Secure Linear Regression on Vertically Partitioned Datasets

Adrià Gascón[1], Phillipp Schoppmann[1], Borja Balle[2], Mariana Raykova[3], Jack Doerner[4], Samee Zahur[4], and David Evans[4]

[1]University of Edinburgh
[2]Lancaster University
[3]Yale University
[4]University of Virginia

We propose multi-party computation protocols for securely computing a linear regression model from a training dataset whose columns are distributed among several parties. Our solution enables organizations to collaborate in the construction of a predictive model while keeping their training data private. Our approach is based on a hybrid MPC protocol combining garbled circuits with an offline phase enabled by a semi-trusted external party. As part of our contribution, we evaluate several algorithms and implementations for solving systems of linear equations using garbled circuts. Experiments conducted with an implementation of our protocols show that our approach leads to highly scalable solutions that can solve data analysis problems with one million records and one hundred features in less than one hour.

1 Introduction

Predictive modelling is an essential technique used in practice by enterprises and governments in spaces as diverse as policy making, medicine, law enforcement, banking, and the work place. While some of such data analyses are unfeasible due to technical constraints, others are limited by ethical/regulatory constraints, or conflicting interests of the parties contributing their data.

For example, nowadays big companies such as Google, Amazon, Netflix and many others use machine learning techniques to build predictive models from the data collected from their users. However, oftentimes different companies have information about the same set of users contributing different features to their profiles. For instance, Google has information about the clickstream of their users on the advertisements that they place on different websites, while an online merchant advertising with Google has information about the purchases of its users. Being able to correlate the two datasets can allow the merchant to build a better advertising strategy. Another common scenario is when the same patients have been treated at different hospitals and the information of their complete medical profiles is distributed between multiple institutions. Medical data is extremely sensitive information and is subject to very strict sharing restrictions, but at the same time being able to analyze complete patient profiles can lead to important insights for their treatment. While being able to build predictive models on joint databases where the contents are contributed by different sources has clear benefits, this can raise several risks and concerns. Sharing data between different companies can lead to many privacy risks for their users. Legal agreements can provide one type of mitigation for such risks, however, such solutions are often imperfect and unable to account for security breaches. Thus, many times the privacy policies that companies offer to protect users explicitly exclude the possibility of data sharing.

These scenarios demonstrate that the assumption that the whole database is available in the clear in the same place might be an unreasonable requirement for many practical applications of machine learning involving private data. In all of these settings having a method to run a machine learning algorithm privately using data from multiple sources will bring a substantial improvement. While there are cryptographic techniques such as secure multiparty computation (MPC) [5,17,40] and fully homomorphic encryption (FHE) [7,8,15] that provide general solutions for the problem of computation on private data, using these techniques in a black-box way rarely scales to problem sizes commonly found in real-world applications requiring privacy-preserving data analysis on massive datasets. The search for practical alternatives has lead to the design of hybrid approaches where carefully engineered protocols combine existing MPC tools in order to taking advantage of specific optimizations applicable to the given scenario.

In this work we consider the question of privately computing a linear regression in settings with vertically partitioned input databases. Linear regression is a fundamental building block of many machine learning algorithms which solves the task of finding a best-fit linear curve through a set of input data points stored in a database as vectors of features. A vertically partitioned database means that the features for each record are partitioned among different input parties, i.e. if each record is a row in the database, then the input parties provide different subsets of the database columns. Next we elaborate on the exact problem that we are addressing.

1.1 Problem statement

Linear regression is a data analysis task where one is interested in fitting a linear model for predicting a real-valued attribute given a vector of features. For example, in a health care environment one might be interested in predicting what is the right dosage of a certain drug for a patient as a function of representative features e.g. age, weight, and concentration of certain chemicals in her blood. Formally speaking, a linear model is a real-valued function on d-dimensional vectors of the form $x \mapsto \langle \theta, x \rangle$, where $x \in \mathbb{R}^d$ is the input vector, $\theta \in \mathbb{R}^d$ is the vector of parameters specifying the model, and $\langle \theta, x \rangle = \sum_{j=1}^{d} \theta_j x_j$ is the inner product between θ and x. The term linear comes from the fact that the output predicted by the function $\langle \theta, x \rangle$ is a linear combination of the features represented in x. In the example above, each coordinate in x would contain the value of one attribute associated to a patient, and the predicted dosage would be obtained as a weighted sum of such attributes.

In order to find this linear model one assumes access to a *training set* of samples representing the input-output relation the model should try to replicate. These are denoted as $(x^1, y^1), \dots, (x^n, y^n)$ with $x^i \in \mathbb{R}^d$ and $y^i \in \mathbb{R}$ for $i \in [n]$. In our example these might represent historic data where each sample represents the characteristics of a patient and the amount of medication an expert decided to give to that patient. A possible motivation for building a model in this case could be to assist general practitioners and reduce the dependence on particular experts. Given the training data, a standard way to find the parameters of the linear model is to solve a *ridge regression* problem given by the optimization

$$\mathsf{argmin}_\theta \frac{1}{n} \sum_{i=1}^{n} (y^i - \langle \theta, x^i \rangle)^2 + \lambda \|\theta\|^2 \ , \tag{1}$$

where $\lambda > 0$ is a regularization parameter. Ignoring the the term weighted by λ for a moment, this optimization can be interpreted as finding a parameter vector θ which minimizes the average squared error between the predictions made by the model on the training examples x^i and the desired outputs y^i. The ridge penalty weighted by λ is used to prevent the model from overfitting when the amount of training data is too small (see Section 3.2 for more details on ridge regression). An important feature of this optimization problem is that its solution can be written in closed form. To simplify the presentation, let us write $X \in \mathbb{R}^{n \times d}$ for a matrix with n rows corresponding to the different vectors x^i, and $Y \in \mathbb{R}^n$ for a vectors with n entries corresponding to the labels y^i. In this case the training data is represented by the pair (X, Y) and the solution to (1) reduces to solving the following system of linear equations:

$$\left(\frac{1}{n} X^\top X + \lambda I \right) \theta = \frac{1}{n} X^\top Y \ . \tag{2}$$

For convenience we henceforth write $A = \frac{1}{n} X^\top X + \lambda I$ and $b = \frac{1}{n} X^\top Y$, in which case the system of equations can be written as $A\theta = b$.

The formula above makes ridge regression an easy problem to solve in general by just relying on numerical linear algebra procedures for solving linear systems. However, an important problem arises when it is not possible to have the whole training data set $\{(x^i, y^i)\}_{i=1}^n$ in a single database before the training of the model begins. In particular, two simple settings where the training data is not contained in a single database can be considered. The first one is the case where the data is *horizontally partitioned*, in which case a fraction of the n training samples is in the possession of one party and the rest of the data is in the possession of another party. This corresponds to splitting the rows of X into two disjoint parts. A second scenario can occur when the data is *vertically partitioned*, in which case a party owns data from all samples corresponding to a fraction of the d features, and another party owns the rest of features and the desired outputs. This corresponds to splitting the columns of X into two disjoint parts. It is easy to imagine how such situations can occurs in applications where the data that has to analyzed is sensitive to privacy concerns. In our medical example horizontally partitioned data can occur when two different health provides want to put together historic records of different patients to get a better predictor than what each of them could do by itself. Similarly, vertically partitioned data can occur when different attributes about the same patients (e.g. historic data and results of blood tests) are held by different healthcare providers. A solution for ridge regression with horizontally partitioned datasets was implicitly provided in [35]. In this paper we focus on the vertically partitioned case, which we will see gives rise to a different set of problems. Next section provides a brief outline of the contributions of the paper.

1.2 Our Contributions

We propose secure computation protocols for solving ridge regression on a vertically partitioned dataset. Our protocols are divided in two phases of independent interest: (1) the secure computation of the coefficients A and b defining the linear system $A\theta = b$ described above; and, (2) the solution of such a system when the coefficients are securely held by two or more parties in the form of additive shares. We develop the following tools in order to solve each of these phases securely:
Phase 1. We construct a two-party computation protocol where several parties that hold vertical partitions of matrix X compute additive shares of $X^\top X$ and $X^\top Y$ securely. This protocol builds upon a protocol for secure inner product that relies on a semi-trusted party,
Phase 2. In this phase we have to solve a private system of linear equations $A\theta = b$ with a positive definite coefficient matrix like the ones arising in ridge regression. Here we assume that the inputs A and b are additively shared between two parties, which is a setup also arising in [35]. We evaluate several methods for solving such systems, and in particular show that iterative methods such as coordinate gradient descent (CGD) can be used to enable analysis on high dimensional data.

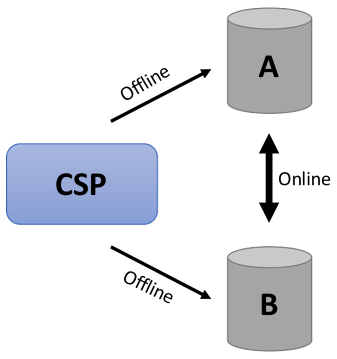
Our protocols are implemented using state of the art two-party computation using garbled circuit. In particular, we used the framework Obliv-C extended with a bigint library to handle fixed point arithmetic computations with arbitrary precision.

1.3 Outline of the paper

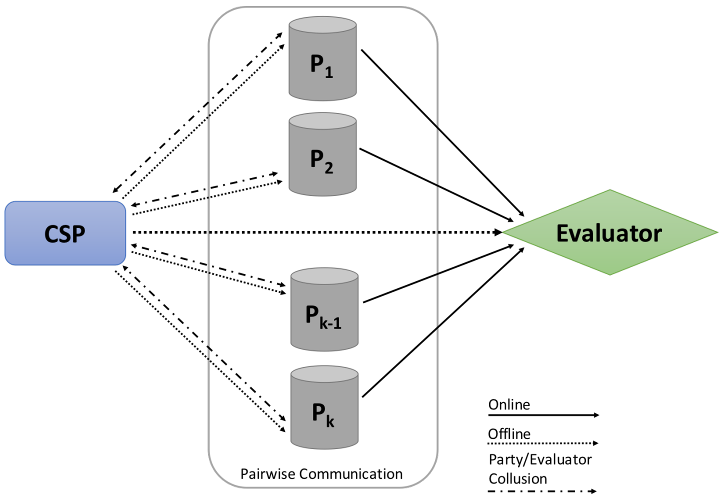
Section 2 gives a high-level description of the architecture and threat models our protocols are based on. Necessary preliminaries on linear systems, ridge regression, and MPC are given in Section 3. The details of our protocol for phase 1 in the 2 party case are given in Section 4. A discussion of different approaches for solving phase 2 is presented in Section 5. Next, in Section 6, we present an architecture and protocols to address the more general setting when X is partitioned among more than 2 parties. In Section 7 we present experimental results obtained from an implementation of our protocols.

2 Architecture and Threat Model

We consider two main settings for our protocols that have different system architecture. The first setting assumes that the input database is vertically partitioned between two parties who want to build a model of the data evaluating a ridge regression over the data. The output model can be either shared between the two parties or obtained by one or both parties. We assume that the parties are semi-honest and follow the prescribed steps in a protocol. Under this assumption we want to guarantee that neither party learns more about the input data than its designated output. Similarly to the architecture presented in the work of Nikolaenko et al. [35] we further assume a trusted initializer for the system, which we also name as in [35] "Crypto Service Provider" (CSP) (see Figure 1a). For the purposes



(a) Two Parties



(b) Many Parties

Figure 1: System Archtecture

of our protocols the CSP will provide correlated randomness for the parties. It is assumed to be semi-honest. Also, all of its work can be done offline before the inputs for the protocol are available.

In the second setting that we consider, the database is vertically partitioned among many parties, which we will call *data providers*. While there are multi-party computation protocols that allow any number of parties to evaluate jointly any functionality, the most efficient such protocols [10, 28] require a preprocessing stage where all parties interact. However, in our setting the parties who will jointly compute the prediction model only have to come online to run the protocol, and to do so, only bilateral computations are needed. Hence, such preprocessing model does not bring any advantage for this scenario. We leverage the CSP party, which is assumed to be semi-honest and not colluding in order to avoid preprocessing that involves all data providers. All preprocessing will be done by the CSP and could be done independently of the exact parties who will be later involved in the computation. We also consider how we can use an additional party to facilitate the efficiency of the protocols, but also give a variant of our protocols where that role can be fulfilled by some of the parties.

More concretely, we in fact adopt the full architecture of Nikolaenko et al. [35], which in addition to the CSP includes an Evaluator, both of which are assumed to be semi-honest (see Figure 1b). The Evaluator collects inputs from all parties and computes a ridge regression model. We allow communication between the parties during the collection process for their inputs. As long as we assume that there is no collusion between input parties and Evaluator we can guarantee that the CSP's work is completely offline. In order to protect against such collusions we require the CSP to interact with the data providers during the input contribution. We note that once we protect against collusions between Evaluator and the input parties, we no longer need to assume that Evaluator is not input contributor and thus can be instantiated by one of the parties.

3 Preliminaries

3.1 Solving Systems of Linear Equations

A linear system with m equations and d unknowns is an expression of the form $A\theta = b$ where $A \in \mathbb{R}^{m \times d}$ and $b \in \mathbb{R}^m$ are the coefficients of the system and $\theta \in \mathbb{R}^d$ contains the unknowns. Finding solutions of such equations is one the most important operations in numerical linear algebra. In this paper we will be dealing exclusively with systems where the coefficient matrix A is symmetric (i.e. $m = d$ and $A^\top = A$) and positive definite (i.e. $\theta^\top A\theta > 0$ for all $\theta \in \mathbb{R}^d$). Such systems are among the easiest to solve, in part because they admit a unique solution, and in part because it is possible to factorize a positive definite matrix in ways that make the system easier to solve.

Many factors can influence the choice of algorithm when a linear system needs to be solved in practice. Roughly speaking, algorithms for solving linear systems can be classified in three large groups. The first group is that of direct algorithms which apply a variant of Gaussian elimination to reduce the coefficient matrix to row echelon form and afterwards solve the resulting system via forward substitution. Different variants can be obtained depending on the details of the reduction algorithm, and sophisticated pivoting strategies are sometimes implemented in order to obtain numerically stable algorithms. The second group of algorithms apply a pre-processing step in which the coefficient matrix is factorized as the product of two or more coefficient matrices of easily solvable linear systems, and then the corresponding systems are solved sequentially until the solution of the original problem is obtained. The efficiency and stability of these algorithms heavily depends on that of the particular factorization being used, and are commonly used for classes of matrices admitting a particular class of decompositions. The third method casts the solution of $A\theta = b$ as the problem of minimizing $\|A\theta - b\|$ and then applies an iterative optimization algorithm. The main advantage of this approach is that the method produces a sequence of refined solutions at each iteration and provides a way to find an approximate solution even if the algorithms is stopped before convergence. The number of iterations required to find a good solution typically depends on the condition number of A.

Details of the particular algorithms we implement to solve phase 2 of our protocols are given in Section 5. In particular we implement two algorithms based on matrix factorizations, and one iterative algorithm. All these algorithms are especially designed to solve systems of equations with positive definite coefficient matrices.

3.2 Statistical Setup of Ridge Regression

Recall that in a linear regression problem one is given a dataset with n pairs of the form (x^i, y^i) with $x^i \in \mathbb{R}^d$, $y^i \in \mathbb{R}$, $1 \leq i \leq n$, and is asked to find a vector of parameters $\hat{\theta} \in \mathbb{R}^d$ that can be used to make predictions of the form $\langle \hat{\theta}, x^i \rangle$ about the y^i associated to a given x^i. One way to justify the least-squares and ridge regression algorithms is by assuming the input pairs satisfy $y^i = \langle \theta^*, x^i \rangle + \epsilon^i$ for some unknown $\theta^* \in \mathbb{R}^d$ and ask for a method to estimate a parameter vector $\hat{\theta}$ converging to θ^* as $n \to \infty$. Two common assumptions in this setup are that the errors $\epsilon^i \sim \mathcal{N}(0, \sigma^2)$ are independent random Gaussian noises with zero mean and variance σ^2, and that the observed feature vectors x^i are sampled i.i.d. from some unknown probability distribution on $\mathbb{R}^d$.

Under these assumptions it can be shown that the asymptotically optimal solution as $n \to \infty$ is to take the least squares solution $\hat{\theta} = \mathsf{argmin}_\theta \frac{1}{n} \sum_{i=1}^{n} (y^i - \langle \theta, x^i \rangle)^2$. In cases where the amount of available samples n is not big enough, this approach might lead to overfitting and yield parameters $\hat{\theta}$ with poor predictive performance on unobserved x's. This is typically remedied by considering a regularized optimization problem where the least squares criterion is augmented with a ridge penalty, leading to the well-known formulation of ridge regression given in (1). A statistical analysis of the generalization error of the output of this algorithm suggests that a good choice for the regularization parameter is $\lambda = \sigma^2 d/n \|\theta^*\|^2$. This is the setup we will be using in the experiments described in Section 5. Note that this choice of λ depends on quantities that might not be known when working with real data, in which case the data analysis pipeline has to include a parameter tuning strategy, e.g. cross-validation. The implementation of a secure data analysis pipeline including hyper-parameter tuning is beyond the scope of this paper; here we only focus on solving a secure ridge regression problem with fixed parameters.

3.3 Garbled Circuits

Cryptographic methods for two party computation allow two parties with inputs x_1 and x_2 to evaluate a function f that depends on their private inputs without revealing anything more than the output of the computation $f(x_1, x_2)$. One of the most popular and widely used approaches for two party computation is the protocol based on Yao garbled circuits (GCs) [40]. A garbling protocol consists of the following algorithms ($\mathsf{GC.GarbCirc}$, $\mathsf{GC.GarbInp}$, $\mathsf{GC.Eval}$), which work as follows:

$\mathsf{GC.GarbCirc}(C)$: this protocol takes as input a boolean circuit C and outputs a garbled circuit $\tilde{C}$, which is computed as follows. For each wire i of the circuit the garbler generates two random values $(w_{i,0}, w_{i,1})$, which can be used as keys for a symmetric encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$. For each gate g in the circuit that has input wires i and j and output wire k, the garbler computes a garbled gate $\tilde{g}$, which consists of the following four ciphertexts in randomly permuted order:

$$\begin{gathered}
\mathsf{Enc}_{w_{i,0}} \left(\mathsf{Enc}_{w_{j,0}} (w_{k,g(0,0)}) \right) \\
\mathsf{Enc}_{w_{i,0}} \left(\mathsf{Enc}_{w_{j,1}} (w_{k,g(0,1)}) \right) \\
\mathsf{Enc}_{w_{i,1}} \left(\mathsf{Enc}_{w_{j,0}} (w_{k,g(1,0)}) \right) \\
\mathsf{Enc}_{w_{i,1}} \left(\mathsf{Enc}_{w_{j,1}} (w_{k,g(1,1)}) \right),
\end{gathered}$$

where $g(b_1, b_2)$ is the output bit of the gate on input bits b_1 and b_2. Then, the garbled circuit $\tilde{C}$ is defined to be $\{\tilde{g}\}_{g \in C}$.

$\mathsf{GC.GarbInp}(x, \{(w_{i,0}, w_{i,1})\}_{i \in [n]})$: Let $x = x[1] \dots x[n]$ be the bit representation of the inputs. The corresponding garbled input $\tilde{x}$ is defined to be $\{w_{i,x[i]}\}_{i \in [n]}$.

$\mathsf{GC.Eval}(\tilde{C}, \tilde{x})$: The evaluator evaluates the gates of the circuit in topological order which guarantees that a gate is evaluated when its input wires have already been assigned values. Thus, for each gate g of the circuit the evaluator proceeds as follows: it has two input garbled values w_{i,b_1} and w_{j,b_2} and uses them to decrypt one of the ciphertexts in $\tilde{g}$. It assigns the decrypted value to be the garbled value for the output wire of the gate.

A two party computation protocol using garbled circuits proceeds as follows: one of the parties generates a garbled circuit and sends it to the other party together with the garbled values for its own inputs. For each input bit of the evaluator the two parties run an oblivious transfer protocol (OT) [32,37] with inputs the garbled values for that input wire $(w_{i,0}, w_{i,1})$ and the input bit b, which enables the evaluator to learn only the value $w_{i,b}$. Once the evaluator has all input garbled labels, it can evaluate $\mathsf{GC.Eval}(\tilde{C}, \tilde{x})$. If the evaluator needs to learn the output of the protocol, then it is also given the mappings between real and garbled values on the output wires.

3.4 Fixed-Point Arithmetic

In many data analysis problems the training data is given as real numbers encoded using a floating-point representation. Secure multiparty computation solutions capable of performing floating-point arithmetic are the subject of current research. For example, recent advances have provided proof-of-concept implementations of floating point primitives using garbled circuits [11]. However, the publicly available tools in this area do not yet scale to the throughtput required for data analysis tasks involving large datasets. Therefore, the protocols presented in this paper rely on fixed-point encodings of real numbers. We will see that under proper normalizations of the input data and an adequate choice of the precision, these encodings can guarantee that the computations in our protocols are very efficient and only incur a small, controlled amount of error with respect to the value one would obtain using floating-point arithmetic. Additionally, this error decreases as the number of bits used in the fixed point encoding increases, yielding a useful knob for trading off speed and accuracy in our protocols.

Our fixed-point encoding strategy follows closely previous works on secure linear regression [9, 19, 35]. However, unlike these works, we shall undertake a formal analysis of the errors such encondings introduce when used to simulate operations in floating-point arithmetic. The encoding has two parts: one mapping reals into integers, and another mapping integers to integers modulo some large q. The purpose of the first part is to represent all reals up to a certain precision using integers, thus creating equivalence classes of reals that cannot be distinguished using the given precision. The second part makes models the fact that a computer can only represent finitely many integers, and gives a way to simulate operations on the integer-encoded reals in a finite ring provided q is large enough to avoid overflows. In addition, using the fact that one can sample uniformly at random from finite rings it is possible to leverage this representation to construct statistically secure multiparty computation protocols. The encoding and decoding maps that we introduce next are summarized in the following diagram:

$$\begin{tikzcd} \mathbb{R} \arrow[r, shift left, "\phi_\delta"] & \mathbb{Z} \arrow[l, shift left, "\tilde{\phi}_\delta"] \arrow[r, shift left, "\varphi_q"] & \mathbb{Z}_q \arrow[l, shift left, "\tilde{\varphi}_q"] \end{tikzcd}$$

The effect of the precision when encoding real numbers using integers is controlled by a parameter $\delta > 0$ which is used to define an mapping from reals to integers given by $\phi_\delta(r) = [r/\delta]$. Here $[\cdot]$ returns the rounding of a real number to the closest integer, with ties favoring the number closest to 0. For example, for a given δ one can see that ϕ maps the interval $[-\delta/2, \delta/2]$ to 0, $(\delta/2, 3\delta/2]$ to 1, and $[-3\delta/2, -\delta/2)$ to -1. Thus, δ can be interpreted as the smallest number that can be represented by the encoding ϕ, e.g. if $\delta = 10^{-\Delta}$ for some integer Δ then $\phi_\delta(r)$ yields a representation of r truncated to the Δth decimal. When integers are represented in binary it is convenient to take $\delta = 2^{-\Delta}$. Using the precision parameter δ we also define a decoding of integers to real numbers acting as an approximate inverse of the encoding ϕ_δ. This decoding is given by $\tilde{\phi}_\delta(z) = z\delta$, and it is easy to see that it satisfied $|r - \tilde{\phi}_\delta(\phi_\delta(r))| \leq \delta$ for any real number r.

The most important property of a fixed-point encoding is how well the ring operations on $\mathbb{Z}$ operating on encoded reals can approximate the corresponding operations in real arithmetic. Addition is simple since using linearity of the decoding map it easy to show that for any reals r, r' one has $|(r+r') - \tilde{\phi}_\delta(\phi_\delta(r)+\phi_\delta(r'))| \leq 2\delta$. Multiplications are slightly more involved because in general 2Δ decimals are required to represent exactly the fractional part of the result of multiplying two numbers with Δ decimals. Taking this into account, one can show the following bound on the error introduced by performing multiplications in fixed-point: $|(rr') - \tilde{\phi}_{\delta^2}(\phi_\delta(r)\phi_\delta(r'))| \leq (|r| + |r'|)\delta + \delta^2$. Note this bound depends on the magnitude of the numbers being multiplied and the decoding is done using precision δ^2.

When using ϕ_δ to encode reals bounded by M, i.e. $|r| \leq M$, we obtain integers in the finite range $[-M/\delta] \leq \phi_\delta(r) \leq [M/\delta]$. There are at most $K = 2\lceil M/\delta \rceil + 1$ integers in this range, and therefore it is possible to map them injectively into the ring $\mathbb{Z}_q$ of integers modulo q with $q > K$ using the map $\varphi_q(z) = z \mod q$. For integers in the range $-q/2 \leq z \leq q/2$ this map is given by $\varphi_q(z) = z$ if $z \geq 0$ and $\varphi_q(z) = q + z$ for $z < 0$. Thus it makes sense to define a decoding map $\tilde{\varphi}_q : \mathbb{Z}_q \to \mathbb{Z}$ with $\tilde{\varphi}_q(u) = u$ if $0 \leq u \leq q/2$ and $\tilde{\varphi}_q(u) = u - q$ for $q/2 < u \leq q - 1$. Then we have $\tilde{\varphi}_q(\varphi_q(z)) = z$ for any $-q/2 \leq z \leq q/2$.

Although φ_q is a ring homomorphism translating operations in $\mathbb{Z}$ into operations in $\mathbb{Z}_q$, decoding from $\mathbb{Z}_q$ to $\mathbb{Z}$ after performing ring operations on encoded integers will not in general yield the desired result due to the occurrence of overflows. To avoid such overflows one must check that the result fall in the interval where the coding φ_q is the inverse of $\tilde{\varphi}_q$: suppose z, z' are integers such that $|z|, |z'| \leq q/2$, then

1. if $|z + z'| \leq q/2$, then $z + z' = \tilde{\varphi}_q(\varphi_q(z) + \varphi_q(z'))$, and
2. if $|zz'| \leq q/2$, then $zz' = \tilde{\varphi}_q(\varphi_q(z)\varphi_q(z'))$.

4 Two Party Case

In this section we consider the two party setting for the problem where the database is vertically partitioned between two parties A and B and at the end of the protocol the parties hold shares of the model constructed by solving ridge regression on the data. We will use this case to introduce important observation that we use later in our protocol for an arbirary number of parties.

As explained above, our problem reduces to securely solving a system of linear equations $A\theta = b$ where $A = \frac{1}{n}X^\top X + \lambda I$ and $b = \frac{1}{n}X^\top Y$, where $X \in \mathbb{R}^{n \times d}$ is a matrix with n rows corresponding to the different vectors x^i, and $Y \in \mathbb{R}^n$ for a vectors with n entries corresponding to the labels y^i.

Note that in most applications of data analysis one can assume $n \gg d$. Hence, it is critical for scalability purposes that the computations that depend on n are very efficient.

A naive approach to solve the ridge regression problem is to execute the whole computation using one large garbled circuit. However, this computation results in a the evaluation of a huge circuit, since only calculating $X^\top X$ involves computing inner products over vector of length n.

Instead we take a more careful look at output that the two parties need to compute. In particular, we can divide the problem in two well defined phases as mentioned above:

1. At the end of the first phase, the parties A and B hold a share of the matrix $X^\top X + \lambda I$ and the vector $X^\top Y$. In other words, our first phase is a two-party computation resulting in an additive share of the coefficients of the system $A\theta = b$. Note that this computation has cost $O(nd^2)$.

2. In our second phase, we run a two-party multiparty computation to solve the shared system obtained in the previous phase. Note that the input of this phase has size is of the order of d^2, and hence is independent of the number of records in the database.

A similar partitioning of the problem was used in [35] to tackle the variant of our problem where the input database is horizontally partitioned among the parties. Moreover, solving a linear system of equations is a central problem in machine learning, and hence our contributions regarding the second phase of our solution are of independent interest.

4.1 Phase 1: Securely Computing $X^\top X$

In this section, we focus of the main task of the first phase of our protocol: computing shares of $X^\top X$. We note that once this computation is done, obtaining A is easy because it amounts to adding a publicly known matrix to $X^\top X$. In addition, the computation of $X^\top Y$ can be done using exactly the same protocol.

We start by having a closer look at what needs to be computed. Recall that X is partitioned vertically among the two parties A and B and, for simplicity, assume that each party holds half of the features. More concretely, as illustrated in Figure 2 Party A holds a matrix X_{A} which contains half of the features for each record in the database and party B holds a matrix X_{B} which contains of the other half of the features for records. Figure 2 shows how the content of the output matrix depends on the inputs of the two parties. We observe that the upper left part of the matrix $M = X^\top X$ depends only on the input of party A and the lower right part depends only on the input of party B. In that case, the corresponding entries of M can be computed locally by A, while B simply has to set her shares of those entries to 0. On the other hand, for entries M_{ij} of M such that features i and j are held by distinct parties, the parties need to compte an inner product between a column vector from X_{A} and a column vector from X_{B}. To do so, we rely on a protocol for computing secure distributed inner product, which we present next.

4.1.1 Secure inner product

In this section, we present a two party computation protocol which enables two parties, each of which holds an input vector, to compute the inner product of their vectors and obtain additive shares of the result. As mentioned above, our numbers are represented as elements of a finite field $\mathbb{Z}_q$, and hence such functionality can be described as

$$f(\vec{x}, \vec{y}) = (r, \langle \vec{x}, \vec{y} \rangle - r), \text{where } \vec{x}, \vec{y} \in \mathbb{Z}_q^n, r \leftarrow U(\mathbb{Z}_q)$$

There are several techniques that can be used for this task as, essentially, it corresponds to secure multiplication. Note that this task is the only point in the whole linear regression

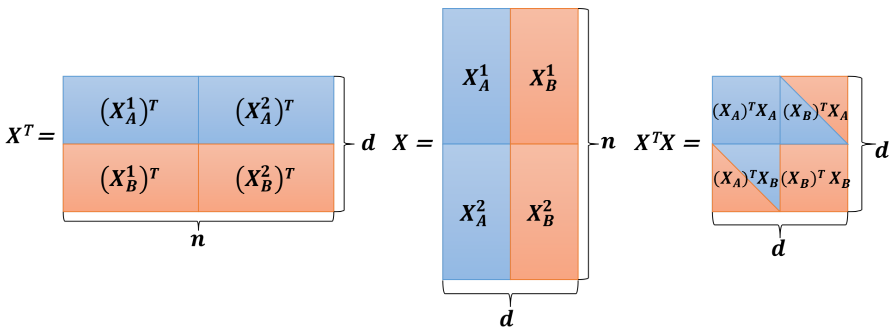


Figure 2: Computation of $X^\top X$.

Parties: A, B, and trusted initializer T.
Inputs: $\mathsf{A} : \vec{a} \in \mathbb{Z}_q^n$; $\mathsf{B} : \vec{b} \in \mathbb{Z}_q^n$.
Outputs: $\mathsf{A} : r_\mathsf{A} \in \mathbb{Z}_q$; $B : r_\in \mathbb{Z}_q$ such that $r_\mathsf{A} + r_\mathsf{B} = \langle \vec{a}, \vec{b} \rangle$.

Protocol:

1. T generates random vectors $\vec{x}, \vec{y} \in \mathbb{Z}_q^n$ and a random number $r \in \mathbb{Z}_q$, and sets $z = \langle \vec{x}, \vec{y} \rangle - r$. It sends $(\vec{x}, r)$ to A, and $(\vec{y}, z)$ to B.
2. A sends $\vec{a} + \vec{x}$ to B.
3. B sends $\vec{b} - \vec{y}$ to A.
4. A computes its output share as $r_\mathsf{A} = \langle \vec{a}, \vec{b} - \vec{y} \rangle - r$.
5. B computes its output share as $r_\mathsf{B} = \langle \vec{a} + \vec{x}, \vec{y} \rangle - z$.

Figure 3: Secure two-party inner product/multiplication supported by a trusted initializer.

computation where we need to perform computations on data whose size is of the order of n, namely the number of records in the database.

A first alternative is based in additive homomorphic encryption, where A encrypts all the entries in her vector and sends them to B. Then, B uses the homomorphic properties of the encryption to compute the encrypted inner product and also subtract its own share, and finally sends the result to A, which decrypts the ciphertext and obtains its output share. This protocol requires expensive homomorphic operations. Instead, we use a very efficient protocol, presented in Figure 3, that uses only symmetric key operations. The protocol leverages the use of a trusted initializer party, which in our architecture is instantiated by the CSP, who provides correlated randomness for the execution of the protocol. The work of this party can be done offline before the inputs of the computation are available. Our protocol is a modification of the construction presented by Du and Attalah [13] and is presented in Figure 3.

We now argue correctness and security of our protocol for inner product in Figure 3.

Correctness. To establish correctness, simply note that

$$\begin{aligned} r_{\mathsf{A}} + r_{\mathsf{B}} &= \langle \vec{a}, \vec{b} - \vec{y} \rangle - r + \langle \vec{a} + \vec{x}, \vec{y} \rangle - (\langle \vec{x}, \vec{y} \rangle - r) \\ &= \langle \vec{a}, \vec{b} \rangle - \langle \vec{a}, \vec{y} \rangle - r + \langle \vec{a}, \vec{y} \rangle + \langle \vec{x}, \vec{y} \rangle - \langle \vec{x}, \vec{y} \rangle + r \\ &= \langle \vec{a}, \vec{b} \rangle \end{aligned}$$

Security. We show semi-honest security by providing simulators $\mathsf{Sim}_{\mathsf{A}}$ and $\mathsf{Sim}_{\mathsf{B}}$ that produce A's and B's view of the protocol given the corresponding input and output. To prove security, our simulators satisfy that the generated views are undistinguishable from the view of the corresponding party in the actual protocol. For relevant precise definition we refer the reader to [16].
$\mathsf{Sim}_{\mathsf{A}}$ generates a view for A given $\vec{a}$ and A's output in the ideal functionality, denoted r^*_{A}. The message from the TI is simulated as $(\vec{x'}, \langle \vec{a}, \vec{y'} \rangle - r^*_{\mathsf{A}})$, while the message from B is simulated as $\vec{y'}$, where $\vec{x'}, \vec{y'} \in U(\mathbb{Z}_q^n)$.
Similarly, $\mathsf{Sim}_{\mathsf{B}}$ generates a view for B given $\vec{b}$ and r^*_{B} as follows: the message from the TI is simulated as $(\vec{y'}, \langle \vec{x'}, \vec{y'} \rangle - r^*_{\mathsf{B}})$, while the message from A is simulated as $\vec{x'}$, where $\vec{x'}, \vec{y'} \in U(\mathbb{Z}_q^n)$.

To conclude this section, we analyze our protocol also from an accuracy perspective, taking into account that, although our protocols operate on a finite ring, they correspond to the encoding real numbers.
Accuracy. We can also provide a bound on the accuracy of the result of our protocol when the vectors in $\mathbb{Z}_q^n$ are in fact encodings of real vectors. Suppose $\vec{a} = \varphi_q(\phi_\delta(u))$ and $\vec{b} = \varphi_q(\phi_\delta(v))$ for some parameters $\delta 0$ and q. Here u, v are n-dimensional real vectors with entries bounded by R: $|u_i|, |v_i| \leq R$ for $i \in [n]$. If the size of $\mathbb{Z}_q$ satisfies $q \geq 2nR^2/\delta^2$ then we have

$$|\langle u, v \rangle - \tilde{\phi}_{\delta^2}(\tilde{\varphi}_q(\langle \vec{a}, \vec{b} \rangle))| \leq 2nR\delta + n\delta^2 \ .$$

To get this bound, we first observe that any number occurring in the computation of the inner product $\langle \phi_\delta(u), \phi_\delta(v) \rangle$ of the integer encodings of u and v is bounded by nR^2/δ^2. Therefore, the condition on q ensures there are no overflows in the computation in $\mathbb{Z}_q$ and therefore $\langle \phi_\delta(u), \phi_\delta(v) \rangle = \tilde{\varphi}_q(\langle \vec{a}, \vec{b} \rangle)$. Now we use the formulas for the error of sum and product of integer encodings from Section 3.4 and the linearity of $\tilde{\phi}_{\delta^2}$ to show that

$$\begin{aligned} |\langle u, v \rangle - &\tilde{\phi}_{\delta^2}(\langle \phi_\delta(u), \phi_\delta(v) \rangle)| \\ &= |\sum_{i=1}^n u_i v_i - \tilde{\phi}_{\delta^2}(\sum_{i=1}^n \phi_\delta(u_i)\phi_\delta(v_i))| \\ &\leq \sum_{i=1}^n |u_i v_i - \tilde{\phi}_{\delta^2}(\phi_\delta(u_i)\phi_\delta(v_i))| \\ &\leq n(2R\delta + \delta^2) \ . \end{aligned}$$

We note that R can be chosen arbitrarily because it only involves a normalization that the parties can do locally without any information or interaction with the other parties (as long as the value R is known to both parties). Thus, in practice it is convenient to normalize the real vectors u and v such that their entries are bounded by $1/\sqrt{n}$. In this case, the bounds above can be used to show that if a final accuracy of $\varepsilon > 0$ is desired, this can be achieved by taking $\delta = \varepsilon/2\sqrt{n}$ and $q = 8n/\varepsilon^2$. Using these expression we see that encodings with $O(\log(n/\varepsilon))$ bits are enough to achieve accuracy ε in n-dimensional inner products.

5 Secure linear system solving

The second phase of our algorithm, as presented in the previous section, consists in solving a linear system shared between two parties P_1 and P_2 holding secret additive shares (A_i, b_i) of the matrix A and vector b in a linear system $A\theta = b$; that is, $A = A_1 + A_2$ and $b = b_1 + b_2$.

We implement three well-known methods typically used in numerical linear algebra for solving systems with positive definite coefficient matrices: Cholesky, LDLT, and conjugate gradient descent. Details about each method and a discussion of their relative merits from an MPC point of view are given next. After the methods are presented, we delve into the intricacies of our implementations based on Yao's garbled circuits protocol [40] using the Obliv-C framework [43]. Finally, we present illustrative experiments showing that conjugate gradient descent is the best choice for dealing with high-dimensional linear systems, and that a careful implementation using fixed-point arithmetic can achieve comparable results to floating-point implementations while yielding much more scalable algorithms. It is important to note that most of this section is application-agnostic, implying that our findings and implementations can be used to solve generic positive definite linear systems in a secure MPC setting not necessarily arising from ridge regression.

5.1 Solution Methods

Several variants of our protocol for solving phase 2 of the private multi-party ridge regression problem can be obtained by considering different methods for solving systems of linear equations with positive definite coefficient matrices. Now we present the details of three of these algorithms.

The first algorithm is based on the Cholesky decomposition. Any positive definite matrix admits a unique *Cholesky decomposition* of the form $A = LL^\top$, where L is a $d \times d$ lower triangular matrix with $L_{ij} = 0$ for all $i < j$. If this decomposition is given, then the system $LL^\top\theta = b$ can be solved by first finding the solution of $L\theta' = b$ and then the solution of $L^\top\theta = \theta'$. By exploiting the lower triangular structure of L, the first system can be efficiently solved by forward substitution taking $\theta'_1 = b_1/L_{11}$ and $\theta'_i = (b_i - \sum_{j<i} L_{ij}\theta'_j)/L_{ii}$ for $i = 2, \ldots, d$. Similarly, the second system is solved by backward substitution taking $\theta_d = \theta'_d/L_{dd}$ and $\theta_i = (\theta'_i - \sum_{j>i} L_{ji}\theta_j)/L_{ii}$ for $i = d-1, \ldots, 1$. Note that these substitutions are a particular instance of Gaussian elimination where the initial coefficient matrices are already in almost echelon form (with the exception that the diagonal coefficients may be different from 1). Since the Cholesky decomposition of a positive definite matrix A can be efficiently computed in $O(d^3)$ floating point operations, this yields an efficient algorithm for solving the system $A\theta = b$. Cholesky's algorithm enjoys two important properties that make it a very natural choice for solving private linear systems: it does not involve any pivoting strategy, thus yielding data-agnostic algorithms; and, it is numerically robust, making it suitable for implementations using finite-precision numeric representations.

A drawback of Cholesky's decomposition $A = LL^\top$ is that the algorithm requires computing $O(d)$ square roots. Since our goal is to implement linear system solvers on top of a secure multi-party computation architecture, this can represent a problem since MPC technologies generally only provide basic arithmetic operations like addition and multiplication. Though it is possible to implement iterative algorithms for computing square roots using MPC approaches, an alternative approach is to use a different matrix decomposition whose computation does not involve square roots. A common option along these lines is the *LDLT decomposition*, a slight variation on the Cholesky decomposition for positive definite matrices where one writes $A = LDL^\top$ with L lower triangular and D diagonal with non-negative entries. It is easy to see that the linear system $LDL^\top \theta = b$ also admits an efficient substitution algorithm very similar to the one given above for the Cholesky decomposition. Solving $A\theta = b$ via the LDLT decomposition of A enjoys the same properties as the Cholesky-based solution, with the difference that instead of the square roots one has to perform one more substitution phase when solving the factorized system.

Factorizing the coefficient matrix A in order to make the solution easier to compute is not the only way to solve a positive definite linear system. An entirely different approach is that of iterative algorithms which construct a monotonically improving sequence of approximate solutions converging to the desired solution. These algorithms typically involve a few vector inner products and one or more matrix-vector products per iteration and their asymptotic running time is again $O(d^3)$ if run until convergence. However, they can also be used to find approximate solutions for systems where the matrix A is too big to be factorized entirely by incorporating an early stopping criterion. When the coefficient matrix is positive definite, the default iterative solution is the *conjugate gradient descent* (CGD) algorithm. One approach to derive the CGD algorithm is to realize the solution to $A\theta = b$ is also the solution to the optimization problem $\mathsf{argmin}_\theta \|A\theta - b\|^2$ and solve it using the method of conjugate gradients where the descent direction chosen at each iteration is orthogonal to all previous descent directions. For the purpose of this paper, the importance of GCD is twofold: first, since intensive computations inside an MPC framework can be expensive, iterative algorithms provide a natural way to spend a fixed computational budget on finding an approximation to the desired solution; and second, that for linear regression problems with noisy data it usually suffices to find an approximate solution whose accuracy is on the same order as the noise present in the data. See [31] for a thorough discussion on how to implement CGD with finite-precision arithmetic.

5.2 Implementation Details

The three algorithms described above are quite standard and detailed implementation guidelines can be found in the literature; e.g. [36]. On the one hand, for LDLT and Cholesky, our implementations are based on these standard pseudo-codes, with the difference that we used fixed-point arithmetic instead of floating-point. On the other hand, to obtain a CGD algorithm with a good trade-off between accuracy and scalabity when executed as a garbled circuit required a taylored approach.

To set up the secure multi-party computation infrastructure required for Yao's garbled circuit protocol we use the Obliv-C system. In this system code get compiled from a high-level language based on C and onto a garbled circuit that will be securely executed using inputs from two or more parties.

In order to deal with real numbers, we implement fixed-point arithmetic on top of Obliv-C basic types, including the square root computation of [35]. Our implementation represents

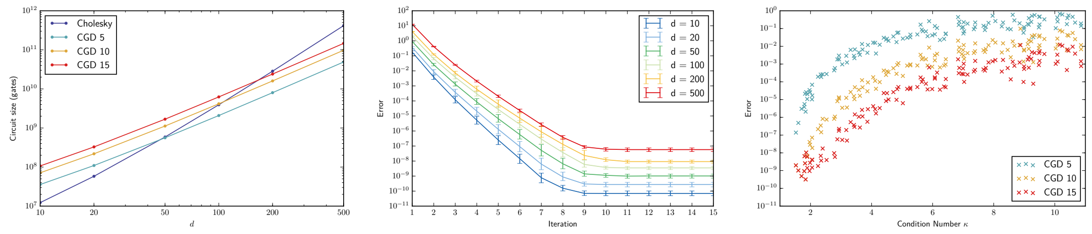


Figure 4: Comparison between different methods for solving linear systems. (left) Gate count for the garbled circuits implementing Cholesky and CGD (with 5, 10, and 15 iterations) as a function of input dimension. (middle) Convergence of CGD with the number of iterations. For various values of d and $n = 100000$, the error is plotted against the number of iterations used, averaged over 30 samples with average condition number 1.77. (right) Difference in ridge regression objective value between floating-point and fixed-point implementations of CGD (100 different problems with $d = 20$).

real numbers using 32 bits split between the integral part (4) and the fractional part (28). To make the most of the fixed-point representation, in our implementation each party locally normalizes its input data to ensure that the entries in the matrix A all satisfy $|A_{ij}| \leq 1/d$. This is particularly useful for CGD, since it helps prevent overflows of the integral part when computing intermediate values of the form $\theta^\top A\theta$ required by the line search step of each iteration.

5.3 Experimental Results

Now we present experimental results in order to illustrate two important points about the implementation of linear system solvers in secure MPC settings. The first point is that for high-dimensional systems running a few iterations CGD yields a more scalable algorithm than using any of the exact methods at a comparatively very small price in terms of accuracy. The second point is that for the types of systems occurring in ridge regression problems implementations of CGD with fixed-point arithmetic can yield results comparable to those obtained with floating-point implementations.

To investigate the first point we start by showing in Figure 4 (left) the number of gates of the circuit resulting from the Obliv-C compilation of the algorithms implementing Cholesky, LDLT, and CGD with different number of iterations. We observe that, for dimensions larger than 100, CGD with 10 iterations corresponds to smaller circuits than the one for solving the full systems via Cholesky or LDLT. For CGD with 15 iterations, such break even point is at $d = 200$.

On the other hand, in Figure 4 (middle) we show the convergence of CGD with the number of iterations in our implementation, when securely solving linear systems arising from ridge regression with several values of d and $n = 10^5$. The error is measured as $|LS(s) - LS(s_{cgd})|/LS(s)$, where LS denotes least squares funtion to be minimized, s denotes the exact solution obtained with floating point arithmetic, and s_{cgd} denotes the solution obtained using our implementation of CGD with fixed point arithmetic.

Note that the plot in Figure 4 (middle) already indicates that a fixed-point implementation of CGD using the normalizations discussed in the previous section can yield results comparable to that of a floating-point implementation. Since it is known that the speed of convergence of CGD can be affected by the condition number of the coefficient matrix A, and the condition number of such a matrix in a ridge regression problem depends on the number of samples n, we must also investigate if there is a substantial difference in the speeds of convergence between the fixed-point and the floating-point implementation of CGD depending on the condition number. For that purpose we generate 100 different ridge regression using random data with different sizes, and we consider the same error measure than in the previous plot: difference in objective value between the floating-point solution and the solution obtained by our fixed-point implementation. For each setting of d and n, we sample n data points x^i from a standard d-dimensional Gaussian distribution, a vector of parameters θ^* with independent coordinates sampled uniformly in the interval $[0, 1]$, and the labels are taken to be $y^i = \langle\theta^*, x^i\rangle + \epsilon^i$ where $\epsilon^i \sim \mathcal{N}(0, 0.1)$. The regularization parameter is set to the optimal choice according to the discussion in Section 3.2. The results, shown in Figure 4 (right), confirm the effect of the condition number of the input matrix, and indicate that the benefit of running CGD for more iterations decreases as the condition number increases. On the other hand, the precision of Cholesky remains stable accross all tested condition numbers.

6 Many Input Parties

Next we consider the case when the initial database in vertically partitioned among several parties. Currently the most efficient multi-party computation protocol [28] is based on the BMR construction [2] combined with a preprocessing stage, which relies on a multi-party variant of garbled circuits. Our construction will be a different variant of multi-party garble circuits since as we discussed in Section 2 we will leverage the assumptions for the CSP to avoid preprocessing that involves all parties.

We also introduce the Evaluator party mentioned in Section 2 to facilitate the protocol providing an evaluator of the garbles circuits that contributes no input and takes care of the most costly computation. The naive approach, similarly to the two party case, is have each party share its database part between the CSP and the Evaluator and then execute a two party computation using a garbled circuit to compute $A = X^\top X$ and then solve the system of linear equations. However, this approach will require multiplication of large dimension matrices to compute $A = X^\top X$ from the parties shares, which is inefficient using GCs. Instead, analogously to what we did in the two party case, we will construct a tailored protocol for Phase I to enable the CSP and the Evaluator to obtain additive shares of the matrix $A = X^\top X$.

Similarly to the two-party case we observe that the vertical partitioning of the database means that each party holds a subset of the columns of X and a corresponding subset of the rows of $X^\top$. Since each value in A is a dot product between a row vector of $X^\top$ and a column vector of X, this means that each value in A depends on the inputs of at most two input parties. Thus, using the inner product protocol from Section 4 between corresponding pairs of parties we can obtain A in a form where each entry is either known to one of the inputs parties or is shared between some pair of input parties.

At this point, in the second phase of our protocol, all parties contribute their shares of the entries of A in a garbled form to Evaluator and it will reconstruct A inside the garbled circuit, which subsequently implement the solution for the system of linear equations from Phase II presented in Section 5. The garbled circuit is generated by the CSP and provided to the Evaluator. We consider two options for the input garbling part of the protocol. In the first setting we assume that neither input party is colluding with the Evaluator – in this case for each input wire the CSP can directly send both input garbled labels to the corresponding party, which based on its input bit will forward the correct input garbled label to the Evaluator. If we do not want to make the assumption for non-collusion between input parties and Evaluator, we run an oblivious transfer protocol between the CSP and the corresponding party for each input wire. At the end all parties hold the garbling of their inputs, which they prove to the Evaluator. Figure 5 describes the steps in the protocol.

Although the architecture and overall distribution of responsibilities resembles the protocols presented in [35] for the horizontally partition case, there are a some important differences between their approach and ours:

- Our parties have to communicate with each other during the online phase of the protocol (this is due to the fact that in contrast with the horizontally partitioned setting, in the vertically partitioned setting each entry in the coefficient matrix A involves products between numbers held by two different parties).
- Our two proposed approaches do not use HE (this is achieved by using additive shares; this optimization can also be applied to the protocol of Nikolaenko et al. by having the CSP distributed correlated randomness between the parties holding the data).
- One of our solutions does not need any oblivious transfer, at the cost of assuming that the Evaluator does not collude with any of the parties (note this assumption is reasonable in scenarios where the Evaluator represents a provider of outsourced computation).

7 Experiments

The experiments were conducted using Amazon EC2 C4 instances, each having 4 CPU cores, 7.5 GiB of RAM and 1 Gbps bandwidth. Datasets were generated as described in Section 5.3, and a precision of 64 bits was used (with 54 bits for the fractional part).

We built the multi-party protocols atop Obliv-C [43], a derivative of the C language supporting two party computation with a number of recent optimizations, including Free XOR [26], Garbled Row Reduction [33], Fixed Key Block Ciphers [3], and Half Gates [44]. To support arbitrary precision arithmetic, we used a big integer library [1] for multi-party computation. This library represents values in two's compliment binary, split between an arbitrary number of machine-native garbled "digits". It implements all the operations we require using common, efficient algorithms for extended precision arithmetic; in particular, it implements the Karatsuba-Comba [22] method for multiplication, and Knuth's algorithm D [25] for division. We extended this library to support fixed precision arithmetic via the

Parties: k input parties $\mathsf{P}_1, \ldots, \mathsf{P}_k$, and CSP, $\mathsf{Evaluator}$.
Inputs: $\mathsf{P}_i : X_i \in \mathbb{Z}_q^{d_i \times n}$ for all $i \in [k]$, where $X = X_1 | \cdots | X_k$.
Outputs: $\mathsf{A} \leftarrow \perp$; $\mathsf{P}_i \leftarrow \perp$ and $\mathsf{Evaluator} \leftarrow \beta$.
Circuit: Let C be a circuit that first reconstructs A from input shares ($A[i, j]$ is obtained from two input shares if $X[i]$ and $X[j]$ are hold by different parties, and is provided as input otherwise), and then solves $Ax = b$.

Protocol:

1. For all $i, j \in [d]$, let $\mathsf{P}_{\mathsf{id}_i}$ and $\mathsf{P}_{\mathsf{id}_j}$ be the parties that hold rows i and j from X. If $\mathsf{id}_i = \mathsf{id}_j$, then party P_{id_i} computes $A[i, j] = \langle X[i], X[j] \rangle$. Otherwise, parties $\mathsf{P}_{\mathsf{id}_i}$ and $\mathsf{P}_{\mathsf{id}_j}$ run the two-party inner product protocol from Figure 3 to compute $A[i, j]$, where the CSP acts as the trusted initializer.

2. CSP generates garbled circuit and input garbled labels $(\tilde{C}, \{w_{i,0}, w_{i,1}\}_{i \in N}) \leftarrow \mathsf{GC.GarbCirc}(C)$ and sends $\tilde{C}$ to $\mathsf{Evaluator}$.

3. For all $i \in [N]$, let $\mathsf{P}_{\mathsf{id}_i}$ be the party who contributes the i-th input bit and b_i be the value of its input bit:

   Non-Colluding Parties The CSP sends $\{w_{i,0}, w_{i,1}\}$ to the party $\mathsf{P}_{\mathsf{id}_i}$ and $\mathsf{P}_{\mathsf{id}_i}$ sends w_{i,b_i} to $\mathsf{Evaluator}$.

   Colluding Parties The CSP and the party $\mathsf{P}_{\mathsf{id}_i}$ run an oblivious transfer protocol with inputs $\{w_{i,0}, w_{i,1}\}$ and b_i respectively. P_{id_i} obtains w_{i,b_i} and sends it to $\mathsf{Evaluator}$.

4. The $\mathsf{Evaluator}$ evaluates $\mathsf{GC.Eval}(\tilde{C}, \{w_{i,b_i}\}_{i \in [N]})$ to obtain the output.

Figure 5: Multi-party computation of linear regression.

same techniques used for native types. This arrangement incurs some overhead relative to using native types, even when the number of bits is equivalent.

We implemented both versions of the protocol of Figure 5. We found that there is no significant improvement in removing the oblivious transfers under the assumptions that the Evaluator and the parties do not collude. We therefore assume potentially colluding parties in the sequel. Moreover, as the time spent in computing oblivious transfers in between phases is negligible, the total running time is, with very high accuracy, the sum of the times spent in each of the two phases. For this reason, we report the experiments for each of our phases independently in this section. For example, our implementation of ridge regression with $n = 10^6$ runs in less than 3 hours if $d = 200$, less than 1 hour if $d = 100$, and less than 2 minutes if $d = 20$.

7.1 Phase 1

The first phase of our protocol was executed with different numbers of parties, with the inputs shared evenly among them. The computation time was recorded for CSP and each party, so was the total running time. Table 1 shows our results. For $n = 10^6$ and two data providers, the total running time is less than an hour when $d = 200$, less than 5 minutes if $d = 100$ and less than 5 seconds if $d = 10$. To compare our approach (based on a semi-trusted CSP) to a purely garbled circuit based approach, note that, using only Obliv-C, a *single inner product* of two secret vectors of length n takes over 50 seconds, 5 minutes, and 90 minutes, for $n = 10^4$, $n = 10^5$, and $n = 10^6$, respectively.

Note that while the average computation time for the data providers unsurprisingly decreases with increasing number of parties, both the computation time for the CSP and the total running time increase. The former can be attributed to the fact that with increasing number of parties, a smaller portion of the covariance matrix can be computed locally, which is why the CSP has to generate more random numbers. The latter is because our implementation is not fully parallel, leaving some room for improvement by future applications.

Similar behaviour can be seen in Table 2. Here, the total amount of data transferred from the CSP and the average data provider is shown, as well as the input size for comparison. Again, the CSP has more work with increasing number of parties, while each data provider's work decreases.

7.2 Phase 2

While Figure 4 in Section 5.3 shows gate counts for several variants of CGD and Cholesky, Figure 6 shows the corresponding timings, which are, as expected, correlated with circuit size. Again, it becomes clear that CGD with a fixed number of iterations outperforms Cholesky for larger values of d. For $d = 200$, both Cholesky and CGD with 15 iterations take less than 2 hours. For $d = 500$, Cholesky takes more than 24 hours while CGD 15 takes less than 9 hours.

8 Related Work

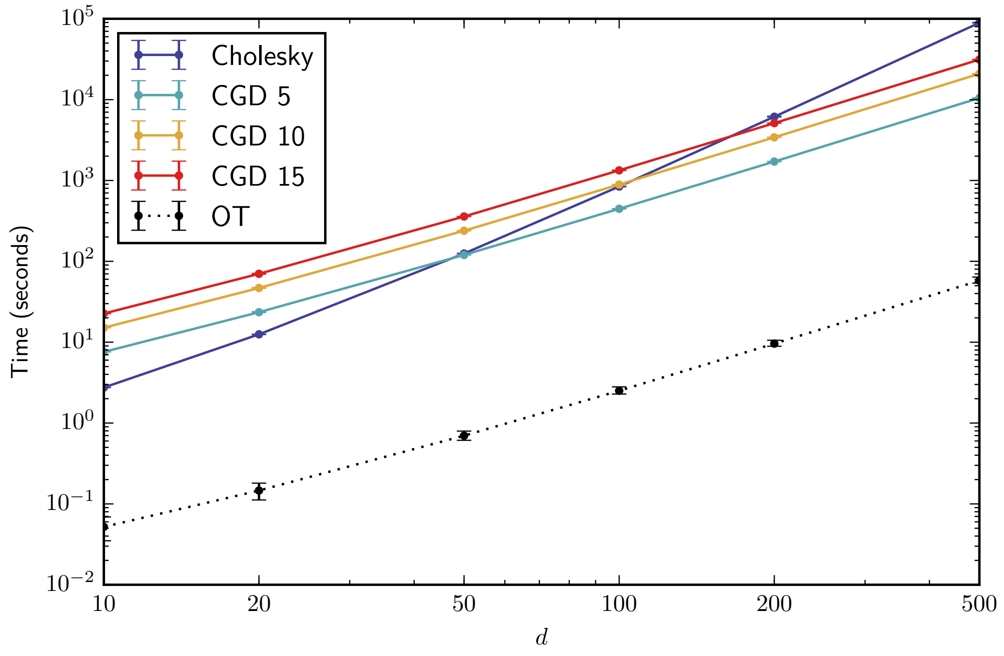
Cryptographic solutions for the secure computation of any functionality have been known for than thirty years [5,17,40]. Yet, implementations of MPC protocols and evaluations of their

		Number of parties (data providers)								
n	*d*	2			3			5		
		CSP	DP	Total	CSP	DP	Total	CSP	DP	Total
1000	10	0.005	0.007	0.020	0.006	0.007	0.021	0.007	0.006	0.019
	20	0.013	0.021	0.070	0.017	0.020	0.077	0.021	0.016	0.074
	50	0.073	0.092	0.377	0.097	0.087	0.449	0.120	0.067	0.470
	100	0.286	0.349	1.495	0.377	0.312	1.783	0.474	0.232	1.913
	200	1.121	1.335	5.947	1.488	1.230	7.136	1.869	0.865	7.635
	500	7.049	8.878	28.212	9.268	7.748	36.822	11.561	5.651	45.959
2000	10	0.008	0.014	0.030	0.010	0.013	0.030	0.012	0.012	0.029
	20	0.025	0.040	0.098	0.034	0.037	0.110	0.042	0.030	0.108
	50	0.142	0.180	0.528	0.196	0.163	0.631	0.235	0.127	0.678
	100	0.550	0.664	2.051	0.764	0.588	2.497	0.950	0.437	2.725
	200	2.205	2.498	7.994	3.017	2.203	9.789	3.624	1.618	10.833
	500	13.785	15.968	41.563	18.653	13.466	52.856	22.387	9.815	64.572
5000	10	0.018	0.034	0.057	0.024	0.032	0.061	0.029	0.029	0.061
	20	0.060	0.093	0.174	0.081	0.086	0.199	0.102	0.072	0.209
	50	0.348	0.420	0.892	0.482	0.381	1.124	0.577	0.299	1.226
	100	1.356	1.485	3.334	1.840	1.343	4.292	2.283	1.023	4.854
	200	5.386	5.584	13.003	7.324	4.998	16.558	8.644	3.694	19.260
	500	33.565	35.374	73.895	45.352	31.320	97.195	53.921	22.389	116.003
10000	10	0.034	0.068	0.101	0.050	0.064	0.111	0.062	0.057	0.111
	20	0.122	0.185	0.299	0.157	0.170	0.351	0.202	0.143	0.376
	50	0.688	0.848	1.499	0.940	0.766	1.875	1.158	0.601	2.088
	100	2.699	3.015	5.664	3.622	2.684	7.140	4.439	2.031	8.163
	200	10.783	11.339	21.708	14.258	10.115	27.916	17.383	7.451	32.246
	500	67.171	70.143	124.308	88.791	61.758	163.028	108.381	44.654	195.330
20000	10	0.066	0.137	0.188	0.088	0.128	0.207	0.110	0.114	0.212
	20	0.233	0.371	0.553	0.310	0.341	0.649	0.394	0.288	0.701
	50	1.365	1.706	2.766	1.868	1.533	3.429	2.375	1.207	3.889
	100	5.390	6.027	10.216	7.181	5.402	13.026	8.880	4.076	15.055
	200	21.174	22.752	39.594	29.031	20.291	51.259	34.254	14.939	59.812
	500	133.874	139.446	234.949	180.684	123.391	309.776	217.714	88.795	367.198
50000	10	0.164	0.342	0.453	0.214	0.319	0.502	0.273	0.286	0.522
	20	0.576	0.932	1.327	0.774	0.857	1.566	0.971	0.728	1.757
	50	3.714	4.319	6.655	4.528	3.928	8.445	5.547	3.061	9.577
	100	13.384	15.229	24.525	17.926	13.709	31.510	21.709	10.359	36.890
	200	53.209	57.567	95.244	70.266	51.090	122.332	85.390	37.814	144.797
	500	327.277	349.400	568.443	436.707	316.512	767.693	526.281	224.212	892.206
100000	10	0.332	0.686	0.907	0.436	0.645	1.009	0.552	0.576	1.043
	20	1.164	1.905	2.718	1.551	1.751	3.214	1.897	1.481	3.598
	50	6.993	8.782	13.675	9.202	7.940	16.957	11.189	6.244	19.706
	100	27.197	30.899	49.773	36.473	27.963	64.077	43.098	21.033	75.125
	200	106.098	116.875	191.752	141.750	104.366	249.667	169.827	77.018	293.731
	500	651.692	713.298	1153.575	880.312	629.066	1516.056	1049.437	457.018	1802.101
200000	10	0.646	1.400	1.852	0.878	1.315	2.181	1.100	1.166	2.313
	20	2.339	4.008	5.801	3.126	3.657	6.862	3.824	3.057	7.689
	50	13.791	18.993	29.870	18.415	16.853	36.626	22.133	13.032	41.877
	100	54.552	65.691	106.936	71.151	58.871	135.884	85.957	44.149	159.043
	200	216.787	252.963	420.397	284.148	223.026	539.589	338.485	163.462	632.491
	500	1305.504	1533.265	2507.263	1750.723	1343.198	3261.326	2095.598	965.710	3839.518
500000	10	1.660	3.521	4.883	2.203	3.283	5.766	2.704	2.901	6.642
	20	6.030	9.969	14.817	7.903	9.035	18.042	9.778	7.539	21.066
	50	35.708	47.658	76.599	47.104	42.060	95.111	57.257	32.419	111.185
	100	136.578	170.530	286.751	184.484	149.449	360.543	221.563	110.986	417.770
	200	540.070	655.201	1132.075	734.639	573.464	1447.904	886.833	412.219	1662.864
	500	3382.436	4001.079	6832.068	4477.094	3451.451	8755.041	5440.699	2456.102	10 252.712
1000000	10	3.444	6.998	9.980	4.468	6.542	11.786	5.521	5.790	13.451
	20	12.365	19.821	30.779	16.109	17.992	37.435	19.682	15.091	43.889
	50	72.054	94.408	159.847	96.601	83.677	201.290	118.332	64.426	233.262
	100	285.408	340.410	601.394	376.270	297.646	761.816	451.015	220.557	888.193
	200	1129.410	1303.508	2356.076	1507.230	1127.955	2995.634	1828.218	827.767	3561.551

Table 1: Phase 1 computation time (seconds) for 2, 3, and 5 data providers and different values of n (number of records) and d (number of features). For each number of data providers, computation time of the CSP (left) and the data providers (middle, averaged) is reported, as well as total running time (right).

n	d	Input Size	Number of parties (data providers)					
			2		3		5	
			CSP	DP	CSP	DP	CSP	DP
1000	10	78.1 KiB	557.8 KiB	278.6 KiB	725.0 KiB	241.5 KiB	892.3 KiB	178.3 KiB
	20	156.2 KiB	2.0 MiB	1021.4 KiB	2.6 MiB	891.6 KiB	3.2 MiB	653.8 KiB
	50	390.6 KiB	11.8 MiB	5.9 MiB	15.7 MiB	5.2 MiB	18.9 MiB	3.8 MiB
	100	781.2 KiB	46.3 MiB	23.1 MiB	61.7 MiB	20.5 MiB	74.1 MiB	14.8 MiB
	200	1.5 MiB	183.4 MiB	91.6 MiB	244.4 MiB	81.4 MiB	293.4 MiB	58.6 MiB
2000	10	156.2 KiB	1.1 MiB	556.6 KiB	1.4 MiB	482.6 KiB	1.7 MiB	356.4 KiB
	20	312.5 KiB	4.0 MiB	2.0 MiB	5.2 MiB	1.7 MiB	6.4 MiB	1.3 MiB
	50	781.2 KiB	23.6 MiB	11.8 MiB	31.3 MiB	10.4 MiB	37.7 MiB	7.5 MiB
	100	1.5 MiB	92.5 MiB	46.2 MiB	123.3 MiB	41.1 MiB	148.0 MiB	29.6 MiB
	200	3.1 MiB	366.3 MiB	183.1 MiB	488.3 MiB	162.7 MiB	586.1 MiB	117.2 MiB
5000	10	390.6 KiB	2.7 MiB	1.4 MiB	3.5 MiB	1.2 MiB	4.3 MiB	890.5 KiB
	20	781.2 KiB	10.0 MiB	5.0 MiB	13.0 MiB	4.3 MiB	15.9 MiB	3.2 MiB
	50	1.9 MiB	58.9 MiB	29.4 MiB	78.3 MiB	26.1 MiB	94.2 MiB	18.8 MiB
	100	3.8 MiB	231.0 MiB	115.5 MiB	308.0 MiB	102.6 MiB	369.7 MiB	73.9 MiB
	200	7.6 MiB	915.1 MiB	457.5 MiB	1.2 GiB	406.6 MiB	1.4 GiB	292.8 MiB
10000	10	781.2 KiB	5.4 MiB	2.7 MiB	7.1 MiB	2.4 MiB	8.7 MiB	1.7 MiB
	20	1.5 MiB	19.9 MiB	10.0 MiB	26.1 MiB	8.7 MiB	31.9 MiB	6.4 MiB
	50	3.8 MiB	117.8 MiB	58.9 MiB	156.5 MiB	52.2 MiB	188.4 MiB	37.7 MiB
	100	7.6 MiB	462.0 MiB	231.0 MiB	615.8 MiB	205.2 MiB	739.2 MiB	147.8 MiB
	200	15.3 MiB	1.8 GiB	914.8 MiB	2.4 GiB	813.0 MiB	2.9 GiB	585.5 MiB
20000	10	1.5 MiB	10.9 MiB	5.4 MiB	14.1 MiB	4.7 MiB	17.4 MiB	3.5 MiB
	20	3.1 MiB	39.9 MiB	19.9 MiB	52.2 MiB	17.4 MiB	63.8 MiB	12.8 MiB
	50	7.6 MiB	235.5 MiB	117.7 MiB	313.0 MiB	104.3 MiB	376.8 MiB	75.4 MiB
	100	15.3 MiB	923.8 MiB	461.9 MiB	1.2 GiB	410.5 MiB	1.4 GiB	295.6 MiB
	200	30.5 MiB	3.6 GiB	1.8 GiB	4.8 GiB	1.6 GiB	5.7 GiB	1.1 GiB
50000	10	3.8 MiB	27.2 MiB	13.6 MiB	35.3 MiB	11.8 MiB	43.5 MiB	8.7 MiB
	20	7.6 MiB	99.6 MiB	49.8 MiB	130.4 MiB	43.5 MiB	159.4 MiB	31.9 MiB
	50	19.1 MiB	588.7 MiB	294.3 MiB	782.5 MiB	260.8 MiB	941.9 MiB	188.4 MiB
	100	38.1 MiB	2.3 GiB	1.1 GiB	3.0 GiB	1.0 GiB	3.6 GiB	739.0 MiB
	200	76.3 MiB	8.9 GiB	4.5 GiB	11.9 GiB	4.0 GiB	14.3 GiB	2.9 GiB
100000	10	7.6 MiB	54.3 MiB	27.2 MiB	70.6 MiB	23.5 MiB	86.9 MiB	17.4 MiB
	20	15.3 MiB	199.2 MiB	99.6 MiB	260.8 MiB	86.9 MiB	318.8 MiB	63.8 MiB
	50	38.1 MiB	1.1 GiB	588.7 MiB	1.5 GiB	521.6 MiB	1.8 GiB	376.7 MiB
	100	76.3 MiB	4.5 GiB	2.3 GiB	6.0 GiB	2.0 GiB	7.2 GiB	1.4 GiB
	200	152.6 MiB	17.9 GiB	8.9 GiB	23.8 GiB	7.9 GiB	28.6 GiB	5.7 GiB
200000	10	15.3 MiB	108.7 MiB	54.3 MiB	141.3 MiB	47.1 MiB	173.9 MiB	34.8 MiB
	20	30.5 MiB	398.5 MiB	199.2 MiB	521.6 MiB	173.9 MiB	637.6 MiB	127.5 MiB
	50	76.3 MiB	2.3 GiB	1.1 GiB	3.1 GiB	1.0 GiB	3.7 GiB	753.5 MiB
	100	152.6 MiB	9.0 GiB	4.5 GiB	12.0 GiB	4.0 GiB	14.4 GiB	2.9 GiB
	200	305.2 MiB	35.7 GiB	17.9 GiB	47.6 GiB	15.9 GiB	57.2 GiB	11.4 GiB
500000	10	38.1 MiB	271.7 MiB	135.8 MiB	353.2 MiB	117.7 MiB	434.7 MiB	86.9 MiB
	20	76.3 MiB	996.2 MiB	498.1 MiB	1.3 GiB	434.7 MiB	1.6 GiB	318.8 MiB
	50	190.7 MiB	5.7 GiB	2.9 GiB	7.6 GiB	2.5 GiB	9.2 GiB	1.8 GiB
	100	381.5 MiB	22.6 GiB	11.3 GiB	30.1 GiB	10.0 GiB	36.1 GiB	7.2 GiB
	200	762.9 MiB	89.3 GiB	44.7 GiB	119.1 GiB	39.7 GiB	142.9 GiB	28.6 GiB
1000000	10	76.3 MiB	543.4 MiB	271.7 MiB	706.4 MiB	235.5 MiB	869.4 MiB	173.9 MiB
	20	152.6 MiB	1.9 GiB	996.2 MiB	2.5 GiB	869.4 MiB	3.1 GiB	637.6 MiB
	50	381.5 MiB	11.5 GiB	5.7 GiB	15.3 GiB	5.1 GiB	18.4 GiB	3.7 GiB
	100	762.9 MiB	45.1 GiB	22.6 GiB	60.1 GiB	20.0 GiB	72.2 GiB	14.4 GiB
	200	1.5 GiB	178.6 GiB	89.3 GiB	238.1 GiB	79.4 GiB	285.8 GiB	57.2 GiB

Table 2: Amount of data transferred in phase 1 by the CSP (left) and the data providers (right, averaged). Similarly to Table 1, the amount of work for the CSP increases with the number of data providers, while each data provider's work decreases.



d	OT	Cholesky	CGD 5	CGD 10	CGD 15
10	0.052	2.751	7.585	15.099	22.608
20	0.146	12.507	23.492	46.798	70.089
50	0.698	124.918	120.002	239.209	358.467
100	2.509	841.372	446.744	890.811	1334.814
200	9.608	6144.301	1713.717	3417.499	5121.407
500	57.791	89 193.308	10 474.579	20 888.350	31 300.942

Figure 6: Comparison between different methods for solving linear systems: Running time (seconds) of our Cholesky and CGD (with 5, 10, and 15 iterations) implementations as a function of input dimension. While Cholesky is faster than CGD for lower values of d, it is quickly overtaken by the latter as d increases. This shows that for high-dimensional data, iterative methods are preferable in terms of computation time. The time spent running oblivious transfers is also shown, and corresponds to a small fraction of the running time.

practical efficiency have become well established part of the research agenda in this area only in the last several years [4,6,18,20,21,24,29,30,34,35,39,42,45]. Popular applications used for the empirical evaluation of implemented systems include graph algorithms [6, 18, 24, 29, 34, 45], data structure algorithms [24,29,42], string matching and distance algorithms [20,21,39] and AES [4,20,21,39]. Questions of privacy preserving data mining and private computation of machine learning algorithms including linear regression have been considered in several works [12, 14, 23, 27, 38, 41] which offer theoretical protocols without implementation and practical evaluations of their efficiency. The work of Hall et al. [19] proposes a protocol for computing linear regression on vertically partitioned database based on homomorphic encryption and runs a simulation for the protocol, which is several orders of magnitude slower than our results (for database of size $n \approx 50000$ with $d \approx 20$ features their protocol runs for two days).

The most relevant previous work to our paper is the work of Nikolaenko et al. [35] which also considers the question of secure computation for ridge regression but in a different setting. It addresses the setting of a horizontally partitioned database, where each record is contributed by a different user. They divide their computation protocol in the same two phases that we use. The horizontal partitioning of the database affects the first phase of the computation. In particular in this case each entry of the covariance matrix M can be expressed as a sum of terms, each of which can be computed by a single party. This enables the authors of [35] to use additively homomorphic encryption for the inputs of different users and compute M. In our setting of vertically partitioned data the entries of A require multiplication of inputs coming from different parties. Non-interactivity for the first phase was important in the case of [35], since this does not require users that contribute records in the database to be online and participate in the protocol. In our setting the potential input parties that hold vertical partitions of the database will be several companies/organizations which come together to build a model over their joint data. Interactivity in this scenario is not of vital importance and we choose to construct interactive protocols that achieve better computation efficiency avoiding expensive homomorphic encryption, which in our case will have to handle both addition and a single multiplication.

For the second phase of the secure ridge regression protocol, Nikolaenko et al. [35] implement Cholesky's algorithm for solving systems of linear equations. We implemented three different techniques for solving systems of linear equations: Cholesky, LDLT and CGD, and showed that iterative methods like the latter enable higher dimensionality at the cost of a resonable accuracy loss. In fact, phase II of our protocol coincides with the one in [35], and hence our contribution is also valuable in the setting considered there.

9 Discussion

The problem of securely running machine learning algorithms when the training data is distributed among several parties is an important milestone for the development of privacy-preserving data analysis tools. In this paper we focus on a linear regression task widely used in practical applications. We showed how the setting in which the training database is vertically partitioned gives rise to technical problems different from the ones encountered in the horizontally partitioned setup previously considered in the literature. We present a hybrid secure multi-party computation protocol for solving this problem that involves an inner product protocol and a linear system solving protocol. By using tools based on shared randomness and garbled circuits we obtain a highly scalable solution that can efficiently solve linear regression problems on large-scale datasets. Our experiments show that it is possible to apply these ideas to high-dimensional problems using an implementation of conjugate gradient descent with fixed-point arithmetic and early stopping. In future work we plan to extend our protocols to support secure hyper-parameter tuning in order to deal with the entire data analysis pipeline. In addition we will extend our implementation to deal with non-linear regression problems based on kernel ridge regression by implementing secure multi-party evaluation of kernel functions typically used in machine learning.

References

[1] Absentminded crypto kit. `https://bitbucket.org/jackdoerner/absentminded-crypto-kit/`.

[2] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. In *Proceedings of the Twenty-second Annual ACM Symposium on Theory of Computing*, STOC '90, 1990.

[3] M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway. Efficient garbling from a fixed-key blockcipher. *2014 IEEE Symposium on Security and Privacy*, 0:478–492, 2013.

[4] M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway. Efficient garbling from a fixed-key blockcipher. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13, pages 478–492, 2013.

[5] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, 1988.

[6] M. Blanton, A. Steele, and M. Alisagari. Data-oblivious graph algorithms for secure computation and outsourcing. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, ASIA CCS '13, 2013.

[7] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 309–325, 2012.

[8] Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 97–106, 2011.

[9] M. D. Cock, R. Dowsley, A. C. A. Nascimento, and S. 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.

[10] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, pages 643–662, 2012.

[11] D. Demmler, G. Dessouky, F. Koushanfar, A.-R. Sadeghi, T. Schneider, and S. Zeitouni. Automated synthesis of optimized circuits for secure computation. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1504–1517. ACM, 2015.

[12] W. Du and M. J. Atallah. Privacy-preserving cooperative scientific computations. In *Proceedings of the 14th IEEE Workshop on Computer Security Foundations*, CSFW '01, 2001.

[13] W. Du and M. J. Atallah. Protocols for secure remote database access with approximate matching. In *E-Commerce Security and Privacy*, pages 87–111. 2001.

[14] W. Du, Y. S. Han, and S. Chen. Privacy-preserving multivariate statistical analysis: Linear regression and classification. In *Proceedings of the 2004 SIAM International Conference on Data Mining*, 2004.

[15] C. Gentry. *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford, CA, USA, 2009. AAI3382729.

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

[17] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, 1987.

[18] S. D. Gordon, J. Katz, V. Kolesnikov, F. Krell, T. Malkin, M. Raykova, and Y. Vahlis. Secure two-party computation in sublinear (amortized) time. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, 2012.

[19] R. Hall, S. E. Fienberg, and Y. Nardi. Secure multiple linear regression based on homomorphic encryption. *Journal of Official Statistics*, 27, 2011.

[20] W. Henecka, S. Kögl, A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. Tasty: Tool for automating secure two-party computations. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, 2010.

[21] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, 2011.

[22] A. Karatsuba and Y. Ofman. Multiplication of many-digital numbers by automatic computers. In *Proceedings of the USSR Academy of Sciences 145*, pages 293–294, 1962.

[23] A. F. Karr, X. Lin, A. P. Sanil, and J. P. Reiter. Regression on distributed databases via secure multi-party computation. In *Proceedings of the 2004 Annual National Conference on Digital Government Research, DG.O 2004, 2004*, 2004.

[24] M. Keller and P. Scholl. Efficient, oblivious data structures for MPC. In *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II*, pages 506–525, 2014.

[25] D. E. Knuth. *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[26] V. Kolesnikov and T. Schneider. Improved Garbled Circuit: Free XOR Gates and Applications. *Automata, Languages and Programming*, 2008.

[27] Y. Lindell and B. Pinkas. Privacy preserving data mining. In *Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '00, 2000.

[28] Y. Lindell, B. Pinkas, N. P. Smart, and A. Yanai. Efficient constant round multi-party computation combining BMR and SPDZ. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, pages 319–338, 2015.

[29] C. Liu, X. S. Wang, K. Nayak, Y. Huang, and E. Shi. Oblivm: A programming framework for secure computation. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, SP '15, 2015.

[30] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay—a secure two-party computation system. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, 2004.

[31] G. Meurant. *The Lanczos and conjugate gradient algorithms: from theory to finite precision computations*, volume 19. SIAM, 2006.

[32] M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '01, 2001.

[33] M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM Conference on Electronic Commerce*, EC '99, pages 129–139, New York, NY, USA, 1999. ACM.

[34] K. Nayak, X. S. Wang, S. Ioannidis, U. Weinsberg, N. Taft, and E. Shi. Graphsc: Parallel secure computation made easy. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 377–394, 2015.

[35] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. 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.

[36] W. H. Press. *Numerical recipes 3rd edition: The art of scientific computing.* Cambridge university press, 2007.

[37] M. Rabin. How to Exchange Secrets by Oblivious Transfer. Technical Report TR-81, Harvard Aiken Computation Laboratory, 1981.

[38] A. P. Sanil, A. F. Karr, X. Lin, and J. P. Reiter. Privacy preserving regression modeling via distributed computation. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, 2004.

[39] E. M. Songhori, S. U. Hussain, A.-R. Sadeghi, T. Schneider, and F. Koushanfar. Tiny-garble: Highly compressed and scalable sequential garbled circuits. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, SP '15, pages 411–428, 2015.

[40] A. C. Yao. How to generate and exchange secrets. In *Symposium on Foundations of Computer Science (FOCS)*, 1986.

[41] H. Yu, J. Vaidya, and X. Jiang. Privacy-preserving svm classification on vertically partitioned data. In *Proceedings of the 10th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, PAKDD'06, 2006.

[42] S. Zahur and D. Evans. Circuit structures for improving efficiency of security and privacy tools. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 493–507, 2013.

[43] S. Zahur and D. Evans. Obliv-c: A language for extensible data-oblivious computation. Cryptology ePrint Archive, Report 2015/1153, 2015.

[44] S. Zahur, M. Rosulek, and D. Evans. Two Halves Make a Whole: Reducing Data Transfer in Garbled Circuits Using Half Gates. In *EUROCRYPT*, 2015.

[45] S. Zahur, X. Wang, M. Raykova, A. Gascón, J. Doerner, D. Evans, and J. Katz. Revisiting square-root oram: Efficient random access in multi-party computation. In *2016 IEEE Symposium on Security and Privacy*, 2016.