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*
[2] *NXP Semiconductors, San Jose, CA, USA*

February 1, 2018

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 modelling 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.

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 quantify the relationship between 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, because 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. This method is extremely popular (see the survey in [McD09]) and has found applications in several different fields, from biology [PQCF07] and medicine [NJFM14, War09] to economics and finance [LD15]. To enhance the efficacy of the learned model, prior experience in model 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

*This paper is a merge of [Joy17, GJPY17]

"data silos" collected by different entities. However, in many applications (*e.g.,* personalized medicine [War09]) 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 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).

If we assume the existence of a third-party trusted by all entities in the system, then this party can collect the data from each entity (in the clear) and performs collaborative analysis. However, finding agreement among all participants on trusting one third-party can be challenging. To avoid the need of trusting the third-party, we could use fully-homomorphic encryption [Gen09]: this allows to encrypt the data before outsourcing them to the third-party, that can still compute an arbitrary function on them. Unfortunately, fully-homomorphic encryption still has a large cost nowadays, making this strategy impractical for real-world applications. Here, we want to look to efficient solutions that limit the amount of trust to be placed on such third-party.

Our paper takes up the above challenge and proposes an efficient solution in the *two-server model* [KMR11], 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)[1] 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 [BGW88] or the 2-party protocol by Yao [Yao86]) 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 [NWI+13]. 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 as Paillier's cipher [Pai99]. We show this via an evaluation of our system that uses synthetically-generated and real-world data. Overall, our experiments show that, for many

[1] Another line of research focuses on studying and preventing the privacy threats that arise from releasing a model trained using private data. This is known as the differential privacy paradigm [Dwo06]. Our approach is orthogonal to differential privacy since we consider a different threat model.

real scenarios, LHE is all you need to *privately* yet efficiently train a ridge regression model on *distributed* data. As 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 minutes with the guarantee of confidentiality for the data silos of each research group involved (details can be found in Appendix A.4).

Related work The question of privacy-preserving machine learning was introduced in 2000 by two pioneering works [LP00, AS00]. Later on, privacy-preserving linear regression was considered in a number of different works (*e.g.,* [KLSR04, DHC04, SKLR04, KLSR05, KLSR09, HFN11, CDNN15, AHPW]). In 2013, Nikolaenko *et al.* [NWI+13] introduced the scenario we consider in this paper: privacy-preserving linear regression protocol in the two-server model. The solution in [NWI+13] 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 [NWI+13], 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 A and $\boldsymbol{b}$ 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 [GSB+17], the system presented in [NWI+13] 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.*[GSB+17] 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 [NWI+13] 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 [MZ17] using mini-batch stochastic gradient decedent, 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 [BCF17] 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.[2] 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 minutes[3] and produces a communication overhead of 1.3 MB. The system presented in

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

[NWI+13] needs more than 50 minutes and 270 MB exchanged data to perform a similar computation.[4] Finally, we notice that gradient descent based solutions (*e.g.,* [GSB+17, MZ17]) 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.

Roadmap In Section 2 we recall ridge linear regression and the cryptographic primitives involved in the design of our system. In Section 3 we describe the general framework of our system (*e.g.,* parties involved, security assumptions, security definitions, etc.). We also provide an overview of its design. In Section 4 we describe in detail the protocols that form our two-phase system. Finally, Section 5 reports on our implementation and experimental results.

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 $\boldsymbol{w}^{*\mathsf{T}}\boldsymbol{x}_i \approx y_i$ 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 $\boldsymbol{x}_i^\mathsf{T}$ 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.,* $\mathrm{rk}(X) = d$). Specifically, $\boldsymbol{w}^*$ is computed by minimizing a linear combination of the aforementioned error function and a regularization term, that is $\boldsymbol{w}^* \in \operatorname{argmin}_{\boldsymbol{w}\in\mathbb{R}^d} f_{X,\boldsymbol{y}}(\boldsymbol{w}) + \lambda R(\boldsymbol{w})$ 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, $\nabla F_{\text{ridge}}(\boldsymbol{w}) = 2X^\mathsf{T}(X\boldsymbol{w} - \boldsymbol{y}) + 2\lambda\boldsymbol{w}$, we have that $\boldsymbol{w}^*$ is computed solving the linear system

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

where $A = X^\mathsf{T}X + \lambda I$ (symmetric $d \times d$ matrix) and $\boldsymbol{b} = X^\mathsf{T}\boldsymbol{y}$ (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 the 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 to transform a message m from $\mathcal{M}$ (plaintext space) into a ciphertext, $c \leftarrow \mathsf{Enc}_{pk}(m)$.

[4] Timing on a 1.9 GHz 64 GB RAM machine running Linux 12.04; 80-bit security.

3. the decryption algorithm Dec is a deterministic function that takes as input a ciphertext and the secret key and recovers the original plaintext (*i.e.,* $\Pr[\mathsf{Dec}_{sk}(c) = m] = 1$ where the probability is taken over the encryption algorithm's random choice).

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). Known instantiations of this primitive include Paillier's scheme [Pai99], and its generalization by Damgård and Jurik [DJ01], Regev's scheme [Reg09] and Joye-Libert scheme [JL13].

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 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 [Gen09]. 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.* [BCF17] 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 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 ciphertext. 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 [BCF17, Section 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 ring, and let $F : \{0,1\}^s \times \mathcal{L} \to \mathcal{M}$ be a pseudo-random function with seed space $\{0,1\}^s$ ($s = \mathrm{poly}(\kappa)$) and input 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 $pk_i = \mathsf{Enc}_{pk}(\sigma_i)$. 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 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 $d_1, \ldots, d_n$, we can "add" them 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 computes $\sigma_i = \mathsf{Dec}_{sk}(pk_i)$ and $\sigma_j = \mathsf{Dec}_{sk}(pk_j)$. Then, 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 $\delta > 0$)[5] 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[6] 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 [WGD82, FSW02] 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 (2) in Section 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.

[5] In other words, $\delta = \max\{\|X\|_\infty, \|\boldsymbol{y}\|_\infty\}$ for the original X and $\boldsymbol{y}$.
[6] We assume that $\lambda \in \mathbb{R}$ has at most 2ℓ digits in the fractional part.

- 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 Section 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 Section 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 key. Phase 2 is realized by an interactive protocol for MLE and the CSP that 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 R and a random vector $\boldsymbol{r}$ 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".

- 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 [BB89], a similar method is 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 [WRWW13], 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.

We assume that the data parameters (*i.e.,* n, d, and δ), the system parameters (*i.e.,* ℓ, m and κ) and the regularization parameter λ are public values.

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 $\mathbb{Z}_N$.

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 $\mathcal{D}_k = \big\{(\boldsymbol{x}_{n_{k-1}+1}, y_{n_{k-1}+1}), \ldots, (\boldsymbol{x}_{n_k}, y_{n_k})\big\}$, for $k = 1, \ldots, m$ ($0 = n_0 < n_1 < \cdots < n_m = n$). In this case, as already noticed in [NWI+13], 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 Fig. 1).

Arbitrarily-partitioned setting Assume that each DO_k holds some elements of X and $\boldsymbol{y}$. That is, DO_k holds $\mathcal{D}_k = \big\{X[i,j] = \boldsymbol{x}_i[j] \mid (i,j) \in D_k\big\} \cup \big\{\boldsymbol{y}[i] = y_i \mid (i,0) \in D_k\big\}$, 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 $A[i,i] = \sum_{t=1}^{n} \boldsymbol{x}_t[i]\boldsymbol{x}_t[j] + \lambda$. To allow this, we use

[7] Notice that the system presented in [WRWW13] 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 [CL16] for more details on this.

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

Protocol $\Pi_{1,\mathrm{hor}}$

– *Parties*: CSP and MLE with no input, DO_k with input $\mathcal{D}_k$ (as defined in (3)).

– *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

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

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

labeled-homomorphic encryption. As we recalled in Section 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 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 $\Pi_{1,\mathrm{arb}}$ in Fig. 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

[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 $\Pi_{1,\text{arb}}$

– *Parties*: CSP and MLE with no input, DO_k with input $\mathcal{D}_k$ (as defined in (4)).

– *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 computes $\sigma_k = \mathsf{Dec}_{sk}(pk_k)$ and $b_{ij} = F(\sigma_k, (i, j))$ 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

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

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

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

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 Section 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 (Fig. 3). The correctness is easy to 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 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 as threshold encryption [DJ01]. 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 $\boldsymbol{w}^*$.

Protocol Π_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 $R \leftarrow (\mathbb{Z}_N^{d\times d})^*$ 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 $\tilde{\boldsymbol{w}} \equiv C^{-1}\boldsymbol{d} \mod N$ and sends it $\tilde{\boldsymbol{w}}$ to MLE.

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

Figure 3: Protocol Π_2 implements Phase 2.

4.2.1 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 [WGD82, FSW02] 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 (see Appendix A.1), 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 Section 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

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

4.2.2 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 Appendix A.3). In particular, our approach significantly improves the communication complexity compared to the previous solutions that use Yao's scheme [NWI+13, GSB+17]. Indeed, the latter requires CSP sending the garbled representation of a boolean circuit of millions of gates (see [NWI+13, Fig. 5] and [GSB+17, Fig. 7]) to MLE. In [NWI+13] 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. 4). For example, in the horizontally-partitioned setting [NWI+13] uses same techniques we deploy in $\Pi_{1,\text{hor}}$ and Yao's protocol. In particular, [NWI+13] 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 [GSB+17], a secure inner-product protocol based on additive secret-sharing and Beaver's triples [Bea91] 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, [GSB+17] 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 only 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 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 [MZ17] 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 [MZ17] has complexity $\Theta(nd + n)$. Thus, in terms of communication complexity, [MZ17] performs better than our solution in the arbitrarily-partitioned case. Our system, however, is preferable if the training dataset is horizontally-partitioned and $n >> d$ (*e.g.*, $n = \Theta(d^{2.5})$). For example, if $d = 100$ and $n = 10^5$ the system in [MZ17] has an overheard of 200 MB for the pre-processing phase only (see [MZ17, Table II]), while the total cost of $\Pi_{1,\text{hor}} + \Pi_2$ is less than 120 MB.

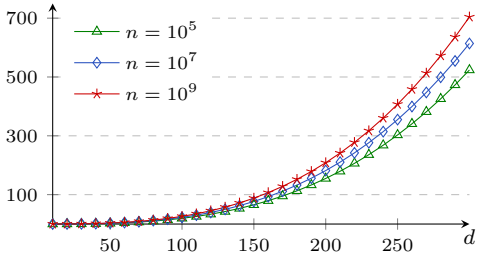

Figure 4: Communication overhead in MB of Π_2 ($\delta = 1$, 80-bit security, $\ell = 3$, Paillier's scheme, $\lambda = 0$).

5 Implementation

In this section we describe our implementation case study of the system described in Section 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. *Evaluate 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

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

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

Setup

We implemented our system using Paillier's scheme with message space $\mathcal{M} = \mathbb{Z}_N$ where N is a large RSA modulus (see Appendix A.2). In order to assure a security level of at least 100 bits[12], decrease the running time and the communication overhead, and satisfy (2), 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 (2). We wrote our software in Python3 5.2 using the phe1.3 library[13] to implement Paillier encryption/decryption and operations on ciphertexts. We use the gmpy2 library[14] to implement arithmetic operations with large integers. To compute the determinant function and to solve linear systems, we use the Gaussian elimination. 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 $\boldsymbol{y}$. In both cases, after having implemented Phase 1, we release variables no longer needed and we run the garbage collector to clean up memory before running Π_2.

Experiments results

All experiments whose results are reported in this section were run on a machine with the following specifics. OS : Scientific Linux7.4, CPU : 40 core (Intel(R) Xeon(R) CPU E5-2660 v2 2.20GHz), Memory: 500GB. All the timings are reported in seconds, all the values are averaged on 5 repetitions of the same experiment.

[11] `https://archive.ics.uci.edu/ml/datasets.html`
[12] According to NIST standard, a factoring 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 setting ($\ell = 3$).

	n	d	$\log_2(N)$	R_{MSE}	Phase 1			Phase 2		
					Step 1	Step 2	Step 3	Step 1	Step 2	Step 3
HP	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	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

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

	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 1, we measure R_{MSE} for different values of ℓ for synthetically-generated data (see Appendix A.4) in both the HP and VP setting (see Fig. 5). 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 (2) the value of this parameter 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 system efficiency.

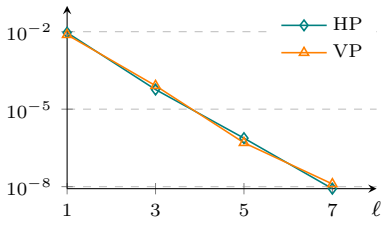

Figure 5: Error rate R_{MSE} (log scale) in function of ℓ ($n = 10^3, d = 10$).

To answer question 2 and assess the effect of the 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 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 the running time of $\Pi_{1,\text{hor}}$ and Π_2. Finally, for Π_2 the results in Table 1 show that Step 1 requires longer time respect 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 seconds for both the steps for a dataset of one hundred thousands instances with 40 features).

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 Table 5 in Appendix A.4). Some of these datasets were used also in [NWI+13] and [GSB+17]. For example, [NWI+13] reports a running time of 45 seconds and a communication overhead of 83 MB (69 MB, resp.) for the Phase 2 of their system run on the dataset "forest" ("wine", resp.) ([NWI+13, Table I]). Our protocol Π_2 for the same datasets has a running time of about 3 seconds with less then 83 kB sent. Phase 2 of the system presented in [GSB+17] runs on the dataset "student" in 19 seconds ([GSB+17, Table 3]) and we estimate an overhead of 3 GB (20 iterations for the CGD). Protocol Π_2 on the same dataset runs in about 40 seconds with 484 kB of overhead.

References

[AHPW] Yoshinori Aono, Takuya Hayashi, Le Trieu Phong, and Lihua Wang. Fast and secure linear regression and biometric authentication with security update. Cryptology ePrint Archive 2015/692.

[AS00] Rakesh Agrawal and Ramakrishnan Srikant. Privacy-preserving data mining. In *ACM SIGMOD*. ACM Press, 2000.

[BB89] Judit Bar-Ilan and Donald Beaver. Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In *8th Annual ACM Symposium on Principles of Distributed Computing*. ACM Press, 1989.

[BCF17] Manuel Barbosa, Dario Catalano, and Dario Fiore. Labeled homomorphic encryption: Scalable and privacy-preserving processing of outsourced data. In *ESORICS 2017*, volume 10492 of *LNCS*, pages 146–166. Springer, 2017.

[Bea91] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO '91*, volume 576 of *LNCS*. Springer, 1991.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *20th Annual ACM STOC*. ACM Press, 1988.

[CDNN15] Martine De Cock, Rafael Dowsley, Anderson C. A. Nascimento, and Stacey C. Newman. Fast, privacy preserving linear regression over distributed datasets based on pre-distributed data. In *8th ACM Workshop on Artificial Intelligence and Security*. ACM Press, 2015.

[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 Appendix A.5.

[CL16] Zhengjun Cao and Lihua Liu. Comment on "harnessing the cloud for securely outsourcin large-scale systems of linear equations". *IEEE Trans. Parallel Distrib. Syst.*, 27(5):1551–1552, 2016.

[DHC04] Wenliang Du, Yunghsiang S. Han, and Shigang Chen. Privacy-preserving multivariate statistical analysis: Linear regression and classification. In *Fourth SIAM International Conference on Data Mining*. SIAM, 2004.

[DJ01] Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In *PKC Proceedings*, 2001.

[Dwo06] Cynthia Dwork. Differential privacy. In *ICALP*, LNCS. Springer, 2006.

[FSW02] Pierre-Alain Fouque, Jacques Stern, and Jan-Geert Wackers. Cryptocomputing with rationals. In *Financial Cryptography*, volume 2357 of *LNCS*. Springer, 2002.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *41st Annual ACM STOC*. ACM Press, 2009.

[GJPY17] Irene Giacomelli, Somesh Jha, C. David Page, and Kyonghwan Yoon. Privacy-preserving ridge regression on distributed data. Cryptology ePrint Archive, Report 2017/707, 2017.

[Gol04] Oded Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.

[GSB+17] Adrià Gascón, Phillipp Schoppmann, Borja Balle, Mariana Raykova, Jack Doerner, Samee Zahur, and David Evans. Privacy-preserving distributed linear regression on high-dimensional data. *Proceedings on Privacy Enhancing Technologies*, 2017(4):248–267, 2017.

[HFN11] Rob Hall, Stephen E. Fienberg, and Yuval Nardi. Secure multiple linear regression based on homomorphic encryption. *J. of Official Statistics*, 27(4):669–691, 2011.

[JL13] Marc Joye and Benoît Libert. Efficient cryptosystems from 2^k-th power residue symbols. In *EUROCRYPT 2013*, LNCS. Springer, 2013.

[Joy17] Marc Joye. Privacy-preserving ridge regression without garbled circuits. Cryptology ePrint Archive, Report 2017/732, 2017.

[KLSR04] Alan F. Karr, Xiaodong Lin, Ashish P. Sanil, and Jerome P. Reiter. Regression on distributed databases via secure multi-party computation. In *2004 Annual National Conference on Digital Government Research*. Digital Government Society of North America, 2004.

[KLSR05] Alan F. Karr, Xiaodong Lin, Ashish P. Sanil, and Jerome P. Reiter. Secure regression on distributed databases. *J. of Computational and Graphical Statistics*, 14(2), 2005.

[KLSR09] Alan F. Karr, Xiaodong Lin, Ashish P. Sanil, and Jerome P. Reiter. Privacy-preserving analysis of vertically partitioned data using secure matrix products. *J. of Official Statistics*, 25(1), 2009.

[KMR11] Seny Kamara, Payman Mohassel, and Mariana Raykova. Outsourcing multiparty computation. Cryptology ePrint Archive, Report 2011/272, 2011.

[LD15] Boqiang Lin and Kerui Du. Measuring energy rebound effect in the Chinese economy: An economic accounting approach. *Energy Economics*, 50:96–104, 2015.

[LP00] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. In *CRYPTO 2000*, volume 1880 of *LNCS*. Springer, 2000.

[McD09] Gary C. McDonald. Ridge regression. *Wiley Interdisciplinary Reviews: Computational Statistics*, 1(1):93–100, 2009.

[MZ17] Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2017.

[NJFM14] Elias Chaibub Neto, In Sock Jang, Stephen H. Friend, and Adam A. Margolin. The Stream algorithm: Computationally efficient ridge-regression via Bayesian model averaging, and applications to pharmacogenomic prediction of cancer cell line sensitivity. In *Pacific Symposium on Biocomputing*. NIH Public Access, 2014.

[NWI+13] Valeria Nikolaenko, Udi Weinsberg, Stratis Ioannidis, Marc Joye, Dan Boneh, and Nina Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *2013 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2013.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT '99*, volume 1592 of *LNCS*. Springer, 1999.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2013.

[PQCF07] Eduardo da Cruz Gouveia Pimentel, Sandra Aidar de Queiroz, Roberto Carvalheiro, and Luiz Alberto Fries. Use of ridge regression for the prediction of early growth performance in crossbred calves. *Genetics and Molecular Biology*, 30(3):536–544, 2007.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6):34:1–34:40, 2009.

[SKLR04] Ashish P. Sanil, Alan F. Karr, Xiaodong Lin, and Jerome P. Reiter. Privacy preserving regression modelling via distributed computation. In *10th ACM SIGKDD*. ACM Press, 2004.

[War09] The International Warfarin Pharmacogenetics Consortium. Estimation of the Warfarin dose with clinical and pharmacogenetic data. *The New England J. Medecine*, 360(8):753–764, 2009.

[WGD82] Paul S. Wang, M. J. T. Guy, and James H. Davenport. P-adic reconstruction of rational numbers. *ACM SIGSAM Bulletin*, 16(2):2–3, 1982.

[WRWW13] Cong Wang, Kui Ren, Jia Wang, and Qian Wang. Harnessing the cloud for securely outsourcing large-scale systems of linear equations. *IEEE Trans. Parallel Distrib. Syst.*, 24(6):1172–1181, 2013.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *27th FOCS*. IEEE Computer Society, 1986.

A Appendix

A.1 Standard notations

For any integer $N > 1$, $\mathbb{Z}_N$ denotes the ring of integers modulo N and $\mathbb{Z}_N^*$ denotes its group of units. For an integer a, $a \bmod N$ represents the smallest integer in $\{0, 1, \ldots, N-1\}$ that is congruent to a modulo N. We use bold notation for vectors and capital letters for matrices (*e.g.,* $\boldsymbol{x} \in \mathbb{R}^n$ is a column vector, $X \in \mathbb{R}^{n \times d}$ is matrix with n rows and d columns, both with real-value entries). We indicate the i-th component of a vector $\boldsymbol{x}$ with $\boldsymbol{x}[i]$ and the i-th component of the j-th column of a matrix X with $X[i, j]$. The p-norm of a vector $\boldsymbol{x} \in \mathbb{R}^n$ is defined by $\|\boldsymbol{x}\|_p = \sqrt[p]{\sum_{i=1}^{n} |\boldsymbol{x}[i]|^p}$. The sup-norm of a matrix (or vector) $X \in \mathbb{R}^{n \times d}$ is defined by $\|X\|_\infty = \max_{i,j}\{|X[i, j]|\}$. If A is a $d \times d$ matrix, then the adjunct of A is defined as $\mathrm{adj}(A) = C^\intercal$ with $C[i, j] = (-1)^{i+j} A_{ij}$ and A_{ij} is the determinant of the $(d-1) \times (d-1)$ matrix that results from deleting row i and column j of A (*i.e.,* the (i, j) minor of A). Note that $\mathrm{adj}(A) = \det(A)A^{-1}$. Finally, we recall that Hadamard's inequality implies that $|\det(A)| \leq (\sqrt{d}\,\|A\|_\infty)^d$, and $\det(A) \leq \|A\|_\infty^d$ if A is positive definite.

A.2 Paillier's scheme

We briefly recall here Paillier's scheme [Pai99] used in the implementation presented in Section 5. Given a security parameter κ, Gen samples p and q, two prime integers of same bit-length, and defines $N = pq$ and $\nu = \mathrm{lcm}(p-1, q-1)$. It sets $pk = N$, $sk = \nu$ and $\mathcal{M} = \mathbb{Z}_N$, $\mathcal{C} = \mathbb{Z}_{N^2}^*$. To encrypt $m \in \mathcal{M}$, Enc randomly chooses r in $\mathbb{Z}_N^*$ and computes $c = (1 + mN)r^N \bmod N^2$. To decrypt $c \in \mathcal{C}$, Dec first computes $\bar{m} = \frac{(c^\nu \bmod N^2)-1}{N}$ and returns $m = \bar{m}/\nu \bmod N$. The correctness follows by observing that $c \equiv (1 + N)^m\, r^N \pmod{N^2}$ and that $(r^N)^\nu \equiv 1 \pmod{N^2}$.

For Paillier's scheme, $\odot$ is the standard product in $\mathbb{Z}_{N^2}^*$; indeed: $c_1 \cdot c_2 \equiv [(1 + N)^{m_1} r_1^N] \cdot [(1+N)^{m_2} r_2^N] \equiv (1+N)^{m_1+m_2}\,(r_1 r_2)^N \pmod{N^2}$ and $\mathsf{cMult}(a, c_1) \equiv (c_1)^a \equiv [(1+N)^{m_1} r_1^N]^a \equiv (1 + N)^{am_1}(r_1^a)^N \pmod{N^2}$.

A.3 Complexity analysis

Table 3 presents the communication complexity in terms of number of plaintexts and ciphertexts sent at each step. We use the following public parameters: n (number of instances), d (total number of features) and d_k (in the arbitrarily-partitioned setting, the data-owner DO_k holds d_k entries of $X, \boldsymbol{y}$. That is d_k is the size of D_k). Notice that $\sum_{k=1}^{m} d_k = d(n+1)$. Notice that because of the use of rational reconstruction, the bit-length of a plaintext (and therefore also the bit-length of a ciphertext) depends on the parameters n and d (see Eq. (2)). It follows that both Protocol $\Pi_{1,\mathrm{hor}}$ and Protocol Π_2 have complexity $O(d^3 \log(nd))$ bits, while Protocol $\Pi_{1,\mathrm{arb}}$ has complexity $O((nd^2 + d^3)\log(nd))$ bits (we assume ℓ and m constant).

Table 4 summarizes the computational complexity in terms of number of elementary operations (*e.g.,* arithmetic operations on plaintexts, arithmetic operations on ciphertexts, encryptions, decryptions, etc.). Beside the aforementioned public parameters n and d we use m (number of data-owners) and n_k (in the horizontally-partitioned setting, the data-owner DO_k holds the instances from $n_{k-1}+1$ to n_k). The notation "mult." (resp."add.") represents a multiplication (resp. an addition) on plaintext messages (*i.e.,* with the arithmetic of $\mathbb{Z}_N$); "enc-add." represents for one operation $\odot$ on ciphertexts. Notice that the number of features d influences the computational complexity of all the steps of our system, while the parameter n influences the complexity of some of the steps in Phase 1 only (specifically, Step 2 of $\Pi_{1,\mathrm{hor}}$ and Step 2 and 3 of $\Pi_{1,\mathrm{arb}}$). Since, each operation considered for Table 4 has a different execution time (*e.g.,* operation on plaintexts are much faster than operation on ciphertexts),

Table 3: Summary of communication complexity.

$\Pi_{1,\text{hor}}$ (Fig. 1)	– CSP sends pk to each party – DO_k sends $\frac{d(d+1)}{2} + d$ ciphertexts to MLE
$\Pi_{1,\text{arb}}$ (Fig. 2)	– CSP sends pk to each party – DO_k sends pk_k to CSP and MLE – CSP sends $\frac{d(d+1)}{2} + d$ ciphertexts to MLE – DO_k sends d_k ciphertexts and d_k plaintexts to MLE
Π_2 (Fig. 3)	– MLE sends $d^2 + d$ ciphertexts to CSP – CSP sends d plaintexts to MLE

Table 4: Summary of computational complexity.

	Step 1		Step 2 (DO_k)	Step 3 (MLE)
$\Pi_{1,\text{hor}}$	CSP 1 execution of Gen	DO_k	$(n_k - n_{k-1})\left(\frac{d(d+1)}{2} + d\right)$ mult. $(n_k - n_{k-1})\left(\frac{d(d+1)}{2} + d\right)$ add. $\frac{d(d+1)}{2} + d$ enc.	$m\left(\frac{d(d+1)}{2} + d\right)$ enc-add.
$\Pi_{1,\text{arb}}$	1 execution of Gen m dec. $\frac{d(d+1)}{2} + d$ enc.	1 enc.	d_k add. d_k enc.	$n\left(\frac{d(d+1)}{2} + d\right)$ mult. $n\left(\frac{d(d+1)}{2} + d\right)$ enc. $2n\left(\frac{d(d+1)}{2} + d\right)$ cMult $(3n+1)\left(\frac{d(d+1)}{2} + d\right)$ enc-add.
	Step 1 (MLE)		Step 2 (CSP)	Step 3 (MLE)
Π_2	$d^3 + d^2 + d$ enc-add. $d^3 + d^2$ cMult		$d^2 + d$ dec. 1 solution of $d \times d$ linear system	$d^2 + d$ add. d^2 mult. d rational reconstruction

for concrete running times we refer to Section 5, where results of the implementation of our system are presented.

A.4 More details on experiments

Computational resource All experiments were executed on a machine with 500 GB of RAM and 40 core CPU. Our goal was to simulate a setting where the CSP and data-owners have available only commodity servers, whereas MLE use a more powerful machine. Thus, we run CSP and each DO_k using a single core of the machine, and we run MLE using parallel computing through multi-core processors.

Synthetic datasets To evaluate the effect of the parameters on our system's performance we run experiments on synthetically generated datasets. For any pair of n and d, each $\boldsymbol{x}_i$ is sampled from a standard d-dimensional Gaussian distribution ($i = 1, \ldots, n$). The coefficients of the vector $\boldsymbol{w}^*$ are sampled independently and uniformly at random from the real interval $[0, 1]$. Finally $y_i = \boldsymbol{x}_i^\intercal \boldsymbol{w}^* + \epsilon_i$, where ϵ_i is sampled from a Gaussian distribution with zero mean and variance $\sigma^2 = 0.1$. As suggested by the statistical theory, the regularization parameter λ is set to $\sigma^2 d/n \|\boldsymbol{w}^*\|_2^2$. The error rate R_{MSE} is computed using a test set sampled from the same distribution and with size equal to $n/10$.

Table 5: References for the UCI datasets.

Dataset	Reference
forest	https://archive.ics.uci.edu/ml/datasets/Forest+Fires
boston	https://archive.ics.uci.edu/ml/datasets/housing
facebook	https://archive.ics.uci.edu/ml/datasets/Facebook+metrics
air	https://archive.ics.uci.edu/ml/datasets/Air+Quality
energy	https://archive.ics.uci.edu/ml/datasets/Appliances+energy+prediction
beijing	https://archive.ics.uci.edu/ml/datasets/Beijing+PM2.5+Data
wine	https://archive.ics.uci.edu/ml/datasets/Wine+Quality
bike	https://archive.ics.uci.edu/ml/datasets/bike+sharing+dataset
student	http://archive.ics.uci.edu/ml/datasets/student+performance

UCI datasets In Section 5, we ran our system on different real-world datasets downloaded from the UCI repository. References about each one of the datasets are given in Table 5. For each dataset, we removed the data points with missing values and we use 1-of-k encoding to convert nominal features to numerical ones. Moreover we split the dataset in two subset: the first one, with 90% of the total instances, is used for training and the complement is used to compute R_{MSE}. The regularization parameter λ is computed using cross-validation. The results of our experiments on the UCI datasets are reported in Table 2 in Section 5. For Phase 1 in the HP setting, we report the total running time (in seconds) of $\Pi_{1,\text{hor}}$ assuming that the 10 data-owners execute Step 2 in parallel, and the size in kilobytes of the message sent from one data-owner to MLE. For Phase 1 in the VP setting, we report the total running time (in seconds) of $\Pi_{1,\text{arb}}$ assuming that the 3 data-owners execute Step 2 in parallel, and the average size in kilobytes of the message sent from one data-owner to MLE (in Step 2) plus the message sent from CSP to MLE (setup in Step 1). In both cases, for Phase 2 we report the total running time (in seconds) of Π_2 and the overall communication required between CSP and MLE (in kilobytes).

Warfarin experiment We want to prove the concrete utility of our system considering here its application to an existing medical scenario: the *Warfarin dosing model*. Warfarin is a popular anticoagulant, used for instance to prevent stokes in patients suffering from atrial fibrillation. In 2009 the International Warfarin Pharmacogenetics Consortium (IWPC 2009)[16] proposed an accurate dosing model trained using linear regression on a database containing clinical and genetic information of 4043 patients. The database was the result of the merge of the data of different patients collected by 21 research groups. The model proposed by IWPC 2009 was tested on a validation cohort of 1009 patients, on which it achieved a MAE (mean absolute error) of 8.5 mg per week (as baseline, notice that a fixed-dose approach of 35 mg per week has a MAE of 13 mg per week). We downloaded[17] the database used for this study and, after removing the instances with missing values, we randomly split it in a training set (80%) and validation set (20%). We run our system (Protocol $\Pi_{1,\text{hor}}$ + Protocol Π_2) with $m = 21$ and $\ell = 5$ on the training set and we compute the MAE of the learned model using the validation set. The average result of this experiment on 30 repetitions is a MAE of 8.8 mg per week. That is, the MAE increases of 3.35% only. Notice that our system in this setting runs in less than 3 minutes on a commodity server[18] and produces an overall communication overhead of less then 2.5 MB.

[16] https://www.pharmgkb.org/page/iwpc

[17] `https://www.pharmgkb.org/downloads`

[18] Timing on a 2.6 GHz 8 GB RAM machine running Linux 16.04. 80-bit security.

A.5 Further optimization

When the logarithm in base 2 of the left-hand of (2) is greater or equal to 4095 (*i.e.,* $\beta \geq 4095$), in order to reduce the communication overhead of all the protocols and the running time of the steps involving operations on ciphertexts ($\odot$ and cMult) we can use Damgård and Jurik's scheme [DJ01]. This cryptosystem is a generalization of the Paillier's scheme cryptosystem that has message space $\mathbb{Z}_{N^s}$ and ciphertext space $\mathbb{Z}_{N^{s+1}}$ where N is an RSA modulus and s is (positive) natural number (Paillier's scheme is the special case with $s = 1$). If $\beta \geq 4095$ we can use Damgård and Jurik's scheme with $s = \left\lfloor \frac{\beta+1}{2048} \right\rfloor \geq 2$ and $\log_2(N) = 2048$. With this choice of parameters, Damgård and Jurik's still guarantees 100-bit security and works with ciphertexts of bit length $\leq \beta + 2049$. While Paillier's scheme with $\log_2(N) = \lfloor \beta \rfloor + 1$ works with ciphertexts of bit length $2(\lfloor \beta \rfloor + 1)$.

A.6 Security proof

Let $\left(\mathcal{M}^{d\times d}\right)^*$ be the set of all invertible matrices with coefficients in the ring $\mathcal{M}$.

Lemma A.1. Let $(A, \boldsymbol{b}) \in \left(\mathcal{M}^{d\times d}\right)^* \times \mathcal{M}^d$. Assume that R is sampled uniformly at random from $\left(\mathcal{M}^{d\times d}\right)^*$ and that $\boldsymbol{r}$ is sampled uniformly at random from $\mathcal{M}^d$. Then, the distribution of $(AR, \boldsymbol{b} + A\boldsymbol{r})$ is the uniform distribution over $\left(\mathcal{M}^{d\times d}\right)^* \times \mathcal{M}^d$.

Proof. Fix $(M, \boldsymbol{v}) \in \left(\mathcal{M}^{d\times d}\right)^* \times \mathcal{M}^d$, then

$$\begin{aligned}\Pr[AR = M, \boldsymbol{b} + A\boldsymbol{r} = \boldsymbol{v}] &= \Pr[R = A^{-1}M, \boldsymbol{r} = A^{-1}(\boldsymbol{v} - \boldsymbol{b})] \\ &= \frac{1}{|\left(\mathcal{M}^{d\times d}\right)^* \times \mathcal{M}^d|} \ .\end{aligned}$$

□

To formally prove security, we use the standard simulation-based definition [Gol04]. Consider a public function $\phi : (\{0,1\}^k)^n \to \{0,1\}^\ell$ and let $P_1, \ldots, P_n$ be n players modelled as PPT machines. Each player P_i holds the value $\boldsymbol{a}_i \in \{0,1\}^k$ and wants to compute the value $\phi(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n)$ while keeping his input private. The players can communicate among them using point-to-point secure channels in the synchronous model. If necessary, we also allow the players to use a broadcast channel. To achieve their goal, the players jointly run a *n-party MPC protocol* Π. The latter is a protocol for n players that is specified via the next-message functions: there are several rounds of communication and in each round the player P_i sends to other players a message that is computed as a deterministic function of the internal state of P_i (his initial input $\boldsymbol{a}_i$ and his random tape $\boldsymbol{k}_i$) and the messages that P_i has received in the previous rounds of communications. The *view* of the player P_j, denoted by $\mathsf{View}_{P_j}(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n)$, is defined as the concatenation of the private input $\boldsymbol{a}_j$, the random tape $\boldsymbol{k}_j$ and all the messages received by P_j during the execution of Π. Finally, the output of Π for the player P_j can be computed from the view View_{P_j}. In order to be private, the protocol Π needs to be designed in such a way that a curious player P_i cannot infer information about $\boldsymbol{a}_j$ with $j \neq i$ from his view $\mathsf{View}_{P_i}(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n)$.

More precisely, we have the following definition.

Definition A.1 ([Gol04, Definition 7.5.1])**.** We say that the protocol Π realizes ϕ with *correctness* if for any input $(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n)$, it holds[19] that $\Pr \phi(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n) \neq$ output of Π for $P_i = 0$ for all $i \in [n]$. Let $\mathcal{A}$ a subset of at most $n - 1$ players, the protocol Π_f realizes ϕ with *privacy* against $\mathcal{A}$ if it is correct and there exists a PPT algorithm Sim such that $(\mathsf{View}_{P_i}(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n))_{P_i \in \mathcal{A}}$ and $\mathsf{Sim}((\boldsymbol{a}_i)_{P_i \in \mathcal{A}}, \phi(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n))$ are computationally indistinguishable for all inputs.

[19] The probability is over the choice of the random tapes $\boldsymbol{k}_i$.

Protocol Π

– *Parties*: CSP and MLE with no input, DO_i with input $\mathcal{D}_i$ for $i = 1, \ldots, m$.

– *Output*: each party gets $\boldsymbol{w}^*$

Phase 1: MLE, CSP and $\mathrm{DO}_1, \ldots, \mathrm{DO}_m$ jointly run $\Pi_{1,\mathrm{hor}}$ or $\Pi_{1,\mathrm{arb}}$.

Phase 2: – MLE and CSP jointly execute Π_2;
– MLE sends $\boldsymbol{w}^*$ to the other parties.

Figure 6: Protocol Π implements our system.

The protocol Π described in Fig. 6, which summarizes the privacy preserving system described in Section 4, can be seen as an MPC for $m+2$ parties: $\mathrm{DO}_1, \ldots, \mathrm{DO}_m$, MLE and CSP. The input of DO_k is $\mathcal{D}_k$ defined by:

$$\mathcal{D}_k = \left\{(\boldsymbol{x}_{n_{k-1}+1}, y_{n_{k-1}+1}), \ldots, (\boldsymbol{x}_{n_k}, y_{n_k})\right\} \tag{3}$$

with $0 = n_0 < n_1 < \cdots < n_m = n$ for the horizontally-partitioned (HP) setting, and by:

$$\mathcal{D}_k = \left\{X[i,j] = \boldsymbol{x}_i[j] \mid (i,j) \in D_k\right\} \cup \left\{\boldsymbol{y}[i] = y_i \mid (i,0) \in D_k\right\} \tag{4}$$

where $D_k \subseteq \{1, \ldots, n\} \times \{0, 1, \ldots, d\}$ for the arbitrarily-partitioned setting. MLE and CSP have no input. Notice that we assume here that all the entries in the local dataset are integers number in in the interval $[-10^\ell\delta, 10^\ell\delta]$ (see Section 2). Moreover we assume that $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a LHE scheme with plaintext space $\mathcal{M} = \mathbb{Z}_N$ and that Eq. (2) is satisfied. Finally define ϕ the function that computes the ridge regression model from the data in the clear ($\phi(\mathcal{D}_1, \ldots, \mathcal{D}_m) = A^{-1}\boldsymbol{b}$). With this assumption we have the following.

Theorem 1. *Let $D \subset \{1, \ldots, m\}$, then Π (Fig. 6) realizes ϕ with correctness and privacy against the adversaries $\mathcal{A}_1 = \{\mathrm{MLE}\} \cup \{\mathrm{DO}_i \mid i \in D\}$ and $\mathcal{A}_2 = \{\mathrm{CSP}\} \cup \{\mathrm{DO}_i \mid i \in D\}$.*

Proof. Correctness: Using the homomorphic properties of the underlying encryption scheme, it easy to verify that at the end of Phase 1 of Π, the MLE knows A' and $\boldsymbol{b}'$ such that $\mathsf{Dec}_{sk}(A') = A$ and $\mathsf{Dec}_{sk}(\boldsymbol{b}') = \boldsymbol{b}$. It is also easy to verify that in Step 3 in Π_2 we have

$$\begin{aligned}\tilde{\boldsymbol{w}}^* = R\tilde{\boldsymbol{w}} - \boldsymbol{r} = R(C^{-1}\boldsymbol{d}) - \boldsymbol{r} \mod N &= R((AR)^{-1} + (\boldsymbol{b} + A\boldsymbol{r})) \mod N \\ &= A^{-1}\boldsymbol{b} \mod N\end{aligned}$$

Since Eq. (2) is satisfied, applying the rational reconstruction to $\tilde{\boldsymbol{w}}^*$ we obtain the model $\boldsymbol{w}^* = A^{-1}\boldsymbol{b}$ in $\mathbb{Q}^d$.

Privacy: To prove privacy we construct two simulators Sim_1 and Sim_2 which simulate the view of the parties in $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively. Let $\bar{\boldsymbol{w}}^* = \phi(\mathcal{D}_1, \ldots, \mathcal{D}_m)$.

$\mathsf{Sim}_1(\{\mathcal{D}_i\}_{i\in D}, \boldsymbol{w}^*)$ in the HP setting is defined by the following steps:

1. Run $(pk, sk) \leftarrow \mathsf{Gen}(\kappa)$;
2. For all $k = 1, \ldots, m$, if $k \in D$ compute A'_k and $\boldsymbol{b}'_k$ as in Step 2 of $\Pi_{1,\mathrm{hor}}$. Otherwise compute A'_k and $\boldsymbol{b}'_k$ as component-wise encryption of the identity $d \times d$ matrix and the zero vector (d components) ($i = n_{k-1} + 1, \ldots, n_k$);

3. Sample R and $\boldsymbol{r}$ as in the protocol;
4. Compute $\tilde{\boldsymbol{w}} = R^{-1}(\boldsymbol{w}^* + \boldsymbol{r}) \mod N$;
5. Output $(\{\mathcal{D}_i\}_{i\in D}, pk, enc, \tilde{\boldsymbol{w}}, \boldsymbol{w}^*)$ where enc contains the encryptions of step (2).

$\mathsf{Sim}_1(\{\mathcal{D}_i\}_{i\in D}, \boldsymbol{w}^*)$ in the arbitrarily-partitioned setting is defined by the following steps:

1. Run $(pk, sk) \leftarrow \mathsf{Gen}(\kappa)$ and run $(pk_i, sk_i) \leftarrow \mathsf{labGen}(\kappa)$ for $i = 1, \ldots, m$;
2. For all $k = 1, \ldots, m$, if $k \in D$ compute $\boldsymbol{c}_{ij}$ for all $(i, j) \in D_k$ as in Step 2 of $\Pi_{1,\text{arb}}$. Otherwise compute $\boldsymbol{c}_{ij}$ as encryption of 0.
3. Sample R and $\boldsymbol{r}$ as in the protocol;
4. Compute $\tilde{\boldsymbol{w}} = R^{-1}(\boldsymbol{w}^* + \boldsymbol{r}) \mod N$;
5. Output $(\{\mathcal{D}_i\}_{i\in D}, pk, \{pk_i\}_{i=1,\ldots,m}, enc, \tilde{\boldsymbol{w}}, \boldsymbol{w}^*)$ where enc contains the encryptions of step (2).

It follows from the semantic security of the encryption scheme that the simulation output has the same distribution of the views of the corrupted parties in $\mathcal{A}_1$ in the protocol Π.

$\mathsf{Sim}_2(\{\mathcal{D}_i\}_{i\in D}, \boldsymbol{w}^*)$ is defined by the following steps:

1. Run $(pk, sk) \leftarrow \mathsf{Gen}(\kappa)$;
2. Sample R and $\boldsymbol{r}$ as in the protocol;
3. Compute $\mathsf{Enc}_{pk}(R)$ and $\mathsf{Enc}_{pk}(\boldsymbol{r})$;
4. Output $(\{\mathcal{D}_i\}_{i\in D}, pk, \mathsf{Enc}_{pk}(R), \mathsf{Enc}_{pk}(\boldsymbol{r}), \boldsymbol{w}^*)$

It follows from Lemma A.1 that the simulation output has the same distribution of the views of the corrupted parties in $\mathcal{A}_2$ in the protocol Π. □

A.7 Beyond passive security

The protocols described in Sections 4.1 and 4.2 guarantee privacy when all the parties follow the protocol (passive security). Here we briefly discuss the security of these protocols in the case when the CSP or the MLE are corrupted and arbitrarily deviate from the protocol. We still assume that they do not collude.

If we have the guarantee that MLE always follows the protocol and only the CSP can be malicious, an easy solution is possible. First of all, notice that we can assume that the encryption scheme is initialized in the correct way and that all users obtain a valid public key using standard techniques as Certificate Authorities. Nevertheless, if CSP is corrupted, in Phase 2 (Protocol Π_2) it can send to the MLE a faulty $\tilde{\boldsymbol{w}}$ causing the computation of a wrong model. To prevent this, it is enough to add a simple verification step run by the MLE at the end of Π_2. Assume that $\tilde{\boldsymbol{w}}^*$ is the model in $\mathbb{Z}_N^d$ computed by MLE during Step 3 in Π_2. In other words, $\tilde{\boldsymbol{w}}^* = R\tilde{\boldsymbol{w}} - \boldsymbol{r} \mod N$ where $\tilde{\boldsymbol{w}}$ is the –possibly wrong– masked model sent by the CSP. Since in ridge regression the model is computed as solution of the system (1), if $\tilde{\boldsymbol{w}}^*$ satisfies $A\tilde{\boldsymbol{w}}^* - \boldsymbol{b} = 0$ in $\mathbb{Z}_N^d$, then the MLE has the correct model. Recall that at the end of Phase 1 the MLE gets the encryptions of the entries of the matrix A and the vector $\boldsymbol{b}$. Therefore, the MLE can easily compute the encryption of the components of the vector $A\tilde{\boldsymbol{w}}^* - \boldsymbol{b}$ in $\mathbb{Z}_N^d$ using the homomorphic property of the underlying encryption scheme. At this point the problem of checking the validity of $\tilde{\boldsymbol{w}}^*$ is equivalent to the following problem: the MLE (honest party) has a ciphertext $\mathsf{Enc}_{pk}(x)$ and it needs to be convinced that $x = 0$ by interacting with the CSP who knows sk but is malicious. This can be easily solved in

the following manner: MLE samples $r \leftarrow \mathbb{Z}_N$, computes $c = \mathsf{Enc}_{pk}(x) \odot \mathsf{Enc}_{pk}(r)$ and sends c to CSP. The latter decrypts and sends the result back to MLE, who accepts the proof if and only if the received value is equal to r. If $x \neq 0$, the probability of a malicious CSP to convince MLE of the opposite is $1/N$.

The case when also the MLE can be malicious is more involved. If the MLE is corrupted, then it can decide to ignore (o replace with encryption of dummy values) the ciphertexts received during Phase 1 from some of the m data-owners. In this way, at the end of Phase 2 the MLE learns a model that is trained only on the data from a small subset of the m data-owners (potentially only one them); such a model may reveal extra information about the private datasets held by these users. Moreover, if the verification step described before is implemented, then a malicious MLE could use the CSP as decryption oracle and break the privacy of the data-owners. Therefore, we need to find another solution that guarantees the correctness of $\tilde{\boldsymbol{w}}^*$. These issues can be mitigated considering the modification of our system described at the end of Section 4.2. There we show how to use threshold LHE to avoid releasing the model to MLE. In this case, the data-owners use the shared decryption functionality to compute $\boldsymbol{w}^*$, while MLE only sees $\mathsf{Enc}_{pk}(\boldsymbol{w}^*)$. In particular, 1) since MLE does not see the model in the clear, it can not use it to gain information about the private training datasets, 2) each data-owner can locally check the model on a local validation set. In this way, they indirectly check the computation of $\tilde{\boldsymbol{w}}^*$.
If we prefer not to modify the system, active security can be achieved by a mechanism that requires 1) CSP to prove in zero knowledge to MLE the correctness of $\tilde{\boldsymbol{w}}^*$, 2) MLE to prove in zero knowledge to CSP that $\mathsf{Enc}_{pk}(C)$ and $\mathsf{Enc}_{pk}(\boldsymbol{d})$ are computed in the correct way using the ciphertexts from all the DOs (*e.g.,* $\mathsf{Enc}_{pk}(A_1), \ldots, \mathsf{Enc}_{pk}(A_m)$ for the horizontally-partitioned case). A mechanism with this property can be obtained using a zero-knowledge argument protocol for proving generic statements (*e.g.,* [PHGR13]). Investigating this enhancement in details is left for future work.