^{j\in M^*}, \big\{(y_j, \pi_j, R_{\mathsf{S}_j}, \rho_j)\big\}_{j\in M^*}\Big) \rightarrow y^*$$

*and outputs 1 if $y^* \neq y = \sum_{i\in N^*} x_i$, otherwise 0.*

*The primitive is said to be* ***verifiable*** *if* $Pr[\mathsf{Exp}^{\mathsf{ver}}(\mathcal{A}) = 1] < \mathsf{negl}$.

### 3.1 NIVA Instantiation

In this section, we provide our instantiation of Def. 3, called NIVA. In a nutshell, NIVA incorporates into the Shamir's SS scheme of Sec. 2, the usage of a key-agreement scheme between the users and the servers. This allows the creation of a *"proving value"* used during the verification phase which **must** be correctly computed by the servers or, otherwise, the verification process fails.

**Definition 6 (NIVA).** *Let* $(\mathsf{KSetup}, \mathsf{KGen}, \mathsf{KAgree})$ *be a key agreement (Def. 1) with public parameters* $\mathsf{pp} \leftarrow \mathsf{KSetup}(\lambda)$*, defined over a cyclic group* $\mathbb{G}$ *with prime order* $p$*. Let* $n \in \mathbb{N}$ *be the number of users* $\mathsf{U}_i$ *and* $m \in \mathbb{N}$ *be the number of servers* $\mathsf{S}_j$*. Let* $I$ *be a secret input's space closed under summation such that the* $\mathsf{dLog}_I$ *problem of Assumption 2 is hard. Let* $N \subseteq [1,n]$ *be a users' subset and* $M \subseteq [1,m]$ *a servers' subset. We refer to* $\mathsf{S}_j \in M$ *with* $j \in M$*. Let* $t \in \mathbb{N}$ *be the evaluation threshold such that* $0 < t < m$*. Define* NIVA *with algorithms:*

- $\mathsf{Setup}(\lambda) \rightarrow (\mathsf{sk}_I, \mathsf{pk}_I)$ *: given the security parameter* $\lambda$*, the setup algorithm executes* $\mathsf{KGen}(\mathsf{pp})$ *and outputs the result* $(\mathsf{sk}_I, \mathsf{pk}_I) = \big(\mathsf{sk}_I, g^{\mathsf{sk}_I}\big)$*. The* Setup *algorithm is evaluated by each user* $\mathsf{U}_i$ *and server* $\mathsf{S}_j$*. All the public keys of the servers* $\{\mathsf{pk}_{\mathsf{S}_j}\}_{j=1}^m$ *are publicly released.*
- $\mathsf{SGen}\big(x_i, \mathsf{sk}_{\mathsf{U}_i}, t, \{\mathsf{pk}_{\mathsf{S}_j}\}_{j=1}^m\big) \rightarrow \big(\mathsf{pk}_{\mathsf{U}_i}, \{\widehat{x_{ij}}\}_{j=1}^m, R_i, \{\tau_{ij}\}_{j=1}^m\big)$ *: given a secret input* $x_i \in I$ *and the user's* $\mathsf{U}_i$ *secret key* $\mathsf{sk}_{\mathsf{U}_i}$*, the designed threshold amount* $0 < t < m-1$*, the list of servers' public keys* $\{\mathsf{pk}_{\mathsf{S}_j}\}_{j=1}^m$ *from which we obtain the list of servers' identities* $\{\mathsf{S}_j\}_{j=1}^m$*, the share generation algorithm instantiates a* $(t,m)$*-threshold additive homomorphic secret sharing scheme by executing* $\mathsf{SS.Share}\big(x_i, t, \{\mathsf{S}_j\}_{j=1}^m\big)$ *which returns the shares* $\widehat{x_{ij}}$ *for all* $j \in [1,m]$*. Then,* $\mathsf{U}_i$ *uses its secret key* $\mathsf{sk}_{\mathsf{U}_i}$ *to compute the shared secrets* w.r.t. *each server* $\mathsf{S}_j$*,* i.e. $\mathsf{KAgree}(\mathsf{sk}_{\mathsf{U}_i}, \mathsf{pk}_{\mathsf{S}_j}) \rightarrow s_{ij}$*. The algorithm samples a random value* $r_i \in [0, p-1]$*, computes* $R_i = g^{r_i}$*, and the verification coefficients*

$$\tau_{ij} = \mathsf{pk}_{\mathsf{S}_j}^{x_i} \cdot R_i^{s_{ij}} = g^{\mathsf{sk}_{\mathsf{S}_j} x_i + r_i \cdot s_{ij}} \tag{1}$$

  *The algorithm outputs* $\big(\mathsf{pk}_{\mathsf{U}_i}, \{\widehat{x_{ij}}\}_{j=1}^m, R_i, \{\tau_{ij}\}_{j=1}^m\big)$*. Each user publicly releases the values* $\{\tau_{ij}\}_{j=1}^m$*.*
- $\mathsf{Agg}\big(\{(\mathsf{pk}_{\mathsf{U}_i}, \widehat{x_{ij}}, R_i)\}_{i \in N}, \mathsf{sk}_{\mathsf{S}_j}\big) \rightarrow (y_j, \pi_j, R_{\mathsf{S}_j}, \rho_j)$ *: given a set of public keys, shares and random values* $(\widehat{x_{ij}}, R_i, \mathsf{pk}_i)$ *for a list of users* $\mathsf{U}_i$ *in the subset* $N \subseteq [1,n]$*, the aggregation algorithm performs all the key-agreements between* $\mathsf{U}_i$ *and* $\mathsf{S}_j$ *as* $\mathsf{KAgree}(\mathsf{sk}_{\mathsf{S}_j}, \mathsf{pk}_{\mathsf{U}_i}) \rightarrow s_{ij}$*, the partial evaluation and proofs as:*

$$\begin{aligned} y_j &\leftarrow \mathsf{SS.Eval}\big(\{\widehat{x_{ij}}\}_{i \in N}\big) & \pi_j &= \sum_{i \in N} s_{ij} \\ R_{\mathsf{S}_j} &= \prod_{i \in N} R_i & \rho_j &= \prod_{i \in N} R_i^{-\sum_{k \in N}^{k \neq i} s_{kj}} \end{aligned} \tag{2}$$

  *The algorithm outputs* $(y_j, \pi_j, R_{\mathsf{S}_j}, \rho_j)$*.*
- $\mathsf{Ver}\Big(t, \{\tau_{ij}\}_{i \in N}^{j \in M}, \big\{(y_j, \pi_j, R_{\mathsf{S}_j}, \rho_j)\big\}_{j \in M}\Big) \rightarrow \{y, \perp\}$ *: given the threshold* $t$*, a set of servers* $M$ *with* $t+1 \leq |M| \leq m$*, given partial evaluations and proofs*

$(y_j, \pi_j, R_{\mathsf{S}_j}, \rho_j)$ *and a set of verification coefficients* $\{\tau_{ij}\}_{i\in N}^{j\in M}$ *for a subset of users* $N$*, the verification algorithm verifies that for any* $\mathsf{S}_j, \mathsf{S}_j{}' \in M$*, it holds* $R_{\mathsf{S}_j} = R_{\mathsf{S}_j{}'} = R$*. If not,* Ver *outputs* $\perp$*. Otherwise, the algorithm verifies that for* ***all*** *the subsets* $T_i \subseteq M$ *of* $t{+}1$ *partial evaluations, the reconstruction algorithm* $\mathsf{SS.Recon}\big(t, \{y_j\}_{j\in T_i}\big)$ *returns always the same output* $y$*. If not,* Ver *outputs* $\perp$*. Otherwise, the algorithm computes*

$$\prod_{j\in M_l}^{i\in N} \tau_{ij} \stackrel{?}{=} \left(\prod_{j\in M_l} \mathsf{pk}_{\mathsf{S}_j}\right)^{y} \cdot \prod_{j\in M_l} R^{\pi_j} \cdot \rho_j \tag{3}$$

*for all the* $|M|$ *subsets* $M_l \subset M$ *such that* $|M_l| = |M| - 1$*. If any check fails, then the verification algorithm outputs* $\perp$*. Otherwise, the verification algorithm outputs* $y$*.*

**Corollary 1.** *NIVA allows the definition of the algorithm:*

– $\mathsf{OptVer}\Big(t, \{\tau_{ij}\}_{i\in N}^{j\in M}, \big\{(y_j, \pi_j, R_{\mathsf{S}_j}, \rho_j)\big\}_{j\in M}\Big) \to \{y, \perp\}$ *: given the threshold* $t$*, a set of servers* $M$ *with* $t{+}1 \leq |M| \leq m$*, given partial evaluations and proofs* $(y_j, \pi_j, R_{\mathsf{S}_j}, \rho_j)$ *and a set of verification coefficients* $\{\tau_{ij}\}_{i\in N}^{j\in M}$ *for a subset of users* $N$*, the verification algorithm verifies that for any* $\mathsf{S}_j, \mathsf{S}_j{}' \in M$*, it holds* $R_{\mathsf{S}_j} = R_{\mathsf{S}_j{}'} = R$*. If not,* Ver *outputs* $\perp$*. Otherwise, the algorithm verifies that for* ***all*** *the subsets* $T_i \subseteq M$ *of* $t{+}1$ *partial evaluations, the reconstruction algorithm* $\mathsf{SS.Recon}\big(t, \{y_j\}_{j\in T_i}\big)$ *returns always the same output* $y$*. If not, the algorithm outputs* $\perp$*. Otherwise, the algorithm computes, for each* $\mathsf{S}_l \in M$*:*

$$\prod_{i\in N} \tau_{ij} \stackrel{?}{=} \big(\mathsf{pk}_{\mathsf{S}_l}\big)^{y} \cdot R^{\pi_l} \cdot \rho_l$$

*If any check fails, then the algorithm outputs* $\perp$*. Otherwise, it outputs* $y$*.*

*Remark 1.* The main difference *w.r.t.* Ver is that OptVer takes the $|M|$ different subsets $M_l$ to be defined as servers' singletons, *i.e.* $M_l = \{\mathsf{S}_l\}$ and $|M_l| = 1$. This reduces the amount of computation needed to verify Eq. (3). The possibility of using OptVer might depend on *application constraints*, *e.g.* the server Agg's outputs might not be directly published but further aggregated by a third party before reaching the final public verification.

NIVA's is **correct** for both the verification algorithms Ver and OptVer.

*Proof (NIVA Correctness).* For any list of key pairs $\mathsf{Setup}(\lambda) \to (\mathsf{sk}_I, \mathsf{pk}_I)$, for any party $I$ being a user $\mathsf{U}_i$ or server $\mathsf{S}_j$ for $i \in [1, n], j \in [1, m]$, for any user choice of secret inputs $x_i \in [0, p{-}1]$, for all computed shares $\mathsf{SGen}\big(x_i, \mathsf{sk}_{\mathsf{U}_i}, t, \{\mathsf{pk}_{\mathsf{S}_j}\}_{j=1}^{m}\big) \to \big(\mathsf{pk}_{\mathsf{U}_i}, \{\widehat{x_{ij}}\}_{j=1}^{m}, R_i, \{\tau_{ij}\}_{j=1}^{m}\big)$, and for all the aggregated values $(y_j, \pi_j, R_{\mathsf{S}_j}, \rho_j)$ computed as $\mathsf{Agg}\big(\{(\mathsf{pk}_{\mathsf{U}_i}, \widehat{x_{ij}}, R_i)\}_{i\in N}, \mathsf{sk}_{\mathsf{S}_j}\big)$, for any subset of users $N$, for any servers' subset $M \subseteq \mathcal{S}$ such that $|M| \geq t + 1$, the verification algorithm, *i.e.* $\mathsf{Ver}\Big(t, \{\tau_{ij}\}_{i\in N}^{j\in M}, \big\{(y_j, \pi_j, R_{\mathsf{S}_j}, \rho_j)\big\}_{j\in M}\Big)$, finds that for any $\mathsf{S}_j, \mathsf{S}_j{}' \in M$, and for

any subset $T_i$ of $t+1$ partial evaluations, $\mathsf{SS.Recon}\big(t, \{y_j\}_{j\in T_i}\big)$ always returns the same $y$ from the correctness of the secret sharing scheme.

Finally, consider Eq. (1), the verification algorithm correctly verifies

$$\begin{aligned}
\prod_{j\in M_l}^{i\in N} \tau_{ij} &= \prod_{j\in M_l}^{i\in N} \mathsf{pk}_{\mathsf{S}_j}^{x_i} R_i^{\mathsf{s}_{ij}} = \left(\prod_{j\in M_l} \mathsf{pk}_{\mathsf{S}_j}\right)^{\sum_{i\in N} x_i} \prod_{i\in N} R_i^{\sum_{j\in M_l} \mathsf{s}_{ij}} \\
&= \left(\prod_{j\in M_l} \mathsf{pk}_{\mathsf{S}_j}\right)^{y} \prod_{i\in N} R_i^{\sum_{k=i}^{j\in M_l} \mathsf{s}_{kj} + \sum_{k\in N, k\neq i}^{j\in M_l} \mathsf{s}_{kj} - \sum_{k\in N, k\neq i}^{j\in M_l} \mathsf{s}_{kj}} \\
&= \left(\prod_{j\in M_l} \mathsf{pk}_{\mathsf{S}_j}\right)^{y} \cdot \prod_{i\in N} R_i^{\sum_{i\in N}^{j\in M_l} \mathsf{s}_{ij}} \left(\prod_{i\in N} R_i\right)^{-\sum_{k\in N, k\neq i}^{j\in M_l} \mathsf{s}_{kj}} \qquad (4) \\
&= \left(\prod_{j\in M_l} \mathsf{pk}_{\mathsf{S}_j}\right)^{y} \cdot \prod_{j\in M_l} \left(\prod_{i\in N} R_i\right)^{\sum_{i\in N} \mathsf{s}_{ij}} \prod_{j\in M_l} \rho_j = \left(\prod_{j\in M_l} \mathsf{pk}_{\mathsf{S}_j}\right)^{y} \cdot \prod_{j\in M_l} R^{\pi_j} \rho_j
\end{aligned}$$

for each subset $M_l \subset M$ with $|M_l| = |M| - 1$. The verification algorithm outputs $y$, thus proving the correctness of the scheme.

Trivially, the same is true whenever considering the subsets $M_l = \{\mathsf{S}_l\}$, *i.e.* the verification executed by the $\mathsf{OptVer}$ algorithm. □

**Theorem 2 (NIVA Security).** *If we assume the negligible probability $\epsilon_{\mathsf{dLog}_I}$ of solving the $\mathsf{dLog}_I$ problem for the input subset $I$ and the additive homomorphic secret sharing scheme's security, then NIVA is **secure** (Def. 4).*

*Proof.* Let $\mathsf{dLog}_I$ be a hard problem and assume the existence of an adversary $\mathcal{A}$ able to break the $\mathsf{Exp}^{\mathsf{sec}}_{\mathsf{NIVA}}(\mathcal{A})$ experiment of Def. 4.

The reduction $\mathcal{R}$ receives a $\mathsf{dLog}_I$ challenge $Z = g^z$, creates the $m$ servers' key-pairs $(\mathsf{sk}_{\mathsf{S}_j}, \mathsf{pk}_{\mathsf{S}_j})$ and provides the corrupted to $\mathcal{A}$. $\mathcal{A}$ replies with the amount of challenged user $n$. The reduction therefore obtains the $n$ users' key-pairs $(\mathsf{sk}_{\mathsf{U}_i}, \mathsf{pk}_{\mathsf{U}_i})$ and samples $n-1$ secret inputs $\{x_i\}_{i=2}^n$ such that $\sum_{i=2}^n x_i = 0$.

Of these secrets, the reduction correctly computes the shares, for $i \in [2, n]$, as $\big(\mathsf{pk}_{\mathsf{U}_i}, \{\widehat{x_{ij}}\}_{j=1}^t, R_i\big) \leftarrow \mathsf{SGen}\big(x_i, \mathsf{sk}_{\mathsf{U}_i}, t, \{\mathsf{pk}_{\mathsf{S}_j}\}_{j=1}^m\big)$.

Regarding $i = 1$, $\mathcal{R}$ samples $m$ uniformly random values $\widehat{x_{1j}}$ to be proposed as shares, correctly randomly sample $r_1$ and computes $R_1 = \mathsf{g}^{r_1}$. $\mathcal{R}$ computes the shares secrets $\mathsf{s}_{1j}$ and verification values $\tau_{1j}$ computed as:

$$\tau_{1j} = Z^{\mathsf{sk}_{\mathsf{S}_j}} \cdot R_1^{\mathsf{s}_{1j}} \qquad \forall\, j \in [1, m] \qquad (5)$$

Observe that the computed $\tau_{1j}$ are equal to the correct verification coefficient obtained by using $x_1 = z$ as the secret input, formally:

$$\tau_{1j} = Z^{\mathsf{sk}_{\mathsf{S}_j}} \cdot R_1^{\mathsf{s}_{1j}} = \mathsf{pk}_{\mathsf{S}_j}^{z} \cdot R_i^{\mathsf{s}_{1j}} \qquad \forall\, j \in [1, m] \qquad (6)$$

The reduction returns to $\mathcal{A}$ the shares $\big(\mathsf{pk}_{\mathsf{U}_i}, \{\widehat{x_{ij}}\}_{j=1}^t, R_i\big)$ and all the verification coefficients $\{\tau_{ij}\}_{j=1}^m$ for each $i \in [1, n]$.

Observe that the adversary $\mathcal{A}$ is unable to reconstruct the final aggregated value $y = \sum_{i=1}^{n} x_i$ since it only posses $t$ shares out of the necessary $t+1$ required by the security of the secret sharing scheme. In other words, the $t$ randomly generated shares of $\mathsf{U}_1$ cannot be used to (even) identify that they are not correctly computed since the provided communication and a correct execution have the same distribution. At this point, $\mathcal{A}$ replies with the guess $y^*$ which is forwarded by $\mathcal{R}$ to the $\mathsf{dLog}_I$ challenger and observe that:

$$y^* = \sum_{i=1}^{n} x_i = x_1 + \sum_{i=2}^{n} x_i = z + 0 = z$$

If $\mathcal{A}$ has a non-negligible advantage to win the $\mathsf{Exp}^{\mathsf{sec}}_{\mathsf{NIVA}}(\mathcal{A})$ experiment, $\mathcal{R}$ has the same non-negligible advantage to break $\mathsf{dLog}_I$ which is assumed to be hard, which is a contradiction. Thus $\mathsf{Pr}\left[\mathsf{Exp}^{\mathsf{sec}}_{\mathsf{NIVA}}(\mathcal{A}) = 1\right] = \epsilon_{\mathsf{dLog}_I} < \mathsf{negl}$. □

**Theorem 3 (NIVA Verifiability).** *Consider $n$ users and $m$ servers, with threshold $t$ such that the order $p$ of the cyclic group $\mathbb{G}$ used for the key-agreement does not divide $m-1$. Let $\mathcal{A}$ be a PPT adversary that maliciously controls $k < m$ servers,* w.l.o.g. *$\{S_j\}_{j=1}^{k}$. It holds that NIVA is **verifiable** (Def. 5).*

*Proof.* The adversary $\mathcal{A}$ provides $N^*$, it is unable to reconstruct the final aggregated value $y = \sum_{i=1}^{n} x_i$ since it only posses $t$ shares out of the necessary $t+1$ required by the security of the secret sharing scheme. Additionally, the choice of $N^*$ does not have any security impact and it is only necessary for the challenger $\mathcal{C}$ to correctly compute the partial evaluations.

Consider $\{y_j, \pi_j, R_{\mathsf{S}_j}, \rho_j\}_{j=1}^{k}$ the honestly computed partial evaluations and observe that $\mathcal{A}$'s tamper $\{{y_j}^*, {\pi_j}^*, {R_{\mathsf{S}_j}}^*, {\rho_j}^*\}_{j=1}^{k}$ must, when reconstructed, obtain the final output $y^*$ to be $y + \Delta$ for any subset of $t+1$ partial evaluation in $M^*$ for some $\Delta \neq 0$. We can denote the tampers as, for any $j$, ${\pi_j}^* = \pi_j + \epsilon_j$ and ${\rho_j}^* = \rho_j \cdot \xi_j$. Observe that, all the ${R_{\mathsf{S}_j}}^*$ must be equal to the correct $R$ obtained from the mandatory uncorrupted server $\mathsf{S} \in M^*$.

Let us focus on Eq. (3), and observe that for the subset $M_l$,

$$\begin{aligned}
\prod_{j \in M_l}^{i \in N} \tau_{ij} &= \left( \prod_{j \in M_l} \mathsf{pk}_{\mathsf{S}_j} \right)^{y^*} \cdot \prod_{j \in M_l} {R^{\pi_j}}^* {\rho_j}^* \\
&= \left( \prod_{j \in M_l} \mathsf{pk}_{\mathsf{S}_j} \right)^{y} \cdot \prod_{j \in M_l} R^{\pi_j} \rho_j \cdot \left( \prod_{j \in M_l} \mathsf{pk}_{\mathsf{S}_j} \right)^{\Delta} \cdot \prod_{j \in M_l} R^{\epsilon_j} {\xi_j}^*
\end{aligned}$$

where, in order to be correctly verified, it must hold, for each $M_l$,

$$\prod_{j \in M_l} \mathsf{pk}_{\mathsf{S}_j}^{\Delta} \cdot R^{\epsilon_j} {\xi_j}^* = 1 \qquad \forall\, M_l \subset M \tag{7}$$

where each $M_l$ can be split into the corrupted and the honest servers subsets, *i.e.* ${M_l}^* = M_l \cap \{\mathsf{S}_j\}_{j \in [1,k]}$ and $\widetilde{M_l} = M_l \cap \{\mathsf{S}_j\}_{j \in [k+1,m]}$ where $\mu_l = |{M_l}^*|$. We can

therefore expand Eq. (7) and obtain the system of equations:

$$\begin{cases} \prod_{j \in {M_1}^*} \mathsf{pk}_{\mathsf{S}_j}^{\Delta} \cdot R^{\epsilon_j} {\xi_j}^* = \prod_{j \in \widetilde{M_1}} \mathsf{pk}_{\mathsf{S}_j}^{-\Delta} \\ \vdots \\ \prod_{j \in {M_\mu}^*} \mathsf{pk}_{\mathsf{S}_j}^{\Delta} \cdot R^{\epsilon_j} {\xi_j}^* = \prod_{j \in \widetilde{M_\mu}} \mathsf{pk}_{\mathsf{S}_j}^{-\Delta} \end{cases} \tag{8}$$

which can be seen as $|M| = \mu$ equations in $|{M_l}^*| \leq \mu - 1$ variables, *i.e.* $\mathcal{A}$ sees $\mathsf{pk}_{\mathsf{S}_j}^{\Delta} R^{\epsilon_j} {\xi_j}^*$ as the variable $\mathsf{g}^{x_j}$ for each $\mathsf{S}_j \in {M_l}^*$. This system has the same solution space as the one obtained by considering the exponents. In other words,

$$\begin{cases} \prod_{j \in {M_1}^*} \mathsf{g}^{x_j} = \prod_{j \in \widetilde{M_1}} \mathsf{g}^{-\Delta \mathsf{sk}_{\mathsf{S}_j}} \\ \vdots \\ \prod_{j \in {M_\mu}^*} \mathsf{g}^{x_j} = \prod_{j \in \widetilde{M_\mu}} \mathsf{g}^{-\Delta \mathsf{sk}_{\mathsf{S}_j}} \end{cases} \iff \begin{cases} \sum_{j \in {M_1}^*} x_j = \sum_{j \in \widetilde{M_1}} -\Delta \mathsf{sk}_{\mathsf{S}_j} \\ \vdots \\ \sum_{j \in {M_\mu}^*} x_j = \sum_{j \in \widetilde{M_\mu}} -\Delta \mathsf{sk}_{\mathsf{S}_j} \end{cases} \tag{9}$$

Denote the vector of variables with $\boldsymbol{x} = (x_1, \ldots, x_{\mu_l})$ and the known coefficient with $\boldsymbol{b} = (-\mathsf{sk}_{\mathsf{S}_j} \Delta)_{j=\mu_l}^{\mu}$.By properly ordering the servers, Eq. (9) form the non-homogeneous linear system

$$\begin{pmatrix} D_{\mu_l} \\ 1_{(\mu - \mu_l)}^{\mu_l} \end{pmatrix} \cdot \boldsymbol{x} = \begin{pmatrix} 1_{\mu_l}^{(\mu - \mu_l)} \\ D_{(\mu - \mu_l)} \end{pmatrix} \boldsymbol{b} \tag{10}$$

which can be seen as the homogeneous system

$$D_\mu \cdot \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{b} \end{pmatrix} = \left( \begin{array}{c|c} D_{\mu_l} & 1_{\mu_l}^{(\mu - \mu_l)} \\ 1_{(\mu - \mu_l)}^{\mu_l} & D_{(\mu - \mu_l)} \end{array} \right) \cdot \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{b} \end{pmatrix} = 0_\mu \tag{11}$$

where $1_r^c$ denotes a matrix of $r$ rows and $c$ columns with all entries equal to one, $0_\mu$ is the zero vector of length $\mu$, $\mathsf{id}_k$ is the identity matrix and $D_k = 1_k^k - \mathsf{id}_k$, *i.e.* a $k$-square matrix of ones and null diagonal. Observe that we can verify the existence of a solution for Eq. (10) by using the Rouché-Capelli's theorem of Thm. 1. To do so, let us first prove that $D_k$ always has maximum rank.

**Lemma 1.** *Consider the matrices defined over a field of characteristic $p$ prime. For any $k \in \mathbb{N}$, $k > 0$ and such that $p$ does not divide $k - 1$, $D_k$ has maximum rank,* i.e. *$D_k$ is invertible.*

*Proof.* By reducing the matrix via the Euclidean algorithm,

$$D_k \Longrightarrow \begin{pmatrix} 1 & 1 & \dots & 1 & 0 \\ 1 & \dots & \iddots & 0 & 1 \\ \vdots & & \iddots & & \vdots \\ 1 & 0 & \iddots & \dots & 1 \\ 0 & 1 & \dots & 1 & 1 \end{pmatrix} \Longrightarrow \begin{pmatrix} 1 & 1 & \dots & 1 & 0 \\ 0 & \dots & \iddots & -1 & 1 \\ \vdots & & \iddots & & \vdots \\ 0 & -1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 1 & 1 \end{pmatrix} \Longrightarrow \begin{pmatrix} 1 & 1 & \dots & 1 & 0 \\ 0 & 1 & \dots & 0 & -1 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \dots & 1 & -1 \\ 0 & 0 & \dots & 0 & (k-1) \end{pmatrix}$$

Thus, the determinant of $D_k$ is $\det(D_k) = k - 1$. Since $p$ does not divide $k - 1$, we have that $\det(D_k) \neq 0$ thus $D_k$ is invertible and of maximum rank. □

By applying the lemma, we conclude that the system of Eq. (11) has rank $\mu$ while the one in Eq. (10) has rank $\mu_l = |M_l{}^*| \leq \mu - 1 < \mu$. Rouché-Capelli guarantees that **no solution exists** that satisfies the system. Thus, $\mathcal{A}$ is unable to provide a correct tamper, thus NIVA is verifiable and $\mathsf{Pr}\left[\mathsf{Exp}^{\mathsf{ver}}_{\mathsf{NIVA}}(\mathcal{A}) = 1\right] = 0$. □

Observe that in the definition of the verification algorithm Ver, the servers' subset $M$ *always allows* the existence of $|M| = \mu$ *different* subsets $M_l \subset M$ with $|M_l| = \mu - 1$ obtained as $M_l = M \setminus \{\mathsf{S}_l\}$ for each $\mathsf{S}_l \in M$. We require $M$ to have at least $t + 1$ elements in order to execute the SS reconstruction SS.Recon.

**Corollary 2.** *NIVA achieves verifiability even in the case of using OptVer as the verification algorithm.*

*Proof.* The OptVer's correspondent system of Eq. (10) is

$$\begin{pmatrix} \mathsf{id}_{\mu_l} \\ 0^{\mu_l}_{(\mu-\mu_l)} \end{pmatrix} \cdot \boldsymbol{x} = \begin{pmatrix} 0^{\mu_l}_{(\mu-\mu_l)} \\ \mathsf{id}_{(\mu-\mu_l)} \end{pmatrix} \boldsymbol{b}$$

where $0^c_r$ denotes a matrix of $r$ rows and $c$ columns with all entries equal to zero. This can be seen as the homogeneous system

$$\mathsf{id}_\mu \cdot \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{b} \end{pmatrix} = \left(\begin{array}{c|c} \mathsf{id}_{\mu_l} & 0^{\mu_l}_{(\mu-\mu_l)} \\ 0^{\mu_l}_{(\mu-\mu_l)} & \mathsf{id}_{(\mu-\mu_l)} \end{array}\right) \cdot \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{b} \end{pmatrix} = 0_\mu$$

Since the identity function has maximal rank, the two systems have different ranks and, exactly as in the proof, Rouché-Capelli guarantees us that no solution exists thus $\mathcal{A}$ is unable to provide a correct tamper. □

### 3.2 Additional Properties and Extensions

In this subsection, we discuss how additional properties presented by concurrent primitives/protocols [32,33,34,25] apply to NIVA.

***Multiple Executions.*** In the FL setting, it is required to execute the aggregation multiple times. NIVA is described for a single execution but the same generated key pairs allow the execution of multiple aggregation/verification calls.

***Decentralization.*** Several published protocols [34,25] do not consider this decentralized scenario making their server a single point-of-failure, *i.e.* if the centralized server halts, the protocol cannot be terminated. NIVA decentralizes the aggregation between several servers and only a predefined amount is necessary for the correct reconstruction and verification of the output. This allows to overcome realistic problems such as *"complete the aggregation in case of failing servers"* or introduce *"responsibilities distribution"*, *i.e.* the servers might be owned by different *independent* entities and not by a single organisation.

***Non-Interactivity and User Drop-Out.*** The aggregation problem discussed in this section can be solved either with an *interactive* protocol or a *non-interactive* primitive. The first allows the use of a *"challenge-response"* interaction that facilitates the computation of more complex verification protocols **but** introduces the **users' drop-out** problem, *i.e.* the user might drop-out during the communication thus are not able to finish the aggregation protocol, forcing the servers to abort the protocol. To overcome this issue, the protocol must be able to identify the drop-outs and recover the user's information to complete the aggregation or, if not possible, having a procedure for removing the user's initial participation. In a non-interactive solution, such as NIVA, a user cannot drop-out since there is no interaction. A dropping user in the non-interactive communication is equivalent to a user that never participated. Thus any non-interactive solution is trivially able to overcome the users' drop-out problem. On the other hand, interactivity allows to easily introduce *input's range proof* [22,7,19], *i.e.* a proof, generally in zero-knowledge, that allows the server to verify that the values obtained are indeed related to the user's secret input without revealing it. It might be possible to transform these zero-knowledge protocols into non-interactive proofs at the cost of introducing additional assumptions, *e.g.* the random oracle model for the Fiat-Shamir's transformation [12]. NIVA design's principle is simplicity with a small amount of assumption required; thus, allowing a more general deployment for different application/security models.

***Authentication and Publishing.*** In this work, we do not consider *malicious* adversaries that are able to diverge from the correct communication. Similarly to the non-interactivity discussion, it might be possible to prevent active attacks by achieving communication authentication by, for example, force the registration of the servers' public keys on a public key infrastructure and using authenticated communications, *e.g.* communicating over a TLS channel. Additionally, NIVA requires the existence of an untamperable public *"space"* (*e.g.* a bulletin board) in which the partial proofs $\tau_{ij}$ to be used in the verification phase, will be stored. These requirements must be carefully considered whenever NIVA is used in a framework where active adversaries are a possibility.

***Differential Privacy.*** Specific applications related to privacy preserving aggregation require a higher-level of privacy, especially when multiple aggregation outputs are published and from which it might be possible to infer information on a specific user/group. This is the case study for *differential privacy* [10] and the framework that implements it. Without entering tedious details, it is possible to

utilize NIVA for differential private and distributed aggregation since it is possible to *introduce* the correctly sampled noise by using the additive-homomorphic property. The specific protocol for fairly and publicly generating the noise are tangent to NIVA's definition and to other abstract frameworks.

## 4 Implementation and Comparisons

In this section, we provide relevant statistics and performance measurements retrieved after implementing our primitive NIVA. We conclude by comparing NIVA with the results obtained by Segal *et al.*'s protocol [25] and Xu *et al.*'s VerifyNet [34]. NIVA is implemented as a prototype in Python 3.8.3 and we execute the tests on MacOS 10.13.6 over a MacBookPro (mid 2017) with processor Intel i5-7267U CPU @ 3.1GHz, with 16GB LPDDR3 2133MHz RAM, 256kB L2 cache and 4MB L3 cache. The source code of our implementation is publicly released[5]. For our experiments, the key agreement used is Diffie-Hellman over the elliptic curve secp256k1 and the additive homomorphic SS is Shamir's SS. The execution time is expressed in milliseconds (ms) and the bandwidth in kilobytes (kB).

The NIVAprimitive is executed with respect to $n$ users, $m$ servers with the threshold parameter $t$ and $\mu$ denoting the size of the verification set $M$. The total communication cost, *i.e.* users and servers' output data, is expected to be linearly dependent *w.r.t.* the numbers $m$ and $n$, since each server has a constant size output, while the users are in total communicating $nm$ shares $x_{ij}$ **and** verification values $\tau_{ij}$. Fig. 2 reports the expected behaviour.

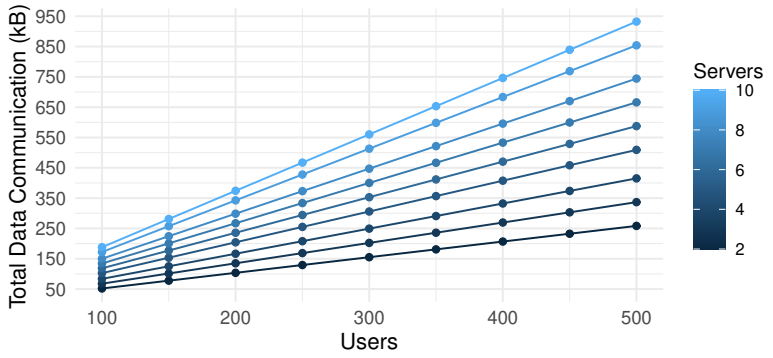
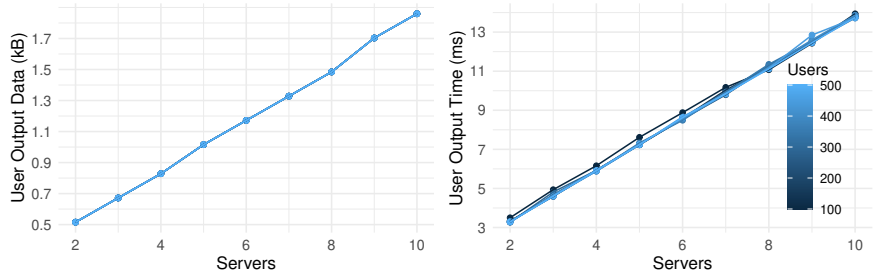



Figure 2: NIVA's total communication bandwidth for a different number $n$ of users and $m$ of servers and fixed $t = 1$ and $\mu = 2$.

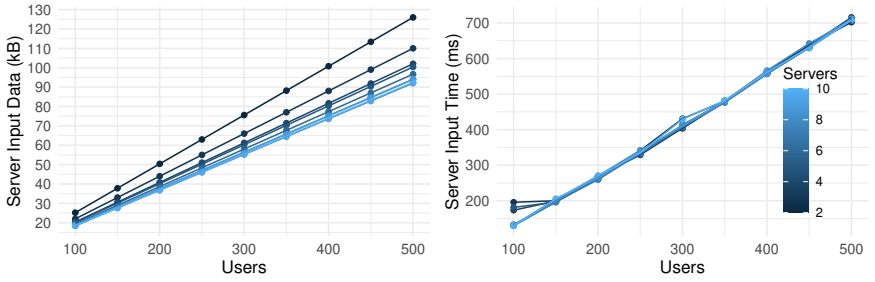
Consider the metrics for a single user U and a server S, depicted in Figures 3a and 3b. As expected, U's output data depends linearly on the amount of servers $m$. The same applies for S's bandwidth and execution time, since they are linear *w.r.t.* the $n$ users. Despite expecting S's input data to be always constant when considering different amount of servers and fixed $n$, our experiments present a

[5] https://bitbucket.org/CharlieTrip/nivacode/src/main/

decreasing S's data when increasing the amount of server. This is due to the approximation introduced by the Python data-measuring package used.




(a) User's data and computation time for a different number $m$ of servers .



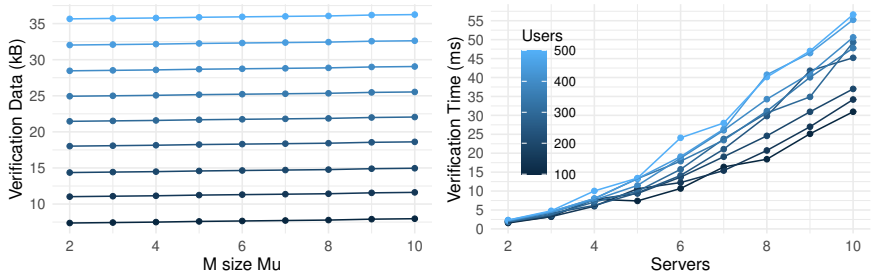

(b) Server's input data and timing per server for a different number $n$ of users'.

Figure 3: User and server's bandwidth and computation time performance.

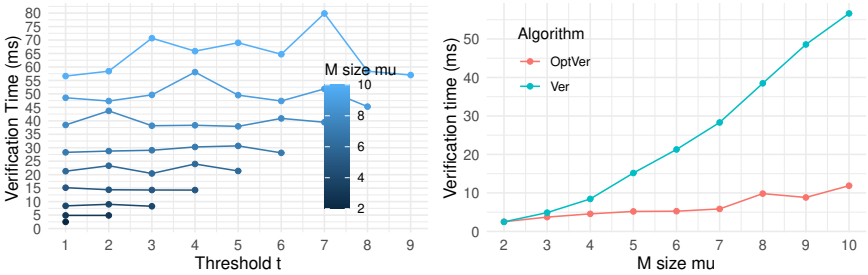
As represented in Fig. 4b, the verification algorithm Ver has input data size proportional to the number $\mu$ of servers used in the verification. By considering the maximum verification set possible, Ver's execution time increases quadratically in the number of users and servers. In Fig. 4a, we observe that the optimal choice for $\mu$ is always $\mu = t + 1$. This is true because, for every $\mu \in [t + 1, m]$, a successful verification requires *(1)* $\mu$ checks of the form of Eq. (3); and *(2)* $\binom{\mu}{t+1}$ calls to SS.Recon. The first is proportional *w.r.t.* the parameters $n$ and $\mu$, but it does not depend on $t$, while the latter has a maximal number of calls whenever $\mu$ is near the integer $2(t + 1)$. This consideration suggests that it is optimal to minimise the verification set size $\mu$ to be $\mu = t+1$. Additionally, the optimized verification algorithm OptVer of Corollary 1 is always faster than Ver, due to the reduced amount of multiplications required during the verification of Eq. (3).

### 4.1 Comparison to Related Work

We compare the performance of our solution with Segal *et al.*'s PPML [25] and Xu *et al.*'s VerifyNet [34] protocols. Segal *et al.*'s results are obtained from a Java implementation running on an Intel Xeon E5-1650 v3 CPU @ 3.50GHz, with 32 GB of RAM while, Xu *et al.*'s are obtained from an Intel Xeon E5-2620




(a) Ver input data size and computation time for a different number $n$ of users , verification set's size $\mu$ and amount of servers $m$.




(b) Ver's computation time for different $\mu$ and $t$ and comparison between Ver and OptVer's computation time for different $\mu$ ($t = 1$).

Figure 4: Communication cost and execution time for Ver and OptVer.

CPU @ 2.10GHz, 16GB RAM on the Ubuntu 18.04 operating system. Both of them have not publicly released their implementations, thus, making it hard to fairly compare the computation times. Additionally, since the considered related works are designed as interactive protocols, we can only compare total bandwidth/execution time and we will mainly focus on the user's and verification algorithm performance metrics since, in the FL scenario, the server enjoys high computational power.

In both the PPML and VerifyNet experiments, the users provide secret *vectors* of length $K$ as input to the aggregation protocol and, additionally, the entries of the vector might be of small size, *e.g.* our implementation represents an integer with $B = 36$ bytes, while the *vector entries* considered in the PPML protocol are $b = 3$ bytes long. To fairly compare, we repeatedly execute NIVA $K\frac{b}{B}$ times in order to achieve the same amount of aggregated value bytes. In other words, we simulate the packing of a vector of small integers into a single bigger integer, as described in the VerifyNet's implementation [34]. PPML assumes that the vector entries are of length $b = 3$ bytes, while VerifyNet was tested on entries of the same size $B$ as NIVA. Since NIVA is the **only** decentralized primitive compared, we test it at the minimal distributed setting possible, *i.e.* $m = 2$ servers both needed for the reconstruction, or threshold $t = 1$.

VerifyNet uses as standard vector size $K = 1000$. Fig. 6a depicts that NIVA is more space efficient than VerifyNet whenever introducing a larger amount of users. Furthermore, NIVA requires a lower amount of users' data than VerifyNet. We should note though that whenever increasing the vector size $K$, it must be observed that NIVA has a slightly steeper angle, which means that there exists a vector size $\hat{k}$ from which VerifyNet becomes more efficient than NIVA. Differently, Fig. 6c collects the required user execution (computation) time in which NIVA results to be always more efficienct than VerifyNet.

PPML is defined with a standard vector of size $K = 10^5$, 100 times bigger than VerifyNet, and **does not** achieve the verification of the aggregated output. Additionally, each vector entry is described with $b = 3$ bytes, 12 times smaller than NIVA's input. As shown in Fig. 6b and Fig. 6d, our primitive seems to never be able to compete with the PPML protocol because of the elevated value $K$. PPML's protocol minimizes the communication cost, thus the execution time, for bigger vector sizes $K$, while it is linearly dependent on the number of users. In contrast, NIVA has a fixed user's communication cost that only depends on the vector size $K$ and the amount of servers $m$. For this reason, we consider $K = 10^5$ and extrapolate the PPML's linear dependency between data and users $n$. We observe that NIVA overtakes PPML regarding both the user's execution time and the communicated data whenever the user size is $\sim 10^4$.

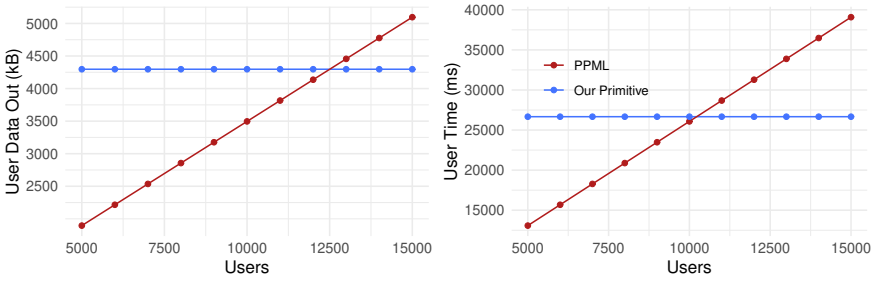
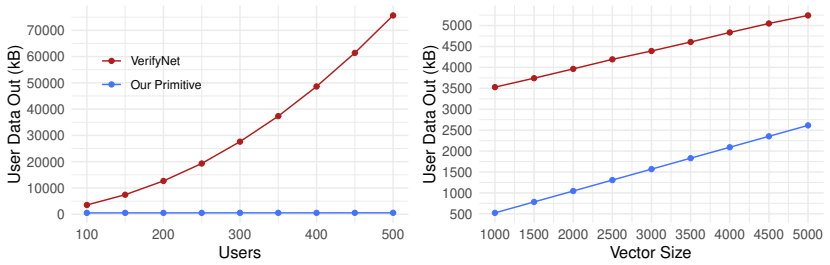
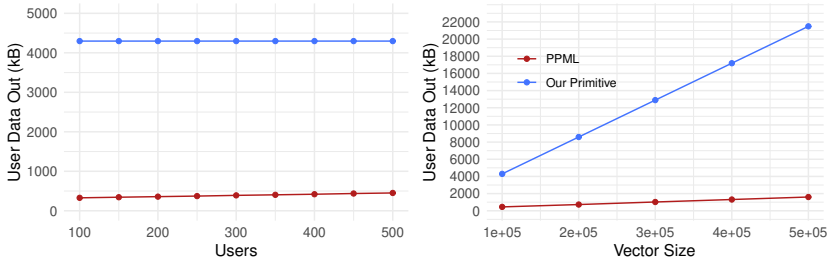



Figure 5: Extrapolated user's data usage and execution time for PPML and NIVA with fixed vector size $K = 10^5$.

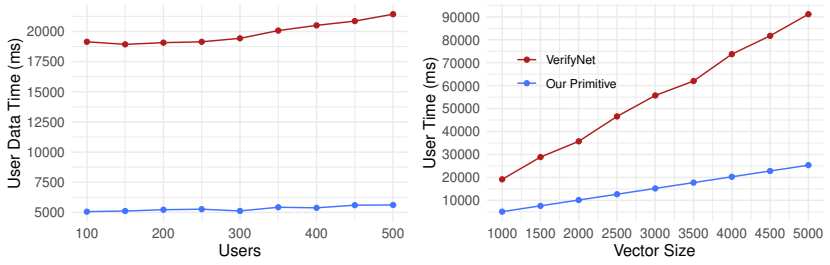
This allows us to conclude that NIVA is better suited than both PPML and VerifyNet for scenarios where the number of users $n$ that participate in a FL model aggregation/update is substantial, *i.e.* over $10^5$. For example, we have simulated a scenario where $n = 10^5$ users participate with a limited vector of $K = 1000$ entries of $b = 3$ bytes each and found out that NIVA has a constant user communication cost of $\sim$43.33kB and execution time of $\sim$282.5ms. In comparison and with the same hypothesis used for Fig. 5, PPML would require *each* user to communicate $\sim$31.55MB for a total of $\sim$4.33 minutes putting it over 3 order of magnitude worse than NIVA. Of course, NIVA's servers have a higher computational demand. In our experiments, each server took $\sim$106.56 hours to handle $\sim$4.00GB of data and the verification algorithm required $\sim$573.33MB of data from users and servers and was executed in $\sim$25.33s. The reason for this



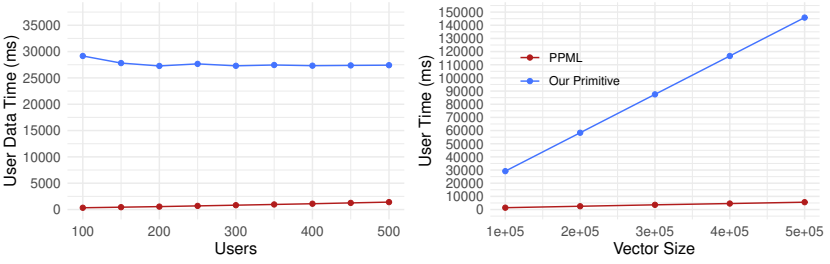

(a) User's data cost comparison between VerifyNet and NIVA for fixed vector size $K = 1000$ and number of users $n = 100$.




(b) User's data cost comparison between PPML and NIVA for fixed vector size $K = 10^5$ and number of users $n = 500$




(c) Timing comparison between VerifyNet and NIVA for fixed vector size $K = 1000$ and number of users $n = 100$.




(d) Timing comparison between PPML and NIVA for fixed vector size $K = 10^5$ and number of users $n = 500$.

Figure 6: Data and time comparisons between PPML, VerifyNet and NIVA.

high cost is the necessity to re-execute the primitive $K \cdot \frac{b}{B}$ times. This can be overcome by, for example, increasing $B$, thus, considering a key agreement based on *very-big* cyclic groups $\mathbb{G}$ such as an elliptic curve over a finite field of 512 bits which should allow to almost double $B$ from 36 to 64. It remains open if it is possible to extend NIVA to work more efficiently with vectors as secret inputs.

**Acknowledgment.** Part of this work was carried out while the third author was a post-doc at Chalmers University of Technology. This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation; and WASP expedition project "Massive, Secure, and Low-Latency Connectivity for IoT Applications".

## References

1. Beimel, A.: Secret-Sharing Schemes: A Survey. In: Coding and Cryptology (2011). https://doi.org/10.1007/978-3-642-20901-7_2
2. Benaloh, J.C.: Secret Sharing Homomorphisms: Keeping Shares of a Secret Secret (Extended Abstract). In: Advances in Cryptology — CRYPTO' 86 (1987). https://doi.org/10.1007/3-540-47721-7_19
3. Boyle, E., Gilboa, N., Ishai, Y.: Group-Based Secure Computation: Optimizing Rounds, Communication, and Computation. In: Advances in Cryptology – EUROCRYPT 2017 (2017). https://doi.org/10.1007/978-3-319-56614-6_6
4. Brickell, E.F.: Some Ideal Secret Sharing Schemes. In: Advances in Cryptology — EUROCRYPT '89 (1990). https://doi.org/10.1007/3-540-46885-4_45
5. Cai, C., Zheng, Y., Du, Y., Qin, Z., Wang, C.: Towards Private, Robust, and Verifiable Crowdsensing Systems via Public Blockchains. IEEE Trans. Dependable and Secure Comput. (2019). https://doi.org/10.1109/TDSC.2019.2941481
6. Capelli, A., Garbieri, G.: Corso Di Analisi Algebrica: 1: Teorie Introduttorie (1886)
7. Chaabouni, R., Lipmaa, H., Zhang, B.: A Non-interactive Range Proof with Constant Communication. In: Financial Cryptography and Data Security (2012). https://doi.org/10.1007/978-3-642-32946-3_14
8. Chor, B., Goldwasser, S., Micali, S., Awerbuch, B.: Verifiable secret sharing and achieving simultaneity in the presence of faults. In: 26th Annual Symposium on Foundations of Computer Science (Sfcs 1985) (Oct 1985). https://doi.org/10.1109/SFCS.1985.64
9. Diffie, W., Hellman, M.E.: New Directions in Cryptography. IEEE Trans. Inf. Theory **22**(6) (1976). https://doi.org/10.1109/TIT.1976.1055638
10. Dwork, C.: Differential Privacy, vol. 4052 (Jul 2006)
11. Fazio, N., Gennaro, R., Jafarikhah, T., Skeith, W.E.: Homomorphic Secret Sharing from Paillier Encryption. In: Provable Security (2017). https://doi.org/10.1007/978-3-319-68637-0_23
12. Fiat, A., Shamir, A.: How To Prove Yourself: Practical Solutions to Identification and Signature Problems. In: Advances in Cryptology — CRYPTO' 86, vol. 263 (2006). https://doi.org/10.1007/3-540-47721-7_12
13. Ghodsi, Z., Gu, T., Garg, S.: 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 (2017)

14. Gordon, S.D., Katz, J.: Rational Secret Sharing, Revisited. In: Security and Cryptography for Networks (2006). https://doi.org/10.1007/11832072_16
15. Hitaj, B., Ateniese, G., Pérez-Cruz, F.: Deep models under the GAN: Information leakage from collaborative deep learning. In: Proc. of CCS (2017)
16. Koblitz, N.: Elliptic curve cryptosystems. Math. Comput. **48**(177) (1987)
17. Krawczyk, H.: Secret Sharing Made Short. In: Advances in Cryptology — CRYPTO' 93 (1994). https://doi.org/10.1007/3-540-48329-2_12
18. Lai, R.W.F., Malavolta, G., Schröder, D.: Homomorphic Secret Sharing for Low Degree Polynomials. In: Advances in Cryptology – ASIACRYPT 2018 (2018). https://doi.org/10.1007/978-3-030-03332-3_11
19. Li, K., Yang, R., Au, M.H., Xu, Q.: Practical Range Proof for Cryptocurrency Monero with Provable Security. In: Information and Communications Security (2018). https://doi.org/10.1007/978-3-319-89500-0_23
20. Liu, Y., Ma, S., Aafer, Y., Lee, W.C., Zhai, J., Wang, W., Zhang, X.: Trojaning attack on neural networks. In: 25th Annual Network and Distributed System Security Symposium, NDSS (2018)
21. McMahan, B., Moore, E., Ramage, D., Hampson, S., y Arcas, B.A.: Communication-efficient learning of deep networks from decentralized data. In: Proc. of AISTATS (2017)
22. Peng, K., Bao, F.: An Efficient Range Proof Scheme. In: 2010 IEEE Second International Conference on Social Computing (Aug 2010). https://doi.org/10.1109/SocialCom.2010.125
23. Phong, L.T., Aono, Y., Hayashi, T., Wang, L., Moriai, S.: Privacy-preserving deep learning via additively homomorphic encryption. IEEE Trans Inf. Forensics Secur. **13**(5) (2018)
24. Pollard, J.M.: Kangaroos, Monopoly and Discrete Logarithms. J. Cryptology **13**(4) (Sep 2000). https://doi.org/10.1007/s001450010010
25. Segal, A., Marcedone, A., Kreuter, B., Ramage, D., McMahan, H.B., Seth, K., Bonawitz, K., Patel, S., Ivanov, V.: Practical secure aggregation for privacy-preserving machine learning. In: CCS (2017)
26. Shamir, A.: How to share a secret. Commun. ACM **22**(11) (Nov 1979). https://doi.org/10.1145/359168.359176
27. Shokri, R., Shmatikov, V.: Privacy-preserving deep learning. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (2015). https://doi.org/10.1145/2810103.2813687
28. Stadler, M.: Publicly Verifiable Secret Sharing. In: Advances in Cryptology — EUROCRYPT '96 (1996). https://doi.org/10.1007/3-540-68339-9_17
29. Thompson, B., Haber, S., Horne, W.G., Sander, T., Yao, D.: Privacy-Preserving Computation and Verification of Aggregate Queries on Outsourced Databases. In: Privacy Enhancing Technologies, vol. 5672 (2009). https://doi.org/10.1007/978-3-642-03168-7_11
30. Tramèr, F., Boneh, D.: Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. In: Proceedings of ICLR (2019)
31. Tsaloli, G., Banegas, G., Mitrokotsa, A.: Practical and provably secure distributed aggregation: Verifiable additive homomorphic secret sharing. Cryptogr. **4**(3), 25 (2020). https://doi.org/10.3390/cryptography4030025
32. Tsaloli, G., Liang, B., Mitrokotsa, A.: Verifiable Homomorphic Secret Sharing. In: Provable Security (ProvSec), 2018. vol. 11192 (2018). https://doi.org/10.1007/978-3-030-01446-9_3
33. Tsaloli, G., Mitrokotsa, A.: Sum it up: Verifiable additive homomorphic secret sharing. In: Information Security and Cryptology – ICISC 2019 (2020)

34. Xu, G., Li, H., Liu, S., Yang, K., Lin, X.: VerifyNet: Secure and verifiable federated learning. IEEE Trans Inf. Forensics Secur. **15** (2020)
35. Xu, W., Evans, D., Qi, Y.: 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 (2018)