Privacy-Preserving Ridge Regression with only Linearly-Homomorphic Encryption

Irene Giacomelli[1(✉)], Somesh Jha[1], Marc Joye[2], C. David Page[1], and Kyonghwan Yoon[1]

[1] University of Wisconsin-Madison, Madison, WI, USA
irene.giacomelli29@gmail.com
[2] NXP Semiconductors, San Jose, CA, USA

Abstract. Linear regression with 2-norm regularization (*i.e.,* ridge regression) is an important statistical technique that models the relationship between some explanatory values and an outcome value using a linear function. In many applications (*e.g.,* predictive modeling in personalized health-care), these values represent sensitive data owned by several different parties who are unwilling to share them. In this setting, training a linear regression model becomes challenging and needs specific cryptographic solutions. This problem was elegantly addressed by Nikolaenko *et al.* in S&P (Oakland) 2013. They suggested a two-server system that uses linearly-homomorphic encryption (LHE) and Yao's two-party protocol (garbled circuits). In this work, we propose a novel system that can train a ridge linear regression model using only LHE (*i.e.,* without using Yao's protocol). This greatly improves the overall performance (both in computation and communication) as Yao's protocol was the main bottleneck in the previous solution. The efficiency of the proposed system is validated both on synthetically-generated and real-world datasets.

Keywords: Ridge regression · Linear regression · Privacy
Homomorphic encryption

1 Introduction

Linear regression is an important statistical tool that models the relationship between some explanatory values (features) and an outcome value using a linear function. Despite its simple definition, a linear regression model is very useful. Indeed, it can be used to quantitatively relate the features and the outcome (*e.g.,* identify which features influence more directly the outcome) and for future prediction (*e.g.,* if a new vector of features with no known outcome is given, the model can be used to make a prediction about it). *Ridge regression* is one of the most widely-used forms of regression; see the survey in [21]. It lessens the overfitting of ordinary least squares regression without adding computational cost. In practice, this is achieved giving preference to models with small Euclidean norm. To enhance the efficacy of the learned model, prior experience in model

© Springer International Publishing AG, part of Springer Nature 2018
B. Preneel and F. Vercauteren (Eds.): ACNS 2018, LNCS 10892, pp. 243–261, 2018.
https://doi.org/10.1007/978-3-319-93387-0_13

training suggests using training data from a large and diverse set. Indeed, it is known that having more data (more relevant features and/or more data points) typically improves the ability to learn a reliable model. A simple way to obtain such training dataset is to merge data contained in "data silos" collected by different entities. However, in many applications (*e.g.,* personalized medicine [28]) the data points encode *sensitive* information and are collected by possibly mutually distrustful entities. Often, these entities will not (or cannot) share the private data contained in their silos, making collaborative analysis on joint data impossible.

Consider the following example: We would like to use a given linear regression method in order to predict the weight of a baby at birth on the basis of some ultrasound measurements made during the last month of pregnancy (*e.g.,* head circumference, femur length,...). On one hand, in order to avoid computing a biased model, we would like to run the selected learning algorithm on data points collected in different hospitals in various locations. On the other hand, each hospital legally cannot share (in the clear) patients' sensitive data (the measurements) with other hospitals or with a third party (*e.g.,* a cloud-computing server). This real-life case exemplifies the challenge on which we focus on: *training a linear regression model on joint data that must be kept confidential and/or are owned by multiple parties.* Moreover, we want to run such collaborative analysis without exposing an entity's sensitive data to any other party in the system (*i.e.,* no entity in the system is trusted to handle the data in the clear).

Our paper takes up the above challenge and proposes an efficient solution in the *two-server model* [16], commonly used by previous works on privacy-preserving machine learning (*e.g.,* see [12,22,23]), where no party needs to be trusted to handle the data in the clear. In this setting, the computation of the model from the merged data is outsourced to two *non-colluding* (but not necessarily trusted) third-parties. After a first phase of collecting private data *in encrypted form* from possibly many data-owners, the two third parties then engage in a second phase for the computation of the model itself. The system is designed in such a way that no extra information (beside that released by the model itself) is revealed to these two parties if they do not collude (condition that can, for example, be enforced by law). Our solution is based only on a simple cryptographic primitive that can be implemented via efficient constructions. Indeed, our system is designed using just a *linearly-homomorphic encryption* (LHE) scheme, that is, an encryption scheme that enables computing the sum of encrypted messages. Previous solutions to the problem considered here are based on multi-party computation protocols (*e.g.,* secret-sharing based protocols like BGW [6] or the 2-party protocol by Yao [29]) or on somewhat-homomorphic encryption (*i.e.,* encryption schemes that support a limited number of arithmetic operations on encrypted messages). A hybrid approach that uses both homomorphic encryption and Yao's scheme was presented in [23]. In this work, *we present the first approach to privacy-preserving ridge regression that uses only linearly-homomorphic encryption.* We believe that this result is interesting both from the theoretical and the practical points of view. Indeed our system

can be seen as a new black-box application of LHE and shows that this basic crypto-primitive can be used alone to handle involved tasks (*i.e.,* ridge regression over distributed data). Furthermore, our system achieves practical performances when implemented using a standard encryption scheme like Paillier's cipher [24]. We show this via an evaluation of our system that uses synthetically-generated and real-world data. Overall, our experiments demonstrate that, for many real scenarios, LHE is all you need to *privately* yet efficiently train a ridge regression model on *distributed* data. As an illustrative example, consider the following existing medical scenario: the *Warfarin dosing model.* Warfarin is a popular anticoagulant for which the International Warfarin Pharmacogenetics Consortium proposed an accurate dosing model trained using linear regression on a medical database that was the merge of the data silos collected by 21 research groups. Using a commodity machine, our system can compute the same model in less than 3 min with the guarantee of confidentiality for the data silos of each research group involved.

Related Work. The question of privacy-preserving machine learning was introduced in 2000 by two pioneering works [1,20]. Later on, privacy-preserving linear regression was considered in a number of different works (*e.g.,* [2,8,10,15,17–19,25]). In 2013, Nikolaenko *et al.* [23] introduced the scenario we consider in this paper: privacy-preserving linear regression protocol in the two-server model. The solution in [23] considers ridge regression on a *horizontally-partitioned* dataset in which each party has some of the data points that form the training set (*e.g.,* two or more hospitals, each of which collects the same medical data on different sets of patients). Their solution is based on LHE and Yao's protocol. The latter is a two-party protocol that allows the evaluation of a circuit C on a pair of inputs (a, b) such that one party knows only a and the other party knows only b. At the end of the protocol, the value $C(a, b)$ is revealed but no party learns extra information beyond what is revealed by this value. In [23], the ridge regression model is computed using Yao's protocol to compute the solution of a linear system of the form $A\boldsymbol{w} = \boldsymbol{b}$ where the entries of matrix A and vector $\boldsymbol{b}$ are encrypted (and must be kept private). The solution $\boldsymbol{w}^*$ is the model. The circuit C is the one that solves a linear system computing the Cholesky decomposition of the coefficient matrix. Recently, in [12], the system presented in [23] was extended to *vertically-partitioned* datasets in which the features in the training dataset are distributed among different parties (*e.g.,* two or more hospitals, each of which collects different medical data on the same set of patients). Gascón *et al.* [12] achieve this result using multiparty-computation techniques to allow the data-owners to distribute shares of the merged datasets to the two parties active in the second phase. Moreover, Gascón *et al.* also improve the running time of the second phase of the protocol presented in [23] by designing a new conjugate gradient descent algorithm that is used as circuit C in the place of Cholesky decomposition. This approach was subsequently further improved by Mohassel and Zhang [22] using mini-batch stochastic gradient descent, and extended to logistic regression and neural networks on arbitrarily partitioned datasets.

Our Contribution. Our paper follows this line of work and presents a novel system for ridge regression in the two-server model. For the first phase, we extend the approach used by Nikolaenko *et al.* to datasets that are arbitrarily partitioned using the techniques of labeled-homomorphic encryption [4] to support multiplications among pairs of ciphertexts encrypted via an LHE scheme. In this way we show that a solution based only on LHE can handle scenarios more complicated than the horizontally-partitioned case. For the second phase, we avoid Yao's protocol by designing an ad-hoc two-party protocol that solves $A\boldsymbol{w} = \boldsymbol{b}$ using only the linear homomorphic property of the underlying encryption scheme. This allows to boost the overall performance and, in particular, to considerably reduce the communication overhead.[1] As a highlight, if we horizontally partition (into ten equal-sized parts) a dataset of 10 millions instances and 20 features, our privacy-preserving regression method runs in under 2 min[2] and produces a communication overhead of 1.3 MB. The system presented in [23] needs more than 50 min and 270 MB exchanged data to perform a similar computation.[3] Finally, we notice that gradient descent based solutions (*e.g.,* [12,22]) use iterative algorithms and present the problem of estimating the number of iterations t. Either t is fixed to a high value that ensures finding a good approximation of the model, which incurs higher complexity for the protocol; either t is chosen adaptively based on the dataset, which can be infeasible in the privacy-preserving setting. Our solution for solving $A\boldsymbol{w} = \boldsymbol{b}$ does not present this problem.

2 Background

Linear Regression. A linear regression learning algorithm is a procedure that on input n points $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$ (where $\boldsymbol{x}_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$) outputs a vector $\boldsymbol{w}^* \in \mathbb{R}^d$ such that $\boldsymbol{w}^{*\intercal}\boldsymbol{x}_i \approx y_i$ for all $i = 1, \ldots, n$. One common way to compute such a model $\boldsymbol{w}^*$ is to use the squared-loss function and the associated empirical error function (mean squared error): $f_{X,\boldsymbol{y}}(\boldsymbol{w}) = \|X\boldsymbol{w} - \boldsymbol{y}\|_2^2$. Here $X \in \mathbb{R}^{n \times d}$ is the matrix with the vector $\boldsymbol{x}_i^\intercal$ as i^{th} row and $\boldsymbol{y} \in \mathbb{R}^n$ is the vector with the value y_i as i^{th} component. We assume that X is always full-rank (*i.e.,* $\text{rk}(X) = d$). Specifically, $\boldsymbol{w}^*$ is computed by minimizing a linear combination of the aforementioned error function and a regularization term, that is, $\boldsymbol{w}^* \in \text{argmin}_{\boldsymbol{w} \in \mathbb{R}^d} f_{X,\boldsymbol{y}}(\boldsymbol{w}) + \lambda R(\boldsymbol{w})$ where $\lambda \geq 0$ is fixed. The regularization term is added to avoid over-fitting the training dataset and to bias toward simpler models. In practice, one of the most common regularization terms is the 2-norm ($R(\boldsymbol{w}) = \|\boldsymbol{w}\|_2^2$), which generates a model with overall smaller components. In this case (called *ridge regression*), the model $\boldsymbol{w}^*$ is computed by minimizing the function $F_{\text{ridge}}(\boldsymbol{w}) = \|X\boldsymbol{w} - \boldsymbol{y}\|_2^2 + \lambda\|\boldsymbol{w}\|_2^2$. Since, $\nabla F_{\text{ridge}}(\boldsymbol{w}) = 2X^\intercal(X\boldsymbol{w} - \boldsymbol{y}) + 2\lambda\boldsymbol{w}$, we have that $\boldsymbol{w}^*$ is computed solving the linear system

$$A\boldsymbol{w} = \boldsymbol{b} \tag{1}$$

[1] Size of the messages exchanged among the parties running the system.
[2] Timing on a 2.6 GHz 8 GB RAM machine running Linux 16.04; 80-bit security.
[3] Timing on a 1.9 GHz 64 GB RAM machine running Linux 12.04; 80-bit security.

where $A = X^\intercal X + \lambda I$ (symmetric $d \times d$ matrix) and $\boldsymbol{b} = X^\intercal \boldsymbol{y}$ (vector of d components). Notice that since X is full-rank, A is positive definite and therefore $\det(A) > 0$ (in particular A is invertible).

Cryptographic Tools. To design our privacy-preserving system, we utilize homomorphic encryption. Let $(\mathcal{M}, +)$ be a finite group. A *linearly-homomorphic encryption* (LHE) scheme for messages in $\mathcal{M}$ is defined by three algorithms:

1. The key-generation algorithm Gen takes as input the security parameter κ and outputs a matching pair of secret and public keys, $(sk, pk) \leftarrow \mathsf{Gen}(\kappa)$.
2. The encryption algorithm Enc is a randomized algorithm that uses the public key pk to transform a message m from $\mathcal{M}$ (plaintext space) into a ciphertext, $c \leftarrow \mathsf{Enc}_{pk}(m)$.
3. The decryption algorithm Dec is a deterministic function that uses the secret key sk to recover the original plaintext from a ciphertext c.

The standard security property (semantic security) says that it is infeasible for any computationally bounded algorithm to gain extra information about a plaintext when given only its ciphertext and the public key pk. Moreover, we have the homomorphic property: Let $\mathcal{C}$ be the set of all possible ciphertexts, then there exists an operation $\odot$ on $\mathcal{C}$ such that for any a-tuple of ciphertexts $c_1 \leftarrow \mathsf{Enc}_{pk}(m_1), \ldots, c_a \leftarrow \mathsf{Enc}_{pk}(m_a)$ (a positive integer), it holds that $\Pr[\mathsf{Dec}_{sk}(c_1 \odot \cdots \odot c_a) = m_1 + \cdots + m_a] = 1$. This implies that, if $c = \mathsf{Enc}_{pk}(m)$, $\mathsf{Dec}_{sk}(\mathsf{cMult}(a, c)) = am$, where $\mathsf{cMult}(a, c) = c \odot \cdots \odot c$ (a times).

In some cases being able to perform only linear operations on encrypted messages is not sufficient. For example, when considering arbitrarily partitioned datasets, we will need to be able to compute the encryption of the product of *two* messages given the encryptions of the individual messages. An LHE scheme cannot directly handle such an operation. On the other hand, a general solution to the problem of computing on encrypted data can be obtained via the use of fully-homomorphic encryption [13]. Since full fledged constructions of fully-homomorphic encryption are still inefficient, more efficient solutions have been designed for evaluating low-degree polynomials over encrypted data functionalities (somewhat-homomorphic encryption). In a recent work, Barbosa *et al.* [4] introduce the concept of *labeled-homomorphic encryption* (labHE); this new primitive significantly accelerates homomorphic computation over encrypted data when the function that is being computed is known to the party that decrypts the result. Since in this paper we consider that the machine-learning algorithm and the data distribution among the participants is publicly known, the previous assumption is satisfied and we can make use of labHE. In particular, Barbosa *et al.* show how to design an homomorphic encryption scheme that supports the evaluation of degree-two polynomials using only an LHE and a pseudo-random function. The new scheme is public-key and works in the multi-user setting: two or more users encrypt different messages, an encryption of the evaluation of a degree-two polynomial on these messages can be constructed by any party having access to the public key and the ciphertexts. Then the party

holding the secret key can decrypt and reveal the result of the evaluation (the polynomial is public, the correspondence user-ciphertext is known). We briefly recall here their construction [4, Sect. 5] in the case that the polynomial is evaluated on messages encrypted only by two different users.

Let $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an LHE scheme with security parameter κ and message space $\mathcal{M}$. Assume that a multiplication operation is given in $\mathcal{M}$; *i.e.,* $(\mathcal{M}, +, \cdot)$ is a finite ring. Let also $F\colon \{0,1\}^s \times \mathcal{L} \to \mathcal{M}$ be a pseudo-random function with seed space $\{0,1\}^s$ ($s = \mathrm{poly}(\kappa)$) and label space $\mathcal{L}$. Define:

- $\mathsf{labGen}(\kappa)$: On input κ, it runs $\mathsf{Gen}(\kappa)$ and outputs (sk, pk).
- $\mathsf{localGen}(pk)$: For each user i and with the public key as input, it samples a random seed σ_i in $\{0,1\}^s$ and computes $pk_i = \mathsf{Enc}_{pk}(\underline{\sigma_i})$ where $\underline{\sigma_i}$ is an encoding of σ_i as an element of $\mathcal{M}$. It outputs (σ_i, pk_i).
- $\mathsf{labEnc}_{pk}(\sigma_i, m, \tau)$: On input a message $m \in \mathcal{M}$ with label $\tau \in \mathcal{L}$ from the user i, it computes $b = F(\sigma_i, \tau)$ and outputs the labeled ciphertext $\boldsymbol{c} = (a, c) \in \mathcal{M} \times \mathcal{C}$ with $a = m - b$ in $\mathcal{M}$ and $c = \mathsf{Enc}_{pk}(b)$.
- $\mathsf{labMult}(\boldsymbol{c}, \boldsymbol{c}')$: On input two labeled ciphertexts, $\boldsymbol{c} = (a, c)$ and $\boldsymbol{c}' = (a', c')$, it computes a "multiplication" ciphertext $d = \mathsf{labMult}(\boldsymbol{c}, \boldsymbol{c}')$ as $d = \mathsf{Enc}_{pk}(a \cdot a') \odot \mathsf{cMult}(a, c') \odot \mathsf{cMult}(a', c)$.
  Observe that $\mathsf{Dec}_{sk}(d) = m \cdot m' - b \cdot b'$. Moreover, notice that given two or more multiplication ciphertexts $d_1, \ldots, d_n$, they can be "added" using the operation of the underlying LHE scheme: $d_1 \odot \cdots \odot d_n$. Assume that user i and user j have both encrypted n messages, $m_1, \ldots, m_n$ and $m'_1, \ldots, m'_n$, respectively. Let $\tilde{c} \in \mathcal{C}$ be the ciphertext obtained as
  $$\bigodot_{t=1}^{n} \mathsf{labMult}\big(\mathsf{labEnc}_{pk}(\sigma_i, m_t, \tau_t), \mathsf{labEnc}_{pk}(\sigma_j, m'_t, \tau'_t)\big).$$
- $\mathsf{labDec}_{sk}(pk_i, pk_j, \tilde{c})$: On input $\tilde{c}$, it first recovers σ_i and σ_j from $\mathsf{Dec}_{sk}(pk_i)$ and $\mathsf{Dec}_{sk}(pk_j)$. Next, it computes $b_t = F(\sigma_i, \tau_t)$ and $b'_t = F(\sigma_j, \tau'_t)$ for all $t = 1, \ldots, n$. Finally, it computes $\tilde{b} = \sum_{t=1}^{n} b_t \cdot b'_t$ and $\tilde{m} = \mathsf{Dec}_{sk}(\tilde{c}) - \tilde{b}$. It is easy to verify that $\tilde{m} = \sum_{t=1}^{n} m_t \cdot m'_t$.

Data Representation. In order to use the cryptographic tools described in the former section, we need to represent the real values that form the input datasets as elements in the finite set $\mathcal{M}$ (the message space). Without loss of generality, we assume that $\mathcal{M} = \mathbb{Z}_N$ for some big integer N and that the entries of X and $\boldsymbol{y}$ are numbers from the real interval $[-\delta, \delta]$ (with $\delta > 0$)[4] with at most ℓ digits in their fractional part. In this case, the conversion from real values to elements in $\mathcal{M}$ can be easily done by rescaling all the entries of X and $\boldsymbol{y}$ and then mapping the integers in $\mathbb{Z}_N$ using the modular operation. For this reason, from now on we consider that the entries of X and $\boldsymbol{y}$ are integers from 0 to $N - 1$. This implies that we consider the matrix A and the vector $\boldsymbol{b}$ having positive integer entries[5]

[4] In other words, $\delta = \max\{\|X\|_\infty, \|\boldsymbol{y}\|_\infty\}$ for the original X and $\boldsymbol{y}$.

[5] We assume that $\lambda \in \mathbb{R}$ has at most 2ℓ digits in the fractional part.

and, finally, that we assume that the model $\boldsymbol{w}^*$ is a vector in $\mathbb{Q}^d$. Notice that for the integer representation of A and $\boldsymbol{b}$ it holds that $\|A\|_\infty, \|\boldsymbol{b}\|_\infty \leq 10^{2\ell}(n\delta^2 + \lambda)$. Therefore, if $10^{2\ell}(n\delta^2 + \lambda) \leq \frac{N-1}{2}$, then A and $\boldsymbol{b}$ are embedded in $\mathbb{Z}_N$ without overflow for their entries. However, if the linear system (1) is now solved over $\mathbb{Z}_N$, then clearly the entries of the solution are given as modular residues of $\mathbb{Z}_N$ and may be different from the entries of the desired model $\boldsymbol{w}^*$ in $\mathbb{Q}^d$. In order to solve this problem and recover the model in $\mathbb{Q}^d$ from the model computed over $\mathbb{Z}_N$, we can apply the *rational reconstruction* technique component-wise. With rational reconstruction [11,27] we mean the application of the Lagrange-Gauss algorithm to recover a rational $t = r/s$ from its representation in $\mathbb{Z}_N$ as $t' = r\, s^{-1} \bmod N$, for N big enough (see (4) in Sect. 4).

3 Threat Model and System Overview

We consider the setting where the training dataset is not available in the clear to the entity that wants to train the ridge regression model. Instead, the latter can access encrypted copies of the data and, for this reason, needs the help of the party handling the cryptographic keys in order to learn the desired model. More precisely, protocols in this paper are designed for the following parties:

- The *Data-Owners*: There are m data-owners $\mathrm{DO}_1, \ldots, \mathrm{DO}_m$; each data-owner DO_i has a private dataset $\mathcal{D}_i$ and is willing to share it only if encrypted.
- The *Machine-Learning Engine* (MLE): This is the party that wants to run a linear regression algorithm on the dataset $\mathcal{D}$ obtained by merging the local datasets $\mathcal{D}_1, \ldots, \mathcal{D}_m$, but has access only to the encrypted copies of them. For this reason, MLE needs the help of the Crypto Service Provider.
- The *Crypto Service Provider* (CSP) takes care of initializing the encryption scheme used in the system and interacts with MLE to help it in achieving its task (computing the linear regression model). CSP manages the cryptographic keys and is the only entity capable of decrypting.

We assume that MLE and CSP do not collude and that all the parties involved are honest-but-curious. That is, they always follow the instructions of the protocol but try to learn extra information about the dataset from the messages received during the execution of the protocol (*i.e., passive security*). Moreover, we assume that for each pair of parties involved in the protocol there exists a private and authenticated peer-to-peer channel. In particular, communications between any two players cannot be eavesdropped.

The goal is to ensure that MLE obtains the model while both MLE and CSP do not learn any other information about the private datasets $\mathcal{D}_i$ beyond what is revealed by the model itself. Even in the case that one of the two servers (MLE or CSP) colludes with some of the data-owners, they should learn no extra information about the data held by the honest data-owners. In order to achieve this goal we design a system that can be seen as multi-party protocol run by the $m + 2$ parties mentioned before and specified by a sequence of steps. This system (described in Sect. 4) has the following two-phase architecture:

Phase 1 (merging the local datasets): CSP generates the key pair (sk, pk), stores sk and makes pk public; each DO_i sends to MLE specific ciphertexts computed using pk and the values in $\mathcal{D}_i$. MLE uses the ciphertexts received and the homomorphic property of the underling encryption scheme in order to obtain encryptions of A and $\boldsymbol{b}$ (coefficient matrix and vector in (1)).

Phase 2 (computing the model): MLE uses the ciphertexts $\mathsf{Enc}_{pk}(A)$ and $\mathsf{Enc}_{pk}(\boldsymbol{b})$ and private random values in order to obtain encryptions of new values that we call "masked data"; these encryptions are sent to the CSP; the latter decrypts and runs a given algorithm on the masked data. The output of this computation ("masked model") is a vector $\tilde{\boldsymbol{w}}$ that is sent back from the CSP to the MLE. The latter computes the output $\boldsymbol{w}^*$ from $\tilde{\boldsymbol{w}}$.

Informally, we say that the system is *correct* if the model computed by the MLE is equal to the model computed by the learning algorithm in the clear using $\mathcal{D}$ as training data. And we say that the system is *private* if the distribution of the masked data sent by the MLE to the CSP is independent of the distribution of the local inputs. Thus, no information about $\mathcal{D}_1, \ldots, \mathcal{D}_m$ is revealed by the messages exchanged during Phase 2.

As we will see in Sect. 4, the specific design of the protocol realizing Phase 1 depends on the distributed setting: horizontally- or arbitrarily-partitioned datasets. However, in both cases, the data-owners input encryptions of local values and the MLE gets the encryptions of A and $\boldsymbol{b}$. The CSP simply takes care of initializing the cryptographic primitive and generates the relative keys. Phase 2 is realized by an interactive protocol between the MLE and the CSP. CSP takes on input the encryptions of A and $\boldsymbol{b}$ from the MLE and returns the solution of the system $A\boldsymbol{w} = \boldsymbol{b}$ following this pattern (we refer to this as the *"masking trick"*):

- The MLE samples a random invertible matrix[6] $R \in \mathrm{GL}(d, \mathcal{M})$ and a random vector $\boldsymbol{r} \in \mathcal{M}$ and it uses the linear homomorphic property of the underlying encryption scheme to compute $C' = \mathsf{Enc}_{pk}(AR)$ and $\boldsymbol{d}' = \mathsf{Enc}_{pk}(\boldsymbol{b} + A\boldsymbol{r})$. The values $C = AR$ and $\boldsymbol{d} = \boldsymbol{b} + A\boldsymbol{r}$ are the "masked data." We slightly abuse notation here; $\mathsf{Enc}_{pk}(\cdot)$ is applied component-wise in the computation of C and of $\boldsymbol{d}'$.
- The CSP decrypts C' and $\boldsymbol{d}'$ and computes $\tilde{\boldsymbol{w}} = C^{-1}\boldsymbol{d}$. The vector $\tilde{\boldsymbol{w}}$ is the "masked model" sent back to the MLE.
- The MLE computes the desired model as $\boldsymbol{w}^* = R\tilde{\boldsymbol{w}} - \boldsymbol{r}$. Indeed, it is easy to verify that $R\tilde{\boldsymbol{w}} - \boldsymbol{r} = R(AR)^{-1}(\boldsymbol{b} + A\boldsymbol{r}) - \boldsymbol{r} = A^{-1}\boldsymbol{b}$.

Informally, the security of the encryption scheme assures privacy against an honest-but-curious MLE. On the other hand, if R and $\boldsymbol{r}$ are sampled uniformly at random, then the distribution of the masked data is independent of A and $\boldsymbol{b}$. This guarantees privacy against an honest-but-curious CSP. Similar masking tricks have been previously used in different settings. In [3], a similar method is

[6] $\mathrm{GL}(d, \mathcal{M})$ denotes the general linear group of degree d over the ring $\mathcal{M}$; namely, the group of $d \times d$ invertible matrices with entries from $\mathcal{M}$.

used to design a secret-shared based MPC protocol for the evaluation of general functions. In this work, we tailor the masking trick for the goal of solving the linear system $A\boldsymbol{w} = \boldsymbol{b}$ gaining in efficiency. In [26], masking with random values is used to outsource a large-scale linear system to an untrusted "cloud server". They assume that the coefficient matrix A and vector $\boldsymbol{b}$ of the linear system are known to a "cloud customer" seeking the solution $\boldsymbol{w}$. In this work, A and $\boldsymbol{b}$ are encrypted and the masking is applied "inside the encryption"; to make the masking trick, which works in $\mathbb{Q}$, compatible with the encryption and the modular arithmetic used for it, we make use of rational reconstruction.[7]

Notice that the two-server model allows for different implementations in practice. If we consider applications in which the majority of data-owners are willing to help to run collaborative analysis but don't want to (or cannot) spend to much resources to execute it, then the role of MLE and CSP can be taken by two semi-trusted[8] third-parties (*e.g.,* two independent research institutions). This setting offers the practical advantage that the involvement of all data-owners is minimal. Otherwise, since CSP and MLE are only required to be non-colluding, their role can be taken by two disjoint subsets of data-owners (*e.g.,* for $m \geq 2$, we can have DO_1 and DO_2 playing the role of MLE and CSP, respectively). In this case, no third-parties are required to implement the system.

4 Protocols Description

In this section we describe how to implement Phase 1 and Phase 2. Let $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an LHE scheme with security parameter κ and message space $\mathcal{M} = \mathbb{Z}_N$.

4.1 Phase 1: Merging the Dataset

Horizontally-Partitioned Setting. Assume that the dataset represented by the matrix X and the vector $\boldsymbol{y}$ is horizontally-partitioned in m datasets. That is, the data-owner DO_k holds

$$\mathcal{D}_k = \left\{(\boldsymbol{x}_{n_{k-1}+1}, y_{n_{k-1}+1}), \ldots, (\boldsymbol{x}_{n_k}, y_{n_k})\right\}, \tag{2}$$

for $k = 1, \ldots, m$ $(0 = n_0 < n_1 < \cdots < n_m = n)$. In this case, as already noticed in [23], defining $A_k = \sum_{i=n_{k-1}+1}^{n_k} \boldsymbol{x}_i \boldsymbol{x}_i^\intercal$ and $\boldsymbol{b}_k = \sum_{i=n_{k-1}+1}^{n_k} y_i \boldsymbol{x}_i$, we have that $A = \sum_{k=1}^{m} A_k + \lambda I$ and $\boldsymbol{b} = \sum_{k=1}^{m} \boldsymbol{b}_i$. In Protocol $\Pi_{1,\mathrm{hor}}$, each data-owner DO_k computes and sends to MLE encryptions of the entries of A_k and $\boldsymbol{b}_k$; then MLE computes encryptions of the entries of A and $\boldsymbol{b}$ using the above formulas and the operation $\odot$ (details in Protocol 1).

[7] Notice that the system presented in [26] fails because no techniques are used to make the arithmetic over $\mathbb{Q}$ compatible with the modular arithmetic used by the underling LHE (*i.e.,* Paillier's scheme). See [7] for more details on this.

[8] That is, trusted to be non-colluding.

Protocol 1. $\Pi_{1,\mathrm{hor}}$: Phase 1 in the horizontally-partitioned setting.

– *Parties*: CSP, MLE, and DO_k with input $\mathcal{D}_k$ (as defined in (2)) for all $k = 1, \ldots, m$.
– *Output*: MLE gets A' and $\boldsymbol{b}'$ (*i.e.,* encryptions of A and $\boldsymbol{b}$, respectively).

Step 1: (*key-generation*) CSP runs $(sk, pk) \leftarrow \mathsf{Gen}(\kappa)$ and makes pk public, while it keeps sk secret.
Step 2: (*local computation*) For all $k = 1, \ldots, m$, DO_k computes $A_k = \sum_i \boldsymbol{x}_i \boldsymbol{x}_i^\intercal$ and $\boldsymbol{b}_k = \sum_i y_i \boldsymbol{x}_i$ with $n_{k-1} + 1 \leq i \leq n_k$; next, DO_k encrypts them, $A'_k[i,j] = \mathsf{Enc}_{pk}(A_k[i,j])$, $\boldsymbol{b}'_k[i] = \mathsf{Enc}_{pk}(\boldsymbol{b}_k[i])$ for all $i, j = 1, \ldots, d$ and $j \geq i$; finally, DO_k sends all A'_k and $\boldsymbol{b}'_k$ to MLE.
Step 3: (*datasets merge*) For all $i, j = 1, \ldots, d$ and $j \geq i$, MLE computes

$$A'[i,j] = \begin{cases} \left(\bigodot_{k=1}^{m} A'_k[i,i]\right) \odot \mathsf{Enc}_{pk}(\lambda) & \text{if } j = i \\ \bigodot_{k=1}^{m} A'_k[i,j] & \text{if } j > i \end{cases}, \quad \boldsymbol{b}'[i] = \bigodot_{k=1}^{m} \boldsymbol{b}'_k[i].$$

Arbitrarily-Partitioned Setting. Assume that each DO_k holds some elements of X and $\boldsymbol{y}$. That is, DO_k holds

$$\mathcal{D}_k = \left\{ X[i,j] = \boldsymbol{x}_i[j] \mid (i,j) \in D_k \right\} \cup \left\{ \boldsymbol{y}[i] = y_i \mid (i,0) \in D_k \right\}, \tag{3}$$

where $D_k \subseteq \{1, \ldots, n\} \times \{0, 1, \ldots, d\}$. Assume that each data-owner sends encryptions of the elements it knows to MLE. Then, in order to compute encryptions of the entries of A and $\boldsymbol{b}$, MLE needs to multiply two ciphertexts. Indeed, we have $\boldsymbol{b}[i] = \sum_{t=1}^{n} \boldsymbol{x}_t[i] \boldsymbol{y}[t]$ and $A[i,j] = \sum_{t=1}^{n} \boldsymbol{x}_t[i] \boldsymbol{x}_t[j]$ if $j \neq i$, otherwise $A[i,i] = \sum_{t=1}^{n} \boldsymbol{x}_t[i] \boldsymbol{x}_t[i] + \lambda$. To allow this, we use labeled-homomorphic encryption. As we recalled in Sect. 2, the latter can be constructed on top of any LHE scheme and it enhances the underlying scheme with the multiplication command $\mathsf{labMult}$. In particular, after having received labeled-encryptions of the input from the data-owners,[9] MLE can compute the encryptions of the entries of A and $\boldsymbol{b}$ using formulas of the form $\bigodot_{t=1}^{n} \mathsf{labMult}\big(\mathsf{labEnc}(\boldsymbol{x}_t[i]), \mathsf{labEnc}(\boldsymbol{x}_t[j])\big)$. Remember that the output of the command $\mathsf{labMult}$ used to compute the encryption of the product of two messages, m_1 and m_2, is in fact an encryption of $m_1 m_2 - b_1 b_2$ where b_1, b_2 are two random values used to compute the labeled-encryptions of the values m_1 and m_2. For this reason, at the end of the procedure described before, MLE obtains encryptions of $A - B$ and $\boldsymbol{b} - \boldsymbol{c}$, instead of encryption of A and $\boldsymbol{b}$, where B and $\boldsymbol{c}$ depend on the random values used to encrypt the entries of the local datasets using the labeled-homomorphic scheme. The matrix B and the vector $\boldsymbol{c}$ can be reconstructed by the party handling the decryption key (*i.e.,* CSP). The decryption procedure of the labeled-homomorphic scheme, labDec, accounts for this. However, in the application we consider here (training a ridge

[9] If $\boldsymbol{x}_t[i]$ and $\boldsymbol{x}_t[j]$ are both held by one DO_k, then the former can send $\mathsf{Enc}_{pk}(\boldsymbol{x}_t[i] \boldsymbol{x}_t[j])$ to MLE, who updates the formulas in Step 3 of $\Pi_{1,\mathrm{arb}}$ accordingly.

Protocol 2. $\Pi_{1,\text{arb}}$: Phase 1 in the arbitrarily-partitioned setting.

– *Parties*: CSP, MLE, and DO_k with input $\mathcal{D}_k$ (as defined in (3)) for all $k = 1, \ldots, m$.
– *Output*: MLE gets A' and $\boldsymbol{b}'$ (*i.e.,* encryptions of A and $\boldsymbol{b}$, respectively).

Step 1: (*key-generation*) CSP runs $(sk, pk) \leftarrow \mathsf{labGen}(\kappa)$ and makes pk public, while it keeps sk secret. For $k = 1, \ldots, m$, DO_k runs $(\sigma_k, pk_k) \leftarrow \mathsf{localGen}(pk)$ and makes pk_k public, while it keeps σ_k secret.

(*setup*) For $k = 1, \ldots, m$, CSP recovers σ_k from $\mathsf{Dec}_{sk}(pk_k)$ and computes $b_{ij} = F(\sigma_k, (i, j))$ with $(i, j) \in D_k$. For $i, j = 1, \ldots, d$ and $j \geq i$, CSP computes $B'[i, j] = \mathsf{Enc}_{pk}(\sum_{t=1}^{n} b_{ti} b_{tj})$ and $\boldsymbol{c}'[i] = \mathsf{Enc}_{pk}(\sum_{t=1}^{n} b_{ti} b_{t0})$. These are sent to MLE.

Step 2: (*local computation*) For $k = 1, \ldots, m$, DO_k computes labeled-encryptions of the known entries of X and $\boldsymbol{y}$. That is, for all $(i, j) \in D_k$, DO_k computes $\boldsymbol{c}_{ij} = (a_{ij}, c_{ij}) = \mathsf{labEnc}_{pk}(\sigma_k, \boldsymbol{x}_i[j], (i, j))$ when $j > 0$ and $\boldsymbol{c}_{i0} = (a_{i0}, c_{i0}) = \mathsf{labEnc}_{pk}(\sigma_k, \boldsymbol{y}[i], (i, 0))$.
For all $k = 1, \ldots, m$, DO_k sends all labeled-ciphertexts $\boldsymbol{c}_{ij}$ to MLE.

Step 3: (*datasets merge*) For all $i, j = 1, \ldots, d$ and $j \geq i$, MLE computes

$$A'[i, j] = \begin{cases} \left(\bigodot_{t=1}^{n} \mathsf{labMult}(\boldsymbol{c}_{ti}, \boldsymbol{c}_{ti})\right) \odot B'[i, i] \odot \mathsf{Enc}_{pk}(\lambda) & \text{if } j = i \\ \left(\bigodot_{t=1}^{n} \mathsf{labMult}(\boldsymbol{c}_{ti}, \boldsymbol{c}_{tj})\right) \odot B'[i, j] & \text{if } j > i \end{cases},$$

$$\boldsymbol{b}'[i] = \left(\bigodot_{t=1}^{n} \mathsf{labMult}(\boldsymbol{c}_{ti}, \boldsymbol{c}_{t0})\right) \odot \boldsymbol{c}'[i].$$

regression model) it is necessary that at the end of Phase 1 the MLE has proper encryptions for A and $\boldsymbol{b}$. Indeed, only in this case we can proceed to Phase 2 and use the masking trick (using the masking trick with labeled-encryptions of A and $\boldsymbol{b}$ doesn't work). For this reason, we need to add one round of communication where CSP sends to MLE encryptions of the entries of B and $\boldsymbol{c}$. This can be done before the beginning of the actual computation (Step 1 of Phase 1) since B and $\boldsymbol{c}$ do not depend on the actual data used to train the regression model. In this way, the MLE can finally gets encryptions of A and $\boldsymbol{b}$. Protocol $\Pi_{1,\text{arb}}$ in Protocol 2 describes this in detail.

4.2 Phase 2: Computing the Model

At the end of Phase 1, MLE knows component-wise encryption of the matrix A and the vector $\boldsymbol{b}$ (both with entries represented in $\mathbb{Z}_N$, the message space of the LHE scheme used in Phase 1). Recall that the final goal of our system is computing $\boldsymbol{w}^* \in \mathbb{Q}^d$ solution of (1). In order to do this in a privacy-preserving manner, in Phase 2 we implement the masking trick described in Sect. 3 and compute $\tilde{\boldsymbol{w}}^*$ that solves (1) in $\mathbb{Z}_N$. Then we use rational reconstruction to find $\boldsymbol{w}^*$. All the details of this are reported in Protocol Π_2 (Protocol 3). The correctness is easy to

Protocol 3. Π_2: Phase 2.

– *Parties*: CSP knows sk, MLE knows $A' = \mathsf{Enc}_{pk}(A)$ and $\boldsymbol{b}' = \mathsf{Enc}_{pk}(\boldsymbol{b})$.
– *Output*: MLE gets $\boldsymbol{w}^*$.

Step 1: (*data masking*) MLE samples $R \leftarrow \mathrm{GL}(d, \mathbb{Z}_N)$ and $\boldsymbol{r} \leftarrow \mathbb{Z}_N^d$ and computes

$$C'[i,j] = \bigodot_{k=1}^{d} \mathsf{cMult}(R[k,j], A'[i,k])$$
$$\boldsymbol{d}'[i] = \boldsymbol{b}'[i] \odot \left(\bigodot_{k=1}^{d} \mathsf{cMult}(\boldsymbol{r}[k], A'[i,k])\right)$$

for all $i, j = 1, \ldots, d$; next, MLE sends C' and $\boldsymbol{d}'$ to CSP.

Step 2: (*masked model computation*) CSP first decrypts C' and $\boldsymbol{d}'$ obtaining C and $\boldsymbol{d}$ ($C[i,j] = \mathsf{Dec}_{sk}(C'[i,j])$, $\boldsymbol{d}[i] = \mathsf{Dec}_{sk}(\boldsymbol{d}'[i])$ for all $i, j = 1, \ldots, d$); then it computes $\tilde{\boldsymbol{w}} \equiv C^{-1}\boldsymbol{d} \bmod N$ and sends it $\tilde{\boldsymbol{w}}$ to MLE.

Step 3: (*model reconstruction*) MLE computes $\tilde{\boldsymbol{w}}^* \equiv R\tilde{\boldsymbol{w}} - \boldsymbol{r} \bmod N$ and uses rational reconstruction on each component of $\tilde{\boldsymbol{w}}^*$ to compute $\boldsymbol{w}^* \in \mathbb{Q}^d$.

verify, indeed we have $R\tilde{\boldsymbol{w}} - \boldsymbol{r} \equiv R(AR)^{-1}(\boldsymbol{b} + A\boldsymbol{r}) - \boldsymbol{r} \equiv A^{-1}\boldsymbol{b} \pmod N$. Security is also straightforward: Protocol Π_2 is secure against a honest-but-curious CSP because the values seen by it (the masked data $AR \mod N$ and $\boldsymbol{b} + A\boldsymbol{r} \mod N$) have a distribution that is unrelated with the input datasets. Moreover, Protocol Π_2 is secure against a honest-but-curious MLE because of the security of the underlying encryption scheme. Indeed, the MLE sees only an encrypted version of A and $\boldsymbol{b}$. See [14, Appendix A.6] for the formal security proof.

In some applications, a desirable property is that the model is delivered only to the data-owners. If the role of MLE and CSP is taken by third-parties, this can be achieved using a standard tool like threshold encryption [9]. In this case, the key generation step of Phase 1 is enhanced with the sharing of sk (*i.e.,* CSP knows sk and each DO_i knows a share for sk). Then, Step 2 of Protocol Π_2 is modified in such a way that CSP sends to MLE the value $\mathsf{Enc}_{pk}(\tilde{\boldsymbol{w}})$, instead of the vector $\tilde{\boldsymbol{w}}$ in the clear. MLE computes $\mathsf{Enc}_{pk}(\tilde{\boldsymbol{w}}^*)$ and broadcasts it to all data-owners. Finally, the DO_i collaborates to jointly decrypt and compute $\boldsymbol{w}^*$.

Choice of Parameters. In the last step of Π_2 we use rational reconstruction to recover the components of $\boldsymbol{w}^* \in \mathbb{Q}^d$ from the solution of $A\boldsymbol{w} = \boldsymbol{b}$ computed in $\mathbb{Z}_N$. According to [11,27] if a rational $t = r/s$ with $-R \leq r \leq R$, $0 < s \leq S$ and $\gcd(s, N) = 1$ is represented as $t' = rs^{-1} \mod N$ in $\mathbb{Z}_N$, then the Lagrange-Gauss algorithm uniquely recovers r and s provided that $2RS < N$. Since $\boldsymbol{w}^* = A^{-1}\boldsymbol{b} = \frac{1}{\det(A)} \operatorname{adj}(A)\boldsymbol{b} \in \mathbb{Q}^d$, in order to choose N that satisfies the condition stated before, we need to bound the $\det(A)$ and the entries of the vector $\operatorname{adj}(A)\boldsymbol{b}$. Let $\alpha = \max\{\|A\|_\infty, \|\boldsymbol{b}\|_\infty\}$, using the Hadamard's inequality, we have that $0 < \det(A) \leq \alpha^d$ (A is a positive definite matrix) and $\|\operatorname{adj}(A)\boldsymbol{b}\|_\infty \leq d(d-1)^{\frac{d-1}{2}}\alpha^d$.

Using the same assumptions of Sect. 2 on the entries of X and $\boldsymbol{y}$ (that is, the entries of X and $\boldsymbol{y}$ are real number in $[-\delta, \delta]$ with at most ℓ digits in the fractional part), we have that $\alpha \leq 10^{2\ell}(n\delta^2 + \lambda)$. It follows that the condition $2RS < N$ is fulfilled when

$$2d(d-1)^{\frac{d-1}{2}}\, 10^{4\ell d}\, (n\delta^2 + \lambda)^{2d} < N. \tag{4}$$

Communication Complexity. The messages sent during Protocol $\Pi_{1,\text{hor}}$ and Protocol Π_2 contain $\Theta(d^2)$ elements from $\mathbb{Z}_N$, while the ones in Protocol $\Pi_{1,\text{arb}}$ contain $\Theta(dn)$ elements. This implies a communication cost of $O(d^3 \log(nd))$ bits for $\Pi_{1,\text{hor}}$ and Π_2, and of $O((nd^2 + d^3)\log(nd))$ bits for $\Pi_{1,\text{arb}}$ (details in [14, Appendix A.3]). In particular, our approach significantly improves the communication complexity compared to the previous solutions that use Yao's scheme [12, 23]. Indeed, the latter requires CSP sending the garbled representation of a boolean circuit of millions of gates (see [23, Fig. 5] and [12, Fig. 7]) to MLE. In [23] the authors show that the garbled representation of one gate is a lookup table of around 30 bytes (80-bit security). This means that a privacy-preserving system based on Yao's scheme, only for sending the garbled circuit and without considering the other steps needs at least hundreds of megabytes. On the other hand, even for large values of n and d, the communication complexity of Π_2 is much smaller than 100 MB (see Fig. 1). For example, in the horizontally-partitioned setting [23] uses same techniques we deploy in $\Pi_{1,\text{hor}}$ and Yao's protocol. In particular, [23] reports that the garbled representation of the circuit that solves (1) with $d = 20$ using Cholesky decomposition (24-bit integer representation) has size 270 MB. On the other hand, for a dataset with 10 millions instances and $d = 20$, the *overall* overhead[10] of $\Pi_{1,\text{hor}} + \Pi_2$ is less than 1.3 MB. In the arbitrarily-partitioned setting, the communication overheard of our system is dominated by the cost of Phase 1 (Protocol $\Pi_{1,\text{arb}}$) because of its linear dependency on the number of instances n. However, this seems to be the case also in other approaches. For example, in [12], a secure inner-product protocol based on additive secret-sharing and Beaver's triples [5] is used to compute the inner product of the columns of the matrix X vertically-partitioned among two or more users. The complexity of this approach for Phase 1 is $\Theta(nd^2 \log(n))$ bits (comparable with the complexity of $\Pi_{1,\text{arb}}$). In Phase 2, [12] use Yao's protocol and conjugate gradient descent (CGD) algorithm to solve (1). They do not report the concrete size of the circuit, but they show the number of gates. For $d = 100$ and 5 iterations of the CGD, more than 10^8 gates are used: this gives an overhead of at least 3 GB only for sending the garbled

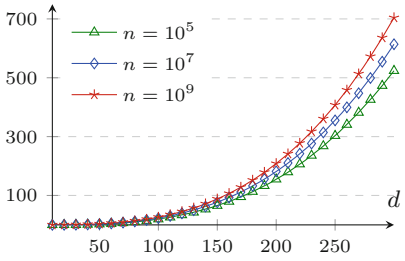

Fig. 1. Communication overhead in MB of Π_2 ($\delta = 1$, 80-bit security, $\ell = 3$, Paillier's scheme, $\lambda = 0$).

[10] In this section, for our system we assume $\ell = 3$ and Paillier's scheme with 80-bit security as underlying LHE.

circuit during Phase 2 (assuming a garbled gate is 30 bytes). On the other hand, the *overall* overhead of $\Pi_{1,\text{arb}} + \Pi_2$ when $d = 100$ for a dataset of 5 thousands instances is less than 1.3 GB.

The SecureML paper [22] uses only additive secret-sharing and Beaver's triples to design a system that assumes an arbitrary partitioning of the dataset. When the pre-processing needed for the triples is implemented via LHE, the linear regression training system proposed in [22] has complexity $\Theta(nd+n)$. Thus, in terms of communication complexity, [22] performs better than our solution in the arbitrarily-partitioned case. Our system, however, is preferable if the training dataset is horizontally-partitioned and $n \gg d$ (*e.g.*, $n = \Theta(d^{2.5})$). For example, if $d = 100$ and $n = 10^5$ the system in [22] has an overheard of 200 MB for the pre-processing phase only (see [22, Table II]), while the total cost of $\Pi_{1,\text{hor}} + \Pi_2$ is less than 120 MB.

5 Implementation

In this section we describe our implementation case study of the system described in Sect. 4. Our goal is to evaluate the effect of the public parameters on the system's accuracy and efficiency, and to test our system on real-world datasets. In particular, the experiments we run are designed to answer the following questions:

1. *Evaluating accuracy*: How does the system parameter ℓ (number of digits in the fractional part of the input data) influence the accuracy of the output model $\boldsymbol{w}^*$? Recall that we assume that the values in X and $\boldsymbol{y}$ are real number with at most ℓ digits in the fractional part. In practice, this means that each user must truncate all the entries in the local dataset after the ℓ^{th} digit in the fractional part. This is done before inputting the values in the privacy-preserving system. On the other hand, in the standard machine learning-setting this requirement is not necessary, and the model is computed using floating point arithmetic on values with more than ℓ digits in the fractional part. For this reason, the model $\boldsymbol{w}^*$, which is trained using our privacy-preserving system, can differ from the model $\bar{\boldsymbol{w}}^*$ learned in the clear (same regularization parameter λ is used). To evaluate this difference we use
$$R_{\text{MSE}} = \left| \frac{\text{MSE}(\boldsymbol{w}^*) - \text{MSE}(\bar{\boldsymbol{w}}^*)}{\text{MSE}(\bar{\boldsymbol{w}}^*)} \right|$$
where MSE is the mean squared error of the model computed on a test dataset (this is a common measure of model accuracy in the machine learning setting). The value R_{MSE} tells the loss in accuracy caused by using the vector $\boldsymbol{w}^*$ instead of $\bar{\boldsymbol{w}}^*$ as model.
2. *Evaluating running-time*: How do the data parameters n and d influence in practice the running time of each step in our privacy-preserving system? In [14, Appendix A.3], we report the number of different elementary operations (*e.g.,* encryptions, modular additions, etc.) for each step in the system, while in this section we report the total running time of each step.

3. *Evaluating efficiency in practice*: How does our system behave when is run on real-world data? In particular, we run our system on datasets downloaded from the UCI repository,[11] which is commonly used for evaluating new machine-learning algorithms in the standard setting (*i.e.,* with no privacy guarantees).

Setup. We implemented our system using Paillier's scheme with message space $\mathcal{M} = \mathbb{Z}_N$. In order to assure a security level of at least 100 bits,[12] decrease the running time and the communication overhead, and satisfy (4), we choose N such that $\log_2(N) = \max\{2048, \lfloor\beta\rfloor + 1\}$ where β is the logarithm in base 2 of the left-hand side of (4). We wrote our software in Python 3 5.2 using the `phe 1.3` library[13] to for Paillier encryption/decryption and operations on ciphertexts, and the `gmpy2` library[14] for arithmetic operations with large integers. Gaussian elimination was used to compute determinants and linear systems.

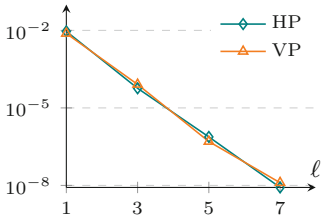

Fig. 2. Error rate R_{MSE} (log scale) in function of ℓ ($n = 10^3, d = 10$).

To test the system composed by $\Pi_{1,\text{hor}} + \Pi_2$, we run experiments in the *horizontally-partitioned* (HP) setting, splitting n data points evenly among 10 data-owners. To test the system $\Pi_{1,\text{arb}} + \Pi_2$, we run experiments in the *vertically-partitioned* (VP) setting, where we assume that d features are evenly split among 3 data-owners and DO_3 also has $\boldsymbol{y}$.

Numerical Results. All experiments were run on a machine with the following specifics. OS: Scientific Linux 7.4, CPU: 40 core (Intel(R) Xeon(R) CPU E5-2660 v2 2.20 GHz), Memory: 500 GB. All the timings are reported in seconds, all the values are averaged on 5 repetitions of the same experiment.

To answer Question 1, we measure the R_{MSE} for different values of ℓ for synthetically-generated data in both the HP and VP settings (see Fig. 2). With the increasing of ℓ, regardless of the values of n and d, the value of R_{MSE} decreases very rapidly, while the efficiency degrades. Indeed, because of (4), the value of ℓ has effect on the bit-length of the plaintexts and ciphertexts. For this reason, we recommend to choose ℓ equal to a small integer (*e.g.,* $\ell = 3$). This choice allows to have a negligible error rate (*e.g.,* R_{MSE} of order 10^{-4}) without degrading the system efficiency.

To answer Question 2 and assess the effect of parameters n and d on our system's performance, we report in Table 1 the running time of each step of the system when it is run on synthetic data. The advantage of this approach is

[11] https://archive.ics.uci.edu/ml/datasets.html.
[12] According to NIST standard, an RSA modulus of 2048 bits gives 112-bit security.
[13] http://python-paillier.readthedocs.io.
[14] https://pypi.python.org/pypi/gmpy2.

Table 1. Running times (secs) for synthetic data in the HP and VP settings ($\ell = 3$).

	n	d	$\log_2(N)$	R_{MSE}	Phase 1			Phase 2		
					Step 1	Step 2	Step 3	Step 1	Step 2	Step 3
HP setting	1000	10	2048	7.21E−05	0.21	1.10	0.03	1.21	0.56	0.04
		20	2048	1.54E−04	0.32	3.88	0.12	7.96	2.15	0.14
		30	2048	1.58E−04	0.18	8.34	0.26	24.76	4.80	0.29
		40	2504	2.01E−04	0.38	26.13	0.62	100.94	14.72	0.67
	10000	10	2048	5.45E−05	0.16	1.11	0.03	1.21	0.57	0.04
		20	2048	1.29E−04	0.09	3.93	0.12	7.99	2.14	0.15
		30	2072	1.90E−04	0.36	8.83	0.26	25.96	5.17	0.32
		40	2768	1.84E−04	0.39	29.81	0.72	120.43	19.34	0.86
	100000	10	2048	1.05E−04	0.13	1.17	0.03	1.22	0.57	0.05
		20	2048	1.08E−04	0.20	4.13	0.12	7.99	2.15	0.16
		30	2270	1.38E−04	0.23	11.65	0.31	33.19	6.26	0.40
		40	3034	1.76E−04	0.61	38.38	0.86	151.37	24.82	1.08
VP setting	1000	10	2048	1.50E−04	1.41	62.06	135.09	1.22	0.56	0.04
		15	2048	8.90E−05	2.52	90.36	220.32	3.51	1.22	0.08
		20	2048	1.78E−04	4.08	118.73	327.48	8.10	2.16	0.14
	2000	10	2048	1.08E−04	1.92	124.35	276.13	1.23	0.59	0.04
		15	2048	6.64E−05	3.54	181.09	443.78	3.56	1.31	0.09
		20	2048	1.67E−04	5.62	236.54	653.06	8.03	2.17	0.14
	3000	10	2048	6.46E−05	2.31	185.89	402.53	1.21	0.57	0.04
		15	2048	1.06E−04	4.38	270.12	659.67	3.52	1.22	0.08
		20	2048	1.36E−04	7.00	355.12	979.89	8.12	2.14	0.14

that we can run experiments for a wide range of parameters values. For Step 2 in Phase 1 (Protocol $\Pi_{1,\text{hor}}$ in the HP setting, Protocol $\Pi_{1,\text{arb}}$ in the VP setting) we report the average running time for one data-owner. In Protocol $\Pi_{1,\text{hor}}$, Step 2 is the most expensive one. Here, the data-owner DO_k computes the $d \times d$ matrix A_k and encrypts its entries. In our setting (n data points evenly split among the ten data-owners), this costs $\Theta(nd^2)$ arithmetic operations on plaintext values and $\Theta(d^2)$ encryptions for one data-owner. We verified that the costs of the encryptions is dominant for all values of n considered here.[15] In Step 3 of $\Pi_{1,\text{hor}}$, the MLE computes the encryption of A and $\boldsymbol{b}$ using approximately $\Theta(d^2)$ ciphertexts additions (*i.e.,* multiplications modulo N), which turns out to be fast. In $\Pi_{1,\text{arb}}$, Step 3 is the most expensive step, here the MLE performs $\Theta(nd^2)$ ciphertexts operation to compute $\mathsf{Enc}_{pk}(A)$ and $\mathsf{Enc}_{pk}(\boldsymbol{b})$. In particular, the running time of $\Pi_{1,\text{arb}}$ is more influenced by the value of n than that of $\Pi_{1,\text{hor}}$ and Π_2. Finally, for Π_2 the results in Table 1 show that Step 1 requires longer time compared to the other two steps because of the $\Theta(d^3)$ operations done on ciphertexts. Step 2 and 3 require $\Theta(d^2)$ decryptions and $\Theta(d^2)$ operations on plaintexts and therefore are faster (*e.g.,* less then 27 s for both the steps for a dataset of one hundred thousands instances with 40 features).

[15] For larger values of n and d, using Damgård and Jurik's scheme instead of Paillier's scheme reduces the running time of operations on ciphertexts. See [14, Appendix A.5].

Table 2. Running times (secs) for UCI datasets in the HP and VP settings.

	Dataset	n	d	ℓ	$\log_2(N)$	R_{MSE}	Phase 1		Phase 2	
							Time	kB	Time	kB
HP	Air	6252	13	1	2048	4.15E−09	1.99	53.24	3.65	96.51
	Beijing	37582	14	2	2048	5.29E−07	2.37	60.93	4.26	110.10
	Boston	456	13	4	2048	2.34E−06	2.00	53.24	3.76	96.51
	Energy	17762	25	3	2724	5.63E−07	12.99	238.26	37.73	451
	Forest	466	12	3	2048	3.57E−09	1.66	46.08	2.81	82.94
	Student	356	30	1	2048	4.63E−07	9.36	253.44	30.40	483.84
	Wine	4409	11	4	2048	2.62E−05	1.71	39.42	2.38	70.40
VP	Boston	456	13	4	2048	2.34E−06	123.76	$1.5\,10^3$	3.73	96.51
	Forest	466	12	3	2048	3.57E−09	115.04	$1.4\,10^3$	2.92	82.94
	Student	356	30	1	2048	4.63E−07	297.52	$2.7\,10^3$	30.54	483.84

To answer Question 3 and show the practicality of our system we report in Table 2 the total running time and communication overhead for seven different UCI datasets (references in [14, Appendix A.4]). Some of these datasets were used also in [12,23]. For example, [23] reports a running time of 45 s and a communication overhead of 83 MB (69 MB, resp.) for the Phase 2 of their system run on the dataset "forest" ("wine", resp.) ([23, Table I]). Our protocol Π_2 for the same datasets takes about 3 s with less then 83 kB sent. Phase 2 of the system presented in [12] runs on the dataset "student" in 19 s ([12, Table 3]) and we estimate an overhead of 3 GB (20 CGD iterations). Protocol Π_2 on the same dataset runs in about 40 s with 484 kB of overhead.

Acknowledgments. This work was partially supported by the Clinical and Translational Science Award (CTSA) program, through the NIH National Center for Advancing Translational Sciences (NCATS) grant UL1TR002373, and by the NIH BD2K Initiative grant U54 AI117924.

References

1. Agrawal, R., Srikant, R.: Privacy-preserving data mining. In: 2000 ACM SIGMOD International Conference on Management of Data, pp. 439–450. ACM Press (2000)
2. Aono, Y., Hayashi, T., Phong, L.T., Wang, L.: Fast and secure linear regression and biometric authentication with security update. Cryptology ePrint Archive, Report 2015/692 (2015)
3. Bar-Ilan, J., Beaver, D.: Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In: Eighth Annual ACM Symposium on Principles of Distributed Computing, pp. 201–209. ACM Press (1989)
4. Barbosa, M., Catalano, D., Fiore, D.: Labeled homomorphic encryption: scalable and privacy-preserving processing of outsourced data. In: Foley, S.N., Gollmann, D., Snekkenes, E. (eds.) ESORICS 2017. LNCS, vol. 10492, pp. 146–166. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66402-6_10

5. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 420–432. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-46766-1_34
6. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: 20th Annual ACM Symposium on Theory of Computing, STOC, pp. 1–10. ACM Press (1988)
7. Cao, Z., Liu, L.: Comment on "harnessing the cloud for securely outsourcing large-scale systems of linear equations". IEEE Trans. Parallel Distrib. Syst. **27**(5), 1551–1552 (2016)
8. Cock, M.D., Dowsley, R., Nascimento, A.C.A., Newman, S.C.: Fast, privacy preserving linear regression over distributed datasets based on pre-distributed data. In: 8th ACM Workshop on Artificial Intelligence and Security, pp. 3–14. ACM Press (2015)
9. Damgård, I., Jurik, M.: A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In: Kim, K. (ed.) PKC 2001. LNCS, vol. 1992, pp. 119–136. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44586-2_9
10. Du, W., Han, Y.S., Chen, S.: Privacy-preserving multivariate statistical analysis: linear regression and classification. In: Fourth SIAM International Conference on Data Mining, pp. 222–233. SIAM (2004)
11. Fouque, P.-A., Stern, J., Wackers, G.-J.: CryptoComputing with rationals. In: Blaze, M. (ed.) FC 2002. LNCS, vol. 2357, pp. 136–146. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36504-4_10
12. Gascón, A., Schoppmann, P., Balle, B., Raykova, M., Doerner, J., Zahur, S., Evans, D.: Privacy-preserving distributed linear regression on high-dimensional data. PoPETS **2017**(4), 248–267 (2017)
13. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: 41st Annual ACM Symposium on Theory of Computing, STOC, pp. 169–178. ACM Press (2009)
14. Giacomelli, I., Jha, S., Joye, M., Page, C.D., Yoon, K.: Privacy-preserving ridge regression with only linearly-homomorphic encryption. Cryptology ePrint Archive, Report 2017/979 (2017)
15. Hall, R., Fienberg, S.E., Nardi, Y.: Secure multiple linear regression based on homomorphic encryption. J. Off. Stat. **27**(4), 669–691 (2011)
16. Kamara, S., Mohassel, P., Raykova, M.: Outsourcing multi-party computation. Cryptology ePrint Archive, Report 2011/272 (2011)
17. Karr, A.F., Lin, X., Sanil, A.P., Reiter, J.P.: Regression on distributed databases via secure multi-party computation. In: 2004 Annual National Conference on Digital Government Research, pp. 108:1–108:2 (2004)
18. Karr, A.F., Lin, X., Sanil, A.P., Reiter, J.P.: Secure regression on distributed databases. J. Comput. Graph. Stat. **14**(2), 263–279 (2005)
19. Karr, A.F., Lin, X., Sanil, A.P., Reiter, J.P.: Privacy-preserving analysis of vertically partitioned data using secure matrix products. J. Off. Stat. **25**(1), 125–138 (2009)
20. Lindell, Y., Pinkas, B.: Privacy preserving data mining. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 36–54. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44598-6_3
21. McDonald, G.C.: Ridge regression. Wiley Interdiscip. Rev.: Comput. Stat. **1**(1), 93–100 (2009)
22. Mohassel, P., Zhang, Y.: SecureML: a system for scalable privacy-preserving machine learning. In: 2017 IEEE Symposium on Security and Privacy, pp. 19–38. IEEE Computer Society (2017)

23. Nikolaenko, V., Weinsberg, U., Ioannidis, S., Joye, M., Boneh, D., Taft, N.: Privacy-preserving ridge regression on hundreds of millions of records. In: 2013 IEEE Symposium on Security and Privacy, pp. 334–348. IEEE Computer Society (2013)
24. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48910-X_16
25. Sanil, A.P., Karr, A.F., Lin, X., Reiter, J.P.: Privacy preserving regression modelling via distributed computation. In: Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 677–682. ACM Press (2004)
26. Wang, C., Ren, K., Wang, J., Wang, Q.: Harnessing the cloud for securely outsourcing large-scale systems of linear equations. IEEE Trans. Parallel Distrib. Syst. **24**(6), 1172–1181 (2013)
27. Wang, P.S., Guy, M.J.T., Davenport, J.H.: *P*-adic reconstruction of rational numbers. ACM SIGSAM Bull. **16**(2), 2–3 (1982)
28. The International Warfarin Pharmacogenetics Consortium: Estimation of the Warfarin dose with clinical and pharmacogenetic data. N. Engl. J. Med. **360**(8), 753–764 (2009)
29. Yao, A.C.C.: How to generate and exchange secrets. In: 27th Annual Symposium on Foundations of Computer Science, FOCS, pp. 162–167. IEEE Computer Society (1986)