Efficient Secure Ridge Regression from Randomized Gaussian Elimination

Frank Blom, Niek J. Bouman, Berry Schoenmakers, and Niels de Vreede*

Dept Math & CS, TU Eindhoven

Abstract. In this paper we present a practical protocol for secure ridge regression. We develop the necessary secure linear algebra tools, using only basic arithmetic over prime fields. In particular, we will show how to solve linear systems of equations and compute matrix inverses efficiently, using appropriate secure random self-reductions of these problems. The distinguishing feature of our approach is that the use of secure fixed-point arithmetic is avoided entirely, while circumventing the need for rational reconstruction at any stage as well.

We demonstrate the potential of our protocol in a standard setting for information-theoretically secure multiparty computation, tolerating a dishonest minority of passively corrupt parties. Using the MPyC framework, which is based on threshold secret sharing over finite fields, we show how to handle large datasets efficiently, achieving practically the same root-mean-square errors as Scikit-learn. Moreover, we do not assume that any (part) of the datasets is held privately by any of the parties, which makes our protocol much more versatile than existing solutions.

1 Introduction

Recent years have seen significant advances in privacy-preserving data mining and machine learning. Secure multiparty computation (MPC) is a promising type of cryptographic protocol for enhancing the security and privacy properties of existing data mining and machine learning algorithms. Handling large datasets, however, still poses practical challenges due to the overhead incurred by MPC.

Secure regression is a problem that received much attention as the resulting cryptographic protocols have the potential of handling relatively large datasets, see, e.g., [DHC04,HFN11,NWI+13,GJJ+18,GSB+17]. When applied to linear and ridge regression, the overhead for MPC is limited because of the highly linear nature of the computation. The bulk of the computation consists of taking inner products, which can be done securely at limited cost in many MPC frameworks.

In this paper we develop particularly efficient m-party protocols for ridge regression tolerating a dishonest minority of up to t passively corrupt parties, $0 \leq t \leq (m-1)/2$. We present a range of practical optimizations, which are

* Email: f.blom.1@tue.nl, n.j.bouman@tue.nl, berry@win.tue.nl, n.d.vreede@tue.nl. This work has received funding from the European Union's Horizon 2020 research and innovation program under grant agreements No 731583 (SODA) and No 780477 (PRIViLEDGE).

combined into a very competitive solution for secure ridge regression. We present experimental results to support our claims using the MPyC framework for secure multiparty computation.

2 Approach

Ridge regression (or, Tikhonov regularization) is a classic problem in statistics. Nowadays, the problem is broadly studied and applied in machine learning, and many algorithms have been proposed covering various types and dimensions of input data. The popular tool Scikit-learn, for instance, provides six different solvers for ridge regression, most of which also use different approaches for sparse and dense data [PVG+11].

The solver used in the present paper directly uses the closed-form solution for ridge regression, cf. Eqs. (2) and (3). An alternative approach is to approximate the solution using an iterative solver, viewing ridge regression as an optimization problem minimizing (1). Well-known iterative solvers are stochastic gradient descent and its many variations (e.g., mini-batch gradient descent).

However, there are two major impediments for adopting iterative solvers in an MPC setting. Firstly, all arithmetic involves real-valued numbers, which needs to be approximated using secure fixed-point arithmetic (as secure floating-point arithmetic is simply too expensive). The use of secure fixed-point numbers incurs a substantial overhead and could lead to numerical stability issues. Secondly, one needs to control the number of iterations. In an MPC setting, evaluation of a stopping criterion may form a bottleneck in itself, and fixing the number of iterations beforehand may demand a high number of iterations (to ensure convergence for all inputs). The advantage of the iterative approach is that it generalizes immediately to related machine learning algorithms such as logistic regression and support vector machines. Adapting the computation of the gradient suffices to solve these problems as well.

As we show in this paper, there are major advantages to solving the ridge regression problem directly. It allows us to avoid fixed-point arithmetic entirely. Issues surrounding rounding errors are limited to the input phase, when real-valued inputs are converted to integral values using appropriate scaling. From that point on all computations are exact, using integer arithmetic only. The main issue left is the growth of the numbers, but we will show that even for huge datasets, our approach is practical and leads to very competitive results in an MPC setting.

The closed-form solution is in fact a matrix equation, which can in turn be solved directly or iteratively, as we will discuss in Section 6.

2.1 Roadmap

We present mathematical preliminaries in Section 3, and the basics on linear regression and ridge regression in Section 4. Next we introduce basic notation for MPC based on Shamir secret sharing in Section 5. In Section 6 we discuss

the relevant choices for solving linear systems of equations in an MPC setting, showing how we avoid the use of rational reconstruction. Section 7 contains the basic protocols for secure linear algebra, which we use in our protocol for secure ridge regression in Section 8. Finally, we discuss the performance in Section 9 and conclude in Section 10.

3 Preliminaries

We use common notation for matrices and vectors. For $d \geq 1$, the group of $d \times d$ invertible matrices over a field $\mathbb{F}$ is denoted by $\mathrm{GL}_d(\mathbb{F})$. The groups of $d \times d$ lower resp. upper triangular invertible matrices are denoted by $\mathrm{L}_d(\mathbb{F}), \mathrm{U}_d(\mathbb{F}) \subseteq \mathrm{GL}_d(\mathbb{F})$, and we use $\mathrm{L}^1_d(\mathbb{F})$ to denote the group of lower triangular matrices with an all-ones diagonal.

A matrix $A \in \mathrm{GL}_d(\mathbb{F})$ is said to have an **LU-decomposition** if $A = LU$ for some $L \in \mathrm{L}^1_d(\mathbb{F})$ and $U \in \mathrm{U}_d(\mathbb{F})$. We use $\mathrm{LU}_d(\mathbb{F})$ to denote the set of all matrices in $\mathrm{GL}_d(\mathbb{F})$ that have an LU-decomposition. Note that the LU-decomposition for each $A \in \mathrm{LU}_d(\mathbb{F})$ is unique. Similarly, a matrix $A \in \mathrm{GL}_d(\mathbb{F})$ is said to have a **Cholesky decomposition** if $A = LL^\mathsf{T}$ for some $L \in \mathrm{L}_d(\mathbb{F})$. The Cholesky decomposition is also unique, and exists over $\mathbb{F} = \mathbb{R}$ if and only if A is symmetric and positive definite.

For $A \in \mathrm{GL}_d(\mathbb{F})$, we use $\operatorname{adj} A = \det(A)A^{-1}$ to denote the **adjugate** of A. For our approach, a key property is that if A is integral then so are $\det A$ and $\operatorname{adj} A$. That is, if A is a matrix over $\mathbb{Z}$, then $\det A \in \mathbb{Z}$ and $\operatorname{adj} A$ is also a matrix over $\mathbb{Z}$. Furthermore, **Hadamard's inequality** states that $|\det A| \leq \prod_{i=1}^{d} \|\boldsymbol{a}_i\|_2$, where $\boldsymbol{a}_i$ are the rows (or columns) of A. For $\alpha = \|A\|_{\max}$, Hadamard's inequality implies $|\det A| \leq d^{d/2}\alpha^d$. If A is symmetric and positive definite, $\det A$ is positive and Hadamard's inequality becomes $\det A \leq \prod_{i=1}^{d} a_{i,i}$ and we get $\det A \leq \alpha^d$. Finally, Hadamard's inequality yields $\|\operatorname{adj} A\|_{\max} \leq (d-1)^{(d-1)/2}\alpha^{d-1}$ as bound for the adjugate.

Gaussian elimination and variations thereof are used to compute $\det A$, $\operatorname{adj} A$, and A^{-1}. For example, A^{-1} is computed by transforming the augmented matrix $(A \mid I)$ into $(I \mid A^{-1})$ by means of Gauss-Jordan elimination. Similarly, if A has an LU-decomposition, applying Gaussian elimination to A amounts to multiplying A from the left by the lower triangular matrix L^{-1}, resulting in $U = L^{-1}A$. Hence, the upper triangular matrix U is obtained without applying any *pivoting* steps. Putting $\det A = \det U = \prod_{i=1}^{d} u_{i,i}$ yields the determinant.

We will perform Gaussian elimination over finite fields of large prime order p, and we will do so for essentially uniformly random matrices only. As a consequence, there will be no need for pivoting and all computations will be exact. Inspired by Bareiss [Bar68], we will combine division-free Gaussian elimination with back substitution such that $\det A$ is obtained at almost no extra cost. See Section 7.3 for further details.

4 Ridge Regression

Ridge regression is a well-known technique in statistics and machine learning [FHT01], which can be seen as a refinement of the ordinary least squares method used in linear regression. Ridge regression provides the user with a handle, the *regularization parameter* $\lambda > 0$, that can be used to reduce the variance of the prediction at the cost of introducing some bias. If λ is set properly, ridge regression can outperform the ordinary least squares method in terms of the root mean-square error, defined below. In high-dimensional problems, ridge regression can help to reduce the problem of overfitting.

Given an overdetermined linear system $X\boldsymbol{w} = \boldsymbol{y}$, the least squares solution $\boldsymbol{w}$ minimizes $\|X\boldsymbol{w} - \boldsymbol{y}\|_2$. Typically, X is an $n \times d$ matrix over $\mathbb{R}$ with $n \gg d$. Each row of X represents an input record with d features, and the corresponding entry of $\boldsymbol{y}$ represents the known output value. The least squares solution $\boldsymbol{w} = (X^\mathsf{T}X)^{-1}X^\mathsf{T}\boldsymbol{y}$, is used as the optimal weight vector for predicting the output values for new input records $\boldsymbol{x}$ by evaluating $\boldsymbol{x}^\mathsf{T}\boldsymbol{w}$.

Ridge regression finds a vector $\boldsymbol{w}$ minimizing

$$\|X\boldsymbol{w} - \boldsymbol{y}\|_2^2 + \lambda\|\boldsymbol{w}\|_2^2, \tag{1}$$

where we note the presence of the regularization parameter in the second term. The solution $\boldsymbol{w}$ minimizing (1) is now given by:

$$\boldsymbol{w} = \left(X^\mathsf{T}X + \lambda I\right)^{-1} X^\mathsf{T}\boldsymbol{y}. \tag{2}$$

To compute $\boldsymbol{w}$, one solves the linear system $A\boldsymbol{w} = \boldsymbol{b}$ with $A = X^\mathsf{T}X + \lambda I$ and $\boldsymbol{b} = X^\mathsf{T}\boldsymbol{y}$. Note that the regularization parameter λ not only suppresses large entries in $\boldsymbol{w}$, but also ensures that A is positive definite, hence invertible: $\boldsymbol{z}^\mathsf{T}\left(X^\mathsf{T}X + \lambda I\right)\boldsymbol{z} = \|X\boldsymbol{z}\|_2^2 + \lambda\|\boldsymbol{z}\|_2^2 > 0$ for any nonzero $\boldsymbol{z}$, since $\lambda > 0$.

In the context of machine learning, the input records X along with the known output values $\boldsymbol{y}$ are called the training set, and the least-squares solution $\boldsymbol{w}$ is called the model. The performance of the model is evaluated in terms of the root-mean-square error (RMSE) of the model's predictions. The model complexity (training error) is defined as the RMSE for the training set, which is equal to $\|X\boldsymbol{w} - \boldsymbol{y}\|_2/\sqrt{n}$. The generalizability (test error) of the model is defined as the RMSE for a test set $(X', \boldsymbol{y}')$, which is equal to $\|X'\boldsymbol{w} - \boldsymbol{y}'\|_2/\sqrt{n'}$. Overall, the goal is to ensure that both RMSEs are small and approximately equal to each other.

The performance of a machine learning algorithm critically depends on the quality of the input data. Extensive data preprocessing may be required in practice to enhance the quality of the input data. In our experiments we will use standard datasets from the UCI repository, for which most of the data preprocessing has already been done. The only two tasks that remain before applying ridge regression to these datasets are (i) feature scaling and (ii) encoding of categorical features.

For feature scaling, we apply min-max scaling to each of the columns of X and to vector $\boldsymbol{y}$ as well. Concretely, all features are scaled to the range $[-1, 1]$.

We prefer this form of data normalization because it requires little processing and does not leak too much information about X and $\boldsymbol{y}$.

To encode categorical features (including Boolean features), we basically use a form of "one-hot encoding" with respect to the range $[-1, 1]$. For Boolean features, we encode the values True and False by 1 and -1, respectively. A categorical feature with s possible values is encoded by s Boolean features, where the value for exactly one of the Boolean features will be set to 1 and the remaining $s-1$ Boolean features are set to -1.

5 MPC Setting

We consider an MPC setting with m parties tolerating a dishonest minority of up to t passively corrupt parties, $0 \leq t \leq (m-1)/2$. The basic protocols for secure addition and multiplication over a finite field rely on Shamir secret sharing [BGW88,GRR98]. For our practical experiments we use the MPyC framework [Sch18], which succeeds the VIFF framework [Gei10].

Let $p > m$ be a prime. We use $[\![a]\!]_p$ or $[\![a]\!]$ to denote a secret-shared value $a \in \mathbb{Z}_p$, where a is interpreted as a signed integer in the range $\{-\lfloor p/2 \rfloor, \ldots, \lfloor p/2 \rfloor\}$. We assume that secure field arithmetic ($+, -, *, /$ modulo p) is supported efficiently as well as secure generation of random numbers (e.g., $[\![r]\!]$ with $r \in_R \mathbb{Z}_p$).

We highlight three auxiliary protocols which are of particular relevance for our approach.

For secure dot products $[\![\boldsymbol{x}]\!] \cdot [\![\boldsymbol{y}]\!]$ with $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{Z}_p^d$, we recall that the complexity is the same as for a single secure multiplication, except for local computations. This extends to secure matrix products $[\![A]\!][\![B]\!]$ with $A, B \in \mathbb{Z}_p^{d \times d}$, for which the complexity is equivalent to d^2 secure multiplications in parallel.

Next, to generate $[\![r]\!]$ and $[\![1/r]\!]$ for a random $r \in_R \mathbb{Z}_p^*$, one proceeds as follows: generate $[\![r]\!]$, $[\![u]\!]$ with $r, u \in_R \mathbb{Z}_p$, open $[\![r]\!][\![u]\!]$ to obtain ru, and output $[\![r]\!]$ and $[\![1/r]\!] = [\![u]\!]/(ru)$. For large p, $ru \neq 0$ will hold with overwhelming probability; if $ru = 0$ simply try again with fresh r and u.

Finally, we will also use secure conversion between secret-shared values in different prime fields. In particular, for primes p and q satisfying $p > q > 2^{\kappa+\ell}$, where κ is a security parameter, we use a secure protocol for converting $[\![a]\!]_q$ into $[\![a]\!]_p$, $-2^{\ell-1} \leq a < 2^{\ell-1}$. Roughly, such a protocol proceeds by generating $[\![r]\!]_q$ and $[\![r]\!]_p$ for a random $r \in [0, 2^{\ell+\kappa})$. Then the value of $a + r$ is opened, which is statistically indistinguishable from random for κ sufficiently large, and one sets $[\![a]\!]_p = a + r - [\![r]\!]_p$.

6 Solving Systems of Linear Equations

As outlined in Section 4, we divide ridge regression into two main stages. In the first stage we compute $A = X^\mathsf{T} X + \lambda I$ and $b = X^\mathsf{T} y$, and in the second stage we solve $A\boldsymbol{w} = \boldsymbol{b}$ to find $\boldsymbol{w}$. For secure ridge regression, the most interesting and challenging part will be the second stage, and in this section we motivate our approach for solving $A\boldsymbol{w} = \boldsymbol{b}$.

In numerical analysis one distinguishes two major types of solution methods for systems of linear equations. Direct methods, such as Gaussian elimination, run in a finite number of steps and compute an exact solution in the absence of rounding errors. Iterative methods, such as conjugate gradient, yield approximate solutions within a limited amount of time, even for very large matrices. In contrast to some other recent work (e.g., [GSB+17]), we will choose a direct method to solve $A\boldsymbol{w} = \boldsymbol{b}$ in our protocols for secure ridge regression. Below we explain our reasons for doing so.

An important observation is that we can actually use exact computation for secure ridge regression. Instead of relying on fixed-point arithmetic—or even worse floating-point arithmetic—in an MPC setting, we will only use exact integer arithmetic in our protocols. This way we take advantage of the fact that secure integer arithmetic is efficient even for large values when using Shamir secret sharing. Moreover, we can borrow techniques from the related setting of secure linear algebra over finite fields [CD01].

We will make sure that the input data (contained in X and $\boldsymbol{y}$) are scaled to integer values, basically by multiplying each input value with 2^α and rounding to the nearest integer for a fixed value of α. The value of α must be sufficiently large to ensure that the final results will be accurate. We will refer to α as the *accuracy* parameter.

Since A is invertible, solving $A\boldsymbol{w} = \boldsymbol{b}$ is equivalent to computing $\boldsymbol{w} = A^{-1}\boldsymbol{b}$. Therefore, even if A contains integer values only, the solution $\boldsymbol{w}$ will in general contain rational values. As $A^{-1} = (\det A)^{-1} \operatorname{adj} A$, however, it suffices to compute $\boldsymbol{w}' = (\operatorname{adj} A)\boldsymbol{b}$ and $z = \det A$, from which $\boldsymbol{w}$ can be recovered as $\boldsymbol{w} = \boldsymbol{w}'/z$. Here, both $\boldsymbol{w}'$ and z are integral. We compute $\boldsymbol{w}'$ and z by first reducing the augmented matrix $(A \mid b)$ to echelon form using Gaussian elimination and then applying back substitution to recover $\boldsymbol{w}$.

To perform *secure* Gaussian elimination on $(A \mid b)$ there are several options. A first idea is to use Gaussian elimination (row reduction) directly, which amounts to repeatedly selecting a pivot and updating the matrix accordingly. However, oblivious row reduction, hiding the position of the pivot and so on, is computationally very costly: searching for a nonzero element in the pivot column is already nontrivial in a secure setting, and obliviously swapping entire rows to move the pivot to the diagonal is even much more costly.

A common technique in numerical analysis to avoid pivot selection is the use of *preconditioning*. Roughly, the idea is to solve the equivalent system $RA\boldsymbol{w} = R\boldsymbol{b}$ for a random matrix R, instead of the original system $A\boldsymbol{w} = \boldsymbol{b}$. Matrix R is assumed to invertible, which is true with overwhelming probability in many settings. When solving linear systems over $\mathbb{R}$, such an approach is numerically unstable and leads to poor results. When solving linear systems over a finite field, however, numerical instability is of no concern. We will follow this approach.

The upshot of computing $(\operatorname{adj} A)\boldsymbol{b}$ and $\det A$ separately is that we will also avoid the use of rational reconstruction. In the next section we will show why this lets us essentially *halve* the size of the prime modulus for the finite field arithmetic compared to other papers. For instance, [GJJ+18] relies on rational

Protocol 1 $\mathrm{Det}([\![A]\!])$ $A \in \mathrm{GL}_d(\mathbb{Z}_p)$

1: Generate $[\![R]\!]$, $[\![\det R^{-1}]\!]$ with $R \in_R \mathrm{LU}_d(\mathbb{Z}_p)$ using Protocol 2.
2: Open $RA \leftarrow [\![R]\!][\![A]\!]$.
3: Compute $[\![\det A]\!] = \det(RA)[\![\det R^{-1}]\!]$.
4: Return $[\![\det A]\!]$.

reconstruction and uses a modulus which should be large enough to "hold" the *product* of $(\mathrm{adj}\, A)\boldsymbol{b}$ and $\det A$.

7 Secure Linear Algebra

We present protocols for computing determinants, matrix inverses, and solutions to linear systems. Given an invertible matrix $A \in \mathbb{Z}^{d\times d}$, we compute the results over $\mathbb{Z}_p$ assuming p is sufficiently large. E.g., for $-p/2 < \det A < p/2$, $\det A \in \mathbb{Z}_p^*$ and A is properly embedded in $\mathbb{Z}_p^{d\times d}$. Assuming further bounds on the entries of A and $\boldsymbol{b}$, we will show how to compute A^{-1} and $A^{-1}\boldsymbol{b}$ over $\mathbb{Z}_p$ as well.

7.1 Secure Determinant

Cramer and Damgård presented a protocol for secure computation of $\det A$ over any finite field [CD01], which is reminiscent of Bar-Ilan and Beaver's protocol for secure multiplicative inverses [BIB89]. The idea is to securely generate a random invertible matrix $[\![R]\!]$ together with its determinant $[\![\det R]\!]$, open the randomized matrix RA, and finally compute $[\![\det A]\!]$. We follow the same approach in Protocol 1, except that we improve upon the way random matrix R is generated in several ways.

Ideally, R is generated as a random matrix in $\mathrm{GL}_d(\mathbb{Z}_p)$. To securely compute $\det R$ as well, matrix R is limited to the slightly smaller range $\mathrm{LU}_d(\mathbb{Z}_p)$ of matrices that have an LU-decomposition. The following lemma shows that uniformly random matrices in $\mathrm{LU}_d(\mathbb{Z}_p)$ are statistically indistinguishable from uniformly random matrices in $\mathrm{GL}_d(\mathbb{Z}_p)$. Therefore, opening RA reveals negligible information on A only.

Lemma 1. *$\Delta(R;G) \le d/p$, for $R \in_R \mathrm{LU}_d(\mathbb{Z}_p)$ and $G \in_R \mathrm{GL}_d(\mathbb{Z}_p)$.*

Proof. Since $\mathrm{LU}_d(\mathbb{Z}_p) \subseteq \mathrm{GL}_d(\mathbb{Z}_p)$ and R and G are both uniform we have

$$\Delta(R;G) = \tfrac{1}{2} \sum_{x\in \mathrm{GL}_d(\mathbb{Z}_p)} |\Pr[R=x] - \Pr[G=x]| = 1 - \frac{|\mathrm{LU}_d(\mathbb{Z}_p)|}{|\mathrm{GL}_d(\mathbb{Z}_p)|}.$$

Since $|\mathrm{LU}_d(\mathbb{Z}_p)| = p^{d^2-d}(p-1)^d$ and $|\mathrm{GL}_d(\mathbb{Z}_p)| \le p^{d^2}$, we have

$$\Delta(R;G) \le 1 - \left(\frac{p-1}{p}\right)^d = 1 - \left(1 - \frac{1}{p}\right)^d \le \frac{d}{p},$$

using Bernoulli's inequality in the last step.

Protocol 2 RndMatDet(d)

1: Generate $[\![L]\!]$ with $L \in_R \mathrm{L}^1_d(\mathbb{Z}_p)$.
2: Generate $[\![U]\!]$, $[\![\det U^{-1}]\!]$ with $U \in_R \mathrm{U}_d(\mathbb{Z}_p)$.
3: Compute $[\![R]\!] = [\![L]\!][\![U]\!]$.
4: Set $[\![\det R^{-1}]\!] = [\![\det U^{-1}]\!]$.
5: Return $[\![R]\!]$, $[\![\det R^{-1}]\!]$.

To sample a matrix R securely from $\mathrm{LU}_d(\mathbb{Z}_p)$, we use Protocol 2. The protocol also outputs the determinant of R, or rather its inverse. Random matrices in $\mathrm{L}^1_d(\mathbb{Z}_p)$ and $\mathrm{U}_d(\mathbb{Z}_p)$ can be generated easily, provided we can securely generate random elements of $\mathbb{Z}_p$. To ensure that U is invertible, we generate $u_{i,i} \in_R \mathbb{Z}_p$ for $i = 1, \ldots, d$, and then apply secure inversion to $\det U = \prod_{i=1}^{d} u_{i,i}$. With negligible probability $\det U = 0$, in which case secure inversion will fail and we have to try again. With overwhelming probability, however, $\det U \neq 0$ and secure inversion will succeed. In total, Protocol 2 roughly uses d^2 random elements from $\mathbb{Z}_p$.

Our protocol for generating random matrices improves upon Cramer and Damgård's protocol Π_0 [CD01, p. 126] in several respects. The main difference is that protocol Π_0 depends on a redundant type of LU-decomposition in which the diagonals of both L and U consist of elements in $\mathbb{Z}_p^*$. By fixing the diagonal of L to all ones, the LU-decomposition used in our protocol is unique. As an immediate consequence, our proof for statistical indistinguishability is much simpler (cf. Lemma 1). Moreover, the complexity of the protocol is reduced as we do not need to generate the diagonal of L at random, and we do not need to compute $\det L$ either. Finally, as a further optimization, we only use one secure inversion throughout the entire protocol (to perform the secure zero-test and inversion for $\det U$ all at the same time).

Apart from generating a random matrix R and its inverse determinant, Protocol 1 mainly performs a secure matrix multiplication. The computation of $\det(RA)$ is done locally, so we might use any algorithm for computing determinants to implement this step. However, Lemma 1 helps us save some work for the local computation as well. The lemma basically implies that RA is statistically close to a uniformly random matrix in $\mathrm{LU}_d(\mathbb{Z}_p)$, and therefore we can perform Gaussian elimination to compute $\det(RA)$ without any pivoting, as shown below.

7.2 Secure Matrix Inversion

We next present Protocol 3 for secure matrix inversion, which is of independent interest. Since A^{-1} will in general have rational entries for a matrix $A \in \mathbb{Z}^{d \times d}$, as discussed above, we will use the pair $(\operatorname{adj} A, \det A)$ as representation of A^{-1}. This way we avoid any rational arithmetic, and, moreover, we can use a similar embedding for A in $\mathbb{Z}_p^{d \times d}$ as for the determinant, using the bound for $\|\operatorname{adj} A\|_{\max}$ from Section 3 to choose p sufficiently large.

Protocol 3 AdjDet($[\![A]\!]$) $A \in \mathrm{GL}_d(\mathbb{Z}_p)$

1: Generate $[\![R]\!]$, $[\![\det R^{-1}]\!]$ with $R \in_R \mathrm{LU}_d(\mathbb{Z}_p)$ using Protocol 2.
2: Open $RA \leftarrow [\![R]\!][\![A]\!]$.
3: Reduce $(RA \mid [\![R]\!])$ to obtain $[\![A^{-1}]\!]$ by Gauss-Jordan elimination over $\mathbb{Z}_p$.
4: Compute $[\![\det A]\!] = \det(RA)[\![\det R^{-1}]\!]$.
5: Compute $[\![\mathrm{adj}\, A]\!] = [\![\det A]\!][\![A^{-1}]\!]$.
6: Return $[\![\mathrm{adj}\, A]\!]$, $[\![\det A]\!]$.

Protocol 4 LinSol($[\![A]\!], [\![\boldsymbol{b}]\!]$) $A \in \mathrm{GL}_d(\mathbb{Z}_p)$, $\boldsymbol{b} \in \mathbb{Z}_p^d$

1: Generate $[\![R]\!]$, $[\![\det R^{-1}]\!]$ with $R \in_R \mathrm{LU}_d(\mathbb{Z}_p)$ using Protocol 2.
2: Open $RA \leftarrow [\![R]\!][\![A]\!]$.
3: Solve $(RA \mid [\![R]\!][\![\boldsymbol{b}]\!])$ to obtain $[\![A^{-1}\boldsymbol{b}]\!]$ by Gaussian elimination over $\mathbb{Z}_p$.
4: Compute $[\![\det A]\!] = \det(RA)[\![\det R]\!]^{-1}$.
5: Compute $[\![(\mathrm{adj}\, A)\boldsymbol{b}]\!] = [\![\det A]\!][\![A^{-1}\boldsymbol{b}]\!]$.
6: Return $[\![(\mathrm{adj}\, A)\boldsymbol{b}]\!]$, $[\![\det A]\!]$.

If we stick to the common approach of computing $A^{-1} = (\det A)^{-1} \mathrm{adj}\, A$ over $\mathbb{Z}_p$, such that $\mathrm{adj}\, A$ and $\det A$ can be recovered using rational reconstruction over $\mathbb{Z}_p$, the required size for p would be roughly twice as large.

7.3 Secure Linear Solver

Finally, we present Protocol 4 for securely solving a linear system, in which we avoid performing a full matrix inversion. In step 3 we apply Gaussian elimination to the augmented matrix $(RA \mid [\![R]\!][\![\boldsymbol{b}]\!])$. As explained in Section 6, this can be done without pivoting. Matrix RA is first transformed into upper-triangular form, and then we apply back substitution to compute $[\![A^{-1}\boldsymbol{b}]\!]$. For Gaussian elimination on $(RA \mid [\![R]\!][\![\boldsymbol{b}]\!])$, we use the division-free variant (see, e.g., [Bar68]). Combined with back substitution, we achieve that $\det(RA)$ is obtained at almost no additional cost. In total we need $\frac{2}{3}d^3 + O(d^2)$ multiplications, $\frac{1}{3}d^3 + O(d^2)$ modular reductions, and exactly n inversions modulo p for step 3.

8 Secure Ridge Regression

In this section we present our protocol for ridge regression, see Protocol 5. All entries of X and $\boldsymbol{y}$ are assumed to be in $[-2^\alpha, 2^\alpha] \cap \mathbb{Z}$ for an appropriate value of the accuracy parameter α (i.e., normalized to $[-1, 1]$ as explained in Section 4, scaled by a factor of 2^α, and rounded to the nearest integer). The regularization parameter λ is scaled accordingly. We note that parameter α is between 5 and 10 in our experiments, cf. Table 2.

The two main stages of ridge regression are performed over two different prime fields. In the first stage, $X^\mathsf{T}X + \lambda I$ and $X^\mathsf{T}\boldsymbol{y}$ are computed over a relatively small field $\mathbb{Z}_q$, while $\boldsymbol{w} = A^{-1}\boldsymbol{b}$ is computed over a substantially larger field $\mathbb{Z}_p$ in the second stage. See Table 2 for some typical sizes of p and q. Since n is

Protocol 5 Ridge($[\![X]\!]_q, [\![\boldsymbol{y}]\!]_q, \lambda$) $\qquad X \in \mathbb{Z}_q^{n\times d}, \boldsymbol{y} \in \mathbb{Z}_q^n, \lambda \in \mathbb{N}$

1: Compute $[\![A]\!]_q = [\![X^\mathsf{T}]\!]_q [\![X]\!]_q + \lambda I$. $\quad \triangleright\ A = X^\mathsf{T}X + \lambda I$
2: Compute $[\![\boldsymbol{b}]\!]_q = [\![X^\mathsf{T}]\!]_q [\![\boldsymbol{y}]\!]_q$. $\quad \triangleright\ \boldsymbol{b} = X^\mathsf{T}\boldsymbol{y}$
3: Convert $[\![(A \mid \boldsymbol{b})]\!]_q$ to $[\![(A \mid \boldsymbol{b})]\!]_p$.
4: Compute $([\![(\operatorname{adj} A)\boldsymbol{b}]\!]_p, [\![\det A]\!]_p) = \mathrm{LinSol}([\![(A \mid \boldsymbol{b})]\!]_p)$.
5: Set $[\![(\det A)\boldsymbol{w}]\!]_p = [\![(\operatorname{adj} A)\boldsymbol{b}]\!]_p$. $\quad \triangleright\ \boldsymbol{w} = A^{-1}\boldsymbol{b}$
6: Return $[\![(\det A)\boldsymbol{w}]\!]_p, [\![\det A]\!]_p$.

typically very large as well, cf. Table 1, secure computation of $X^\mathsf{T}X$ over $\mathbb{Z}_p$ would put excessive demands on time and space utilization.

The conversion in step 3 of the protocol is done as described at the end of Section 5. The sizes for primes p and q are determined in the following lemma.

Lemma 2. *Let* $\beta = \|(X \mid \boldsymbol{y})\|_{\max}$. *Correctness of Protocol 5 follows if*

$$\frac{q}{2} > n\beta^2 + \lambda + 2^\kappa \quad \textit{and} \quad \frac{p}{2} > d(d-1)^{\frac{d-1}{2}}(n\beta^2+\lambda)^d.$$

Proof. For prime q we need that $q/2 > \|(A \mid \boldsymbol{b})\|_{\max}$. Each entry of $(A \mid \boldsymbol{b})$ is a dot product of two length-n vectors with entries bounded in absolute value by β, plus λ for the diagonal of A. Therefore, $\|(A \mid \boldsymbol{b})\|_{\max} \le n\beta^2 + \lambda$. To allow for secure conversion from $\mathbb{Z}_q$ to $\mathbb{Z}_p$, we require $q/2 > n\beta^2 + \lambda + 2^\kappa$.

For prime p we need that $p/2 > \det A$ and $p/2 > \|(\operatorname{adj} A)\boldsymbol{b}\|_\infty$. We use the bounds for $\det A$ and $\operatorname{adj} A$ obtained from Hadamard's inequality in Section 3 as follows. For the determinant of A, we have the bound $\det A \le (n\beta^2+\lambda)^d$ since A is symmetric positive definite. For the adjugate of A, we have the bound $\|(\operatorname{adj} A)\|_{\max} \le (d-1)^{\frac{d-1}{2}}(n\beta^2+\lambda)^{d-1}$. So, together with the bound $\|\boldsymbol{b}\|_\infty \le n\beta^2$, we take $p/2 > d(d-1)^{\frac{d-1}{2}}(n\beta^2+\lambda)^d$ as overall bound.

9 Performance Evaluation

We have performed several experiments using the UCI datasets [DG19] shown in Table 1. Each dataset is randomly split into a 70% training set and a 30% test set (except for dataset *Year prediction MSD*, for which the first 463715 rows form the training set and the remaining 51630 rows are used for testing). The RMSEs reported for training and testing are obtained using the Cholesky solver provided for ridge regression in Scikit-learn [PVG+11], setting $\lambda = 1$. Note that the *Gas Sensor Array* datasets have two targets, for which the RMSEs are reported separately. To handle multiple targets, we have generalized Protocol 5 in the obvious way, replacing vector $\boldsymbol{y}$ by a matrix Y with one column per target.

We have run our protocol for secure ridge regression in a 3-party setting using the values for accuracy parameter α shown in Table 2. For each (normalized) dataset we have tried increasingly larger values for α until the errors became insignificant (below 0.1% relative to the RMSEs of Table 1). We have refrained from tuning the regularization parameter λ, and simply set $\lambda = 2^{2\alpha}$ (which

id	dataset	n	d	target(s)	train RMSE	test RMSE
1	Student Performance	395	30	G3	0.38	0.46
2	Wine Quality red	1,599	11	quality	0.16	0.16
3	Wine Quality white	4,898	11	quality	0.19	0.19
4	Year Prediction MSD	515,345	90	year	0.21	0.21
5	Gas Sensor Array methane	4,178,504	16	ethylene methane	0.29 0.34	0.29 0.34
6	Gas Sensor Array CO	4,208,261	16	ethylene CO	0.34 0.34	0.34 0.34
7	HIGGS	11,000,000	7	class	0.97	0.97

Table 1. UCI datasets. RMSEs for ridge regression with Scikit-learn.

				This work			[NWI+13]	[GJJ+18]		[GSB+17]	
id	α	$\lvert q\rvert$	$\lvert p\rvert$	$(A,\boldsymbol{b})$	$A^{-1}\boldsymbol{b}$	total	HP	HP	VP	VP 32-bit	VP 64-bit
1	6	54	1316	0.13	1.48	1.61	-	39.76	328.06	5 (-0.0%)	35 (-0.0%)
2	7	58	314	0.02	0.08	0.10	39	-	-	-	-
3	8	61	357	0.04	0.12	0.16	45	4.09	-	0 (4.2%)	4 (-0.0%)
4	6	64	3105	237	18.3	255	-	-	-	230 (0.0%)	808 (0.0%)
5	8	71	675	62.8	0.05	62.9	-	-	-	-	-
6	9	73	709	63.0	0.05	63.1	-	-	-	42 (5.2%)	69 (0.0%)
7	5	66	277	34.9	0.12	35.0	-	-	-	-	-

Table 2. Results of this work compared to the literature. All times are in seconds. HP/VP stand for horizontal/vertical partitioning. 32-bit and 64-bit refer to bit lengths used for secure fixed-point arithmetic. Accuracy α yields relative errors below 0.1%. The relative errors reported by [GSB+17] are also given.

corresponds to $\lambda = 1$ after scaling). The bit lengths $|p|$ and $|q|$ are determined from the bounds in Lemma 2, using $\beta = 2^\alpha$ and $\kappa = 30$. The total running time for Protocol 5 comprises two parts: $(A,\boldsymbol{b})$-time represents the time for computing $[\![A]\!]$ and $[\![b]\!]$ (steps 1–2), while $A^{-1}\boldsymbol{b}$-time covers the time for Protocol 4. The time for the conversion in step 3 is negligible.

Our implementation is written in Python using the MPyC package [Sch18, ridgeregression.py]. The experiments were done using three PCs, connected via a Netgear GS208-100PES Ethernet switch. Each PC was running on Windows 8.1 Enterprise (64-bit) with an Intel Core i7-4770 CPU at 3.40GHz and 16GB of RAM. Table 2 compares our results to three other solutions for secure ridge regression from the literature. The times reported are purely indicative, and give a basic idea of the performance of the various solutions.

We note that the previous works shown in Table 2 exploit the locality of the input data, assuming that the data is either partitioned horizontally or vertically. For example, Nikolaenko et al. [NWI+13] assume the dataset is partitioned

n	d	This work	[GSB+17]
50,000	20	1s	2s
50,000	100	24s	32s
500,000	20	11s	18s
500,000	100	3m29s	6m1s
1,000,000	100	6m57s	12m42s
1,000,000	200	28m30s	49m56s

Table 3. Comparison of $(A, \boldsymbol{b})$-times for synthetic data.

horizontally, allowing them to compute A and $\boldsymbol{b}$ using additive homomorphic encryption only. In our work, we do not make any specific assumptions on the distribution of the input data; any data provider simply sends secret shares of its data to the respective parties performing the secure computation (using $\log_2 q$ bits per share). Thus, in our experiments, each party holds shares of the *entire* dataset.

Also, these previous works [NWI+13,GSB+17,GJJ+18] rely on a so-called 2-server approach requiring two non-colluding parties (e.g., a "crypto service provider" and an "evaluator"). For our solution, the number of colluding parties tolerated is scalable, assuming an honest majority.

The competitiveness of our solution is also confirmed by Table 3, showing our results for a range of synthetic datasets compared to the most favorable results reported by Gascón et al. [GSB+17] (for their 3-party setting).

10 Concluding Remarks

Assuming that matrix X is of full column rank, Protocol 5 can also be used for secure linear regression by setting $\lambda = 0$. If matrix X is distributed among several data providers, however, ensuring that X is of full rank need not be trivial. For instance, in a vertical data partitioning it may not that easy to detect a redundant feature (used by multiple data providers). Setting $\lambda > 0$ removes the need to remove such redundant columns.

Our results also extend to the underdetermined case $n < d$. In this case, the closed form solution given by Eq. (2) can be rewritten as

$$\boldsymbol{w} = X^\mathsf{T} \left(XX^\mathsf{T} + \lambda I\right)^{-1} \boldsymbol{y}, \tag{3}$$

using that $(X^\mathsf{T}X + \lambda I)X^\mathsf{T} = X^\mathsf{T}XX^\mathsf{T} + \lambda X^\mathsf{T} = X^\mathsf{T}(XX^\mathsf{T} + \lambda I)$.

Modifying our protocol for ridge regression accordingly results in Protocol 6. In step 4 of the protocol the secret-shared matrix X converted to the large prime field $\mathbb{Z}_p$ is used to compute the output vector $\boldsymbol{w}$. Since typically $d \gg n$, a relatively small number of conversions n per entry of the length-d output vector $\boldsymbol{w}$ are performed. Setting $\lambda = 0$ for this protocol yields a solution for secure linear regression in the case that X is of full row rank.

Protocol 6 Ridge($[\![X]\!]_q, [\![\boldsymbol{y}]\!]_q, \lambda$) $X \in \mathbb{Z}_q^{n \times d}, \boldsymbol{y} \in \mathbb{Z}_q^n, \lambda \in \mathbb{N}$

1: Compute $[\![A]\!]_q = [\![X]\!]_q [\![X^\mathsf{T}]\!]_q + \lambda I$. ▷ $A = XX^\mathsf{T} + \lambda I$
2: Convert $[\![(A \mid X \mid \boldsymbol{y})]\!]_q$ to $[\![(A \mid X \mid \boldsymbol{y})]\!]_p$.
3: Compute $([\![(\operatorname{adj} A)\boldsymbol{y}]\!]_p, [\![\det A]\!]_p) = \text{LinSol}([\![(A \mid \boldsymbol{y})]\!]_p)$.
4: Compute $[\![(\det A)\boldsymbol{w}]\!]_p = [\![X^\mathsf{T}]\!]_p [\![(\operatorname{adj} A)\boldsymbol{y}]\!]_p$. ▷ $\boldsymbol{w} = X^\mathsf{T} A^{-1} \boldsymbol{y}$
5: Return $[\![(\det A)\boldsymbol{w}]\!]_p, [\![\det A]\!]_p$.

The approach presented in this paper is generalized in follow-up work by Bouman and de Vreede [BV19], in which they show how to compute the Moore-Penrose pseudoinverse securely. Pseudoinverses are much harder to compute securely as one needs to hide all information about the rank of the input matrix.

Details about the handling of the input and output for our secure ridge protocols are beyond the scope of this paper. For instance, one needs to decide how much information the parties are willing to leak when normalizing their joint datasets. Also, the parties may jointly need to determine a suitable value for the regularization parameter λ (hyperparameter tuning). Similarly, the output $[\![(\det A)\boldsymbol{w}]\!]_p, [\![\det A]\!]_p$ of our secure ridge protocols can be handled in lots of ways. These two values may simply be revealed, accepting leakage of the exact value of the determinant. Alternatively, these values may be converted to shares over $\mathbb{Z}_{p'}$, where p' is of double length compared to p, followed by rational reconstruction modulo p' to recover $\boldsymbol{w}$ in the clear.

References

Bar68. Erwin H. Bareiss. Sylvester's identity and multistep integer-preserving gaussian elimination. *Mathematics of Computation*, 22(103):565–578, 1968.

BGW88. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proc. 20th Symposium on Theory of Computing (STOC '88)*, pages 1–10, New York, 1988. ACM.

BIB89. J. Bar-Ilan and D. Beaver. Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In *Proc. 8th Symp. on Princip. of Distr. Comp.*, pages 201–209, NY, 1989. ACM.

BV19. Niek J. Bouman and Niels de Vreede. A practical approach to the secure computation of the Moore-Penrose pseudoinverse over the rationals. Cryptology ePrint Archive, Report 2019/470, 2019.

CD01. Ronald Cramer and Ivan Damgård. Secure distributed linear algebra in a constant number of rounds. In *Proc. CRYPTO 2001, Santa Barbara, USA*, pages 119–136. Springer, 2001.

DG19. Dheeru Dua and Casey Graff. UCI machine learning repository, 2019.

DHC04. Wenliang Du, Yunghsiang S Han, and Shigang Chen. Privacy-preserving multivariate statistical analysis: Linear regression and classification. In *Proceedings of the 2004 SIAM International Conference on Data Mining*, pages 222–233. SIAM, 2004.

FHT01. Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

Gei10. M. Geisler. *Cryptographic Protocols: Theory and Implementation.* PhD thesis, Department of Computer Science, University of Aarhus, Denmark, February 2010. viff.dk.

GJJ+18. Irene Giacomelli, Somesh Jha, Marc Joye, C. David Page, and Kyonghwan Yoon. Privacy-preserving ridge regression with only linearly-homomorphic encryption. In Bart Preneel and Frederik Vercauteren, editors, *Applied Cryptography and Network Security - 16th International Conference, ACNS 2018, Leuven, Belgium, July 2-4, 2018, Proceedings*, volume 10892 of *Lecture Notes in Computer Science*, pages 243–261. Springer, 2018.

GRR98. R. Gennaro, M. O. Rabin, and T. Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In *17th annual ACM symposium on Principles of Distributed Computing (PODC '98)*, pages 101–111, New York, 1998. ACM.

GSB+17. Adrià Gascón, Phillipp Schoppmann, Borja Balle, Mariana Raykova, Jack Doerner, Samee Zahur, and David Evans. Privacy-preserving distributed linear regression on high-dimensional data. *Proceedings on Privacy Enhancing Technologies*, 2017(4):345–364, 2017.

HFN11. Rob Hall, Stephen E. Fienberg, and Yuval Nardi. Secure multiple linear regression based on homomorphic encryption, 2011.

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 *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13, pages 334–348, Washington, DC, USA, 2013. IEEE Computer Society.

PVG+11. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Sch18. Berry Schoenmakers. MPyC: Secure multiparty computation in Python. GitHub, May 2018. github.com/lschoe/mpyc.