Privacy-Preserving Ridge Regression on Distributed Data

Irene Giacomelli, Somesh Jha, C. David Page, and Kyonghwan Yoon

University of Wisconsin-Madison, Madison, WI, US

July 18, 2017

Abstract. Linear regression is an important statistical tool 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 healthcare), these values represent sensitive data owned by several different parties that are unwilling to share them. In this setting, training a linear regression model becomes challenging and needs specific cryptographic solutions. In this work, we propose a new system that can train a linear regression model with 2-norm regularization (*i.e.* ridge regression) on a dataset obtained by merging a finite number of private datasets. Our system is composed of two phases: The first one is based on a simple homomorphic encryption scheme and takes care of securely merging the private datasets. The second phase is a new ad-hoc two-party protocol that computes a ridge regression model solving a linear system where all coefficients are encrypted. The efficiency of our system is evaluated both on synthetically generated and real-world datasets.

Keywords: linear regression, distributed data, privacy-preserving system, multiparty computation.

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 $\boldsymbol{x}_i$ 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 $\boldsymbol{w}$ (the model) with d real components such that the value $\boldsymbol{w}(1)\boldsymbol{x}_i(1) + \cdots + \boldsymbol{w}(d)\boldsymbol{x}_i(d)$ 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 value) 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).

Motivation. 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 $\boldsymbol{w}$. This common assumption becomes non-trivial in some relevant areas where linear regression finds application (*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 their private data, making traditional linear regression algorithms difficult (or even impossible) to apply. On the other hand, it is known that having a large training dataset composed of a good variety of data-points (*e.g.* more relevant features or more data-points) 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, etc). 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 countries. 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 are owned by multiple parties.

This work. 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 the setting in which the features in the training dataset are distributed among different parties (*vertically-partitioned* dataset) and the setting in which each party has some of the data-points that form the training set (*horizontally-partitioned* dataset).
Our system is composed of two phases. In the first phase, a public-key encryption scheme with limited homomorphic property is used to let the data-owners securely submit their data to a third party. After this first step, a randomized algorithm is used to compute the desired linear regression model from the encrypted version of the merged dataset. This algorithm is run by the party who collected the encrypted data with the help of a crypto-service provider. Neither one of these two parties have to handle the input data in the clear and no extra information (beside that released by the model itself) is revealed to these two parties (assuming that they do not collude). Moreover, notice that the data-owners are active only in the first phase of the entire system. This makes our solution suitable for all applications in which the majority of data-owners are willing to help in order to run collaborative analysis but don't want to (or cannot) spend to much resources for it.

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. introduced in [NWI$^+$13] the scenario considered in this paper: the privacy-preserving linear regression protocol has two phases. In the first phase there are possibly many data-owners that submit their private data to a third party. In the second phase, the third party is in charge of computing the model with the help of a crypto-service provider. Their solution considers the horizontally-partitioned setting for ridge regression and is based on additive encryption (Paillier's scheme) and Yao's protocol [Yao86,LP09]. The latter is 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 beside 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. Very recently, the system presented in [NWI$^+$13] is extended to the vertically-partitioned setting by the paper [GSB$^+$16]. Gascón et al. achieves this result using MPC techniques to allow the data-owners to compute shares of the merged dataset. Moreover, Gascón et al. also improves the running time of the second phase of the protocol presented in [NWI$^+$13] by designing a new conjugate gradient descent algorithm that is used as circuit C in the place of Cholesky decomposition (both Choleshy decomposition and conjugate gradient descent have complexity $\Theta(d^3)$ where d is the dimension of the square matrix A).

Our paper follows this line of work and presents a new two-phase system. For the first phase, we extend the approach used by Nikolaenko et al. to the vertically-partitioned setting using an homomorphic encryption scheme that supports only depth-1 multiplications among ciphertexts. For the second phase, we design a new two-party protocol that solves the linear system $A\boldsymbol{w} = \boldsymbol{b}$. The first version of this protocol has two rounds of communication and uses only a pseudorandom generator and modular arithmetic (beside the homomorphic properties of the scheme used in

the first phase). We believe this first solution is interesting for its simplicity and efficiency. On the other hand this first protocol leaks $\det(A)$ to the party holding the encryption of A. To avoid this we present a second version that has an extra round of communication and invokes a oracle-functionality to compute division. Notice that this functionality can be implemented by any two-party protocol for secure function evaluation. In particular, using Yao's protocol the circuit C that need to be considered is the one represents $\Theta(d)$ arithmetic operations among large integers (*e.g.* 2048 bit-length).

Organization of the paper. 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.), and we also provide and overview of its design. In Section 4 we describes in detail the protocols that form our two-phases system (in the horizontally-partitioned setting) and provide the formal security proof. In Section 5 we discuss the extension of our system to the vertically-portioned setting and to the case of lasso-penalized regression. Finally, Section 6 reports on our implementation and experiments.

Standard notations. We use bold notation for vectors and capital letter for matrices (*e.g.* $\boldsymbol{x} \in \mathbb{R}^n$ is a column vector, $A \in \mathbb{R}^{n \times d}$ is matrix with n rows and d columns, both with real-value entries). We indicate with $\boldsymbol{x}(i)$ the i-th component of a vector $\boldsymbol{x}$ and with $A(i,j)$ the i-th component of the j-th column of the matrix A. The p-norm of a vector $\boldsymbol{x}$ is defined by $\|\boldsymbol{x}\|_p = \sqrt[p]{\sum_{i=1}^{n} |\boldsymbol{x}(i)|^p}$.

If A is a $d \times d$ matrix with entries in a ring R, then the adjugate of A is defined as $\operatorname{adj}(A) = C^\top$ with $C(i,j) = (-1)^{i+j} A_{ij}$ and A_{ij} (the (i,j) minor of A) is the determinant of the $(d-1) \times (d-1)$ matrix that results from deleting row i and column j of A.

2 Background

2.1 Linear Regression

A linear regression learning algorithm is a procedure that on input n points $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$ (where $\boldsymbol{x}_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$) outputs a vector $\boldsymbol{w}^* \in \mathbb{R}^d$ such that ${\boldsymbol{w}^*}^\top \boldsymbol{x}_i \approx y_i$ for all $i = 1, \ldots, n$. One common way to compute such a model $\boldsymbol{w}^*$ is to use the squared-loss function and the associated empirical error function (mean squared error):

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

where $X \in \mathbb{R}^{n \times d}$ is the matrix with the vector $\boldsymbol{x}_i^\top$ as i^{th} row and $\boldsymbol{y} \in \mathbb{R}^n$ is the vector with the value y_i as i^{th} component (we always assume that X is full-rank, $\operatorname{rk}(X) = d$). 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 $\lambda \geq 0$ is fixed. The regularization term is added to avoid overfitting 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, $\nabla F_{\text{ridge}}(\boldsymbol{w}) = 2X^\top(X\boldsymbol{w} - \boldsymbol{y}) + 2\lambda\boldsymbol{w}$, we have that $\boldsymbol{w}^*$ is computed solving the linear system

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

where $A = X^\top X + \lambda I$ (symmetric invertible $d \times d$ matrix) and $\boldsymbol{b} = X^\top \boldsymbol{y}$ (vector of d components).

2.2 Cryptographic Tools

To design our new privacy-preserving system for linear regression, we employ homomorphic encryption. An **additive encryption scheme** 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, $\mathsf{Gen}(\kappa) \to (\boldsymbol{sk}, \boldsymbol{pk})$.
2. the encryption algorithm Enc is a randomized algorithm that uses the public key to transform an element from $\mathcal{M}$ (finite plaintext space) in a ciphertext, $\mathsf{Enc}_{\boldsymbol{pk}}(\boldsymbol{x}) \to \boldsymbol{c} \in \mathcal{C}$.
3. the decryption algorithm Dec is a deterministic function that takes as input a element from the ciphertext space $\mathcal{C}$ and the secret key and reveal the original plaintext (*i.e.* $\mathsf{Dec}_{\boldsymbol{sk}}(\boldsymbol{c}) = \boldsymbol{x}$).

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 $\boldsymbol{pk}$. Moreover, we have the additive property: we assume that $(\mathcal{M}, +)$ is an additive group and that there exists an operation $\oplus$ on the ciphertext space $\mathcal{C}$ such that for any a-uple of ciphertexts $\mathsf{Enc}_{\boldsymbol{pk}}(\boldsymbol{x}_1) \to \boldsymbol{c}_1, \ldots, \mathsf{Enc}_{\boldsymbol{pk}}(\boldsymbol{x}_a) \to \boldsymbol{c}_a$ (a integer), it holds that $\mathsf{Dec}_{\boldsymbol{sk}}(\boldsymbol{c}_1 \oplus \cdots \oplus \boldsymbol{c}_a) = \boldsymbol{x}_1 + \cdots + \boldsymbol{x}_a$. This implies that, if a is a positive integer, $\mathsf{Dec}_{\boldsymbol{sk}}(\mathsf{mult}(a, \boldsymbol{c})) = a\boldsymbol{x}$, where $\mathsf{mult}(a, \boldsymbol{c}) = \underbrace{\boldsymbol{c} \oplus \cdots \oplus \boldsymbol{c}}_{a \text{ times}}$.

Example 1. (Paillier's scheme [Pai99].) Let $N = pq$ (product of two prime of κ bits) and $g \leftarrow \mathbb{Z}_{N^2}$, we have $\boldsymbol{pk} = (N, g)$ and $\mathcal{M} = \mathbb{Z}_N$. To encrypt $M \in \mathbb{Z}_N$, sample $r \leftarrow \mathbb{Z}_N^*$ and compute $c = g^M r^N \mod N^2$. In this case, $\oplus$ is the standard product in $\mathbb{Z}_{N^2}$, indeed: $c_1 \oplus c_2 = c_1 c_2 \mod N^2 = (g^{M_1} r_1^N \mod N^2)(g^{M_2} r_2^N \mod N^2) = g^{M_1+M_2}(r_1 r_2)^N \mod N^2$ and $\mathsf{mult}(a, c_1) = (c_1)^a \mod N^2 = (g^{M_1} r_1^N \mod N^2)^a = g^{aM_1} r_1^{aN} \mod N^2$.

3 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 Figure 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.
- 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, the 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 the MLE to help it in achieving its task (computing the linear regression model). The 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 instruction of the protocol but try to learn extra information on 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[1].

[1] Using proxy encryption [AFGH06], it is possible to decrease to number of messages sent using private channels. We leave this for future work

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 list of public parameters and a sequence of steps. The system described in this paper (Section 4) has the following two-phase architecture (see Figure 1):

- *Phase 1* (merging the local datasets): the CSP generates key pair $(\boldsymbol{sk}, \boldsymbol{pk})$, stores $\boldsymbol{sk}$ and makes $\boldsymbol{pk}$ public; each DO_i sends to the MLE specific ciphertexts computed using $\boldsymbol{pk}$ 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 $\boldsymbol{b}$.
- *Phase 2* (computing the model): the MLE uses the ciphertexts $\mathsf{Enc}_{\boldsymbol{pk}}(A)$ and $\mathsf{Enc}_{\boldsymbol{pk}}(\boldsymbol{b})$ 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 an approximation of the model computed by the learning algorithm in the clear. And we say that the system is *private* if the distribution of the masked data sent by the MLE to the CSP is independent from the distribution of the local inputs.

As we will see in Section 4 and 5, the specific design of the protocol realising Phase 1 depends on the distributed setting: horizontally- or vertically-partitioned dataset. However, in both cases, the encryptions of A and $\boldsymbol{b}$ are computes by the MLE simply using the homomorphic properties of the encryption scheme used by the DOs to encrypt. The protocol realising Phase 2 is presented in two versions, both of which return the solution of the system $A\boldsymbol{w} = \boldsymbol{b}$ following this "masking trick":

- Given a matrix R and a vector $\boldsymbol{r}$, the MLE uses the homomorphic property of the underlying encryption scheme to compute $C' = \mathsf{Enc}_{\boldsymbol{pk}}(AR)$ (from $\mathsf{Enc}_{\boldsymbol{pk}}(A)$) and $\boldsymbol{d}' = \mathsf{Enc}_{\boldsymbol{pk}}(\boldsymbol{b} + A\boldsymbol{r})$ (from $\mathsf{Enc}_{\boldsymbol{pk}}(\boldsymbol{b})$. 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 from 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. The details of the two versions of the protocol realizing Phase 2 are given in the next section. In particular, the usage of cryptographic tools implies restrictions on the values representation and on the arithmetic used (modular arithmetic). The first version allows for a small leakage ($\det(A)$ is leaked to the MLE) in order to deal with such constrains. The second version avoid the leakage using a simple trusted functionality to implement integer division.

4 Protocols Description

In this section, we describe our two-phases 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 Section 4.1 and 4.2, respectively.

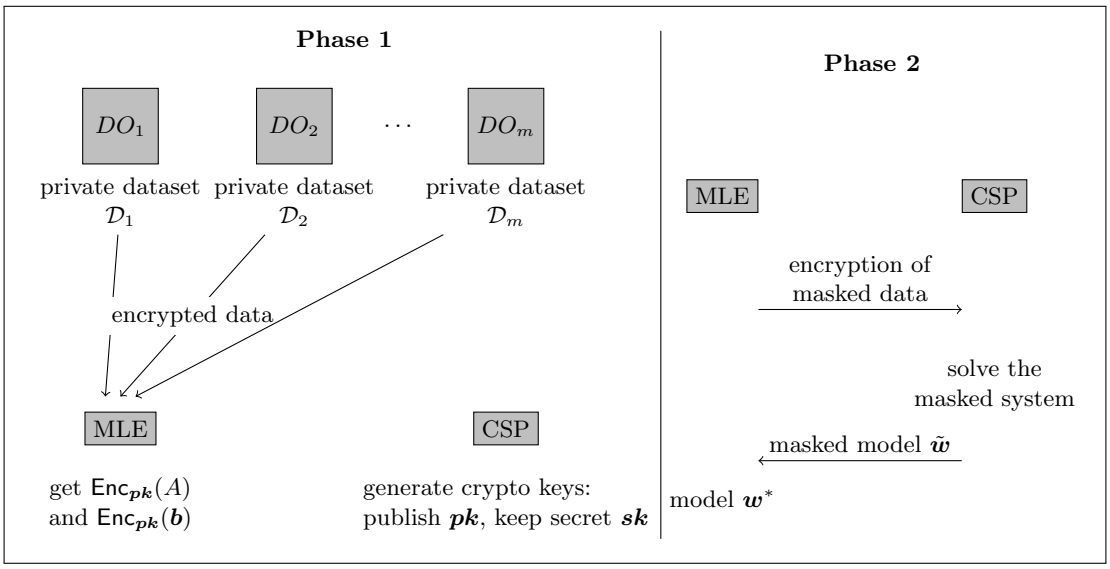

Fig. 1. System overview.

Data representation We assume that the entries of X are real numbers from the interval $[-x, x]$ and the entries of $\boldsymbol{y}$ are real number from the interval $[-y, y]$. In both cases, we consider each value with at most ℓ fractional digits. In other words, there exist the matrix $\bar{X} \in \mathbb{Z}^{n \times d}$ and the vector $\bar{\boldsymbol{y}} \in \mathbb{Z}^n$ such that

$$X = \frac{1}{10^\ell}\bar{X} \quad \text{and} \quad \boldsymbol{y} = \frac{1}{10^\ell}\bar{\boldsymbol{y}}$$

With this assumption, we have that the ridge regression model can be computed as $\boldsymbol{w}^* = \operatorname{argmin}_{\boldsymbol{w}\in\mathbb{R}^d} \|\bar{X}\boldsymbol{w} - \bar{\boldsymbol{y}}\|_2^2 + \bar{\lambda}||\boldsymbol{w}||_2^2$, with $\bar{\lambda} = 10^{2\ell}\lambda$. In particular, from now on we consider $A = \bar{X}^\top \bar{X} + \bar{\lambda} I$ (symmetric invertible matrix in $\mathbb{Z}^{d\times d}$) and $\boldsymbol{b} = \bar{X}^\top \bar{\boldsymbol{y}}$ (vector in $\mathbb{Z}^d$).

4.1 Phase 1: merging the dataset

Assume that the dataset represented by the matrix X and the vector $\boldsymbol{y}$ is horizontally-portioned in m datasets. This means that each DO_i holds some rows of the matrix X. We assume that the correspondence between rows and parties is publicly known and has this form: the data-owner DO_i holds $\mathcal{D}_i$ and

$$\mathcal{D}_i = \{(\boldsymbol{x}_{n_{i-1}+1}, y_{n_{i-1}+1}), \ldots, (\boldsymbol{x}_{n_i}, y_{n_i})\}$$

for $i = 1, \ldots, m$ ($n_0 = 1, n_m = n$). In this case, as already noticed in [NWI+13], we have the following: define $A_i = 10^{2\ell}\boldsymbol{x}_i\boldsymbol{x}_i^\top$ and $\boldsymbol{b}_i = 10^{2\ell} y_i \boldsymbol{x}_i$ for $i = 1, \ldots n$, then

$$A = \sum_{i=1}^{n} A_i + \bar{\lambda} I \quad \text{and} \quad \boldsymbol{b} = \sum_{i=1}^{n} \boldsymbol{b}_i.$$

Each A_i and $\boldsymbol{b}_i$ can be computed locally by the party DO_i; then, if additively homomorphic encryptions of such values are sent to the MLE, the latter can easily compute encryptions of A and $\boldsymbol{b}$. This is depicted in protocol Π^1_{hor} (Figure 2).

See Section 5.1 for the implementation of Phase 1 in the vertically-partitioned dataset setting.

Protocol $\Pi^1_{\mathbf{hor}}$

- *Parties*: CSP and MLE with no input, DO_i with input $\mathcal{D}_i$ for all $i = 1, \ldots, m$.
- *Output*: MLE gets A' and $\boldsymbol{b}'$ encryptions of A and $\boldsymbol{b}$, respectively.
- *Public parameters*: number of features d, total numbers of data points n, parameter λ.

Assume that $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is an additive encryption scheme with plaintext space $\mathcal{M} = \mathbb{Z}_N$ (*e.g.* Paillier's scheme) and security parameter κ.

Step 1: (*key-generation*) the CSP runs $\mathsf{Gen}(\kappa) \to (\boldsymbol{sk}, \boldsymbol{pk})$ and makes $\boldsymbol{pk}$ public, while it keeps $\boldsymbol{sk}$ secret.

Step 2: (*local computation*) for all $k = 1, \ldots, m$, DO_k does the following: Compute $A_k = 10^{2\ell} \sum_i \boldsymbol{x}_i \boldsymbol{x}_i^\top$ and $\boldsymbol{b}_k = 10^{2\ell} \sum_i y_i \boldsymbol{x}_i$ with $i = n_{k-1} + 1, \ldots, n_k$; encrypt

$$A'_k(i,j) = \mathsf{Enc}_{\boldsymbol{pk}}(A_k(i,j))$$
$$\boldsymbol{b}'_k(i) = \mathsf{Enc}_{\boldsymbol{pk}}(\boldsymbol{b}_k(i));$$

for all $i, j = 1, \ldots, d$ and $j \geq i$; finally, send all A'_k and $\boldsymbol{b}'_k$ to the MLE.

Step 3: (*datasets merge*) the MLE does the following

$$A'(i,j) = \begin{cases} \bigoplus_{k=1}^{m} A'_k(i,j) \oplus \mathsf{Enc}_{\boldsymbol{pk}}(10^{2\ell}\lambda) & \text{if } i = j \\ \bigoplus_{k=1}^{m} A'_k(i,j) & \text{otherwise} \end{cases}$$
$$\boldsymbol{b}'(i) = \bigoplus_{k=1}^{m} \boldsymbol{b}'_k(i)$$

for all $i, j = 1, \ldots, d$ and $j \geq i$;

Fig. 2. The protocol Π^1_{hor} implements Phase 1 in the horizontally-distributed setting.

4.2 Phase 2: computing the model

Recall that we want to compute $\boldsymbol{w}^*$ such that $A\boldsymbol{w}^* = \boldsymbol{b}$ where A is an invertible matrix in $\mathbb{Z}^{d \times d}$ and $\boldsymbol{b}$ is a vector in $\mathbb{Z}^d$. First of all, notice that

$$\boldsymbol{w}^* = \frac{1}{\det(A)} \boldsymbol{w}' \tag{2}$$

where $\boldsymbol{w}' = \det(A) A^{-1} \boldsymbol{b} = \mathrm{adj}(A)\boldsymbol{b} \in \mathbb{Z}^d$. Therefore, to compute $\boldsymbol{w}^*$ it is enough to compute $\boldsymbol{w}'$ and then rescale its entries by $\det(A)$. Since $\boldsymbol{w}'$ and $\det(A)$ are integer values, if N is a positive large integer, we can compute them in $\mathbb{Z}_N$ (modulo N). In order to do this in a privacy-preserving manner, protocol Π^2_{ridge} V.1 (Version 1) described in Figure 3 implements the masking trick described in Section 3. The MLE samples R and $\boldsymbol{r}$ uniformly at random with entries in $\mathbb{Z}_N$ and uses the encryption of A and $\boldsymbol{b}$ (from Phase 1) to compute encryptions of $C = AR \mod N$ and $\boldsymbol{d} = \boldsymbol{b} + A\boldsymbol{r} \mod N$ (*data masking*). These ciphertexts are sent to the CSP. The latter decrypts and computes $\tilde{\boldsymbol{w}}$ and $\det(C)$ (*masked model computation*). This values are sent back to the MLE who first computes $\boldsymbol{w}' \mod N$ and then divides it by $\det(A)$ (integer division) to compute $\boldsymbol{w}^*$ (*model reconstruction*).

Protocol $\Pi^2_{\mathbf{ridge}}$ V.1

The protocol Π^1_{hor} has been previously run.

- *Parties*: CSP knows $\boldsymbol{sk}$, the MLE knows $A' = \mathsf{Enc}_{\boldsymbol{pk}}(A)$ and $\boldsymbol{b}' = \mathsf{Enc}_{\boldsymbol{pk}}(\boldsymbol{b})$.
- *Output*: MLE gets $\boldsymbol{w}^*$ and $\det(A)$.
- *Public parameters*: number of features d, total numbers of data points n, $\mathcal{M} = \mathbb{Z}_N$.

Step 1: (*data masking*) the MLE does the following:
- sample $R \leftarrow \mathbb{Z}_N^{d\times d}$ (invertible) and $\boldsymbol{r} \leftarrow \mathbb{Z}_N^d$;
- compute the values

$$C'(i,j) = \bigoplus_{k=1}^{d} \mathsf{mult}(R(k,j), A'(i,k))$$
$$\boldsymbol{d}'(i) = \boldsymbol{b}'(i) \oplus \left(\bigoplus_{k=1}^{d} \mathsf{mult}(\boldsymbol{r}(k) A'(i,k)) \right)$$

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

Step 2: (*masked model computation*) the CSP does the following:
- decrypt

$$C(i,j) = \mathsf{Dec}_{\boldsymbol{sk}}(C'(i,j))$$
$$\boldsymbol{d}(i) = \mathsf{Dec}_{\boldsymbol{sk}}(\boldsymbol{d}'(i))$$

  for all $i, j = 1, \ldots, d$;
- compute

$$\tilde{\boldsymbol{w}} = C^{-1}\boldsymbol{d} \mod N$$
$$c = \det(C) \mod N$$

- send the vector $\tilde{\boldsymbol{w}}$ and the value c to the MLE.

Step 3: (*model reconstruction*) the MLE does the following:
- compute

$$\delta = c \det(R)^{-1} \mod N \quad \% \ \delta = \det(A)$$
$$\boldsymbol{w}' = \delta(R\tilde{\boldsymbol{w}} - \boldsymbol{r}) \mod N$$
$$\boldsymbol{w}^* = \tfrac{1}{\delta}\boldsymbol{w}'$$

- output the vector $\boldsymbol{w}^*$

Fig. 3. Version 1 (V.1) of the protocol Π^2_{ridge} implements Phase 2: the system $A\boldsymbol{w} = \boldsymbol{b}$ is solved and the MLE gets to know $\det(A)$. No other information about A and $\boldsymbol{b}$ is revealed.

Informally, protocol Π^2_{ridge} V.1 is secure against a honest-but-curious CSP because the values seen by it (the masked data $AR \mod N$ and $\boldsymbol{b} + A\boldsymbol{r} \mod N$) have a distribution that is unrelated with the input datasets. More precisely we have the following lemma. Let $\left(\mathbb{Z}_N^{d\times d}\right)^*$ be the set of all invertible matrices with coefficients in $\mathbb{Z}_N$.

Lemma 1. *Assume that R is sampled uniformly at random from $\left(\mathbb{Z}_N^{d\times d}\right)^*$ and that $\boldsymbol{r}$ is sampled uniformly at random from $\mathbb{Z}_N^d$. Then, the distribution of $(AR \mod N, \boldsymbol{b} + A\boldsymbol{r} \mod N)$ is the uniform one over $\left(\mathbb{Z}_N^{d\times d}\right)^* \times \mathbb{Z}_N^d$.*

Proof. Given $(M, \boldsymbol{v}) \in \left(\mathbb{Z}_N^{d\times d}\right)^* \times \mathbb{Z}_N^d$, let $\mathcal{M}_{M,\boldsymbol{v}} = \{(R, \boldsymbol{r}) \in \left(\mathbb{Z}_N^{d\times d}\right)^* \times \mathbb{Z}_N^d \mid AR = M \text{ and } \boldsymbol{b} + A\boldsymbol{r} = \boldsymbol{v}\}$. It is enough to notice that the map $(R, \boldsymbol{r}) \to (RM^{-1}, A^{-1}\boldsymbol{v} - \boldsymbol{r})$ is a bijection between $\mathcal{M}_{M,\boldsymbol{v}}$ and $\mathcal{M}_{I,\boldsymbol{0}}$.

Moreover, protocol Π^2_{ridge} V.1 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 $\boldsymbol{b}$. However, notice that in step 3 of Π^2_{ridge} V.1 the MLE receive the value $\det(A)$, which is used for the computation of the output $\boldsymbol{w}^*$. This value can leak information about the matrix A. In order to avoid this and keep $\det(A)$ private, we present Π^2_{ridge} V.2 (Version 2) in Figure 4. The latter has same blueprint as the first version, but it has an extra round of communication and assumes that the MLE and the CSP has access to an oracle that computes the function $f_{d,N} : \mathbb{Z}_N^{d+1} \times \mathbb{Z}_N^{d+1} \longrightarrow \mathbb{R}^d$ defined by

$$f_{d,N}\big((\boldsymbol{s}, s), (\boldsymbol{v}, v)\big) = \frac{(v - s)(\boldsymbol{v} - \boldsymbol{s}) \mod N}{(v - s) \mod N} \tag{3}$$

The extra round of communication is used to provide the two parties, the MLE and the CSP, with additive shares of the values $\boldsymbol{w}'$ and $\det(A)$ in the ring $\mathbb{Z}_N$. Then, in the last step each party inputs the known shares into the oracle implementing $f_{d,N}$. The latter simply reconstructs the vector $\boldsymbol{w}'$ and the value $\det(A)$ (adding the relative shares) and then computes $\boldsymbol{w}^*$ using the same formula as before, $\boldsymbol{w}^* = \frac{1}{\det(A)}\boldsymbol{w}'$.

Discussion (info leaked by $\det(A)$*)* Assume that it is known that $|A(i, j)| \leq p$ for some prime p. Then, given a non-zero integer d (with $\gcd(p, d) = 1$), we have that $\#\{A \in \mathbb{Z}_p^{d\times d} \mid \det(A) = \delta\} = p^{\binom{d}{2}}(p - 1)^{d-1}[d]!$ where $[d] = \frac{1-p^d}{1-p}$. Therefore, even when $\det(A)$ is know, A remains hidden in a set of cardinality $poly(d)$.

4.3 Security proofs

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 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*[2] *that $\Pr \phi(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n) \neq$ output of Π for $P_i = 0$*

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

Protocol Π^2_{ridge} V.2

The protocol Π^1_{hor} has been previously run.

- *Parties*: CSP knows $\boldsymbol{sk}$, the MLE knows A' and $\boldsymbol{b}'$
- *Output*: MLE gets $\boldsymbol{w}^*$
- *Public parameters*: number of features d, total numbers of data points n, $\mathcal{M} = \mathbb{Z}_N$.

Step 1: (*data masking*) same as in Π^2_{ridge} V.1;

Step 2: (*masked model computation*) the CSP does the following:
- decrypt

$$C(i,j) = \mathsf{Dec}_{\boldsymbol{sk}}(C'(i,j))$$
$$\boldsymbol{d}(i) = \mathsf{Dec}_{\boldsymbol{sk}}(\boldsymbol{d}'(i))$$

  for all $i, j = 1, \ldots, d$;
- compute

$$\tilde{\boldsymbol{w}} = C^{-1}\boldsymbol{d} \mod N$$
$$\tilde{\boldsymbol{w}}' = \mathsf{Enc}_{\boldsymbol{pk}}(\tilde{\boldsymbol{w}})$$
$$c' = \mathsf{Enc}_{\boldsymbol{pk}}(\det(C))$$

- send the vector $\tilde{\boldsymbol{w}}'$ and the value c' to the MLE.

Step 3: the MLE does the following:
- sample $\boldsymbol{s} \leftarrow \mathbb{Z}_N^d$ and $s \leftarrow \mathbb{Z}_N$;
- compute

$$v' = \mathsf{mult}(\det(R)^{-1}, c') \oplus \mathsf{Enc}_{\boldsymbol{pk}}(s)$$
$$\% \; v' \text{ is an encryption of } \det(A) + s \mod N$$
$$\boldsymbol{v}'(i) = \left(\bigoplus_{j=1}^{d} \mathsf{mult}(R(i,j), \tilde{\boldsymbol{w}}'(j))\right) \oplus (-\boldsymbol{r}(i)) \oplus \boldsymbol{s}(i)$$
$$\% \; \boldsymbol{v}' \text{ is an encryption of } R\tilde{\boldsymbol{w}} - \boldsymbol{r} + \boldsymbol{s} \mod N$$

  for all $i = 1, \ldots, d$;
- send the vector $\boldsymbol{v}'$ and the value v' to the CSP.

Step 4: the CSP compute

$$v = \mathsf{Dec}_{\boldsymbol{sk}}(v')$$
$$\boldsymbol{v} = \mathsf{Dec}_{\boldsymbol{sk}}(\boldsymbol{v}') \mod N$$

Step 5: (*model reconstruction*) the MLE and the CSP use an oracle to implement the function $f_{d,N}$ (3). The MLE inputs $\boldsymbol{s}, s$, the CSP inputs $\boldsymbol{v}, v$. In this way, the MLE obtains all coefficients of the vector $\boldsymbol{w}^*$.

Fig. 4. Version 2 (V.2) of the protocol Π^2_{ridge} implements Phase 2 of our system: the system $A\boldsymbol{w} = \boldsymbol{b}$ is solved without any party gaining extra information about A and $\boldsymbol{b}$.

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.

We are ready to state our first theorem:

Theorem 1. *Let Π V.1 be the protocol described in Figure 5 and ϕ be the function computing the linear regression model from the data in the clear ($\boldsymbol{w}^* = A^{-1}\boldsymbol{b}$) augmented with the computation of* $\det(A)$*. Let $D \subseteq \{1, \ldots, m\}$. Then, Π realizes ϕ with correctness and privacy against the adversaries $\mathcal{A}_1 = \{\text{MLE}\} \cup \{\text{DO}_i \,|\, i \in D\}$ and $\mathcal{A}_2 = \{\text{CSP}\} \cup \{\text{DO}_i \,|\, i \in D\}$.*

Proof. Correctness: Using the homomorphic properties of the underlying encryption scheme, it easy to verify that at the end of Phase 1 of Π, the MLE knows A' and $\boldsymbol{b}'$ such that $\mathsf{Dec}_{\boldsymbol{sk}}(A') = A$ and $\mathsf{Dec}_{\boldsymbol{sk}}(\boldsymbol{b}') = \boldsymbol{b}$. It is also easy to verify that after Step 3 in Π^2_{ridge} V.1 we have $\delta = \det(A)$, therefore

$$\frac{\det(A)(R\tilde{\boldsymbol{w}} - \boldsymbol{r}) \mod N}{\det(A) \mod N} = \frac{\boldsymbol{w}' \mod N}{\det(A) \mod N} = \frac{\boldsymbol{w}'}{\det(A)} = \boldsymbol{w}^*$$

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.
$\mathsf{Sim}_1(\{\mathcal{D}_i\}_{i \in D}, \boldsymbol{w}^*, \det(A))$:

1. Run $\mathsf{Gen}(\kappa) \to (\boldsymbol{pk}, \boldsymbol{sk})$;
2. For all $k = 1, \ldots, m$, if $k \in D$ compute A'_i and $\boldsymbol{b}'_i$ as in Step 2 of Π^1_{hor}. Otherwise compute A'_i and $\boldsymbol{b}'_i$ 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 $\tilde{\boldsymbol{w}} = R^{-1}(\boldsymbol{w}^* + \boldsymbol{r}) \mod N$ and $c = \det(A)\det(R) \mod N$;
5. Output $(\{\mathcal{D}_i\}_{i \in D}, \boldsymbol{pk}, enc, \tilde{\boldsymbol{w}}, c, \boldsymbol{w}^*)$ 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 V.1.
$\mathsf{Sim}_2(\{\mathcal{D}_i\}_{i \in D}, \boldsymbol{w}^*)$:

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

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

An *oracle-aided protocol* is an MPC protocol augmented by an oracle-functionality for computing a give function f. If a set of parties send inputs to the oracle, the latter computes f on such inputs and send back to the parties (or possible a subset of them) the result (see Definition 7.5.5 in [Gol04]). In this setting, it is possible to state a *Composition Theorem* (Theorem 7.5.7 in [Gol04]. Informally, this theorem says that if an n-party MPC oracle-aided protocol is proved secure, than it remains secure when the oracle-functionality is realized with privacy and correctness by another m-party MPC protocol.

In the following theorem we prove that version 2 of our system (protocol Π^1_{hor} + protocolΠ^2_{ridge} V.2) is secure when it uses an oracle-functionality for the function $f_{d,N}$ defined in (3). Therefore, because of the aforementioned composition theorem, our system stays secure when the oracle is replaced by any two-party secure MPC protocol for secure evaluation of $f_{d,N}$ run by the MLE and the CSP.

Theorem 2. *Let Π V.2 be the protocol described in Figure 5 and ϕ be the function computing the linear regression model from the data in the clear ($\boldsymbol{w}^* = A^{-1}\boldsymbol{b}$). Let $D \subseteq \{1, \ldots, m\}$. Then, Π realizes ϕ with correctness and privacy against the adversaries $\mathcal{A}_1 = \{\text{MLE}\} \cup \{\text{DO}_i \,|\, i \in D\}$ and $\mathcal{A}_2 = \{\text{CSP}\} \cup \{\text{DO}_i \,|\, i \in D\}$.*

Protocol Π V.1

- *Parties*: CSP and MLE with no input, DO_i with input $\mathcal{D}_i$ for $i = 1, \ldots, m$.
- *Output*: each party gets $\boldsymbol{w}^*$, the MLE gets also $\det(A)$.
- *Public parameters*: N, d, n, λ.

Phase 1: MLE, CSP and $\text{DO}_1, \ldots, \text{DO}_m$ jointly run Π^1_{hor};

Phase 2: - MLE and CSP jointly run Π^2_{ridge} V.1;
- MLE sends to the other parties $\boldsymbol{w}^*$.

Protocol Π V.2

- *Parties*: CSP and MLE with no input, DO_i with input $\mathcal{D}_i$ for $i = 1, \ldots, m$.
- *Output*: each party gets $\boldsymbol{w}^*$
- *Public parameters*: N, d, n, λ.

Phase 1: MLE, CSP and $\text{DO}_1, \ldots, \text{DO}_m$ jointly run Π^1_{hor};

Phase 2: - MLE and CSP jointly run Π^2_{ridge} V.2;
- MLE sends to the other parties $\boldsymbol{w}^*$.

Fig. 5. The 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 $\mathsf{Dec}_{\boldsymbol{sk}}(A') = A$ and $\mathsf{Dec}_{\boldsymbol{sk}}(\boldsymbol{b}') = \boldsymbol{b}$. It is also easy to verify that after Step 4 in Π^2_{ridge} V.2 we have $\boldsymbol{v} = R\tilde{\boldsymbol{w}} - \boldsymbol{r} + \boldsymbol{s} \mod N$ and $v = \det(A) + s \mod N$. Therefore the function $f_{d,N}$ computes:

$$\frac{(v-s)(\boldsymbol{v}-\boldsymbol{s}) \mod N}{(v-s) \mod N} = \frac{\det(A)(R\tilde{\boldsymbol{w}} - \boldsymbol{r}) \mod N}{\det(A) \mod N} = \frac{\boldsymbol{w}'}{\det(A)} = \boldsymbol{w}^*$$

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.
$\mathsf{Sim}_1(\{\mathcal{D}_i\}_{i\in D}, \boldsymbol{w}^*)$:

1. Run $\mathsf{Gen}(\kappa) \to (\boldsymbol{pk}, \boldsymbol{sk})$;
2. For all $k = 1, \ldots, m$, if $k \in D$ compute A'_i and $\boldsymbol{b}'_i$ as in Step 2 of Π^1_{hor}. Otherwise compute A'_i and $\boldsymbol{b}'_i$ 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. Compute the encryption of the zero vector (d components) and the encryption of 0.
4. Output $(\{\mathcal{D}_i\}_{i\in D}, \boldsymbol{pk}, enc, \boldsymbol{w}^*)$ where enc contains the ciphertexts computed in step (2) and (3).

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.
$\mathsf{Sim}_2(\{\mathcal{D}_i\}_{i\in D}, \boldsymbol{w}^*)$:

1. Run $\mathsf{Gen}(\kappa) \to (\boldsymbol{pk}, \boldsymbol{sk})$;
2. Sample R and $\boldsymbol{r}$ as in the protocol;
3. Compute $\mathsf{Enc}_{\boldsymbol{pk}}(R)$ and $\mathsf{Enc}_{\boldsymbol{pk}}(\boldsymbol{r})$;

4. Sample $\boldsymbol{s}$ and s as in the protocol;
5. Compute $\mathsf{Enc}_{\boldsymbol{pk}}(\boldsymbol{s})$ and $\mathsf{Enc}_{\boldsymbol{pk}}(s)$;
6. Output

$$(\{\mathcal{D}_i\}_{i\in D}, \boldsymbol{pk}, \mathsf{Enc}_{\boldsymbol{pk}}(R), \mathsf{Enc}_{\boldsymbol{pk}}(\boldsymbol{r}), \mathsf{Enc}_{\boldsymbol{pk}}(\boldsymbol{s}), \mathsf{Enc}_{\boldsymbol{pk}}(s), \boldsymbol{w}^*)$$

It follows from Lemma 2 that the simulation output has the same distribution of the views of the corrupted parties in $\mathcal{A}_2$ in the protocol Π.

Lemma 2. *Assume that R is sampled uniformly at random from $\left(\mathbb{Z}_N^{d\times d}\right)^*$, $\boldsymbol{r}, \boldsymbol{s}$ are sampled independently and uniformly at random from $\mathbb{Z}_N^d$ and s is sample uniformly at random from $\mathbb{Z}_N$. Then, the distribution of $(AR \mod N, \boldsymbol{b} + A\boldsymbol{r} \mod N, A^{-1}\boldsymbol{b} + \boldsymbol{s} \mod N, \det(A) + s \mod N)$ is the uniform one over $\left(\mathbb{Z}_N^{d\times d}\right)^* \times \mathbb{Z}_N^d \times \mathbb{Z}_N^d \times \mathbb{Z}_N$.*

Proof. Similar to proof of Lemma 1.

Discussion (active security) The protocol Π guarantees privacy when all the parties follow the instructions given in the protocol (passive security). Here we briefly discuss the security of Π in the case when the CSP or the MLE are corrupted and arbitrarily deviate from the protocol. We still assume that they do not collude.

If the CSP is corrupted, during Phase 2 it can send to the MLE a faulty $\tilde{\boldsymbol{w}}$ causing the computation of a wrong model $\tilde{\boldsymbol{w}}^*$. In this case, if we are in the horizontally-partitioned setting, each data-owner DO_i can verify on their local data the received model $\boldsymbol{w}^*$ and catch the cheating CSP. That is, DO_i checks that $(\boldsymbol{w}^*)^\top \boldsymbol{x}_i - y_i \in [-u, u]$ for a small u chosen by the parties. If all data-owners do not complain, the received model is valid. If we are in the vertically-partitioned setting, then the data-owners can jointly run an m-party MPC protocol to securely compute the sum of m inputs and again check that $(\boldsymbol{w}^*)^\top \boldsymbol{x}_i - y_i \in [-u, u]$ for all i.

If the MLE is corrupted, then it can decide to ignore o replace some of the ciphertexts received during Phase 1. This may be reveal extra information about some of the private datasets $\mathcal{D}_1, \ldots, \mathcal{D}_m$. A solution to avoid this threat in the horizontally-partitioned setting is proposed in [NWI+13]. They use one-time MACs, Pedersen commitments and standard zero-knowledge proofs. Extending this solution to the vertically-partitioned using multiplicatively homomorphic commitments is left for future work.

Discussion. Notice that, as the protocol Π works with arithmetic on a finite ring, it computes an approximation of the original linear regression functionality (the one computing a linear regression model directly from real-valued data). In [FIM+06], Feigenbaum et al. show that the output of an approximation may reveal more information than the output of the original function. For this reason, they define the notion of security approximation and give a protocol to make an approximation satisfy such definition. A detailed study of our system in the framework proposed by Feigenbaum et al. is left for future work.

5 Extensions and Future Work

5.1 Vertically-Partitioned Setting

Assume that the dataset represented by the matrix X and the vector $\boldsymbol{y}$ is vertically-portioned in m datasets. In other words, we assume that each DO_i holds some columns of X and $\boldsymbol{y}$. We assume the correspondence between columns and parties is publicly known and can be represented in the following way. For $i = 1, \ldots, m-1$ ($n_0 = 1, n_{m-1} = d$), define

$$X_i = \begin{pmatrix} \boldsymbol{x}_1(n_{i-1}+1) & \ldots & \boldsymbol{x}_1(n_i) \\ \vdots & & \vdots \\ \boldsymbol{x}_n(n_{i-1}+1 & \ldots & \boldsymbol{x}_n(n_i) \end{pmatrix} \quad \text{and} \quad X_m = \boldsymbol{y}$$

and let $\mathcal{D}_i = \{X_i\}$. The data-owner DO_i holds $\mathcal{D}_i$. In this case, we have that

$$A = 10^{2\ell} \begin{pmatrix} X_1^\top X_1 & X_1^\top X_2 & \dots & X_1^\top X_{m-1} \\ \vdots & \vdots & & \vdots \\ X_{m-1}^\top X_1 & X_{m-1}^\top X_2 & \dots & X_{m-1}^\top X_{m-1} \end{pmatrix} + \bar{\lambda} I \tag{4}$$

and

$$\boldsymbol{b} = 10^{2\ell} \begin{pmatrix} X_1^\top X_m \\ \vdots \\ X_{m-1}^\top X_m \end{pmatrix} \tag{5}$$

If all values in X are encrypted, computing an encryption of $X_i^\top X_j$ requires multiplications of pairs of ciphertexts. An additive encryption scheme (*e.g.* Paillier's scheme) cannot handle such operation. On the other hand, an encryption of A and $\boldsymbol{b}$ can be computed from encryptions of the local datasets, if the underling scheme is one-time multiplicative. We say that an encryption scheme is **one-time multiplicative** if it is an additive encryption scheme that also support one multiplication among ciphertext. That is, we assume that in $(\mathcal{M}, +, \cdot)$ is a ring and that there is a product operation $\otimes$ defined on the ciphertext space $\mathcal{C}$ such that for any pair of ciphertexts $\mathsf{Enc}_{\boldsymbol{pk}}(\boldsymbol{x}_1) \to \boldsymbol{c}_1$ and $\mathsf{Enc}_{\boldsymbol{pk}}(\boldsymbol{x}_2) \to \boldsymbol{c}_2$, it holds that $\mathsf{Dec}_{\boldsymbol{sk}}(\boldsymbol{c}_1 \otimes \boldsymbol{c}_2) = \boldsymbol{x}_1 \cdot \boldsymbol{x}_2$. An example of one-time multiplicative scheme can be found in [GHV10][3].

Example 2. (GHV scheme [GHV10].) Let κ, a be integers and p, q primes ($a, q = poly(\kappa)$). We have $\mathcal{M} = \mathbb{Z}_p^{a \times a}$ and

1. $\mathsf{Gen}(\kappa)$: run the trapdoor function in [AP09] and obtain a matrix $A \in \mathbb{Z}_q^{a \times \kappa}$ and an invertible matrix $T \in \mathbb{Z}^{a \times \kappa}$ such that $TA = 0 \mod q$. Define $\boldsymbol{pk} = A$ and $\boldsymbol{sk} = T$;
2. $\mathsf{Enc}_{\boldsymbol{pk}}(M)$: sample S, X and compute $C = AS + pX + M \mod q$;
3. $\mathsf{Dec}_{\boldsymbol{sk}}(C)$: let $E = TCT^\top \mod q$ and output $T^{-1}E(T^\top)^{-1} \mod p$.

Given this, protocol Π^1_{ver} (Figure 6) describes how to compute encryptions of A and $\boldsymbol{b}$ using a one-time multiplicative scheme such as the GHV scheme.

The implementations of Phase 2 in the vertically-partitioned setting remains basically unchanged. Π^2_{ridge} (V.1 or V.2) can be used if we updated it in order to take in to account the change of the plaintext space from $\mathbb{Z}_N$ to $\mathbb{Z}_p^{a \times a}$ and of the block-wise structure of A' and $\boldsymbol{b}'$.

5.2 Lasso Regression

Beside 2-norm, which grantees a model with overall smaller components, another commonly used regularization term is 1-norm ($R(\boldsymbol{w}) = \|\boldsymbol{w}\|_1$). The latter provides a sparse model (*i.e.* a model with only few non-zero components). Note that, 1-norm regularization is often preferred since in practice it performs feature selection. In this case (known in the literature as *lasso regression*), the model $\boldsymbol{w}^*$ is computed by minimizing the function $F_{\mathrm{lasso}}(\boldsymbol{w}) = \|X\boldsymbol{w} - \boldsymbol{y}\|_2^2 + \lambda\|\boldsymbol{w}\|_1$. Unlike ridge regression, lasso-penalized linear regression does not have a closed form solution, if $\lambda > 0$. Moreover, F_{lasso} is not differentiable everywhere, and finding its minimum point cannot even be accomplished by gradient descent. Nevertheless, it is possible to efficiently find an approximation of it. Indeed, it is known that finding the minimum point of F_{lasso} is equivalent to solving the following convex minimization problem [FT07]:

$$\begin{cases} \mathrm{argmin}_{\boldsymbol{w} \in \mathbb{R}^d} \|X\boldsymbol{w} - \boldsymbol{y}\|_2^2 \\ \text{subject to } \|\boldsymbol{w}\|_1 \leq s \end{cases} \tag{6}$$

[3] Similar to the Boneh-Goh-Nassim cryptosystem [BGN05], but with a more efficient encryption and decryption algorithms.

Protocol $\Pi^1_{\mathbf{ver}}$

- *Parties*: CSP and MLE with no input, DO_i with input $\mathcal{D}_i$ for all $i = 1, \dots, m$.
- *Output*: MLE gets A' and $\boldsymbol{b}'$ encryptions of A and $\boldsymbol{b}$, respectively.
- *Public parameters*: d, n, λ, ℓ.

Assume that $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a one-time multiplicative encryption scheme with plaintext space $\mathcal{M} = \mathbb{Z}_p^{a \times a}$ and security parameter κ. Assume $a \geq (n_{i+1} - n_i)$ for all i and let $b = \lceil n/a \rceil$.

Step 1: (*key-generation*) the CSP runs $\mathsf{Gen}(\kappa) \to (\boldsymbol{sk}, \boldsymbol{pk})$ and makes $\boldsymbol{pk}$ public, while it keeps $\boldsymbol{sk}$ secret.

Step 2: (*local computation*) for all $i = 1, \dots, m$, DO_i does the following
- split X_i in b matrices with a rows

$$\boldsymbol{X}_i = \begin{pmatrix} M_{i1} \\ \vdots \\ M_{ib} \end{pmatrix}$$

(eventually padding with zeros to have $M_{ij} \in \mathbb{Z}_p^{a \times a}$) and compute $M'_{ij} = \mathsf{Enc}_{\boldsymbol{pk}}(10^\ell M_{ij})$, $M''_{ij} = \mathsf{Enc}_{\boldsymbol{pk}}(10^\ell M_{ij}^\top)$ for all $j = 1, \dots, b$;

- send all M'_{ij}, M''_{ij} to the MLE.

Step 3: (*datasets merge*) the MLE computes the encryption of $X_i^\top X_j = \sum_k M_{ik}^\top M_{jk}$ with $j \geq i$ and $i, j = 1, \dots m$ as $\bigoplus_{k=1}^b M''_{ik} \otimes M'_{kj}$. The MLE uses these values to forms encryptions of A and $\boldsymbol{b}$ according to (4) and (5).

Fig. 6. The protocol Π^1_{ver} implements Phase 1 in the vertically-distributed setting.

where $s \in \mathbb{R}$ is fixed[4]. An approximation of the solution[5] of (6) can be found using the *conditional gradient descent* method. This method, also known as the Frank-Wolfe (FW) algorithm (see Chapter 3 in [Bub15]), is an iterative algorithm that works in the following way: given an initial point $\boldsymbol{w}_1$ in $\mathbb{R}^d$ with $\|\boldsymbol{w}_1\|_1 \leq s$, then for any $j = 1, 2, \dots$ compute

$$\begin{cases} \boldsymbol{v}_j \in \operatorname{argmin}_{\|\boldsymbol{v}\|_1 \leq s} \{(A\boldsymbol{w}_j - \boldsymbol{b})^\top \boldsymbol{v}\} \\ \boldsymbol{w}_{j+1} = (1 - \gamma_j)\boldsymbol{w}_j - \gamma_j \boldsymbol{v}_j \end{cases}$$

where $A = X^\top X$, $\boldsymbol{b} = X^\top \boldsymbol{y}$ and $\gamma_j = \frac{2}{j+1}$.

We could try to extend the "masking trick" to this setting in the following way. Consider the following problem:

$$\begin{cases} \operatorname{argmin}_{\boldsymbol{w} \in \mathbb{R}^d} \|XR\boldsymbol{w} - (\boldsymbol{y} + X\boldsymbol{r})\|_2^2 \\ \text{subject to } \|R\boldsymbol{w} - \boldsymbol{r}\|_1 \leq s \end{cases}$$

where R is any $d \times d$ matrix and $\boldsymbol{r}$ is a vector of d components. Let $\tilde{\boldsymbol{w}}$ be solution of (5.2). If R is invertible and $\boldsymbol{w}^* = R\tilde{\boldsymbol{w}} - \boldsymbol{r}$, then $\boldsymbol{w}^*$ is solution of (6). If the FW algorithm is used in order

[4] If we allow $s = +\infty$, then standard linear regression is also represented by (6).
[5] If $\mathrm{rk}(X) = d$, then (6) has a unique solution.

to compute an approximation of $\tilde{\boldsymbol{w}}$, then i^{th} iterative step has the form

$$\begin{cases} \text{argmin}_{\boldsymbol{v}\in\mathbb{R}^d}[AR\tilde{\boldsymbol{w}}_{t-1} - (\boldsymbol{b} + A\boldsymbol{r})]^\top \boldsymbol{v} \\ \text{subject to } \|\boldsymbol{v}\|_1 \leq s \end{cases} \tag{7}$$

Therefore, it would seem that our system can be extended to compute lasso regression models: the CSP knows $(AR, \boldsymbol{b}+A\boldsymbol{r})$ and computes the masked model that them is used by the MLE to reconstruct the desired model. Unfortunately, this is not true: the question is that the masking trick $(A, \boldsymbol{b}) \to (AR, \boldsymbol{b}+A\boldsymbol{r})$ is secure only if done over a finite ring (*e.g.* the CSP sees $(AR, \boldsymbol{b}+A\boldsymbol{r})$ mod N), while the problem (7) has not a clear equivalent in such a ring. So, our system does not extend trivially to the computation of lasso regression models and moreover we are are not aware of any system that can efficiently train a lasso regression model on private datasets. Given the large number of current applications of lasso regression in the bio-medical field, we believe that filling this gap is an interesting direction for future work.

6 Implementation

In this section we describe our implementation case study of the system described in Section 4 to train a ridge regression model on encrypted and distributed datasets. We evaluate the system's accuracy and efficiency testing our implementation on different datasets.

6.1 Setup

We implement our system using Paillier's scheme as underlying additive encryption scheme ($N = pq$ and p, q primes of bit-length $\kappa = 1024$). We wrote our software in Python3 5.2 using the phe1.3 library[6] to implement Paillier encryption/decryption and operations on ciphertexts. We use the gmpy2 library[7] to implement arithmetic operations with large integers. To compute the determinant function and to solve linear systems, we use the Gaussian elimination (adapted to work on the ring $\mathbb{Z}_N$). To multiply to square matrices we use the Strassen algorithm.

In each experiment, we split the n data-points evenly among the m data-owners. It is known that the predictive accuracy of the ridge model obtained depends on λ. In machine learning, different techniques are used to tune the regularization parameter and 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 minor 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. All experiments were run on on a machine with a 2.6GHz single CPU under Ubuntu Linux 16.04.

6.2 Experiments results

Synthetic data To evaluate the effect of the public parameters (number of data points n, number of features d, number of fractional digits ℓ, number of parties m) on our system 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 $[-1, 1]^n$ $(i = 1, \ldots, n)$. The coefficients of the vector $\boldsymbol{w}^*$ are sampled independently and uniformly at random from the real interval $[0, 1]$. Finally $y_i = \boldsymbol{x}_i^\top \boldsymbol{w}^* + \epsilon_i$, where ϵ_i is sampled from a Gaussian distribution with zero mean and variance 1.

[6] http://python-paillier.readthedocs.io
[7] https://pypi.python.org/pypi/gmpy2

To evaluate accuracy of our system we compare the output of Π (the vector $\tilde{\boldsymbol{w}}^*$) with the model $\boldsymbol{w}^*$ learned from the data X and $\boldsymbol{y}$ in the clear when the same regularization parameter λ is used. Note that, to run Π we truncate the values in X and $\boldsymbol{y}$ after the ℓ^{th} digit in the fractional part, while the model $\boldsymbol{w}^*$ (in the clear) is computed using floating point representation for X and $\boldsymbol{y}$. The error is measured using the 2-norm (Euclidean distance of the vector $\tilde{\boldsymbol{w}}^*$ from the vector $\boldsymbol{w}^*$, optimal solution)

$$E_2 = \|\tilde{\boldsymbol{w}}^* - \boldsymbol{w}^*\|_2$$

Moreover, we measure the error rate of the system computing

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

We measure E_1 and E_2 for different values of the parameters n and d (with $m = 10$, $\lambda = 0.1$ constant). A summary of the results of this experiment for version 1 (protocol Π V.1) can be read in Figure 7 and Table 1. Both the error measures are of the order of 10^{-5} (or below) when ℓ is at least 5. Notice the running-time of our protocol Π V.1 does not vary significantly by increasing the number ℓ of digits used for the fractional part of the data (while increasing ℓ significantly decreases the error).

n	d	ℓ	E_2	E_1	Phase 1 (Π^1_{hor})	Phase 2 (Π^2_{ridge} V.1)
10^3	10	1	1.17E-01	1.05E-02	1.033	13.831
		3	1.22E-03	6.99E-05	1.082	14.000
		5	1.17E-05	6.66E-07	1.126	13.600
		7	1.21E-07	6.32E-09	1.174	13.266
	20	1	1.22E-01	8.47E-03	3.203	87.407
		3	1.28E-03	7.99E-05	3.173	87.327
		5	1.18E-05	7.05E-07	3.213	87.414
		7	1.20E-07	6.05E-09	3.277	87.441
10^5	10	1	6.39E-02	1.52E-03	1.041	13.226
		3	6.22E-04	7.44E-06	1.122	13.218
		5	6.23E-06	7.53E-08	1.113	13.239
		7	6.22E-08	7.46E-10	1.058	13.217
	20	1	1.49E-01	7.37E-03	3.416	87.763
		3	1.44E-03	9.78E-06	3.476	87.395
		5	1.44E-05	1.01E-07	3.436	87.361
		7	1.44E-07	1.00E-09	3.401	87.360

Table 1. Errors and running times of system Π (version 1) run on synthetically generated data for different values of digits in the fractional part (average on 10 repetitions). All the timings are reported in seconds. System parameters: $\lambda = 0.1, m = 10, \kappa = 1024$.

To evaluate efficiency of our system, we report the running time of Phase 1 (protocol Π^1_{hor}) and Phase 2 (protocol Π^2_{ridge} V.1 and V.2) for different combinations of parameters.

In Phase 1, Step 2 is the most expensive one (see Table 2). Here, each data-owner computes the encryptions of the entries of the locally computed $d \times d$ matrices A_i. In our setting (n data-points evenly split among all DO_ks), this costs $\frac{n}{m}\frac{d(d+1)}{2}$ standard multiplications and $\frac{d(d+1)}{2}$

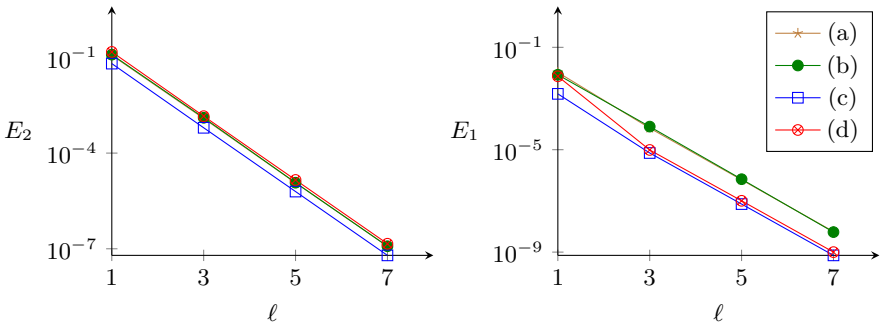

Fig. 7. Relation among the average of error E_1 (E_2 resp.) and the number of fractional digits ℓ for different parameters choice in the protocol Π V.1. (a) $n = 10^3, d = 10$, (b) $n = 10^3, d = 20$, (c) $n = 10^5, d = 20$, (d) $n = 10^5, d = 20$

encryptions for one data-owner. If $\frac{n}{m}$ is small, the costs of the encryptions is dominant[8]. In this case, the efficiency of this step can be improved using batching techniques to pack multiple plaintext values in one ciphertext. However, in order to do this, a careful implementation of the operation done over the chipertext in Step 1 in Π^2_{ridge} (especially for the operation mult) becomes necessary. We leave this optimization for future work. As expected, the running time of protocol Π^1_{hor} is sensitive to the parameters n, d and m, while the efficiency of Phase 2 (protocol Π^2_{ridge}) depends mainly on the number of features d.

In Phase 2 (version 1), Step 1 is the most expensive (see Table 3). Here the MLE needs to compute $O(d^{2.8074})$ operations[9] on ciphertexts (multiplications of a ciphertext by a plaintext and addition of ciphertexts). The use of more performing libraries (or programming languages) for implementing Paillier's scheme will directly improve the efficiency of this step.

For version 2 of protocol Π^2_{ridge}, we report running times for the first 4 steps in Table 4. Step 5 (the oracle) can be implemented by different 2-party protocols. In Appendix A.2, we report on the implementation of Step 5 based on Yao's two-party protocol and using the Obliv-C library[10] to generate the relative code.

Real Datasets Finally, we run our system on 9 different real-world datasets downloaded from the UCI repository[11]. References about each one of the datasets are given in Table 6. The results of experiments running protocol Π V.1 are reported in Table 5. For each dataset, we removed the data points with missing values and we use 1-of-k encoding to convert nominal features to numerical ones. Moreover, we split the dataset in 11 subsets: 10 of them were used to train the model with our system ($m = 10$), the last subset we used to compute the error rate E_1. In Table 5, n indicates the number of data-points used for training.

Discussion (parameters restriction) In order to get the correct output $\boldsymbol{w}^*$ using the protocol Π^2_{ridge}, we need that both $\det(A)$ and $\boldsymbol{w}'$ are integers in the interval $[-\frac{N-1}{2}, \frac{N-1}{2}]$. This implies

[8] In our code we have that the average running time for 1 multiplication is 3.972530e-06 seconds, while the average time for 1 encryption is 0.0136508. Thus if $\frac{n}{m} \leq 3436.3$, the encoding time is the predominant one for Step 2 in Π^1_{hor}.

[9] The asymptotic complexity of the Strassen algorithm to multiply two $d \times d$ matrices is $O(d^{2.8074})$. Notice that in our case the use of the Strassen algorithm in the place of standard matrix multiplication significantly reduces the running time. Indeed the Strassen algorithm reduces the number of multiplications replacing them with additions and subtractions and in the library we use (phe1.3), an encrypted multiplication is about 100 times expensive than an encrypted addition/subtraction.

[10] https://oblivc.org

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

	n	d	Phase 1 (Π^1_{hor})		
			Step1	Step2	Step3
$m = 5$	10^3	10	0.096	0.856	0.005
		20	0.308	3.163	0.022
	10^4	10	0.157	0.910	0.006
		20	0.166	3.239	0.022
	10^5	10	0.178	0.999	0.005
		20	0.185	3.568	0.022
	10^6	10	0.197	2.038	0.005
		20	0.120	7.183	0.020
	10^7	10	0.135	12.281	0.006
		20	0.089	43.035	0.020
$m = 10$	10^3	10	0.188	0.888	0.012
		20	0.160	3.180	0.047
	10^4	10	0.091	0.901	0.013
		20	0.144	3.084	0.042
	10^5	10	0.149	0.905	0.011
		20	0.222	3.196	0.044
	10^6	10	0.234	1.392	0.012
		20	0.281	4.892	0.044
	10^7	10	0.109	6.251	0.011
		20	0.086	21.898	0.044
$m = 20$	10^3	10	0.089	0.856	0.023
		20	0.168	3.088	0.101
	10^4	10	0.199	0.882	0.025
		20	0.399	3.118	0.094
	10^5	10	0.192	0.905	0.024
		20	0.175	3.215	0.092
	10^6	10	0.464	1.185	0.024
		20	0.165	4.133	0.099
	10^7	10	0.202	3.758	0.026
		20	0.287	13.166	0.097

Table 2. Running times of protocol Π^1_{hor} (Phase 1) run on synthetically generated data. All timing are reported in seconds (average on 3 repetitions). The n data-points are evenly split among m data-owners. The columns "Step 2" report the average running time for one data-owner. System parameters: $\ell = 5$, $\lambda = 1$, $m = 10$,$\kappa = 1024$.

n	d	E_2	E_1	Phase 1 (Π^1_{hor})			Phase 2 (Π^2_{ridge} V.1)		
				Step 1	Step 2	Step 3	Step 1	Step 2	Step 3
10^3	10	7.44E-06	5.13E-07	0.094	0.870	0.012	12.912	0.424	0.001
	15	1.33E-05	1.69E-06	0.104	1.768	0.025	34.187	0.934	0.001
	20	1.27E-05	1.14E-06	0.144	3.027	0.044	85.648	1.657	0.002
	25	1.36E-05	1.08E-06	0.169	4.586	0.070	197.747	2.617	0.003
10^5	10	8.17E-06	1.04E-10	0.116	0.913	0.011	12.817	0.433	0.001
	15	1.38E-05	1.26E-07	0.137	1.882	0.024	34.462	0.937	0.001
	20	1.14E-05	5.36E-08	0.151	3.223	0.044	85.662	1.649	0.002
	25	1.90E-05	3.11E-07	0.132	4.870	0.068	197.766	2.567	0.003
10^7	10	8.85E-06	9.06E-10	0.132	6.417	0.012	12.837	0.426	0.001
	15	1.12E-05	1.04E-08	0.171	12.914	0.025	34.542	0.936	0.001
	20	1.24E-05	2.18E-09	0.141	22.236	0.044	85.628	1.673	0.002
	25	1.71E-05	1.57E-08	0.086	33.568	0.067	199.723	2.583	0.003

Table 3. Errors and running times of system Π (version 1) run on synthetically generated data. All the timings are reported in seconds (average on 3 repetitions). System parameters: $\ell = 5, \lambda = 0.1, m = 10, \kappa = 1024$.

n	d	E_2	E_1	Phase 2 (Π^2_{ridge} V.2)			
				Step 1	Step 2	Step 3	Step 4
10^3	10	7.10E-06	6.55E-07	12.867	0.571	1.335	0.042
	15	1.05E-05	5.16E-07	33.999	1.141	2.970	0.060
	20	1.58E-05	6.74E-07	85.700	1.928	5.266	0.079
	25	1.58E-05	2.34E-06	196.783	2.973	8.265	0.098
10^5	10	8.73E-06	9.77E-08	13.201	0.568	1.329	0.041
	15	1.27E-05	5.19E-08	34.379	1.142	2.965	0.060
	20	1.42E-05	3.33E-07	86.276	1.930	5.274	0.079
	25	1.37E-05	1.03E-07	196.673	3.013	8.507	0.100
10^7	10	1.05E-05	1.32E-08	13.630	0.587	1.398	0.047
	15	1.13E-05	1.71E-08	36.253	1.293	3.216	0.067
	20	1.31E-05	9.14E-09	85.951	1.927	5.263	0.079
	25	1.61E-05	1.15E-08	199.360	3.342	8.692	0.098

Table 4. Running times of protocol Π^2_{ridge} V.2 run on synthetically generated data. All the timings are reported in seconds (average on 3 repetitions). Error values reported here assumes that Step 5 is implemented by an ideal oracle. System parameters: $\ell = 5, \lambda = 0.1, \kappa = 1024$.

restrictions on the dataset that our system can handle. Assume that for all i and j, $\boldsymbol{x}_i(j) \in [-x, x]$ and $y_i \in [-y, y]$ for some positive $x, y \in \mathbb{R}$. Then, for all i and j we have, $|(X^\top X)(i,j)| \leq nx^2$ and $|(X^\top \boldsymbol{y})(i)| \leq nxy$. Using the Hadamard's inequality to bound the absolute value of $\det(A)$ and of the entries of $\text{adj}(A)$, we obtain that protocol Π^2_{ridge} produces the correct result when

$$\max\left\{10^{2\ell d}(nx^2+\lambda)^d d^{\frac{d}{2}}, d(d-1)^{\frac{(d-1)}{2}} 10^{2\ell d} nxy(nx^2+\lambda)^{d-1}\right\} \leq \frac{N-1}{2}$$

For example, if $x = 1$, $y = d$, $n = 10^\alpha$ and N integer of 2048 bits, the above is guaranteed to be true if $\frac{d}{2}\log_{10}(d) + (2\ell + \alpha)d \leq 615$.

Dataset	n	d	E_2	E_1	Phase 1 (Π^1_{hor})			Phase 2 (Π^2_{ridge} V.1)		
					Step 1	Step 2	Step 3	Step 1	Step 2	Step 3
forest	470	12	6.32E-07	2.95E-08	0.169	1.248	0.006	20.410	0.649	0.001
boston	460	13	1.12E-04	1.10E-06	0.153	1.441	0.007	31.329	0.760	0.001
facebook	454	17	9.40E-10	1.03E-12	0.201	2.376	0.013	67.742	1.255	0.002
air	6314	13	1.41E-10	2.15E-13	0.193	1.443	0.012	31.179	0.736	0.001
beijing	37960	14	1.13E-06	4.26E-10	0.161	1.590	0.010	31.725	0.819	0.001
wine	4452	11	3.05E-05	2.04E-07	0.222	1.054	0.006	17.871	0.519	0.001
bike	15799	14	1.38E-10	6.47E-15	0.199	1.575	0.009	31.659	0.827	0.001
energy	17940	25	2.64E-05	4.76E-08	0.146	4.624	0.026	196.829	2.591	0.003
student	359	30	9.84E-13	1.89E-14	0.229	6.622	0.033	232.898	3.958	0.005

Table 5. Errors and running times of system Π (version 1) run on real-world datasets from the UCI repository. All the timings are reported in seconds (10-cross validation). System parameters: $\ell = 5, \lambda = 0.1, m = 10, \kappa = 1024$.

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

Table 6. References for the UCI datasets.

7 Conclusions

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. We design a first protocol that is based on simple cryptographic primitives (*i.e.* Paillier's encryption scheme and pseudorandom generators) and a second one that improves the security of the first protocol assuming the existence of a trusted implementation of a simple functionality (division). We present a case study implementation and discuss parameter restrictions. We show that for real-world dataset, our system is fast (*e.g.* version 1 can train a model on a dataset of almost 18 thousands instances with 25 features split evenly in 10 datasets in less than 4 minutes) and accurate (the distance from the produced model and the optimal one is always the order of at most 10^{-4} in all experiments).

References

AFGH06. Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. *ACM Trans. Inf. Syst. Secur.*, 9(1):1–30, 2006.

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.

AP09. Joël Alwen and Chris Peikert. Generating shorter bases for hard random lattices. In *26th International Symposium on Theoretical Aspects of Computer Science, STACS 2009, February 26-28, 2009, Freiburg, Germany, Proceedings*, pages 75–86, 2009.

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.

BGN05. Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, pages 325–341, 2005.

Bub15. Sébastien Bubeck. Convex optimization: Algorithms and complexity. *Foundations and Trends in Machine Learning*, 8(3-4):231–357, 2015.

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.

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.

FIM+06. Joan Feigenbaum, Yuval Ishai, Tal Malkin, Kobbi Nissim, Martin J. Strauss, and Rebecca N. Wright. Secure multiparty computation of approximations. *ACM Trans. Algorithms*, 2(3):435–472, 2006.

FT07. Michael P. Friedlander and Paul Tseng. Exact regularization of convex programs. *SIAM Journal on Optimization*, 18(4):1326–1350, 2007.

GHV10. Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. A simple BGN-type cryptosystem from LWE. In *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings*, pages 506–522, 2010.

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

GSB+16. Adrià Gascón, Phillipp Schoppmann, Borja Balle, Mariana Raykova, Jack Doerner, Samee Zahur, and David Evans. Privacy-preserving distributed linear regression on high-dimensional data. Cryptology ePrint Archive, Report 2016/892, 2016.

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.

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.

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.

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.

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.

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.

ZE15. Samee Zahur and David Evans. Obliv-c: A language for extensible data-oblivious computation. *IACR Cryptology ePrint Archive*, 2015:1153, 2015.

A Appendix

A.1 Further experimental results

Dataset	n	d	ℓ	E_2	E_1	Phase 1 (Π^1_{hor})			Phase 2 (Π^2_{ridge} V.1)		
						Step 1	Step 2	Step 3	Step 1	Step 2	Step 3
beijing	37960	14	1	3.34E-02	6.25E-05	0.194	1.589	0.009	31.380	0.816	0.001
			3	9.88E-05	2.16E-08	0.157	1.587	0.009	31.416	0.821	0.001
			5	1.13E-06	4.26E-10	0.161	1.590	0.010	31.725	0.819	0.001
wine	4452	11	1	1.25E 00	2.66E-02	0.148	1.010	0.005	17.363	0.513	0.001
			3	5.27E-03	2.48E-04	0.200	1.031	0.005	17.367	0.514	0.001
			5	3.05E-05	2.04E-07	0.222	1.054	0.006	17.871	0.519	0.001

Table 7. Running time for system Π (version 1) run on real-world datasets from the UCI repository. All the timings are reported in seconds (10-cross validation). System parameters: $\lambda = 0.1, m = 10, \kappa = 1024$.

A.2 Yao's protocol for Step 5 in Π^2_{ridge} V.2

In this section we present a case study implementation of the oracle functionality used in Step 5 of protocol Π^2_{ridge} V.2. In particular, we assume that the MLE and the CSP use Yao's 2-party protocol [Yao86,LP09] in order to securely compute the function $f_{d,N}$ (defined in Section 4.2) on input $\boldsymbol{s}, s$ from the MLE, and $\boldsymbol{v}, v$ from the CSP. We generate the code for this protocol using the Obliv-C library[12]. Notice that $\boldsymbol{s}, \boldsymbol{v} \in \mathbb{Z}_N^d$ and $N = pq$ (p, q 1024 bits primes). Thus, all components of $\boldsymbol{s}, s, \boldsymbol{v}, v$ need to be represented as 2048-bits integers. On the other hand, in the Obliv-C library the largest C++ type for integer is "long long" which has 64 bits. For this reason, in the following we need to consider integers represented in blocks (block-int). More precisely, in order to allow for overflow during arithmetic operation, we use 32-bits signed integer (int32) in each block. This implies that each component of $\boldsymbol{s}, s, \boldsymbol{v}, v$, and N is represented by $\lceil \frac{2048}{31} \rceil = 67$ blocks. For the addition/subtraction and the multiplication of block-int we use standard arithmetic algorithms. To compute the modulo reduction we use the algorithm in Figure 8. We tested the code on synthetically generated data. Results of this experiment are reported in Table 8.

If high dimension datasets are dealt with, an overall runtime might be too long. However, the computation of function $f_{d,N}$ could be easily parallelized on multi-processors, because each coefficient of the output (the model $\boldsymbol{w}^*$) is computed independently.

[12] Obliv-C [ZE15] is a C++ library that includes an efficient implementation of Yao's protocol (based on Yao's garbling scheme).

Input: $A = \sum_{i=0}^{66} a_i M^i$ and $B = \sum_{i=0}^{66} b_i M^i$ with $A, B = 2048$ bits signed integers, $a_i, b_i = 32$ bits signed integers and $M = 2^{31}$ (because of 1 signed bit).
Output: Let $A = QB + R$, the output is R.

/*preprocess*/
$A' = |A|$
$B' = |B|$
$x = \max(i)$, where $a'_i \neq 0$
$y = \max(i)$, where $b'_i \neq 0$
$z = x - y$
$B' = B' * M^z$

/*compute remainder*/
while
if $(A' > B')$
if $(\frac{a'_x}{b'_y+1} > 1)$ **then** $A' = A' - \left\lfloor \frac{a'_x}{b'_y+1} \right\rfloor B'$ **else** $A' = A' - B'$
else
if $(z > 0)$ **then** $z = z - 1$, $B' = \frac{B'}{M}$ **else** break

/*adjust a sign of remainder*/
if $(A * B > 0)$ **then** return A' **else** return $-A'$

Fig. 8. Algorithm to compute the remainder in a division among two block-integers.

n	d	E_2	E_1	Phase 1 (Π^1_{hor})			Phase 2 (Π^2_{ridge} V.2)				
				Step 1	Step 2	Step 3	Step 1	Step 2	Step 3	Step 4	Step 5
10^3	5	7.81E-06	1.32E-06	0.097	0.263	0.003	1.969	0.193	0.343	0.023	195.025
	10	8.30E-06	6.35E-07	0.143	0.858	0.012	12.984	0.594	1.405	0.044	516.760
	15	1.04E-05	8.81E-07	0.243	1.791	0.026	34.151	1.178	3.152	0.064	964.283

Table 8. Errors and running times for our system Π (version 2) run on synthetically generated data. Step 5 is implemented using Yao's scheme. All the timings are reported in seconds (average on 3 repetitions). System parameters: $\ell = 5, \lambda = 0.1, m = 10, \kappa = 1024$.