Privacy-Preserving Ridge Regression over Distributed Data from LHE[*]

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*

October 6, 2017

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 current 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 linearly-homomorphic encryption (*i.e.,* without using Yao's protocol). This greatly improves the overall performance (both in computation and communications) 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. More precisely, given the data points $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)$ where $\boldsymbol{x}_i = (\boldsymbol{x}_i[1], \ldots, \boldsymbol{x}_i[d])$ is a vector of d real values (features) and y_i is a real value (outcome), a linear regression method is a learning algorithm for finding a vector $\boldsymbol{w} = (\boldsymbol{w}[1], \ldots, \boldsymbol{w}[d])$ (the model) with d real components such that the value $\boldsymbol{w}[1] \cdot \boldsymbol{x}_i[1] + \cdots + \boldsymbol{w}[d] \cdot \boldsymbol{x}_i[d]$ is "close" to y_i for all $i = 1, \ldots, n$. Despite its simple definition, a linear regression model is very useful. Indeed, $\boldsymbol{w}$ 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, $\boldsymbol{w}$ can be used to make a prediction about it). *Ridge regression* (detailed in the next section) 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] to economics and finance [LD15].

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

In the standard statistics setting, it is assumed that the party performing the regression has direct access to all the data points in the training set in order to compute the model. This common assumption becomes non-trivial in some relevant areas where linear regression is applied (*e.g.,* personalized medicine [C$^+$09]) because the data points encode *sensitive* information owned by different and possibly mutually distrustful entities. Often, these entities will not (or cannot) share the private data contained in their "data silos", making traditional linear regression algorithms difficult (or even impossible) to apply. On the other hand, it is known that having more data (more relevant features and/or more data points) typically improves the ability to compute a reliable model. 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 different 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 in this work: *training a linear regression model on data points that must be kept confidential and/or are owned by multiple parties.*

Our paper takes up this challenge and proposes an efficient solution in the setting in which the training set is a combination of data input by different parties (data-owners). Specifically, we consider both *horizontally-partitioned* datasets 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) and *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). Our system is based only on a simple cryptographic primitive that can be implemented via efficient constructions. Indeed, our solution, in both the aforementioned settings, is designed using just a *linearly-homomorphic encryption* 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, LP09]) or on somewhat-homomorphic encryption (*i.e.,* encryption schemes that allow homomorphic computation for two operations, for example addition and multiplication, on ciphertexts). 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* (LHE). We believe that this result is interesting both from the theoretical and the practical points of view. Indeed our system can be seen as 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 a broad evaluation of our system using synthetically-generated and real-world data. Overall, our experiments show that, for many real scenarios, LHE is all you need to *privately* yet efficiently train a ridge regression model on *distributed* data.

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, AHPW15]). In 2013, Nikolaenko *et al.* [NWI$^+$13] introduced the scenario we consider in this paper: privacy-preserving linear regression protocol in the *two-server model.* In this setting, the computation is outsourced to two *non-colluding* (but not necessarily trusted) third parties. This setting offers the practical advantage that the involvement of data-owners is minimal: they just need to submit their private data and do not participate in any further future interaction. After a first phase of collecting private data from possibly many data-owners, the two third parties then engage in a second phase for the computation of the model itself. Neither one of these two parties have to handle the input data in the clear and the system is designed in such a way that no extra information (beside that released by the model itself) is revealed to these two parties (assuming that they do not collude). The solution in [NWI$^+$13] considers the horizontally-partitioned setting for ridge regression and is based on LHE and Yao's protocol [Yao86, LP09]. 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 circuit C is the one that solves a linear system computing the Cholesky decomposition of the coefficient matrix. Recently, the system presented in [NWI$^+$13] was extended to vertically-partitioned setting by the paper [GSB$^+$17]. Gascón *et al.* achieve this result using multiparty computation (MPC) techniques to allow the data-owners to distribute shares of the merged dataset to the two parties active in the second phase. Moreover, Gascón *et al.* also improve the running time of the second phase of the protocol presented in [NWI$^+$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] and extended to logistic regression and neural networks.

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 the vertically-partitioned setting using the techniques of labelled-homomorphic encryption [BCF17] to support multiplications among pairs of ciphertexts encrypted via a LHE scheme. For the second phase, we get rid of Yao's protocol by designing an ad-hoc two-party protocol that solves the linear system $A\boldsymbol{w} = \boldsymbol{b}$ using only the linear homomorphic property of the underlying encryption scheme. This allows to boost the overall performance. 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[1] and produces a communication overhead[2] of 1.3 MB. The system presented in [NWI$^+$13] needs more than 50 minutes and 270 MB exchanged data to perform a similar computation.[3]

Roadmap

In Section 2 we start our presentation recalling ridge linear regression and we provide the definitions of 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

2.1 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 any integer a, $a \mod N$ represents the smallest integer in $\{0, 1, \ldots, N-1\}$ that is congruent to a modulo N.

[1]Timing on a 2.6 GHz 8 GB RAM machine running Linux 16.04.

[2]Size of the messages exchanged among the parties running the system.

[3]Timing on a 1.9 GHz 64 GB RAM machine running Linux 12.04

We use bold notation for vectors and capital letter 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 the 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 a 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 $\operatorname{adj}(A) = C^\intercal$ 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 $\operatorname{adj}(A) = \det(A)A^{-1}$. Finally, we recall that Hadamard's inequality implies that $|\det(A)| \le (\sqrt{d}\,\|A\|_\infty)^d$ for any $d \times d$ matrix A, and $\det(A) \le \prod_{j=1}^d A[j,j] \le \|A\|_\infty^d$ if A is positive definite.

2.2 Linear ridge regression

A linear regression learning algorithm is a procedure that on input n points $\{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_n, y_n)\}$ (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 $i = 1, \dots, n$. 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):

$$f_{X,\boldsymbol{y}}(\boldsymbol{w}) = \|X\boldsymbol{w} - \boldsymbol{y}\|_2^2$$

where $X \in \mathbb{R}^{n \times d}$ 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. Notice that in this work we assume that X is always full-rank (*i.e.,* $\operatorname{rk}(X) = d$). Specifically, $\boldsymbol{w}^*$ is computed by minimizing a linear combination of the aforementioned error function and a regularization term,

$$\boldsymbol{w}^* \in \operatorname*{argmin}_{\boldsymbol{w} \in \mathbb{R}^d} f_{X,\boldsymbol{y}}(\boldsymbol{w}) + \lambda R(\boldsymbol{w})$$

where $\lambda \ge 0$ 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

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

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).

2.3 Cryptographic tools

To design our new privacy-preserving system for linear regression, we employ 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 key, $(sk, pk) \leftarrow \mathsf{Gen}(\kappa)$.

2. the encryption algorithm Enc is a randomized algorithm that uses the public key to transform a message m from $\mathcal{M}$ (plaintext space) in a ciphertext, $c \leftarrow \mathsf{Enc}_{pk}(m)$.

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 $c_1 \leftarrow \mathsf{Enc}_{pk}(m_1), \dots, c_a \leftarrow \mathsf{Enc}_{pk}(m_a)$ (a 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 $c = \mathsf{Enc}_{pk}(m)$ and a is a positive integer, $\mathsf{Dec}_{sk}(\mathsf{cMult}(a, c)) = ax$, where

$$\mathsf{cMult}(a, c) = \underbrace{c \odot \cdots \odot c}_{a \text{ times}} \ .$$

Known instantiations of this primitive include Paillier's scheme [Pai99] (that we briefly recall in Appendix A.1), 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 vertically-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 (FHE) [Gen09]. Since full fledged constructions of FHE are inefficient to be used in practice, more efficient solutions have been designed for evaluating low-degree polynomials over encrypted data functionalities (somewhat-homomorphic encryption). This less powerful variants, even if more efficient than FHE, still require very large key and have poor expansion factor (*i.e.,* a ciphertext is much longer than a plaintext). In a recent work, Barbosa *et al.* [BCF17] introduce the concept of **labelled-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 polynomial 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 (see [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 the security parameter κ as input, 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 $\boldsymbol{s}_i$ in $\{0,1\}^s$ and computes $pk_i = \mathsf{Enc}_{pk}(\boldsymbol{s}_i)$. It outputs $(\boldsymbol{s}_i, pk_i)$.
- $\mathsf{labEnc}_{pk}(\boldsymbol{s}_i, m, \tau)$: On input a message $m \in \mathcal{M}$ with label $\tau \in \mathcal{L}$ from the user i, it computes $b = F(\boldsymbol{s}_i, \tau)$ and outputs the labelled 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 labelled 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 underling LHE scheme: $d_1 \odot \cdots \odot d_n$. Assume that user i and user j have both encrypted n messages, $m_1, \cdots, m_n$ and $m'_1, \cdots, m'_n$, respectively. Let $\tilde{c} \in C$ is the ciphertext obtained as

$$\bigodot_{t=1}^{n} \mathsf{labMult}\big(\mathsf{labEnc}_{pk}(\boldsymbol{s}_i, m_t, \tau_t), \mathsf{labEnc}_{pk}(\boldsymbol{s}_j, m'_t, \tau'_t)\big)$$

- $\mathsf{labDec}_{sk}(pk_i, pk_j, \tilde{c})$: On input the ciphertext $\tilde{c}$, it computes $\boldsymbol{s}_i = \mathsf{Dec}_{sk}(pk_i)$ and $\boldsymbol{s}_j = \mathsf{Dec}_{sk}(pk_j)$. Then, it computes $b_t = F_{\boldsymbol{s}_i}(\tau_t)$ and $b'_t = F_{\boldsymbol{s}_j}(\tau'_t)$ 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 $\tilde{m} = \sum_{t=1}^{n} m_t \cdot m'_t$.

2.4 Data representation

In order to use the cryptographic tools described in the former section to design a privacy-preserving system for ridge regression, we need to represent the real values that form the datasets as values in a 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 the matrix X and the vector $\boldsymbol{y}$ 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] 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 $\|A\|_\infty, \|\boldsymbol{b}\|_\infty \le 10^{2\ell}(n\delta^2 + \lambda)$. Therefore, if $10^{2\ell}(n\delta^2 + \lambda) \le \frac{N-1}{2}$, 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 $t' = r\, s^{-1} \bmod M$. Notice that the modulus N has to be big enough (see Section 4.3).

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 (see Fig. 1):

- 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.

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

[5]We assume that the regularization parameter $\lambda \in \mathbb{R}$ has at most 2ℓ digits in the fractional part and can be rescaled by the factor $10^{2\ell}$ to obtain an integer.

- 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 the MLE and the 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 the MLE obtains the regression model (the vector obtained by running the regression algorithm on $\mathcal{D}$ in the clear) while both the MLE and the CSP do not learn any other information about the private datasets $\mathcal{D}_i$ beyond what is revealed by the model itself.

In order to achieve this goal we design a new system that can be seen as multi-party protocol run by the $m + 2$ parties mentioned before and specified by a sequence of steps. The system described in this paper (Section 4) has the following two-phase architecture (see Fig. 1):

Phase 1 (merging the local datasets): CSP generates the key pair (sk, pk), stores sk and makes pk public; each DO_i sends to the MLE specific ciphertexts computed using pk and the values in $\mathcal{D}_i$. The 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 vertically-partitioned dataset. 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 "masking trick"):

– The MLE samples a random 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. In [BB89], similar masking tricks are 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.

Notice that the data-owners are active only in Phase 1. They are not required to stay online during Phase 2 since their participation to this phase is not necessary.

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.

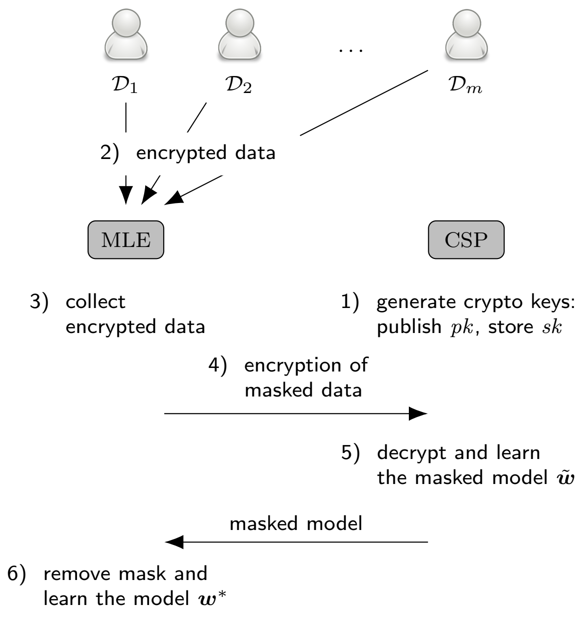
$\mathcal{D}_1$
$\mathcal{D}_2$
...
$\mathcal{D}_m$
2) encrypted data
MLE
CSP
3) collect encrypted data
1) generate crypto keys: publish pk, store sk
4) encryption of masked data
5) decrypt and learn the masked model $\tilde{\boldsymbol{w}}$
masked model
6) remove mask and learn the model $\boldsymbol{w}^*$

Figure 1: System overview —The CSP initializes the encryption; the m data-owners input the datasets $\mathcal{D}_1, \ldots, \mathcal{D}_m$ in encrypted form. The MLE, with the help of the CSP, learns the model trained on the data $\mathcal{D}_1 \cup \ldots \cup \mathcal{D}_m$, without seeing the data in the clear.

4 Protocols Description

In this section, we construct our two-phase system for training a ridge regression model in the setting described in Section 3. Specifically, we describe how to implement Phase 1 (merging the datasets) and Phase 2 (computing the model) in Sections 4.1 and 4.2, respectively.

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. This means that each DO$_k$ holds some rows of the matrix X and the corresponding components of vector $\boldsymbol{y}$. We assume that the correspondence between rows and parties is publicly known and has this form: the data-owner DO_k holds $\mathcal{D}_k$ with

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

for $k = 1, \dots, m$ ($0 = n_0 < n_1 < \dots < n_m = n$). In this case, as already noticed in [NWI+13], we have the following: If we define

$$A_k = \sum_{i=n_{k-1}+1}^{n_k} \boldsymbol{x}_i \boldsymbol{x}_i^\intercal \quad \text{and} \quad \boldsymbol{b}_k = \sum_{i=n_{k-1}+1}^{n_k} y_i \boldsymbol{x}_i$$

then we have that

$$A = \sum_{k=1}^{m} A_k + \lambda I \quad \text{and} \quad \boldsymbol{b} = \sum_{k=1}^{m} \boldsymbol{b}_i \ .$$

The matrix A_k and the vector $\boldsymbol{b}_k$ can be computed locally by the party DO_k; the latter can also compute encryptions of the entries of A_k and $\boldsymbol{b}_k$ and send the ciphertexts to MLE. If an LHE scheme is used for the encryption, then MLE can compute encryptions of the entries of A and $\boldsymbol{b}$ simply using the above formula and the operation $\odot$. This is depicted in detail in protocol $\Pi_{1,\mathrm{hor}}$ (Fig. 2).

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

- *Parties*: CSP and MLE with no input, DO_k with input $\mathcal{D}_k$ as defined in (2) for all $k = 1, \dots, m$.
- *Output*: MLE gets A' and $\boldsymbol{b}'$ (*i.e.,* encryptions of A and $\boldsymbol{b}$, respectively).

Assume that $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is an LHE scheme with plaintext space $\mathcal{M}$ and security parameter κ.

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 $k = 1, \dots, m$, 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 $n_{k-1} + 1 \le i \le n_k$; next, DO_k encrypts them,

$$\begin{cases} A'_k[i,j] = \mathsf{Enc}_{pk}(A_k[i,j]) \\ \boldsymbol{b}'_k[i] = \mathsf{Enc}_{pk}(\boldsymbol{b}_k[i]) \end{cases}$$

for all $i, j = 1, \dots, d$ and $j \ge i$; finally, DO_k sends all A'_k and $\boldsymbol{b}'_k$ to MLE.

Step 3: (*datasets merge*) MLE computes

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

for all $i, j = 1, \dots, d$ and $j \ge i$.

Figure 2: Protocol $\Pi_{1,\mathrm{hor}}$ implements Phase 1 in the horizontally-partitioned setting.

Vertically-partitioned setting

Assume that the dataset represented by the matrix X and the vector $\boldsymbol{y}$ is vertically-partitioned in m datasets. In this case, each DO_k holds some columns of X and $\boldsymbol{y}$. We assume the correspondence between columns and parties is publicly known and, without loss of generality, we represented it in the following way: For $k = 1, \dots, m$, the data-owner DO_k holds the columns from $d_{k-1}+1$ to d_k in X:

$$\mathcal{D}_k = \left\{ \begin{matrix} \boldsymbol{x}_1[d_{k-1}+1] & & \boldsymbol{x}_1[d_k] \\ \vdots & \dots & \vdots \\ \boldsymbol{x}_n[d_{k-1}+1] & & \boldsymbol{x}_n[d_k] \end{matrix} \right\} \tag{3}$$

where $0 = d_0 < d_1 < \cdots < d_m = d$. The party DO_m holds also the vector $\boldsymbol{y}$. By definition, we have

$$\begin{cases} A[i,i] = \sum_{t=1}^{n} \boldsymbol{x}_t[i]\boldsymbol{x}_t[j] + \lambda & \\ A[i,j] = \sum_{t=1}^{n} \boldsymbol{x}_t[i]\boldsymbol{x}_t[j] & \text{if } j \neq i \\ \boldsymbol{b}[i] = \sum_{t=1}^{n} \boldsymbol{x}_t[i]\boldsymbol{y}[t] & \end{cases} .$$

When the columns i, j of X are both held by one data-owner, then this party can directly compute the value $A[i,j]$ and send an encryption of it to the MLE. But if the columns i, j are held by two different data-owners, then this is not possible since neither of them can compute the value $A[i,j]$. To solve this and enable the MLE to get encryptions of all components of the matrix A (and the vector $\boldsymbol{b}$), we use labelled-homomorphic encryption. As we discussed in Section 2, the latter can be constructed on top of any LHE scheme and it enhances the underlying scheme with the command $\mathsf{labMult}$. This is used in order to compute an encryption of the product of two messages from the labelled-encryption of the individual messages. If each data-owner DO_k encrypts the entries of the local dataset $\mathcal{D}_k$ using a labelled-homomorphic encryption scheme and sends the labelled-ciphertexts to the MLE, then the latter can compute an encryption of the entries of A (and the entries of $\boldsymbol{b}$) using formulas of the form:

$$\bigodot_{t=1}^{n} \mathsf{labMult}\big(\mathsf{labEnc}(\boldsymbol{x}_t[i]), \mathsf{labEnc}(\boldsymbol{x}_t[i])\big) .$$

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 labelled-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 labelled-homomorphic scheme. The matrix B and the vector $\boldsymbol{c}$ can be reconstructed by the party handling the decryption key (*i.e.,* in our setting, the CSP). The decryption procedure of the labelled-homomorphic scheme, labDec, accounts for this. However, in the application we consider here (training for a ridge regression model) it is necessary that at the end of Phase 1 the MLE has proper encryptions or 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 labelled-encryptions of A and $\boldsymbol{b}$ doesn't work). For this reason, we need to add one round of communication where the CSP sends to the MLE encryptions of the entries of B and $\boldsymbol{c}$. Notice that this can be done before the beginning of the actual computation (Step 1 of Phase 1) since the value 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 $\Pi_{1,\mathrm{ver}}$ (Fig. 3) 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 $\mathcal{M} = \mathbb{Z}_N$, the message space of the LHE scheme used in

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

– *Parties*: CSP and MLE with no input, DO_k with input $\mathcal{D}_k$ as defined in (3) for all $k = 1, \dots, m$.

– *Output*: MLE gets A' and $\boldsymbol{b}'$ (*i.e.,* encryptions of A and $\boldsymbol{b}$, respectively).

Assume that $(\mathsf{labGen}, \mathsf{labEnc}, \mathsf{labDec})$ is the labelled-homomorphic encryption constructed using an LHE scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ with plaintext space $\mathcal{M}$ and security parameter κ.

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

(*setup*) For $k = 1, \dots, m$, CSP computes $sk_k = \mathsf{Dec}_{sk}(pk_k)$ and $b_{ij} = F(sk_k, (i, j))$ with $1 \leq i \leq n$, and $d_{k-1} + 1 \leq j \leq d_k$ (if $k = m$, then include also $j = 0$). Then, for all $i, j = 1, \dots, d$ 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 ciphertexts are sent to MLE.

Step 2: (*local computation*) For $k = 1, \dots, m$, DO_k computes labelled-encryptions of the known entries of all vectors $\boldsymbol{x}_i$. That is, for $i = 1, \dots, n$ and $j = d_{k-1} + 1 \dots, d_k$, it computes

$$\boldsymbol{c}_{ij} = (a_{ij}, c_{ij}) = \mathsf{labEnc}_{pk}(\boldsymbol{s}_k, \boldsymbol{x}_i[j], \tau)$$

with label $\tau = (i, j)$. DO_m also computes labelled-encryptions of entries of $\boldsymbol{y}$:

$$\boldsymbol{c}_{i0} = (a_{i0}, c_{i0}) = \mathsf{labEnc}_{pk}(\boldsymbol{s}_m, \boldsymbol{y}(i), \tau)$$

with label $\tau = (i, 0)$.

Finally, for all $k = 1, \dots, m$, DO_k sends all labelled-ciphertexts $\boldsymbol{c}_{ij}$ to MLE.

Step 3: (*datasets merge*) MLE computes

$$\begin{cases} A'[i,i] = \bigodot_{t=1}^{n} \mathsf{labMult}(\boldsymbol{c}_{ti}, \boldsymbol{c}_{ti}) \odot \\ \qquad\qquad B'[i,i] \odot \mathsf{Enc}_{pk}(\lambda) \\ A'[i,j] = \bigodot_{t=1}^{n} \mathsf{labMult}(\boldsymbol{c}_{ti}, \boldsymbol{c}_{tj}) \odot B'(i,j) \\ \qquad\qquad\qquad \text{if } j \neq i \\ \boldsymbol{b}'[i] = \bigodot_{t=1}^{n} \mathsf{labMult}(\boldsymbol{c}_{ti}, \boldsymbol{c}_{t0}) \odot \boldsymbol{c}'[i] \end{cases}$$

for all $i, j = 1, \dots, d$ and $j \geq i$.

Figure 3: Protocol $\Pi_{1,\mathrm{ver}}$ implements Phase 1 in the vertically-partitioned setting.

Phase 1). Recall that the final goal of our system is computing $\boldsymbol{w}^* \in \mathbb{Q}^d$ such that $A\boldsymbol{w}^* = \boldsymbol{b}$. In order to do this in a privacy-preserving manner, in Phase 2 we implement the masking trick described in Section 3 in the following way. The MLE samples an invertible matrix R and a vector $\boldsymbol{r}$ uniformly at random from $\mathcal{M}^{d \times d}$ and $\mathcal{M}^d$, respectively. MLE uses these values and the encryption of A and $\boldsymbol{b}$ to compute encryptions of $C \equiv AR \mod N$ and $\boldsymbol{d} \equiv \boldsymbol{b} + A\boldsymbol{r} \mod N$ (*data masking*). These ciphertexts are sent to CSP. The latter decrypts, obtains C and $\boldsymbol{d}$, and computes $\tilde{\boldsymbol{w}} \equiv C^{-1}\boldsymbol{d} \mod N$ (*masked model computation*). This vector is sent back to the MLE who first computes $\tilde{\boldsymbol{w}}^* \equiv R\tilde{\boldsymbol{w}} - \boldsymbol{r} \mod N$ (the model represented in $\mathcal{M}$) and then uses rational reconstruction to compute $\boldsymbol{w}^*$ in $\mathbb{Q}^d$(*model reconstruction*). All the details of this procedure are reported in Protocol Π_2 (Fig. 4).

The correctness is easy to verify, indeed we have:

$$R\tilde{\boldsymbol{w}} - \boldsymbol{r} \equiv R(AR)^{-1}(\boldsymbol{b} + A\boldsymbol{r}) - \boldsymbol{r} \equiv A^{-1}\boldsymbol{b} \mod N$$

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 and $\boldsymbol{b} + A\boldsymbol{r}$) have a distribution that is unrelated with the input datasets. More precisely we have the following lemma. Let $\left(\mathcal{M}^{d\times d}\right)^*$ be the set of all invertible matrices with coefficients in the ring $\mathcal{M}$.

Lemma 4.1. 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

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

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.2 for the formal security proof.

4.3 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 $\mathcal{M} = \mathbb{Z}_N$. 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 $\mathcal{M}$, 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 Section 2.1), 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 2.4 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

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

In Section 5, in our case study implementation the underlying encryption scheme has messages space $\mathbb{Z}_N$ and we adaptively choose N in terms of the public data and system parameters using Eq. (4).

4.4 Complexity

Protocols $\Pi_{1,\text{hor}}$, $\Pi_{1,\text{ver}}$ and Π_2 described in Section 4.1 and Section 4.2 are practically efficient. Before demonstrating this via concrete experiments in Section 5, we discuss here the complexity the aforementioned protocols as a function of the public parameters.

Table 1 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 vertically-partitioned setting, the data-owner DO_k

Protocol Π_2

The protocol $\Pi_{1,\mathrm{hor}}$ or the protocol $\Pi_{1,\mathrm{ver}}$ has been previously executed.

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

– *Output*: MLE gets $\boldsymbol{w}^*$.

– *Public parameters*: d, n, $\mathcal{M} = \mathbb{Z}_N$.

Step 1: (*data masking*) MLE samples $R \leftarrow \left(\mathcal{M}^{d\times d}\right)^*$ and $\boldsymbol{r} \leftarrow \mathcal{M}^d$ and computes

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

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

Step 2: (*masked model computation*) The CSP first decrypts C' and $\boldsymbol{d}'$ obtaining C and $\boldsymbol{d}$,

$$\begin{cases} C[i,j] = \mathsf{Dec}_{sk}(C'[i,j]) \\ \boldsymbol{d}[i] = \mathsf{Dec}_{sk}(\boldsymbol{d}'[i]) \end{cases}$$

for all $i, j = 1, \ldots, d$; then it computes $\tilde{\boldsymbol{w}} \equiv C^{-1}\boldsymbol{d} \mod N$. CSP sends the vector $\tilde{\boldsymbol{w}}$ to the MLE.

Step 3: (*model reconstruction*) The 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 and output the vector $\boldsymbol{w}^*$.

Figure 4: Protocol Π_2 implements Phase 2.

holds the features from $d_{k-1} + 1$ to d_k and DO_m also holds the vector of outputs). 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. (4)). It follows that both Protocol $\Pi_{1,\mathrm{hor}}$ and Protocol Π_2 have complexity $O(d^3(\log(d) + \log(n)))$ bits, while Protocol $\Pi_{1,\mathrm{ver}}$ has complexity $O((nd^2 + d^3)(\log(d) + \log(n)))$ bits (we assume ℓ and m constant and an equal-sized partition of the training dataset).

In practice, 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 For example, [NWI+13] reports that the garbled representation of the circuit that solves (1) with $d = 20$ using Cholesky decomposition and 24-bits integer representation has size 270 MB. On the other hand, even for large values of n and d, the *total* communication complexity of our system in the horizontally-partitioned case ($\Pi_{1,\mathrm{hor}} + \Pi_2$) is smaller than 200 MB (see Fig. 5, where Paillier's scheme is assumed as underlying encryption). For example, if we have a dataset with 10^7 instances and $d = 20$, then Eq. (4) implies that we need to use an encryption scheme with message space $\mathbb{Z}_N$ where the bit-length of N is at least 1774. If we use Paillier's scheme, this implies an overhead of less the 1.3 MB. In the vertically-partitioned case ($\Pi_{1,\mathrm{ver}} + \Pi_2$), the overall communication overheard of our system is dominated by the complexity of Phase 1 because of its linear dependency on the number of instances n. However, this seems to be the case also in other approaches. 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 held by different users. The communication complexity of this approach is $\Theta(nd^2 \log(n))$ bits. The same building blocks are used in [MZ17] to design a system that assumes an arbitrary partitioning of the dataset (*i.e.,* it works for both the vertically- and the horizontally-partitioned setting). 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 vertically-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]). On the other hand, the overall complexity of $\Pi_{1,\mathrm{hor}} + \Pi_2$ for $d = 100$ and $n = 10^7$ is less then 150 MB.

Table 1: Summary of communication complexity.

$\Pi_{1,\mathrm{hor}}$ (Fig. 2)	– CSP sends pk to each party – DO$_k$ sends $\frac{d(d+1)}{2} + d$ ciphertexts to MLE
$\Pi_{1,\mathrm{ver}}$ (Fig. 3)	– 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 $n \times (d_k - d_{k-1})$ ciphertexts and $n \times (d_k - d_{k-1})$ plaintexts to MLE ($k \neq m$) – DO$_m$ sends $n \times (d_k - d_{k-1} + 1)$ ciphertexts and $n \times (d_k - d_{k-1} + 1)$ plaintexts to MLE
Π_2 (Fig. 4)	– MLE sends $d^2 + d$ ciphertexts to CSP – CSP sends d plaintexts to MLE

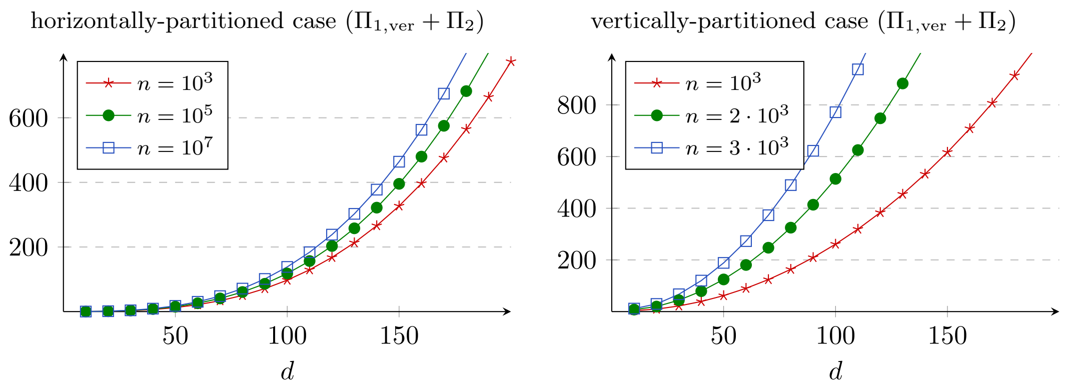

Figure 5: The overall communication complexity in megabytes of our system for different values of n and d ($\delta = 1, \ell = 3$).

Table 2 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 $\Pi_{1,\text{hor}}$ and Step 2 and 3 of $\Pi_{1,\text{ver}}$). Since, each operation considered for Table 2 has a different execution time (*e.g.,* operation on plaintexts are much faster than operation on ciphertexts), for concrete running times we refer to Section 5, where results of the implementation of our system are presented.

Table 2: Summary of computational complexity.

	Step 1		Step 2 (DO_k)	Step 3 (MLE)
	CSP	DO_k	$(n_k - n_{k-1})\left(\frac{d(d+1)}{2}+d\right)$ mult.	$m\left(\frac{d(d+1)}{2}+d\right)$ enc-add.
$\Pi_{1,\text{hor}}$	1 execution of Gen		$(n_k - n_{k-1})\left(\frac{d(d+1)}{2}+d\right)$ add. $\frac{d(d+1)}{2}+d$ enc.	
$\Pi_{1,\text{ver}}$	1 execution of Gen m dec. $\frac{d(d+1)}{2}+d$ enc.	1 enc.	$n(d_k - d_{k-1})$ add. $n(d_k - d_{k-1})$ 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

4.5 Active 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.

The case of a malicious CSP can be handled easily. 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 abstract 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 $1/N$ (*e.g.,* N can be the Paillier's scheme modulus).

The case of a malicious MLE 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. To avoid this threat we need to use a mechanism that allows the CSP to check that the masked data seen in Step 2 of Protocol Π_2 (*i.e.,* $C = AR$ and $\boldsymbol{d} = \boldsymbol{b} + A\boldsymbol{r}$) are computed by the MLE 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 dataset). Moreover, this mechanism needs to be assure that no extra information about the inputs from the DOs or the random values used to mask (*i.e.,* R and $\boldsymbol{r}$) is revealed to CSP. A mechanism with this property can be obtained using a zero-knowledge argument protocol for proving generic statements, as the ones proposed in [PHGR13] or in [GMO16, CDG$^+$]. Investigating this enhancement in details is left for future work.

5 Implementation

In this section we describe our implementation case study of the system described in Section 4 to train a ridge regression model on an encrypted and distributed data. Our goal is to evaluate the effect of the public parameters (*i.e.,* data parameters n and d, and system parameter ℓ) on the system's accuracy and efficiency. Moreover, we are interested in testing our system on real-word datasets to evaluate its effectiveness in practice. In particular, the experiments we run are designed to answer the following questions:

1. *Evaluate correctness*: How does the system parameter ℓ influence the correctness of the output model $\boldsymbol{w}^*$?

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?

3. *Evaluate efficiency in practice*: how does our system behave when is run on real-word data?

To design our system we assumed that the values X and $\boldsymbol{y}$ are real number with at most ℓ digits in the fractional part. In practice, this means that we ask to each user to truncate all the entries in the local dataset after the ℓ^{th} digit in the fractional part. This needs to be done before inputting the values in our 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 represented 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 (the same regularization parameter λ is used). To evaluate the error that this difference can introduce we use the following error rate measure [NWI$^+$13]:

$$E_1 = \left| \frac{F_{\text{ridge}}(\boldsymbol{w}^*) - F_{\text{ridge}}(\bar{\boldsymbol{w}}^*)}{F_{\text{ridge}}(\bar{\boldsymbol{w}}^*)} \right|$$

where F_{ridge} is the objective function defined in Section 2.2. Recall that learning a ridge regression model is equivalent to find the minimum point of F_{ridge}, therefore E_1 tells the loss in accuracy caused by using the vector $\boldsymbol{w}^*$ instead of $\bar{\boldsymbol{w}}^*$ as model. To answer question 1, we measure E_1 for different values of ℓ when the system is run on several datasets. The experimental results reported in Section 5.2 indicate that a negligible error rate (*e.g.,* E_1 of order 10^{-6}) can be achieved already for small values of ℓ (*e.g.,* $\ell = 3$).

To answer question 2 and asses the effect of the parameters n and d on our system's performance in practice, we report the running time of each steps in $\Pi_{1,\mathrm{hor}}$, $\Pi_{1,\mathrm{ver}}$ and Π_2 when the system is run on synthetically generated data. The advantage of this approach is that we can run experiments for a wide range of parameters values. The results of this experiment are reported in details in Section 5.2. As an highlight, when $n = 10^3$ and $d = 20$ Protocol $\Pi_{1,\mathrm{hor}}$ runs in less than 1 second, Protocol $\Pi_{1,\mathrm{ver}}$ runs in less than 20 minutes and Protocol Π_2 runs in less than half a minute.

To answer question 3 and show the practicality of our system we consider real-world datasets. In particular, we run our system on real-world datasets download from the UCI repository[6]. This repository is commonly used for evaluating new machine-learning algorithms in the standard setting (*i.e.,* no privacy guarantees). To evaluate communication and computational efficiency in practice, we report the total running time and communication overhead for eights different UCI datasets (see Section 5.2).

Finally, before moving to the next section, 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) [C$^+$09] proposed an accurate dosing model trained using linear regression on a database containing clinical and genetic information of 4043 patients. The database was the result of the merge of the data of different patients collected by 21 research groups. The model proposed in [C$^+$09] 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[7] 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 $\Pi_{1,\mathrm{hor}}$ + 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 and produces an overall communication overhead of less then 2.5 MB.

5.1 Setup

We implemented our system using Paillier's scheme (see Appendix A.1); recall that Paillier's scheme has message space $\mathcal{M} = \mathbb{Z}_N$ where N is a large RSA modulus. In order to 1) assure a security level of at least 80 bits[8], 2) decrease the running time and the communication overhead and 3) satisfy Eq. (4), we choose N such that $\log_2(N) = \max\{1024, \lfloor\beta\rfloor + 1\}$ where β is the logarithm in base 2 of the left-hand side of Eq. (4). We wrote our software in Python3 5.2 using the phe1.3 library[9] to implement Paillier encryption/decryption and operations on ciphertexts. We use the gmpy2 library[10] to implement arithmetic operations with large integers. To compute the determinant function and to solve linear systems, we use the Gaussian elimination. To multiply to square matrices we use the Strassen algorithm.

In each experiment involving $\Pi_{1,\mathrm{hor}}$ (horizontally-partitioned setting), we split the n data points evenly among the m data-owners and $m = 10$. In the experiments involving $\Pi_{1,\mathrm{ver}}$ (vertically-partitioned setting), we split the d features evenly among the m data-owners and we assume that DO_m also has the vector of outputs. We assume $m = 5$ for this setting.

It is known that for ridge regression the predictive accuracy of the learned model depends on λ. In machine learning, different techniques are used to tune the regularization parameter and

[6]`https://archive.ics.uci.edu/ml/datasets.html`

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

[8]According to NIST standard 80-bit security corresponds to a factoring modulus of at least 1024 bits.

[9]`http://python-paillier.readthedocs.io`

[10]`https://pypi.python.org/pypi/gmpy2`

obtain more accurate models (*e.g.,* cross-validation); many of these techniques split the input dataset in subsets and repeat the following procedure: train on some of the subsets and test on the other ones. This kind of parameter tuning algorithms can be included in our system if ad-hoc modifications are made. Since this is beyond the scope of this implementation case study, in the following we fix the regularization parameter λ to 0.1.

5.2 Experiments results

All experiments were run on a machine with a 2.6GHz single CPU and 8 GB RAM under Ubuntu Linux 16.04. All the timings are reported in seconds, all the values (timings and errors) are averaged on 5 repetitions of the same experiment.

Synthetic data

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 uniformly at random from $[-1,1]^n$ $(i = 1,\dots,n)$. 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 1 and under the condition $|y_i| \le 1$.

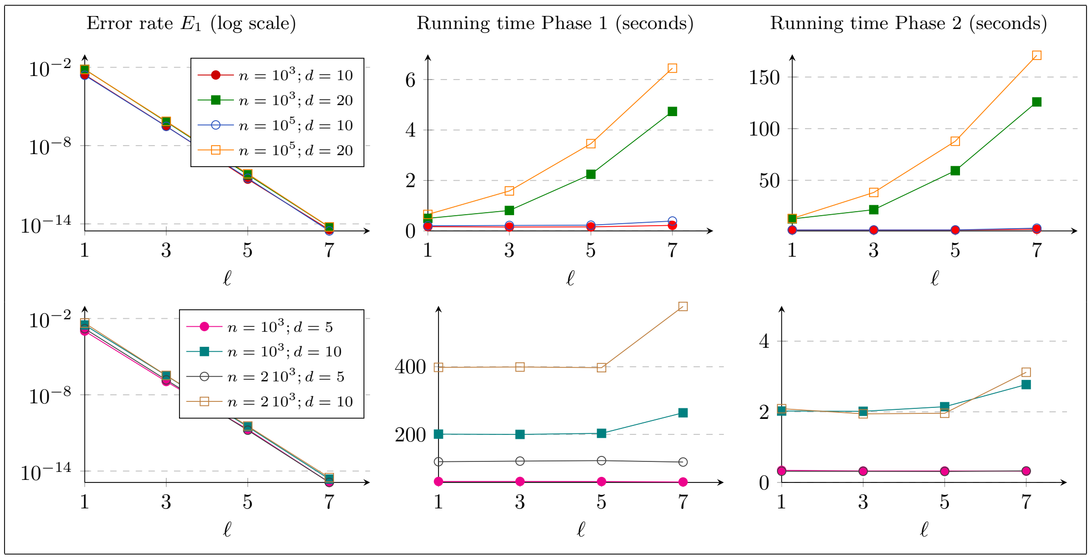

Figure 6: Results of the experiments run on synthetically-generated data for different values of the system parameter ℓ in the horizontally-partitioned setting (top) and in the vertically-partitioned setting (below).

The results from the experiments involving synthetically generated data are reported in Figure 6, Tables 3 and 4 here, and in Tables 2 and 6 in Appendix A. To quantify the impact of the parameter ℓ, we measure the error rate E_1 and the running times when $\ell = 1, 3, 5, 7$ for 4 different synthetically-generated datasets in both settings considered in this work (horizontally- and vertically-partitioned data). Figure 6 shows a summary of the results of this experiment. It is clear that the value of E_1 decreases very rapidly with the increasing of ℓ, regardless of the values of n and d. Moreover, we can infer that the efficiency of our system can be influenced by the value of ℓ. Indeed, because of (4) the value of this parameter has effect on the bit-length of the plaintexts and ciphertexts handled by our system. 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 without degrading our system efficiency.

Table 3: Running times and error rate for UCI datasets in the horizontally-partitioned setting.

Dataset	n	d	ℓ	$\log_2(N)$	E_1	Phase 1		Phase 2	
						Time	kB	Time	kB
forest	517	12	1	1024	1.46E-07	0.234	23	2.891	41.48
facebook	500	17	1	1926	1.11E-15	2.148	81.8	56.203	151.37
air	6946	13	1	1128	0.00E+00	0.323	29.3	5.898	53.09
beijing	41757	14	1	1260	2.40E-08	0.504	37.44	8.468	68.28
wine	4898	11	3	1112	1.59E-06	0.227	21.38	3.213	38.18
energy	19735	25	2	2400	4.53E-08	7.597	209.9	310.826	397.29
student	395	30	1	1740	2.85E-16	4.396	215.22	152.220	410.87
boston	506	13	3	1268	1.96E-08	0.430	33.94	8.110	59.7

Table 4: Running times and error rate for UCI datasets in the vertically-partitioned setting.

Dataset	n	d	ℓ	$\log_2(N)$	E_1	Phase 1		Phase 2	
						Time	kB	Time	kB
forest	517	12	1	1024	1.46E-07	134.410	543.06	2.876	41.48
facebook	500	17	1	1926	1.11E-15	1496.023	1397.5	56.118	151.37
student	395	30	1	1740	2.85E-16	2636.412	712.34	150.000	410.87
boston	506	13	3	1268	1.96E-08	270.957	1854.82	7.880	59.7

To understand the effect of n and d on the speed of our system, we fix $\ell = 3$ and we run the system on synthetic data for large range of values for these two data parameters. The results can be found in Table 5 (horizontally-partitioned case) and in Table 6 (vertically-partitioned case). We report the running time of each step of protocols $\Pi_{1,\text{hor}}$, $\Pi_{1,\text{ver}}$ and Π_2. For Step 2 in $\Pi_{1,\text{hor}}$ and $\Pi_{1,\text{ver}}$ we report the average running time of one data-owner. In $\Pi_{1,\text{hor}}$, Step 2 is the most expensive step (see Table 5). 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 approximately nd^2 arithmetic operations on plaintext values and d^2 encryptions for one data-owner (see Table 2). From the results reported in Table 5, it is clear that the costs of the encryptions is dominant even for large values of n. Indeed, in our implementation using Paillier's scheme one encryption correspond to 2 multiplication and 1 exponentiation modulo N (see Appendix A.1). In Step 3 of $\Pi_{1,\text{hor}}$, the MLE computes the encryption of A and $\boldsymbol{b}$ using approximately $10d^2$ ciphertexts additions (*i.e.,* multiplications modulo N), which turns out to be fast. In $\Pi_{1,\text{ver}}$, Step 3 is the most expensive step (see Table 6). In particular, as expected from the analysis done in Section 4.4, the running time of this step is significantly influenced by the values of n. Finally, for Phase 2 the results in Table 5 and 6 show that in Π_2 Step 1 requires longer time respect to the other two steps because of the operations done on ciphertexts. Step 2 and 3 require operations only on plaintext values and therefore are very fast (*e.g.,* 21 seconds for both the steps for a dataset of one hundred thousands instances with 40 features).

UCI datasets

Finally, 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 8 in the Appendix A. 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. The results of this experiment are reported in Table 3 (horizontally-partitioned case) and in Table 4 (vertically-partitioned case). In Table 3 for Phase 1 we report the total running time (in seconds) of $\Pi_{1,\text{hor}}$ assuming that the 10 data-owners execute

Table 5: Running times and error rate for synthetically-generated data (horizontally-partitioned setting, $\ell = 3$).

n	d	E_1	$\Pi_{1,\text{hor}}$			Π_2		
			Step 1	Step 2	Step 3	Step 1	Step 2	Step 3
10^3	10	3.71E-07	0.026	0.126	0.007	1.844	0.057	0.010
	20	6.19E-07	0.057	0.741	0.034	20.963	0.409	0.045
	30	1.07E-06	0.243	5.402	0.151	192.757	3.045	0.175
	40	1.20E-06	0.367	21.556	0.451	1112.223	12.142	0.477
10^4	10	3.28E-07	0.032	0.135	0.007	1.897	0.059	0.011
	20	7.96E-07	0.039	1.057	0.043	29.757	0.575	0.060
	30	9.62E-07	0.285	7.288	0.188	254.774	4.012	0.224
	40	1.13E-06	0.314	23.617	0.556	1221.701	15.853	0.604
10^5	10	2.71E-07	0.026	0.190	0.007	1.894	0.060	0.013
	20	5.78E-07	0.099	1.624	0.049	40.959	0.786	0.080
	30	8.66E-07	0.314	9.795	0.220	331.589	5.563	0.320
	40	1.07E-06	0.634	30.605	0.642	1559.289	20.373	0.784

Table 6: Running times and error rate for synthetically-generated data (vertically-partitioned setting, $\ell = 3$).

n	d	E_1	$\Pi_{1,\text{ver}}$			Π_2		
			Step 1	Step 2	Step 3	Step 1	Step 2	Step 3
10^3	10	2.34E-07	0.319	4.214	188.206	1.842	0.058	0.010
	15	5.63E-07	0.521	6.124	401.245	4.981	0.129	0.021
	20	7.23E-07	1.139	13.540	1,135.393	20.900	0.409	0.045
$2\,10^3$	10	3.66E-07	0.472	8.389	368.255	1.845	0.058	0.010
	15	4.08E-07	0.809	12.222	781.864	4.983	0.129	0.022
	20	5.84E-07	1.717	30.486	2,573.014	23.620	0.457	0.048
$3\,10^3$	10	2.88E-07	0.633	12.584	552.764	1.855	0.059	0.011
	15	5.58E-07	1.062	18.307	1,174.733	4.976	0.128	0.022
	20	6.68E-07	2.264	46.620	3,916.432	24.027	0.465	0.050

Step 2 in parallel, and the size in kilobytes of the message sent from one data-owner to MLE. In Table 4 for Phase 1 we report the total running time (in seconds) of $\Pi_{1,\text{ver}}$ assuming that the 5 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 tables, 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).

Further optimizations

When the logarithm in base 2 of the left-hand of (4) is greater or equal to 2047 (*i.e.,* $\beta \geq 2047$), 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 $\beta \geq 2047$ we can use Damgård and Jurik's scheme with $s = \left\lfloor \frac{\beta+1}{1024} \right\rfloor$ and $\log_2(N) = 1024$. With this choice of parameters, Damgård and Jurik's still guarantees 80-bit security and works with ciphertexts of bit length at most $\frac{s+1}{s}\beta$. While Paillier's scheme with $\log_2(N) = \lfloor \beta \rfloor + 1$ works with ciphertexts of bit length less or equal to 2β.

Finally, notice that we implement all the commands in our system in a sequential manner. However, the operations described in many steps of both Phases 1 and 2 can be easily be *parallelized*. For example, the MLE can run Step 3 in protocol $\Pi_{1,\text{ver}}$ on k processors in k times faster since the computations of the values $A'[i, j]$ can be done in parallel.

6 Conclusion

In this paper we described a new system to train a ridge regression model on the merge of encrypted datasets held by different (possibly mutually distrustful) parties. Our new system is designed in the two-server model and is the first one based only on linearly-homomorphic encryption (*e.g.,* Paillier's scheme). We discussed parameter selection and scalability of our system. We presented an implementation and showed that the system we propose has low running-time and low communication overhead for many real-world datasets. Using standard computational resources, we were able to train a ridge regression model on a dataset of almost 20 thousands instances with 25 features (result of the horizontal merge of 10 equal-sized datasets) in 5 and half minutes causing a communication overhead of only 2.4 MB.

References

[AHPW15] Yoshinori Aono, Takuya Hayashi, Le Trieu Phong, and Lihua Wang. Fast and secure linear regression and biometric authentication with security update. Cryptology ePrint Archive, Report 2015/692, 2015.

[AS00] Rakesh Agrawal and Ramakrishnan Srikant. Privacy-preserving data mining. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA.*, pages 439–450, 2000.

[BB89] Judit Bar-Ilan and Donald Beaver. Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In *Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing, Edmonton, Alberta, Canada, August 14-16, 1989*, pages 201–209, 1989.

[BCF17] Manuel Barbosa, Dario Catalano, and Dario Fiore. Labeled homomorphic encryption: Scalable and privacy-preserving processing of outsourced data. *IACR Cryptology ePrint Archive*, 2017:326, 2017.

[Bea91] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Proceedings of Crypto*, pages 420–432, Springer Verlag 1991.

[BGW88] Michael BenOr, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, pages 1–10, 1988.

[C$^+$09] International Warfarin Pharmacogenetics Consortium et al. Estimation of the warfarin dose with clinical and pharmacogenetic data. *N Engl J Med*, 2009(360):753–764, 2009.

[CDG$^+$] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. *IACR Cryptology ePrint Archive*, 2017.

[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 *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security, AISec 2015, Denver, Colorado, USA, October 16, 2015*, pages 3–14, 2015.

[DHC04] Wenliang Du, Yunghsiang S. Han, and Shigang Chen. Privacy-preserving multivariate statistical analysis: Linear regression and classification. In *Proceedings of the Fourth SIAM International Conference on Data Mining, Lake Buena Vista, Florida, USA, April 22-24, 2004*, pages 222–233, 2004.

[DJ01] Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *Public Key Cryptography, 4th International Workshop on Practice and Theory in Public Key Cryptography, PKC 2001, Cheju Island, Korea, February 13-15, 2001, Proceedings*, pages 119–136, 2001.

[FSW02] Pierre-Alain Fouque, Jacques Stern, and Jan-Geert Wackers. Cryptocomputing with rationals. In *Financial Cryptography*, volume 2357 of *Lecture Notes in Computer Science*, pages 136–146. Springer, 2002.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 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.

[GMO16] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. Zkboo: Faster zero-knowledge for Boolean circuits. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 1069–1083, 2016.

[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*, 4:248–267, 2017.

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

[JL13] Marc Joye and Benoît Libert. Efficient cryptosystems from 2^k-th power residue symbols. In *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, pages 76–92, 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 *Proceedings of the 2004 Annual National Conference on Digital Government Research*, pages 108:1–108:2. 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. *Journal of Computational and Graphical Statistics*, 14(2):263–279, 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. *Journal of Official Statistics*, 25(1):125, 2009.

[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 *Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings*, pages 36–54, 2000.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of Yao's protocol for two-party computation. *J. Cryptology*, 22(2):161–188, 2009.

[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, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 19–38, 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. Pacific Symposium on Biocomputing*, page 27. 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, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 334–348, 2013.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 238–252, 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, 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 *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, pages 677–682. ACM, 2004.

[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.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, pages 162–167, 1986.

A Appendix

Table 7: Results for synthetically-generated data. Underlying encryption: Paillier's scheme with message space $\mathbb{Z}_N$.

n	d	ℓ	$\log_2(N)$	E_1	$\Pi_{1,\text{hor}}$			Π_2		
					Step 1	Step 2	Step 3	Step 1	Step 2	Step 3
10^3	10	1	1024	2.72E-03	0.033	0.129	0.007	1.856	0.058	0.006
		3	1024	3.07E-07	0.021	0.127	0.007	1.845	0.058	0.010
		5	1024	2.68E-11	0.027	0.126	0.007	1.845	0.058	0.014
		7	1150	3.60E-15	0.044	0.168	0.008	2.493	0.080	0.020
	20	1	1024	7.01E-03	0.038	0.436	0.025	12.393	0.224	0.021
		3	1244	6.70E-07	0.039	0.738	0.034	21.112	0.408	0.045
		5	1776	5.84E-11	0.150	2.032	0.063	58.131	1.087	0.100
		7	2306	5.70E-15	0.285	4.352	0.098	123.408	2.295	0.189
10^5	10	1	1024	2.46E-03	0.027	0.169	0.007	1.843	0.057	0.008
		3	1024	2.98E-07	0.036	0.174	0.007	1.847	0.058	0.012
		5	1024	3.17E-11	0.044	0.177	0.007	1.885	0.059	0.016
		7	1282	2.89E-15	0.081	0.301	0.010	3.528	0.117	0.025
	20	1	1024	6.82E-03	0.040	0.588	0.026	12.798	0.224	0.028
		3	1510	7.33E-07	0.065	1.473	0.047	37.476	0.692	0.069
		5	2040	6.98E-11	0.192	3.186	0.081	86.095	1.602	0.139
		7	2572	5.84E-15	0.290	6.041	0.119	167.641	3.156	0.251

n	d	ℓ	$\log_2(N)$	E_1	$\Pi_{1,\text{ver}}$			Π_2		
					Step 1	Step 2	Step 3	Step 1	Step 2	Step 3
10^3	5	1	1024	1.03E-03	0.154	2.480	58.239	0.322	0.019	0.002
		3	1024	1.11E-07	0.138	2.444	58.416	0.305	0.017	0.003
		5	1024	1.90E-11	0.135	2.457	58.116	0.304	0.017	0.004
		7	1024	1.27E-15	0.128	2.409	56.900	0.296	0.017	0.005
	10	1	1024	3.09E-03	0.337	4.472	195.984	1.954	0.061	0.006
		3	1024	3.26E-07	0.327	4.438	195.218	1.943	0.062	0.010
		5	1024	3.04E-11	0.324	4.481	198.343	2.067	0.062	0.015
		7	1150	2.33E-15	0.368	5.884	257.592	2.662	0.086	0.022
$2\,10^3$	5	1	1024	1.59E-03	0.234	4.856	114.198	0.298	0.016	0.002
		3	1024	1.49E-07	0.234	4.937	115.894	0.296	0.016	0.003
		5	1024	1.63E-11	0.245	5.107	117.163	0.291	0.016	0.004
		7	1024	1.28E-15	0.239	4.837	113.248	0.305	0.017	0.005
	10	1	1024	4.37E-03	0.521	8.892	388.718	2.018	0.064	0.007
		3	1024	3.37E-07	0.480	9.184	389.507	1.872	0.060	0.011
		5	1024	3.52E-11	0.493	8.898	387.770	1.884	0.060	0.015
		7	1170	3.07E-15	0.566	12.950	564.662	3.002	0.095	0.022

Table 8: 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

A.1 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.2 Security proof

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 (Definition 7.5.1 in [Gol04])**.** We say that the protocol Π realizes ϕ with *correctness* if for any input $(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n)$, it holds[11] 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.

The protocol Π described in Fig. 7, 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 $\mathcal{D}_k$ as explained by (2) for the horizontally-partitioned setting, and by (3) for the vertically-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 2.4). 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. (4) 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. 7) realizes ϕ with correctness and privacy against the adversaries $\mathcal{A}_1 = \{\mathrm{MLE}\} \cup \{\mathrm{DO}_i \,|\, i \in D\}$ and $\mathcal{A}_2 = \{\mathrm{CSP}\} \cup \{\mathrm{DO}_i \,|\, i \in D\}$.*

[11]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 $\text{DO}_1, \ldots, \text{DO}_m$ jointly run $\Pi_{1,\text{hor}}$ or $\Pi_{1,\text{ver}}$.

Phase 2: – MLE and CSP jointly execute Π_2;

– MLE sends $\boldsymbol{w}^*$ to the other parties.

Figure 7: Protocol Π implements our system.

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. (4) 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 horizontally-partitioned 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 $\Pi_{1,\text{hor}}$. 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 vertically-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 $i = 1, \ldots, n$ and $j = d_{k-1} + 1, \ldots, d_k$ as in Step 2 of $\Pi_{1,\text{ver}}$. 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 4.1 that the simulation output has the same distribution of the views of the corrupted parties in $\mathcal{A}_2$ in the protocol Π.

$\square$