2013 IEEE Symposium on Security and Privacy

Privacy-Preserving Ridge Regression on Hundreds of Millions of Records

Valeria Nikolaenko*, Udi Weinsberg†, Stratis Ioannidis†, Marc Joye†, Dan Boneh*, Nina Taft†
*Stanford University, {valerini, dabo}@cs.stanford.edu
†Technicolor, {udi.weinsberg, stratis.ioannidis, marc.joye, nina.taft}@technicolor.com

***Abstract*—Ridge regression is an algorithm that takes as input a large number of data points and finds the best-fit linear curve through these points. The algorithm is a building block for many machine-learning operations. We present a system for privacy-preserving ridge regression. The system outputs the best-fit curve in the clear, but exposes no other information about the input data. Our approach combines both homomorphic encryption and Yao garbled circuits, where each is used in a different part of the algorithm to obtain the best performance. We implement the complete system and experiment with it on real data-sets, and show that it significantly outperforms pure implementations based only on homomorphic encryption or Yao circuits.**

I. Introduction

Recommendation systems operate by collecting the preferences and ratings of many users for different items and running a learning algorithm on the data. The learning algorithm generates a model that can be used to predict how a new user will rate certain items. In particular, given the ratings that a user provides on certain items, the model can predict how that user will rate other items. There is a vast array of algorithms for generating such predictive models and many are actively used at large sites like Amazon and Netflix. Learning algorithms are also used on large medical databases, financial data, and many other domains.

In current implementations, the learning algorithm must see all user data *in the clear* in order to build the predictive model. In this paper we ask whether the learning algorithm can operate without the data in the clear, thereby allowing users to retain control of their data. For medical data this allows for a model to be built without affecting user privacy. For books and movie preferences letting users keep control of their data reduces the risk of future unexpected embarrassment in case of a data breach at the service provider.

Roughly speaking, there are three existing approaches to data-mining private user data. The first lets users split their data among multiple servers using secret sharing. These servers then run the learning algorithm using a distributed protocol such as BGW [1] and privacy is assured as long as a majority of servers do not collude. The second is based on homomorphic encryption where the learning algorithm is executed over encrypted data [2] and a trusted third party is trusted to only decrypt the final encrypted model. The third approach is similar to the second, but instead of homomorphic encryption one uses Yao's garbled circuit construction [3] to obtain the final model without learning anything else about user data (we further expand on these in Section VII). A fourth approach uses trusted hardware at the service provider, but we do not consider that here.

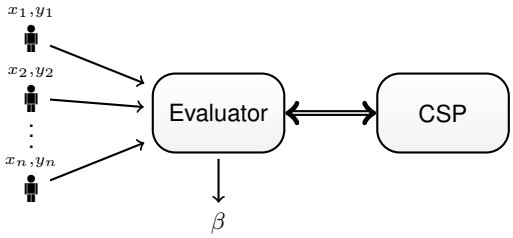


Figure 1: The parties in our system. The Evaluator learns β, the model describing the best linear curve fit to the data $(x_i, y_i), i = 1, \ldots, n$, without seeing the data in the clear.

In this paper we focus on a fundamental mechanism used in many learning algorithms called *ridge regression*. Given a large number of points in high dimension, the regression algorithm (described in Section III-A) produces a best-fit linear curve through these points. Our goal is to perform this computation without exposing any other information about user data. We target a system as shown in Figure 1: Users send their encrypted data to a party called the *Evaluator* who runs the learning algorithm. At certain points the Evaluator may interact with a Crypto Service Provider (CSP), who is trusted not to collude with the Evaluator. The final outcome is the cleartext predictive model.

Our contributions. We present a hybrid approach to privacy-preserving ridge regression that uses both homomorphic encryption and Yao garbled circuits. We separate the regression algorithm into two phases, presented in detail in Section IV. Users submit their data encrypted under a linearly homomorphic encryption system such as Paillier [4] or Regev [5]. The Evaluator uses the linear homomorphism to carry out the first phase of the regression algorithm, which requires only linear operations. This phase generates encrypted data, but the amount of data generated is independent of the number of users n. Consequently, when the system is asked to process a large number of records the bulk of the processing happens in this first phase, while the remainder of the algorithm is independent of n. In the second phase the Evaluator evaluates a Yao garbled circuit

DOI 10.1109/SP.2013.30

that first decrypts the ciphertexts from the first phase and then executes the remaining operations of the regression algorithm (we also show an optimized realization that avoids decryption inside the garbled circuit). This second phase of the regression algorithm requires a fast linear system solver and is highly non-linear. For this step a Yao garbled circuit approach is much faster than current homomorphic encryption schemes. We thus obtain the best of both worlds using linear homomorphisms to handle a large data set and using garbled circuits for heavy non-linear computations.

We show that this approach has many desirable properties. Because we remove the dependence on n from the regression computation, our system runs regression on hundreds of millions of user records in a reasonable time. Also, in this system users can be offline, meaning that they can send encrypted data to the Evaluator whenever they desire, and are not required to stay online to participate.

Our second contribution is an implementation of a complete privacy-preserving regression system, including a Yao garbled circuit for a fast linear system solver based on Cholesky decomposition. Our implementation uses batched Paillier as the additively homomorphic system and uses FastGC [6] as the underlying Yao framework. We extended FastGC with an arithmetic library that supports fixed-point division, square roots, and other operations. We built a real prototype in which the parties communicate over a network and garbled circuits are properly managed in memory.

Our third contribution is an extensive evaluation of the system using real datasets provided by UCI [7]. We demonstrate that the benefits of our hybrid system, compared to a pure Yao implementation, are substantial resulting in execution times that are reduced from days to hours. It also performs substantially better than a recent implementation combining homomorphic encryption with secret-sharing due to Hall, Fienberg, and Nardi [8]. Their system required *two days* to compute the linear regression of roughly 51K input vectors, each with 22 features. Our hybrid system performs a similar computation in 3 minutes. For 100 million user records each with 20 features, our system takes 8.75 hours.

Overall, our experiments show that for specific privacy-preserving computations, a hybrid approach combining both homomorphic encryption and garbled circuits can perform significantly better than either method alone.

II. Settings and Threat Model

A. Architecture and Entities

Our system is designed for many users to contribute data to a central server called the *Evaluator*. The Evaluator performs *regression* over the contributed data and produces a *model*, which can later be used for prediction or recommendation tasks. More specifically, each user $i = 1, \ldots, n$ has a private record comprising two variables $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$, and the Evaluator wishes to compute a $\beta \in \mathbb{R}^d$ — the model — such that $y_i \simeq \beta^T x_i$. Our goal is to ensure that the Evaluator learns nothing about the user's records beyond what is revealed by β, the final result of the regression algorithm. To initialize the system we will need a third party, which we call a "Crypto Service Provider", that does most of its work offline.

More precisely, the parties in the system are the following (shown in Figure 1):

- Users: each user i has private data x_i, y_i that it sends encrypted to the Evaluator.
- Evaluator: runs a regression algorithm on the encrypted data and obtains the learned model β in the clear.
- Crypto Service Provider (CSP): initializes the system by giving setup parameters to the users and the Evaluator. The CSP does most of its work offline before users contribute data to the Evaluator. In our most efficient design, the CSP is also needed for a short one-round online step when the Evaluator computes the model.

B. Threat Model

Our goal is to ensure that the Evaluator and the CSP cannot learn anything about the data contributed by users beyond what is revealed by the final results of the learning algorithm. In the case that the Evaluator colludes with some of the users, they should learn nothing about the data contributed by other users beyond what is revealed by the results of the learning algorithm.

In our setting, we assume that it is the Evaluator's best interest to produce a correct model. Hence, we are not concerned with a malicious Evaluator who is trying to corrupt the computation in the hope of producing an incorrect result. However, the Evaluator is motivated to misbehave and learn information about private data contributed by the users since this data can potentially be sold to other parties, *e.g.*, advertisers. Therefore, even a malicious Evaluator should be unable to learn anything about user data beyond what is revealed by the results of the learning algorithm. In Section IV we describe the basic protocol which is only secure against an honest-but-curious Evaluator. We then explain in Section IV-G how to extend the protocol to ensure privacy against a malicious Evaluator.

Similarly, a malicious CSP should learn nothing about user data and should be unable to disrupt the Evaluator's computation. We show that the Evaluator can efficiently verify that the result of the computation is correct, with high probability, and if the results are correct then privacy against a malicious CSP is ensured. Hence, a malicious CSP will be caught if it tries to corrupt the computation. Informally, we allow the CSP to act as a *covert* adversary in the sense of [9] (see also [10]). Our approach to checking the result of the computation makes use of the specific structure of the regression problem the Evaluator is solving.

Non-threats: Our system is not designed to defend against the following attacks:

- We assume that the Evaluator and the CSP do not collude. Each one may try to subvert the system as discussed above, but they do so independently. More precisely, at most one of these two parties is malicious: this is an inherent requirement without which security cannot be achieved.
- We assume that the setup works correctly, that is all users obtain the correct public key from the CSP. This can be enforced in practice with appropriate use of Certificate Authorities.

III. Background

A. Learning a Linear Model

We briefly review ridge regression, the algorithm that the evaluator executes to learn β. All results discussed below are classic, and can be found in most statistics and machine learning textbooks (*e.g.*, [11]).

Linear Regression: Given a set of n input variables $x_i \in \mathbb{R}^d$, and a set of output variables $y_i \in \mathbb{R}$, the problem of learning a function $f : \mathbb{R}^d \to \mathbb{R}$ such that $y_i \simeq f(x_i)$ is known as *regression*. For example, the input variables could be a person's age, weight, body mass index, *etc.*, while the output can be their likelihood to contract a disease.

Learning such a function from real data has many interesting applications, that make regression ubiquitous in data mining, statistics, and machine learning. On one hand, the function itself can be used for *prediction*, *i.e.*, to predict the output value y of a new input $x \in \mathbb{R}^d$. Moreover, the structure of f can aid in identifying how different inputs affect the output—establishing, *e.g.*, that weight, rather than age, is more strongly correlated to a disease.

Linear regression is based on the premise that f is well approximated by a linear map,[1] *i.e.*,

$$y_i \simeq \beta^T x_i\,, \qquad i \in [n] \equiv \{1, \ldots, n\}$$

for some $\beta \in \mathbb{R}^d$. Linear regression is one of the most widely used methods for inference and statistical analysis in the sciences. In addition, it is a fundamental building block for several more advanced methods in statistical analysis and machine learning, such as kernel methods. For example, learning a function that is a polynomial of degree 2 reduces to linear regression over $x_{ik}x_{ik'}$, for $1 \le k, k' \le d$; the same principle can be generalized to learn any function spanned by a finite set of basis functions.

As mentioned above, beyond its obvious uses for prediction, the vector $\beta = (\beta_k)_{k=1,\ldots,d}$ is interesting as it reveals how y depends on the input variables. In particular, the sign of a coefficient β_k indicates either positive or negative correlation to the output, while the magnitude captures relative importance. To ensure these coefficients are comparable, but also for numerical stability, the inputs x_i are rescaled to the same, finite domain (*e.g.*, $[-1, 1]$).

[1] Affine maps can also be captured through linear regression, by appending a 1 to each of the inputs.

Computing the Coefficients: To compute the vector $\beta \in \mathbb{R}^d$, the latter is fit to the data by minimizing the following quadratic function over $\mathbb{R}^d$:

$$F(\beta) = \sum_{i=1}^{n} (y_i - \beta^T x_i)^2 + \lambda \|\beta\|_2^2 \ . \tag{1}$$

The procedure of minimizing (1) is called *ridge regression*; the objective $F(\beta)$ incorporates a penalty term $\lambda\|\beta\|_2^2$, which favors parsimonious solutions. Intuitively, for $\lambda = 0$, minimizing (1) corresponds to solving a simple least squares problem. For positive $\lambda > 0$, the term $\lambda\|\beta\|_2^2$ penalizes solutions with high norm: between two solutions that fit the data equally, one with fewer large coefficients is preferable. Recalling that the coefficients of β are indicators of how input affects output, this acts as a form of "Occam's razor": simpler solutions, with few large coefficients, are preferable. Indeed, a $\lambda > 0$ gives in practice better predictions over new inputs than the least squares solution based.

Let $y \in \mathbb{R}^n$ be the vector of outputs and $X \in \mathbb{R}^{n \times d}$ be a matrix comprising the input vectors, one in each row: $y = (y_i)_{i=1,\ldots,n}$ and $X = (x_i^T)_{i=1,\ldots,n}$; *i.e.*,

$$y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \quad \text{and} \quad X = \begin{pmatrix} x_{11} & x_{12} & \ldots & x_{1d} \\ x_{21} & x_{22} & \ldots & x_{2d} \\ \vdots & \vdots & & \vdots \\ x_{n1} & x_{n2} & \ldots & x_{nd} \end{pmatrix} .$$

The minimizer of (1) can be computed by solving the linear system

$$A\beta = b \tag{2}$$

where $A = X^T X + \lambda I \in \mathbb{R}^{d \times d}$ and $b = X^T y \in \mathbb{R}^d$. For $\lambda > 0$, the matrix A is symmetric positive definite, and an efficient solution can be found using the Cholesky decomposition, as outlined in Section V.

B. Yao's Garbled Circuits

In its basic version, Yao's protocol (a.k.a. *garbled circuits*) [3] (see also [12]) allows the two-party evaluation of a function $f(x_1, x_2)$ in the presence of semi-honest adversaries. The protocol is run between the input owners (a_i denotes the private input of user i). At the end of the protocol, the value of $f(a_1, a_2)$ is obtained but no party learns more than what is revealed from this output value.

The protocol goes as follows. The first party, called *garbler*, builds a "garbled" version of a circuit computing f. It then gives to the second party, called *evaluator*, the garbled circuit as well as the garbled-circuit input values that correspond to a_1 (and only those ones). We use the notation $\mathrm{GI}(a_1)$ to denote these input values. It also provides the mapping between the garbled-circuit output values and the actual bit values. Upon receiving the circuit, the evaluator engages in a 1-out-of-2 oblivious transfer protocol [13], [14] with the garbler, playing the role of the chooser, so as to obliviously obtain the garbled-circuit input values

corresponding to its private input a_2, $\mathrm{GI}(a_2)$. From $\mathrm{GI}(a_1)$ and $\mathrm{GI}(a_2)$, the evaluator can therefore calculate $f(a_1, a_2)$.

In more detail, the protocol evaluates the function f through a Boolean circuit. To each wire w_i of the circuit, the garbler associates two random cryptographic keys, $K^0_{w_i}$ and $K^1_{w_i}$, that respectively correspond to the bit-values $b_i = 0$ and $b_i = 1$. Next, for each binary gate g (*e.g.*, an OR-gate) with input wires (w_i, w_j) and output wire w_k, the garbler computes the four ciphertexts

$$\mathrm{Enc}_{(K^{b_i}_{w_i}, K^{b_j}_{w_j})}(K^{g(b_i,b_j)}_{w_k}) \quad \text{for } b_i, b_j \in \{0,1\} \ .$$

The set of these four *randomly ordered* ciphertexts defines the garbled gate.

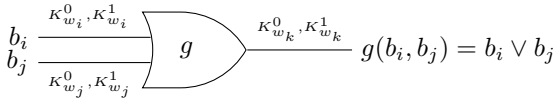


Figure 2: Example of a garbled OR-gate

It is required that the symmetric encryption algorithm Enc, which is keyed by a pair of keys, has indistinguishable encryptions under chosen-plaintext attacks. It is also required that given the pair of keys $(K^{b_i}_{w_i}, K^{b_j}_{w_j})$, the corresponding decryption process unambiguously recovers the value of $K^{g(b_i,b_j)}_{w_k}$ from the four ciphertexts constituting the garbled gate; see *e.g.* [15] for an efficient implementation. It is worth noting that the knowledge of $(K^{b_i}_{w_i}, K^{b_j}_{w_j})$ yields only the value of $K^{g(b_i,b_j)}_{w_k}$ and that no other output values can be recovered for this gate. So the evaluator can evaluate the entire garbled circuit gate-by-gate so that no additional information leaks about intermediate computations.

IV. Our hybrid approach

Recall that, in our setup, each input and output variable x_i, y_i, $i \in [n]$, is private, and held by a different user. The Evaluator wishes to learn the β determining the linear relationship between the input and output variables, as obtained through ridge regression with a given $\lambda > 0$.

As described in Section III-A, to obtain β, one needs the matrix $A \in \mathbb{R}^{d\times d}$ and the vector $b \in \mathbb{R}^d$, as defined in (2). Once these values are obtained, the Evaluator can solve the linear system (2) and extract β. There are several ways to tackle this problem in a privacy-preserving fashion. One option is to use homomorphic encryption. Another is to use Yao garbled circuits.

The naïve way to use Yao's approach is to design a monolithic circuit with inputs x_i, y_i, for $i \in [n]$ and $\lambda > 0$. The circuit first computes the matrices A and b and then solves the system $A\beta = b$. The Evaluator evaluates this large garbled circuit using evaluation keys supplied by the n users. A similar approach has been used for other tasks such as sealed second price auctions [16]. Putting implementation issues aside (such as how to design a circuit that solves a linear system), a major shortcoming of such a solution in our setting is that the size of resulting garbled circuit depends on both the number of users n, as well as the dimension d of β. In practical applications it is common that n is large, in the order of millions of users. In contrast, d is relatively small, in the order of 10s. To obtain a solution that can scale to million of users it is therefore preferable that the size of the circuit does not depend on n. To this end, we reformulate the problem as discussed below.

A. Reformulating the Problem

We note that the matrix A and vector b can be computed in an iterative fashion as follows. Recalling that each pair x_i, y_i is held by a distinct user, each user i can locally compute the matrix $A_i = x_i x_i^T$ and the vector $b_i = y_i x_i$. It is then easily verified that summing these partial contributions yields:

$$A = \sum_{i=1}^{n} A_i + \lambda I \quad \text{and} \quad b = \sum_{i=1}^{n} b_i \ . \tag{3}$$

Equation (3) importantly shows that A and b are the result of a series of additions. The Evaluator's regression task can therefore be separated into two subtasks: (a) collecting the A_i's and b_i's, to construct matrix A and vector b, and (b) using these to obtain β by solving the linear system (2).

Of course, the users cannot send their local shares, (A_i, b_i), to the Evaluator in the clear. However, if the latter are encrypted using a public-key *additive* homomorphic encryption scheme, then the Evaluator can reconstruct the encryptions of A and b from the encryptions of the (A_i, b_i)'s. The remaining challenge is to solve (2), with the help of the CSP, without revealing (to the Evaluator or the CSP) any additional information other than β; we describe two distinct ways of doing so through Yao's garbled circuits below.

More explicitly, let

$$\mathfrak{E}_{\mathsf{pk}} : (A_i; b_i) \in \mathcal{M} \mapsto \mathfrak{c}_i = \mathfrak{E}_{\mathsf{pk}}(A_i; b_i)$$

be a semantically secure encryption scheme indexed by a public key pk that takes as input a pair $(A_i; b_i)$ in the message space $\mathcal{M}$ and returns the encryption of $(A_i; b_i)$ under pk, $\mathfrak{c}_i$. Then it must hold for any pk and any two pairs $(A_i; b_i)$, $(A_j; b_j)$, that

$$\mathfrak{E}_{\mathsf{pk}}(A_i; b_i) \otimes \mathfrak{E}_{\mathsf{pk}}(A_j; b_j) = \mathfrak{E}_{\mathsf{pk}}(A_i + A_j; b_i + b_j)$$

for some public binary operator $\otimes$. Such an encryption scheme can be constructed from any semantically secure additive homomorphic encryption scheme by encrypting component-wise the entries of A_i and b_i. Examples include Regev's scheme [5] and Paillier's scheme [4].

We refer to the phase of aggregating the user shares as *Phase 1*, and note that the addition it involves depends linearly in n. We refer to the subsequent phase, which amounts to computing the solution to Eq. (2) from the

encrypted values of A and b, as *Phase 2*. We note that Phase 2 has *no dependence* on n.

We are now ready to present our protocols. A high-level description is provided in Fig. 3. Note that we assume below the existence of a circuit that can solve the system $A\beta = b$; we present an implementation of such a circuit in Section V.

B. First Protocol

Our first protocol operates as follows. It comprises three phases: a preparation phase, Phase 1, and Phase 2. As will become apparent (see the discussion below in Section IV-D), only Phase 2 really requires an on-line treatment.

Preparation phase. The Evaluator provides the specifications to the CSP, such as the dimension d of the input variables, their value range, and the number of bits used to represent the integer and fractional parts of a number. The CSP prepares a Yao garbled circuit to be used in Phase 2 and makes it available to the Evaluator. The CSP also generates a public key pk_{csp} and a private key sk_{csp} for the homomorphic encryption scheme $\mathfrak{E}$.

Phase 1. Each user i locally computes her partial matrix A_i and vector b_i. She then encrypts these values using additive homomorphic encryption scheme $\mathfrak{E}$ under the public encryption key pk_{csp} of the CSP; *i.e.*,

$$\mathfrak{c}_i = \mathfrak{E}_{\mathsf{pk}_{\text{csp}}}(A_i; b_i) \ .$$

The user i then sends $\mathfrak{c}_i$ to the Evaluator. (We assume the existence of a secure channel between the users and the Evaluator—*e.g.*, through the standard Transport Layer Security (TLS) protocol.)

The Evaluator computes $\mathfrak{c}_\lambda = \mathfrak{E}_{\mathsf{pk}_{\text{csp}}}(\lambda I; \mathbf{0})$. It then aggregates it to all received $\mathfrak{c}_i$'s and gets:

$$\mathfrak{c} = \Big(\bigotimes_{i=1}^{n} \mathfrak{c}_i\Big) \otimes \mathfrak{c}_\lambda = \mathfrak{E}_{\mathsf{pk}_{\text{csp}}}\big(\textstyle\sum_{i=1}^n A_i + \lambda I; \sum_{i=1}^n b_i\big)$$
$$= \mathfrak{E}_{\mathsf{pk}_{\text{csp}}}(A; b) \ . \qquad (4)$$

Phase 2. The garbled circuit provided by the CSP in the preparation phase is a garbling of a circuit that takes as input $\text{GI}(\mathfrak{c})$ and does the following two steps:

1) decrypting $\mathfrak{c}$ with sk_{csp} to recover A and b (here sk_{csp} is embedded in the garbled circuit);
2) solving Eq. (2) and returning β.

In this Phase 2, the Evaluator need only to obtain the garbled-circuit input values corresponding to $\mathfrak{c}$; *i.e.*, $\text{GI}(\mathfrak{c})$. These are obtained using a standard oblivious transfer between the Evaluator and the CSP.

The above hybrid computation decrypts the encrypted inputs within the garbled circuit. As this can be demanding, we suggest to use for example Regev homomorphic encryption [5] as the building block for $\mathfrak{E}$ since the Regev scheme has a very simple decryption circuit.

C. Second Protocol

We now present a modification that avoids decrypting $(A; b)$ in the garbled circuit using random masks. Phase 1 remains broadly the same. We will highlight the Phase 2 (and the corresponding preparation phase).

The idea is to exploit the homomorphic property to obscure the inputs with an additive mask. Note that if $(\mu_A; \mu_b)$ is an element in $\mathcal{M}$ (namely, the message space of homomorphic encryption $\mathfrak{E}$) then it follows from (4) that

$$\mathfrak{c} \otimes \mathfrak{E}_{\mathsf{pk}_{\text{csp}}}(\mu_A; \mu_b) = \mathfrak{E}_{\mathsf{pk}_{\text{csp}}}(A + \mu_A; b + \mu_b) \ .$$

Hence, assume that the Evaluator chooses a random mask $(\mu_A; \mu_b)$ in $\mathcal{M}$, obscures $\mathfrak{c}$ as above, and sends the resulting value to the CSP. Then, the CSP can apply its decryption key and recover the masked values

$$\hat{A} = A + \mu_A \qquad \text{and} \qquad \hat{b} = b + \mu_b \ .$$

As a consequence, we can apply the protocol of the previous section where the decryption is replaced by the removal of the mask. In more detail, we have:

Preparation phase. As before, the Evaluator sets up the evaluation. It provides the specifications to the CSP to build a garbled circuit supporting its evaluation. The CSP prepares the circuit and makes it available to the Evaluator, and generates public and private keys.

The Evaluator chooses a random mask $(\mu_A; \mu_b) \in \mathcal{M}$ and engages in an oblivious transfer protocol with the CSP to get the garbled-circuit input values corresponding to $(\mu_A; \mu_b)$; *i.e.*, $\text{GI}(\mu_A; \mu_b)$.

Phase 1. This is the same as in our first realization. In addition, the Evaluator masks $\mathfrak{c}$ as

$$\hat{\mathfrak{c}} = \mathfrak{c} \otimes \mathfrak{E}_{\mathsf{pk}_{\text{csp}}}(\mu_A; \mu_b) \ .$$

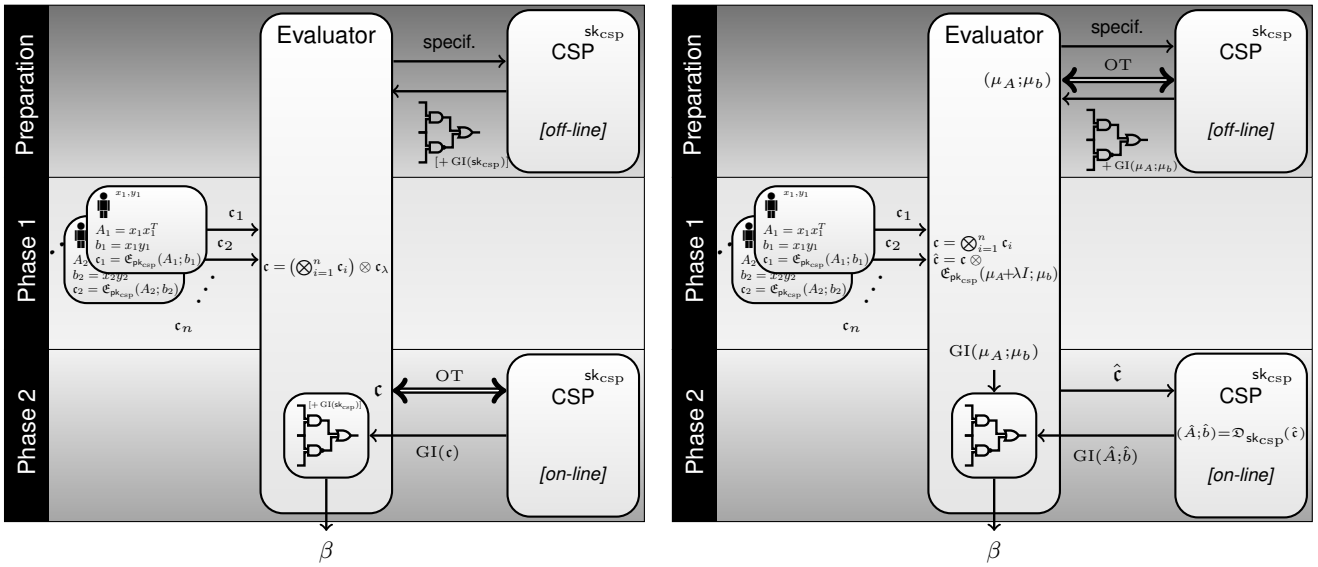
Phase 2. The Evaluator sends $\hat{\mathfrak{c}}$ to the CSP that decrypts it to obtain $(\hat{A}; \hat{b})$ in the clear. The CSP then sends the garbled input values $\text{GI}(\hat{A}; \hat{b})$ back to the Evaluator.

The garbled circuit provided by the CSP in the preparation phase is a garbling of a circuit that takes as input $\text{GI}(\hat{A}; \hat{b})$ and $\text{GI}(\mu_A; \mu_b)$ and:

1) subtracts the mask $(\mu_A; \mu_b)$ from $(\hat{A}; \hat{b})$ to recover A and b;
2) solves Eq. (2) and returns β.

The garbled circuit as well as the garbled-circuit input values corresponding to $(\mu_A; \mu_b)$, $\text{GI}(\mu_A; \mu_b)$, were obtained during the preparation phase. In this phase, the Evaluator need only receive from the CSP the garbled-circuit input values corresponding to $(\hat{A}; \hat{b})$, $\text{GI}(\hat{A}; \hat{b})$. We note that there is no oblivious transfer in this phase.

For this second realization, the decryption is not executed as part of the circuit. We are thus not restricted to homomorphic encryption schemes that can be efficiently implemented as a circuit. Instead of Regev's scheme, we suggest Paillier's scheme [4] or its generalization by Damgård and Jurik [17]



(a) First protocol

(b) Second protocol

Figure 3: High-level description of the two protocols. The first protocol decrypts the data inside the garbled circuit while the second avoid decrypting in the circuit by using random masks.

as the building block for $\mathfrak{E}$. These schemes have a shorter ciphertext expansion than Regev and require smaller keys.

D. Discussion

The proposed protocols have several strengths that make them efficient and practical in real-world scenarios. First, there is no need for users to stay on-line during the process. Since Phase 1 is incremental, each user can submit their encrypted inputs, and leave the system.

Furthermore, the system can be easily applied to performing ridge regression multiple times. Assuming that the Evaluator wishes to perform ℓ estimations, it can retrieve ℓ garbled circuits from the CSP during the preparation phase.[2] Multiple estimations can be used to accommodate the arrival of new users. In particular, since the public keys are long-lived, they do not need to be refreshed too often, meaning that when new users submit more pairs (A_i, b_i) to the Evaluator, the latter can sum them with the prior values and compute an updated β. Although this process requires utilizing a new garbled circuit, the users that have already submitted their inputs do not need to resubmit them.

Finally, significantly less communication is required than in a secret sharing scheme, and only the Evaluator and the CSP communicate using oblivious transfer.

E. Further Optimizations

Recall that the matrix A is in $\mathbb{R}^{d\times d}$ and the vector b is in $\mathbb{R}^d$. Hence letting k denote the bit-size used to encode real numbers, the matrix A and vector b respectively need d^2k bits and dk bits for their representation. Our second protocol requires a random mask $(\mu_A; \mu_b)$ in $\mathcal{M}$. Suppose that the homomorphic encryption scheme $\mathfrak{E}$ was built on top of Paillier's scheme where every entry of A and of b is individually Paillier encrypted. In this case the message space $\mathcal{M}$ of $\mathfrak{E}$ is composed of $(d^2 + d)$ elements in $\mathbb{Z}/N\mathbb{Z}$ for some Paillier modulus N. But as those elements are k-bit values there is no need to draw the corresponding masking values in the whole range $\mathbb{Z}/N\mathbb{Z}$. Any $(k+l)$-bit values for some (relatively short) security length l will do, as long as they statistically hide the corresponding entry. In practice, this leads to fewer oblivious transfers in the preparation phase and to a smaller garbled circuit.

Another way to improve the efficiency is via a standard batching technique, that is packing multiple plaintext entries of A and b into a single Paillier ciphertext. For example, packing 20 plaintext values into a single Paillier ciphertext (separated by sufficiently many 0's) will reduce the running time of Phase 1 by a factor of 20.

F. Malicious CSP: Checking the Final Result

A malicious CSP can corrupt the computation in two ways: it can contribute a garbled circuit for the wrong function and it can incorrectly decrypt the Paillier ciphertexts sent to it by the Evaluator. The second issue can be easily handled by standard techniques that require the CSP to efficiently prove in zero knowledge to the Evaluator that Paillier decryption was done correctly.

The first issue — proving that the garbled circuit computes

[2]Note that, in such a case, our security requirement is that the Evaluator learns only β_ℓ for each ℓ; in particular, we are not concerned with how informative such sequence might be about the x_i's and y_i's of each user.

the correct function — is more interesting. Observe that the correct β is a minimizer of the quadratic function $F(\beta)$ given by (1). As such, the gradient

$$\nabla F = 2\sum_{i=1}^{n}(x_i x_i^T \beta - y_i x_i) + 2\lambda\beta = 2(A\beta - b)$$

must attain a value close to zero at the correct β.

Recall that, at the completion of Phase 1 (in both protocols presented above), the Evaluator has access to $\mathfrak{E}_{\mathsf{pk}_{\mathrm{csp}}}(A; b)$, obtained by applying an additive homomorphic encryption scheme componentwise on A and b. Having obtained β in the clear at the termination of Phase 2, observe that the computation of the d-dimensional gradient vector $\delta = 2(A\beta - b)$ involves only linear operations. Therefore the Evaluator can compute $\mathfrak{E}_{\mathsf{pk}_{\mathrm{csp}}}(\delta)$ from $\mathfrak{E}_{\mathsf{pk}_{\mathrm{csp}}}(A; b)$. Moreover, if β is correct then the norm $\|\delta\|$ must be small.

These observations lead us to the following abstract problem: the Evaluator has a ciphertext $\mathfrak{c} = \mathfrak{E}_{\mathsf{pk}_{\mathrm{csp}}}(x)$ and it wants to convince itself that x is small, say in the range $[-u, u]$. It may communicate with the CSP to do so. The Evaluator will perform this operation for all the components of the vector δ defined above to convince itself that the norm $\|\delta\|$ is small as required. We stress that, as explained in Section II-B, here the CSP may be malicious, but the Evaluator is honestly following the protocol. We discuss a malicious Evaluator in the next subsection.

The protocol works as follows: first the Evaluator selects two random masks $\mu^{(1)}$ and $\mu^{(2)}$ in the plaintext space and constructs the ciphertext

$$\mathfrak{c} = \mathfrak{E}_{\mathsf{pk}_{\mathrm{csp}}}\left(x \cdot \mu^{(1)} + \mu^{(2)}\right)$$

Next, the Evaluator sends $\mathfrak{c}$ to the CSP who decrypts it and sends back $\hat{z} = \mathfrak{D}_{\mathsf{sk}_{\mathrm{csp}}}(\mathfrak{c})$. Finally, the Evaluator subtracts the mask $\mu^{(2)}$ from $\hat{z}$ and divides by $\mu^{(1)}$. If the resulting value is in the range $[-u, u]$ the Evaluator is convinced that x is in $[-u, u]$. Otherwise, the Evaluator rejects β. This step reveals ∇F to the Evaluator, but since it is low-norm and its dimension d is independent of the number of users, revealing this information to the Evaluator seems acceptable.

The malicious CSP cannot cause a large x to be accepted by the Evaluator in the above protocol, even if the CSP knows x. To see why observe that the CSP is given $z = x \cdot \mu^{(1)} + \mu^{(2)}$ and we even allow it to have x. Recall that the function family $x \to \mu^{(1)}x + \mu^{(2)} \bmod N$ is ϵ-pair wise independent for negligible ϵ (here N is the Paillier modulus). Therefore, if the CSP responds with a $\hat{z} \neq z$, the quantity $(\hat{z} - \mu^{(2)})/\mu^{(1)}$ will be close to uniform in $\mathbb{Z}/N\mathbb{Z}$ in the CSP's view and will be in $[-u, u]$ with probability about $2u/N$, which is negligible. Hence, if the CSP responds with $\hat{z} \neq z$, the Evaluator rejects β with high probability.

G. Malicious Evaluator

So far the protocols described in this section are only secure when the Evaluator honestly follows the protocol (*i.e.*, it is honest-but-curious). The Evaluator, however, is motivated to misbehave and potentially learn the cleartext data provided by the users. This would violate user privacy since our goal is that only the final learned model (*i.e.*, β) would become public. One danger, for example, is that the Evaluator ignores many of the values provided by the users and simply runs the regression on the data provided by a small subset of the users, or even on the data provided by just a single user. The resulting output could potentially leak or even completely reveal that user's data.

We briefly outline an approach to preventing the Evaluator from misbehaving by having the CSP validate the Evaluator's work in zero knowledge. In particular, we use homomorphic commitments to force the Evaluator to convince the CSP that the Paillier ciphertexts the Evaluator sent are exactly the product of the Paillier ciphertexts the Evaluator received from all users (and multiplied by a mask known to the Evaluator). The CSP will decrypt these Paillier ciphertexts and send the result to the Evaluator only once it is convinced they were constructed correctly and, in particular, include all the data contributed by an approved list of users. The CSP should learn nothing new from these proofs.

The commitment scheme we use is based on Pedersen commitments [18] and we use standard zero-knowledge protocols to prove modular arithmetic relations between the committed values [19], [20] as well as conjunctions of such arithmetic relations [21]. These protocols are made non-interactive using the standard Fiat-Shamir heuristic in the random oracle model [22].

At a high level our approach is to have the CSP share different random one-time MAC keys with each of the users. We use simple one-time MACs based on a linear pair-wise independent hash as described below. When user i sends a Paillier ciphertext to the Evaluator she also includes a Pedersen commitment to the one-time MAC on that ciphertext along with a short zero-knowledge proof that the opening of this commitment is a valid one-time MAC on the Paillier ciphertext. The Evaluator collects all the ciphertexts and proofs from the users and forms the masked product c of the users' Paillier ciphertexts. It also constructs a zero-knowledge proof that c is indeed the product of these ciphertexts. The Evaluator submits the ciphertext c along with all the commitments and proofs to the CSP. By validating the proofs, the CSP learns that c is indeed the masked product of ciphertexts received from the specified list of users. The one-time MACs ensure that the Evaluator did not modify or drop any of the values received from the users. If the CSP approves the list of users, it decrypts the masked Paillier ciphertext and sends the result back to the Evaluator as in Figure 3.

We stress that the additional data sent from a single user to the Evaluator, *i.e.*, the additional commitments and proofs, is relatively short thanks to the simple one-time MAC we use. Nevertheless, all this data now needs to be forwarded to

the CSP. Consequently, the amount of data sent to the CSP is now linear in the number of users n. In the honest-but-curious protocol from Figure 3 the data sent to the CSP was independent of the number of users. This increase seems inherent since the CSP must be told which users submitted their data to the Evaluator so that it can decide whether running the data-mining algorithm on that set of users is acceptable.

Recall that Paillier ciphertexts are elements in the group $(\mathbb{Z}/N^2\mathbb{Z})^*$ where N is an RSA modulus. To construct Pedersen commitments to such values we use a prime $p = 1 \bmod N^2$ and elements $g, h \in \mathbb{F}_p^*$ of order N^2. The Pedersen commitment to a Paillier ciphertext c is then $\text{Com} = g^c h^r \in \mathbb{F}_p^*$ where $0 \leq r < N^2$ is chosen at random. This way we can prove arithmetic relations on committed values where the arithmetic is modulo N^2. Moreover, the one-time MACs we use are built from linear hash functions $H(x) = ax + b$ where the MAC signing key is the pair of random values (a, b) in $\mathbb{Z}/N^2\mathbb{Z}$. This hash family is a collection of ϵ-pair-wise independent functions for negligible ϵ and therefore forms a secure one-time MAC. Now, suppose we have a Pedersen commitment Com_c to a Paillier ciphertext c and a Pedersen commitment Com_t to its MAC $t = ac + b \in \mathbb{Z}/N^2\mathbb{Z}$. Proving to the CSP that the opening of Com_t is a valid MAC on the opening of Com_c is a simple proof of arithmetic relations.

Finally, recall that to defend against a malicious CSP we added in Section IV-F one more round between the Evaluator and the CSP where the Evaluator submits a ciphertext $\mathfrak{c}$ and the CSP decrypts it. To prevent a malicious Evaluator from abusing this step, we would also augment it with a proof that this $\mathfrak{c}$ is constructed correctly. The Evaluator has all the data it needs to construct this proof. In addition, it would prove to the CSP that it correctly evaluated the garbled circuit, but this is straight forward using the inherent integrity properties of Yao garbled circuits [23].

V. Implementation

To assess the practicality of our privacy-preserving system, we implemented and tested it on both synthetic and real datasets. We implemented the second protocol proposed in Section IV, as it does not require decryption within the garbled circuit, and allows for the use of homomorphic encryption that is efficient for Phase 1 (that only involves summation). We assumed in our implementation the honest-but-curious threat model, and as such, we did not implement the extensions for a malicious Evaluator or CSP discussed in Sections IV-F and IV-G.

A. Phase 1 Implementation

As discussed in Section IV, for homomorphic encryption we opted to use Paillier's scheme with a 1024-bit modulus, which corresponds to a 80-bit security level.

To speed up Phase 1, we also implemented batching as outlined in Section IV-E. Given n users that contribute their inputs, the number of elements that can be batched into one Paillier ciphertext of 1024 bits is $1024/(b + \log_2 n)$, where b is the total number of bits for representing numbers. As we discuss later, b is determined as a function of the desired accuracy, thus in our experiments, we manage to batch between 15 and 30 elements.

B. Circuit Garbling Framework

We built our system on top of FastGC [6], a Java-based open-source framework that enables developers to define arbitrary circuits using elementary XOR, OR and AND gates. Once the circuits are constructed, the framework handles garbling, oblivious transfer and the complete evaluation of the garbled circuit. The security level that FastGC provides is 80-bits which matches the security level of the homomorphic encryption scheme that we are using.

FastGC includes several optimizations. First, the communication and computation cost for XOR gates in the circuit is significantly reduced using the "free XOR" technique [24]. Second, using the garbled-row reduction technique [25], FastGC reduces the communication cost for k-fan-in non-XOR gates by $1/2^k$, which gives a 25% communication saving, since only 2-fan-in gates are defined in the framework. Third, FastGC implements the OT extension [26] which can execute a practically unlimited number of transfers at the cost of k OTs and several symmetric-key operations per additional OT. Finally, the last optimization is the succinct "addition of 3 bits" circuit [27], which defines a circuit with four XOR gates (all of which are "free" in terms of communication and computation) and just one AND gate.

FastGC enables the garbling and evaluation to take place concurrently. The CSP transmits the garbled tables to the Evaluator as they are produced in the order defined by circuit structure. The Evaluator then determines which gate to evaluate next based on the available output values and tables. Once a gate was evaluated its corresponding table is immediately discarded. This amounts to the same computation and communication costs as precomputing all garbled circuits off-line, and also leaks , but brings memory consumption to a constant.

Beyond FastGC, there are several other frameworks for designing and generating Yao's garbled circuits. The first such implementation was Fairplay [15], which requires the circuit to be built and stored in memory before execution, an approach that does not scale to large circuits. TASTY [28] and more recently L1 [29] extend Fairplay by reverting to homomorphic encryption when the latter is more efficient. In contrast to both systems, FastGC [6] introduced the idea of concurrent garbling and execution, which significantly reduces memory consumption and improves scalability to circuits of arbitrary number of gates. As such, it is currently

the fastest implementation providing security against a semi-honest garbler.

Several other frameworks explored extensions to malicious adversaries [25], [30]–[32]. Building on earlier work [33], a recent two-party system by Huang *et al.* [30] provides security against malicious adversaries by running the protocol twice on both parties and comparing the result using a secure equality test. Closer to our setting, Kreuter *et al.* [31] designed a framework for two-party computation in the presence of a malicious garbler. To achieve this type of security, the refined cut-and-choose technique from [32] is applied: the garbler is required to produce several garbled circuits and the Evaluator will ask the garbler to reveal keys for the circuits of her choice, checking that the circuits were generated correctly. As such, if the garbler generates the wrong circuit, she is detected with high probability. In our setup, the correctness of the garbled circuit can be verified with a much lower communication cost through the protocol outlined in Section IV-F, so we forego using any of the above methods in our implementation.

C. Solving a Linear System in a Circuit

One of the main challenges of our approach is designing a circuit that solves the linear system $A\beta = b$, as defined in Eq. (2). When implementing a function as a garbled circuit, it is preferable to use operations that are *data-agnostic*, *i.e.*, whose execution path does not depend on the input. For example, as inputs are garbled, the Evaluator needs to execute all possible paths of an **if-then-else** statement, which leads to an exponential growth of both the circuit size and the execution time in the presence of nested conditional statements. This renders impractical any of the traditional algorithms for solving linear systems that require pivoting, such as, *e.g.*, Gaussian elimination.

A data-agnostic—but very inefficient—exact method of solving a linear system is the standard method of Cramer's rule [34]. An efficient, data-agnostic, but approximate method for solving a linear system is to minimize $\|A\beta - b\|_2^2$ through, *e.g.*, a fixed number of iterations of conjugate gradient descent [34], [35]. This is particularly advantageous over exact methods when A is sparse, which is not the case in our setup. Moreover, using a fixed number of iterations (and, in turn, limiting the depth of the circuit) introduces additional errors in the estimation of β. This has a compounding effect on the error, which is also impacted by the representation of real numbers in a fixed number of bits. We thus opted for an exact method instead.

Data-agnostic methods for inverting the matrix A also exist, which in turn can be used to compute β as $A^{-1}b$. Hall *et al.* [8] propose an iterative matrix inversion algorithm which can also be implemented as a garbled circuit; this approach however is also approximate, and introduces errors when a fixed number of iterations is used. We conducted a preliminary evaluation comparing this method with Cholesky decomposition, and observed that Cholesky required a smaller number of operations for the same level of accuracy. At present, the fastest method for inverting a matrix is block-wise inversion [34], that has a complexity of $\Theta(d^{2.37})$ (the same as matrix multiplication). For the sake of simplicity, we implemented the standard $\Theta(d^3)$ Cholesky algorithm presented below. We note, however, that its complexity can be further reduced to the same complexity as block-wise inversion using similar techniques [36].

Algorithm 1 The Cholesky decomposition of $A \in \mathbb{R}^{d\times d}$

Input: $A = (a_{ij})_{i,j\in\{1,\dots,d\}}$
Output: L, s.t., $A = L^T L$
1: **for** $j = 1, d$ **do**
2: **for** $k = 1, j-1$ **do**
3: **for** $i = j, d$ **do** $a_{ij} \mathrel{-}= a_{ik}a_{jk}$
4: $a_{jj} = \sqrt{a_{jj}}$
5: **for** $k = j+1, d$ **do** $a_{kj} \mathrel{/}= a_{jj}$
6: **return** $L := A$

There are several decomposition methods for solving linear systems, such as LU and QR [34]. Cholesky decomposition is a data-agnostic method for solving a linear system that is applicable only when the matrix A is symmetric positive definite. The main advantage of Cholesky is that it is numerically robust *without the need for pivoting*, as opposed to LU and QR decompositions. In particular, it is well suited for fixed point number representations [37]. Since $A = \lambda I + \sum_i^n x_i x_i^T$ is indeed a positive definite matrix for $\lambda > 0$, we chose Cholesky as the method of solving $A\beta = b$ in our implementation.

We briefly outline the main steps of Cholesky decomposition below. The algorithm constructs a lower-triangular matrix L such that $A = L^T L$. Solving the system $A\beta = b$ then reduces to solving the following two systems:

$$L^T y = b\,, \quad \text{and}$$
$$L\beta = y\ .$$

Since matrices L and L^T are triangular, these systems can be solved easily using back substitution. Moreover, because matrix A is positive definite, matrix L necessarily has non-zero values on the diagonal, so no pivoting is necessary.

The decomposition $A = L^T L$ is described in Algorithm 1. It involves $\Theta(d^3)$ additions, $\Theta(d^3)$ multiplications, $\Theta(d^2)$ divisions and $\Theta(d)$ square root operations. Moreover, the solution of the two systems above through backwards elimination involves $\Theta(d^2)$ additions, $\Theta(d^2)$ multiplications and $\Theta(d)$ divisions. We discuss how we implement these operations as circuits in the next two sections.

D. Representing Real Numbers

In order to solve the linear system (2), we need to accurately represent real numbers in a binary form. We

considered two possible approaches for representing real numbers: floating point and fixed point. Floating point representation of a real number a is given by formula:

$$[a] = [m, p], \quad \text{where } a \approx 1.m \cdot 2^p \ .$$

Floating point representation has the advantage of accommodating numbers of practically arbitrary magnitude. However, elementary operations on floating point representations, such as addition, are difficult to implement in a data-agnostic way. Most importantly, using Cholesky warrants using fixed point representation, which is significantly simpler to implement. Given a real number a, its fixed point representation is given by:

$$[a] = \lfloor a \cdot 2^p \rfloor, \quad \text{where the exponent } p \text{ is fixed} \ .$$

As we discuss below, many of the operations we need to perform can be implemented in a data-agnostic fashion over fixed point numbers. As such, the circuits generated for fixed point representation are much smaller. Moreover, recall that the input variables of ridge regression x_i are typically rescaled the same domain (between -1 and 1) to ensure that the coefficients of β are comparable, and for numerical stability. In such a setup, it is known that Cholesky decomposition can be performed with fixed point numbers without leading to overflows [37]. Finally, given bounds on y_i and the condition number of the matrix A, we can compute the bits necessary to prevent overflows while solving the last two triangular systems in the method. We thus opted for implementing our system using fixed point representations.

The number of bits p for the fractional part can be selected as a system parameter, and as we show later, it creates a trade-off between the accuracy of the system and size of the generated circuits. However, we illustrate that selecting p can be done in a principled way based on the desired accuracy. We represent negative numbers using the standard two's complement representation.

E. Arithmetic Library

To implement Cholesky decomposition we first built a set of sub-circuits that implement all elementary operations over fixed point numbers, namely addition, multiplication, subtraction, square root and division. Using fixed point representation, all these operations reduce to their integral version, making their implementation straightforward. In particular:

- Addition/Subtraction: $[a \pm b] = [a] \pm [b]$;
- Multiplication: $[a \cdot b] = [a] \cdot [b]/2^p$;
- Division: $[a/b] = [a] \cdot 2^p/[b]$;
- Square root: $[\sqrt{a}] = \sqrt{[a] \cdot 2^p}$.

For example, to implement division between two numbers with fixed point representation $[a]$ and $[b]$, it suffices to shift $[a]$ left p times (corresponding to a multiplication by 2^p) and perform an integer division with $[b]$. The resulting number will be a/b in fixed point representation. Similarly, computing the fixed point representation of the square root of a number amounts to shifting it to the left p times and finding the integer part of the square root of this number.

The FastGC library does not include multiplication. We extended it by implementing the Karatsuba multiplication algorithm [38]. This algorithm multiplies two k-digit numbers in at most $3k^{\log_2 3} \approx 3k^{1.585}$ single digit operations, outperforming the standard "school method" that uses k^2 operations. Efficiency in division was less critical since division is used less frequently than multiplication by a factor of d. Hence, we implemented division using the standard school method.

The computation of square roots is often implemented using Newton's method [34, Chapter 9]. This method computes the square root of a number x by starting with an estimate and refining the estimate, by halving the error at each step. Newton's method is problematic over fixed point numbers because it may oscillate when performed over integer arithmetic. Furthermore, the number of iterations needed to reach a target accuracy level depends on the accuracy of our initial guess, and thus we cannot correctly predict the accuracy achieved. Therefore, we implemented a different approach to find square roots that uses a bit-by-bit computation [39].

This method is iterative but data-agnostic, as it always takes a fixed number of steps. It does not require multiplications or divisions, and thus leads to a small, efficient circuit. To explain this method, we denote by r the solution for the square root of x at step i (starting at $i = 0$); then, r contains the first i bits of the desired root. To find the value of the next most significant bit, let e denote the number with zeros everywhere except a 1 in the $i+1$-st bit. The algorithm performs the following simple test: if $(r+e)(r+e) = r^2 + 2er + e^2 \leq x$, then the new bit is set to 1. If this quantity exceeds x, then the bit is set to zero. The whole process is then repeated for each subsequent bit. The actual algorithm is summarized in Algorithm 2, which is an optimized version that avoids the multiplications appearing in the above computation.

VI. Experiments

This section details the evaluation of our proposed hybrid system. Recall our implementation focus is on the second protocol of Section IV, since this design is the most efficient given all the tradeoffs discussed so far.

To understand the effects of input parameters (such as the number of users and number of features) as well as system parameters (such as the number of bits for the fractional representation) on our system's performance, we first use synthetically generated datasets. We look at the performance in terms of the circuit size and the execution time. The advantage of using synthetic data is that we can

Algorithm 2 Integer square root

Input: $x \in \mathbb{Z}$
Output: $r = \lfloor \sqrt{x} \rfloor$

```
Set e to be highest power of four not greater than x (this
  will determine the first digit of the result)
while e ≠ 0 do
    if x ≥ r + e then
        x −= r + e
        r = (r ≫ 1) + e
    else
        r = r ≫ 1
    e = e ≫ 2
return r
```

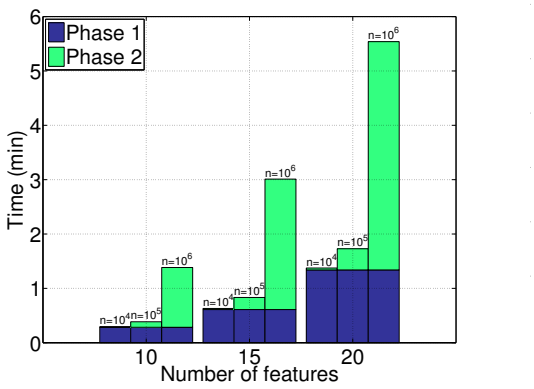


Figure 4: Time comparison between Phase 1 and Phase 2

evaluate the performance of our methods over a wide range of parameter values. We show how someone configuring our system can select the right number of bits for the fractional part of number representation. Finally, we also evaluate our system on a number of real-world datasets, and show that the proposed system is scalable, efficient and accurate.

All experiments were run on a commodity 1.9 GHz, 64GB RAM server, running Ubuntu Linux 12.04, Java Runtime Engine (JRE) 1.7. The Evaluator and the CSP run on different Java Virtual Machines (JVMs), each using a single processor. The users are simulated separately and their inputs are assumed to be ready before the Evaluator starts using their data, thus user-side computations are not accounted for in our results.

A. Generating Synthetic Data

We created a number of different synthetic datasets, each with a different number of users and features, using the method depicted in Alg. 3. The algorithm first creates a random matrix X, which captures the set of user features, and then computes the output variable y by fixing a random linear relationship β, and adding Gaussian noise.

Algorithm 3 Synthetic dataset generation

Input: Number of users n, Number of features d
Output: $X \in \mathbb{R}^{n \times d}$, $y \in \mathbb{R}^n$

```
Generate uniformly at random X ←U [0,1]^{n×d}
Generate uniformly at random β ←U [0,1]^d
Compute y = Xβ + e, where e ~ N(0,1)
```

By adjusting λ in Equation (1) we change the condition number of the input matrix, which enables us to study the stability of the algorithm.

B. Phase 1 Performance

Our implementation of Phase 1 using Paillier's scheme sums n matrices $A_i \in \mathbb{R}^{d \times d}$ and n vectors $b_i \in \mathbb{R}^d$ under homomorphic encryption, thus we expect the summation phase to take $O(n(d + d^2))$, with a timing constant that depends on the hardware. As we discussed in Section V-A, batching in Paillier adds a speedup of roughly 20 over a non-batched implementation. In our experiments we indeed found a perfect linear correlation between the time to execute the phase and the number of users, and using our hardware, we found the constant to be 0.75μsec. For example, using 10 million users and 20 features, Phase 1 takes 52 minutes to complete, and for 100 million users it takes 8.75 hours. We note that we implemented a sequential summation using Paillier's scheme; however this could easily be parallelized and on k processors computed roughly k times faster.

Recall that using homomorphic encryption for Phase 1 leads to a speed-up over naïvely implementing Phase 1 as a circuit, as the summation can be conducted significantly faster. We conducted an experiment with the "pure" Yao approach to illustrate this. Using 10K users and 20 features, Phase 1 takes 5 minutes when implemented as a circuit, as opposed to 2.4 seconds using Paillier; for 10 million users, Phase 1 implemented as a Yao's circuit takes more than 3 days to compute as opposed to 52 minutes using Paillier.

In order to evaluate the scalability of our system with the number of users, we study the time costs of Phase 1 relative to Phase 2. Figure 4 plots the time spent in the two phases for different number of users n and different number of features (the number of bits for number representation is fixed to 20). The plot shows that only when the dataset includes more than hundreds of thousands of users Phase 1 starts to take longer than Phase 2.

C. Accuracy

Since our proposed system implements regression in a numerical way using fixed point numbers it introduces errors. Denote by β^* the solution to ridge regression in the clear (we used Matlab on a 64bit commodity server), and β to be the solution using our system. Recall that $F(\beta)$, as given in Eq. (1), is the objective function we are minimizing.

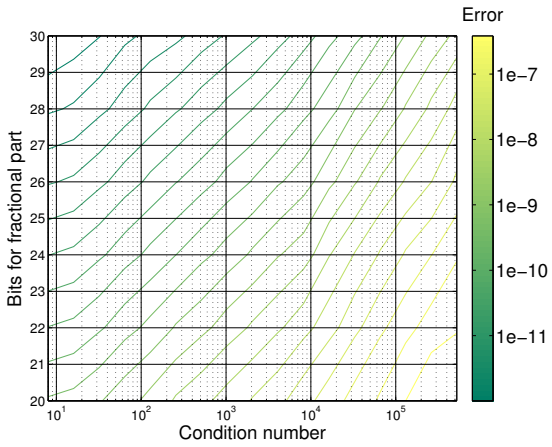


Figure 5: Tradeoffs between number of bits used for the fractional part of number representation, the condition number, and error rates

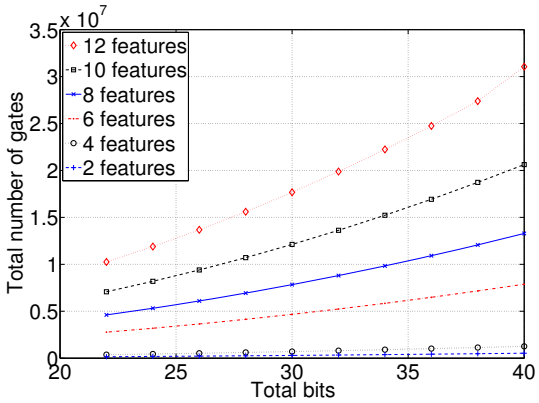


Figure 6: Number of gates in the circuit depending on number of bits and number of features

We define the error of our system as:

$$Err_{\beta^*} = \left| \frac{F(\beta) - F(\beta^*)}{F(\beta^*)} \right| .$$

This error definition allows us to assess the loss of accuracy—as captured by the objective function—that is introduced by our system as opposed to running ridge regression in the clear, without any privacy concerns.

Two of the most dominant features that affect the accuracy of our system are the condition number of the input matrix and the number of bits used to represent fractions. The condition number of A is defined as the ratio between its largest and smallest eigenvalues, and characterizes the stability of the system (*i.e.*, how much the solution β is affected by small perturbations of A) [37].

Through a broad sweep of synthetic experiments, Figure 5 illustrates the tradeoff between the number of bits, the condition number of matrix A and the resulting error rates. This plot provides us an important design tool for configuring our system; for example, if the condition number of the data is expected to be 10^4, and the Evaluator seeks an error on the order of 10^{-6}, then the plot indicates that she should use 25 bits. Overall, this plot allows the Evaluator to select the number of bits to use based on a bound on the condition number and her target error rate. Although the Evaluator can decide to use more bits to lower her error, as we show next, this comes as the cost of a larger and slower circuit.

D. Phase 2 Performance

Next, we study how the inputs and parameters of our system affect its execution time and the size of the resulting circuit. Figure 6 shows the number of gates in the resulting circuit for increasing numbers of bits used for representing numbers, and for different numbers of features. We note that because FastGC implements the "free" XOR technique, only OR and AND gates are accounted for in the plot (XOR gates constitute roughly 70% of the circuits). The plot shows that the circuits are relatively large, spanning tens of millions of gates. The garbled representation of each gate is a 30 Bytes lookup table, thus a complete circuit is hundreds of megabytes large. For example, a complete circuit of 20 features using 24 bits takes 270 MBytes. Such a circuit is easy to store in the memory of any modern commodity server. Furthermore, the size of the circuit also determines the communication overhead, which is practical even for Internet-scale deployments, where the Evaluator and CSP are not co-located. For example, using a standard 10Mbps link, it takes only a few minutes to communicate the 270 MBytes circuit between the parties.

As expected from the analysis of Cholesky and the complexity of our implementation of division and square root, the circuit size is cubic in the number of features and quadratic in the number of bits of fixed-point representation.

When computing execution time, we include the time from the beginning of the garbling process by the CSP, until the end of the execution process by the Evaluator. Recall that in FastGC the garbling and execution are almost concurrent, and delays occur mostly because of Java's garbage collection process. The communication overhead is negligible because the Evaluator and CSP are executed as two processes on the same machine.

Figure 7 shows the execution time as a function of the number of bits for representing numbers and the number of features (the noisy trend is the result of Java's garbage collection behavior). For example, a circuit with 30 bits and 12 features takes roughly 50 seconds to complete. Since we do not parallelize the execution, the time it takes the Evaluator to execute the circuit is proportional to the circuit size shown in Figure 6, *i.e.*, it has the same dependencies in the number of features and number of bits.

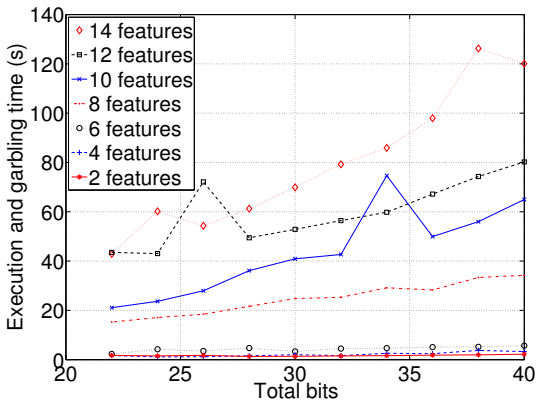


Figure 7: Time for execution of garbled circuit depending on number of bits and number of features

E. Comparison to State of the Art

Hall *et al.* [8] propose a privacy-preserving linear regression solution based on secret sharing, under the semi-honest model. Using their method, linear regression on 51K records, 22 features, and a low accuracy of 10^{-3}, took two days to complete out of which a full day is spent on the inversion of the matrix alone. By extrapolating the results in Figure 7 combined with the observation that our system scales cubically with the number of features, we found that our implementation will take roughly 3 minutes to carry out a similarly sized problem.

Graepel *et al.* [35] propose solving a linear system in a privacy-preserving manner through a fixed number of iterations of gradient descent. This allows them to use Somewhat Homomorphic Encryption that supports only a fixed number of multiplications. On a dataset of 100 2-dimensional points, 1 iteration of gradient descent using their method lasts 1 minute, while 2 iterations last 75 seconds. Our system can handle such a dataset within 2 seconds, at an accuracy of 10^{-5}. The relative performance benefit of our system increases for larger datasets. Overall, these timings demonstrate that our approach can outperform other state-of-the-art implementations by one or two orders of magnitude.

F. Real Datasets

Finally, we ran our system on real datasets retrieved from the UCI repository [7]. The repository includes a set of 24 real-world datasets that are commonly used for evaluating various machine-learning algorithms. Table I summarizes our findings. For each of the datasets, we indicate the number of entries n and features d. The number of bits was chosen in order to achieve a low target error of 10^{-5}. The penalty coefficient λ was chosen using cross-validation. The table provides the overall communication required between the Evaluator and CSP for sending garbled tables, and the overall time it took the Evaluator to execute the circuit.

Table I: Experimental results using UCI datasets

Name	n	d	Bits	Comm. (MB)	Time (s)
automobile.I	205	14	31	189	100
automobile.II	205	14	24	118	91
autompg	398	9	21	39	21
challenger	23	2	13	2	2
communities	1994	20	21	234	122
communities11.I	2215	20	21	234	130
communities11.II	2215	20	21	234	126
communities11.III	2215	20	23	271	146
communities11.IV	2215	20	21	234	314
computerhardware	209	7	19	21	15
concreteslumptest.I	103	7	15	15	10
concreteslumptest.II	103	7	17	18	14
concreteslumptest.III	103	7	19	21	11
concreteStrength	1030	8	19	27	17
forestFires	517	12	23	83	46
insurance	9822	14	21	102	55
flare1.I	323	20	17	170	92
flare1.II	323	20	19	200	108
flare1.III	323	20	19	200	109
flare2.I	1066	20	19	200	115
flare2.II	1066	20	21	234	132
flare2.III	1066	20	21	234	140
winequality-red	1599	11	21	60	39
winequality-white	4898	11	23	69	45

We note that the OTs execute in less than 2 seconds for all of the datasets. Furthermore, the communication overhead is very small, for example, for the "forestFires" dataset the overall communication during OT is roughly 52Kbytes, which is insignificant when compared to the 83MBytes it takes to communicate the garbled circuit.

The smallest dataset ("challenger") has only 2 features, and it resulted in a small 2MB communication overhead, and was executed in 2 seconds. The slowest execution time of slightly more than 5 minutes is observed for the "communities1 1.IV" dataset, that has 2215 samples and 20 features. However, this is still extremely fast for most practical uses.

VII. Related Work

Secure linear regression over private data is a subject that has received considerable attention, although in different settings than the one considered here. Previous work mostly focused on data that is partitioned either horizontally or vertically across a few databases [8], [40]–[44]. The database owners wish to learn a linear model over their joint data without explicitly disclosing any records they own. Some approaches use protocols based on secret sharing [8], [45]–[47] while others use additive homomorphic encryption (for horizontally partitioned data) [48]. These proposed solutions are not suited for our setting—millions of distinct users and a single aggregator —since all contributing parties must remain present and online throughout the entire computation. Moreover, as shown in Section VI-E, our approach improves the execution time in [8] by two orders of magnitude, from days to minutes.

A different approach makes use of two party computation based on Yao garbled circuits [3]. Several frameworks implement Yao garbled circuits and describe many applications [6], [49]. However, we are not aware of any work applying garbled circuits to regression.

A third approach for privacy-preserving computation is using fully homomorphic encryption (FHE) [2], which eschews the requirement of two non-colluding parties. Linear regression performs a large number of both multiplication and addition steps. In such a setting, current FHE schemes [35], [50] are not as efficient as our system. Lauter *et al.* [50] mention using FHE for regression, but do not quantify performance. Also as demonstrated in Section VI-E, we can reduce the execution time in Graepel *et al.* [35] from a minute to a few seconds.

Similar to our system, some works introduced hybrid approaches, such as combining HE and garbled circuits for face and fingerprints recognition [51], [52], and combining secret sharing with garbled circuits for learning a decision tree [53]. Constructing circuits for such discrete-valued functions is significantly different than the regression task we consider here; to the best of our knowledge, this work is the first to design and implement a circuit for solving a positive definite linear system.

Much recent research focuses on database privacy under the differential privacy paradigm [54]–[56]. Our approach is orthogonal to differential privacy, as we operate under a different threat model. In differential privacy, a database owner has full access to the data in the clear. The privacy threat arises from releasing a function over the data to a third party, which may use it to infer data values of users in the database. In our work, the database owner itself (*i.e.*, the "Evaluator") poses a threat. Moreover, our solution can be augmented to provide differential privacy guarantees against a public release of β through appropriate addition of noise (see, *e.g.*, [55], [56]) within the regression circuit.

VIII. Conclusions

We presented a practical system that learns ridge regression coefficients for a large number of users without learning anything else about the users' data. The system efficiently scales with the number of users and features while providing the required accuracy. To do so we employ a hybrid system that uses homomorphic encryption to handle the linear part of the computation and Yao garbled circuits for the non-linear part. Using a commodity server our implementation learned a regression model for 100 million user records, each with 20 features, in less than 8.75 hours.

The practicality of our design invites further research on privacy-preserving machine learning tasks, particularly ones that use regression as a building block. For example, recommender systems employ techniques like matrix factorization to extract item profiles from ratings their users generate. A widely used method for matrix factorization (termed the alternating least-squares method), involves iterative executions of ridge regressions. Extending our system to this application will enable, *e.g.*, online video service providers to profile their catalog, without learning potentially private information about their users.

Beyond linear regression, many classification methods such as logistic regression and support vector machines (see, *e.g.*, [11]) also build models as solutions of convex optimization problems, parametrized by user data, similar to the one we considered here. Extending our hybrid approach to these settings, and providing efficient solvers for such problems as garbled circuits, is a promising future direction for this work.

Acknowledgments. The fifth author is supported by NSF, DARPA, IARPA, an AFO SR MURI award, and a grant from ONR. Funding by IARPA was provided via DoI/NBC contract number D11PC20202. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, IARPA, DoI/NBC, or the U.S. Government.

References

[1] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation," in *ACM STOC*, 1988.
[2] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *ACM STOC*, 2009.
[3] A. C.-C. Yao, "How to generate and exchange secrets," in *IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, 1986.
[4] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology – EUROCRYPT '99*. Springer, 1999.
[5] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *J. ACM*, vol. 56, no. 6, 2009.
[6] Y. Huang, D. Evans, J. Katz, and L. Malka, "Faster secure two-party computation using garbled circuits," in *20th USENIX Security Symposium*. USENIX Association, 2011.
[7] UCI, "Machine Learning Repository." [Online]. Available: http://archive.ics.uci.edu/ml/datasets.html
[8] R. Hall, S. E. Fienberg, and Y. Nardi, "Secure multiple linear regression based on homomorphic encryption," *J. Official Statistics*, vol. 27, no. 4, 2011.
[9] Y. Aumann and Y. Lindell, "Security against covert adversaries: Efficient protocols for realistic adversaries," *J. Cryptology*, vol. 23, no. 2, 2010.
[10] M. K. Franklin and M. Yung, "Communication complexity of secure computation," in *ACM STOC*, 1992.
[11] J. Friedman, T. Hastie, and R. Tibshirani, *The Elements of Statistical Learning: Data Mining, Inference and Prediction*, 2nd ed. Springer, 2009.
[12] Y. Lindell and B. Pinkas, "A proof of security of Yao's protocol for two-party computation," *J. Cryptology*, vol. 22, no. 2, 2009.
[13] M. O. Rabin, "How to exchange secrets by oblivious transfer," Harvard University, Tech. Rep. TR-81, 1981.
[14] S. Even, O. Goldreich, and A. Lempel, "A randomized protocol for signing contracts," *Commun. ACM*, vol. 28, no. 6, 1985.

[15] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella, "Fairplay – Secure two-party computation system," in *13th USENIX Security Symposium*. USENIX Association, 2004.

[16] M. Naor, B. Pinkas, and R. Sumner, "Privacy preserving auctions and mechanism design," in *1st ACM Conference on Electronic Commerce*. ACM Press, 1999.

[17] I. Damgård and M. Jurik, "A generalisation, a simplification and some applications of Paillier's probabilistic public-key system," in *Public-Key Cryptography – PKC 2001*. Springer, 2001.

[18] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *CRYPTO '91*, 1992.

[19] J. Camenisch and M. Michels, "Proving in zero-knowledge that a number is the product of two safe primes," in *Advances in Cryptology – EUROCRYPT '99*. Springer, 1999.

[20] S. Brands, "Untraceable off-line cash in wallets with observers," in *CRYPTO '93*. Springer, 1994.

[21] R. Cramer, I. Damgård, and B. Schoenmakers, "Proofs of partial knowledge and simplified design of witness hiding protocols," in *CRYPTO '94*. Springer, 1994.

[22] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *CRYPTO '86*. Springer, 1987.

[23] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers," in *CRYPTO 2010*. Springer, 2010.

[24] V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free XOR gates and applications," in *Automata, Languages and Programming (ICALP)*. Springer, 2008.

[25] B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams, "Secure two-party computation is practical," in *Advances in Cryptology – ASIACRYPT 2009*. Springer, 2009.

[26] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank, "Extending oblivious transfers efficiently," in *CRYPTO 2003*. Springer, 2003.

[27] V. Kolesnikov, A.-R. Sadeghi, and T. Schneider, "Improved garbled circuit building blocks and applications to auctions and computing minima," in *Cryptology and Network Security (CANS)*. Springer, 2009.

[28] W. Henecka, S. K. ögl, A.-R. Sadeghi, T. Schneider, and I. Wehrenberg, "TASTY: tool for automating secure two-party computations," in *17th ACM Conference on Computer and Communications Security (CCS)*. ACM Press, 2010.

[29] A. Schropfer, F. Kerschbaum, and G. Muller, "L1 - an intermediate language for mixed-protocol secure computation," in *35th Annual Computer Software and Applications Conference (COMPSAC)*. IEEE Computer Society, 2011.

[30] Y. Huang, J. Katz, and D. Evans, "Quid-pro-quo-tocols: Strengthening semi-honest protocols with dual execution," in *IEEE Symposium on Security and Privacy*, 2012.

[31] B. Kreuter, A. Shelat, and C.-H. Shen, "Billion-gate secure computation with malicious adversaries," in *21st USENIX Security Symposium*. USENIX Association, 2012.

[32] Y. Lindell, B. Pinkas, and N. P. Smart, "Implementing two-party computation efficiently with security against malicious adversaries," in *Security and Cryptography for Networks (SCN)*. Springer, 2008.

[33] P. Mohassel and M. Franklin, "Efficiency tradeoffs for malicious two-party computation," in *Public-Key Cryptography (PKC)*. Springer, 2006.

[34] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes: The Art of Scientific Computing*, 3rd ed. Cambridge University Press, 2007.

[35] T. Graepel, K. Lauter, and M. Naehrig, "ML confidential: Machine learning on encrypted data," Cryptology ePrint Archive, Report 2012/323, 2012.

[36] F. G. Gustavson and I. Jonsson, "Minimal-storage high-performance Cholesky factorization via blocking and recursion," *IBM J. Res. Dev.*, vol. 44, 2000.

[37] J. H. Wilkinson, *Rounding Errors in Algebraic Processes*. Prentice Hall, 1963.

[38] D. E. Knuth, *The Art of Computer Programming, Vol. 2, Semuninumerical Algorithms*, 3rd ed. Addison-Wesley, 1997.

[39] J. W. Crenshaw, "Integer square roots," *Embedded Systems Programming*, Feb. 1998.

[40] A. P. Sanil, A. F. Karr, X. Lin, and J. P. Reiter, "Privacy preserving regression modelling via distributed computation," in *ACM KDD*, 2004.

[41] H. Yu, J. Vaidya, and X. Jiang, "Privacy-preserving SVM classification on vertically partitioned data," in *Advances in Knowledge Discovery and Data Mining (PAKDD)*, 2006.

[42] W. Du and M. J. Atallah, "Privacy-preserving cooperative scientific computations," in *14th IEEE Computer Security Foundations Workshop (CSFW-14)*. IEEE Computer Society, 2001.

[43] W. Du, Y. S. Han, and S. Chen, "Privacy-preserving multivariate statistical analysis: Linear regression and classification," in *4th SIAM International Conference on Data Mining (SDM 2004)*. SIAM, 2004.

[44] J. Vaidya, C. W. Clifton, and Y. M. Zhu, *Privacy Preserving Data Mining*. Springer, 2006.

[45] A. F. Karr, X. Lin, A. P. Sanil, and J. P. Reiter, "Secure regression on distributed databases," *J. Computational and Graphical Statistics*, vol. 14, no. 2, 2005.

[46] A. F. Karr, W. J. Fulp, F. Vera, S. S. Young, X. Lin, and J. P. Reiter, "Secure, privacy-preserving analysis of distributed databases," *Technometrics*, vol. 49, no. 3, 2007.

[47] A. F. Karr, X. Lin, A. P. Sanil, and J. P. Reiter, "Privacy-preserving analysis of vertically partitioned data using secure matrix products," *J. Official Statistics*, vol. 25, no. 1, 2009.

[48] J. F. Canny, "Collaborative filtering with privacy," in *IEEE Symposium on Security and Privacy*, 2002.

[49] Y. Huang, D. Evans, and J. Katz, "Private set intersection: Are garbled circuits better than custom protocols?" in *Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2012.

[50] K. Lauter, M. Naehrig, and V. Vaikuntanathan, "Can homomorphic encryption be practical?" in *3rd ACM Cloud Computing Security Workshop (CCSW)*. ACM Press, 2011.

[51] A.-R. Sadeghi, T. Schneider, and I. Wehrenberg, "Efficient privacy-preserving face recognition," in *Information, Security and Cryptology – ICISC 2009*. Springer, 2010.

[52] Y. Huang, L. Malka, D. Evans, and J. Katz, "Efficient privacy-preserving biometric identification," in *Network and Distributed System Security Symposium (NDSS 2011)*. The Internet Society, 2011.

[53] Y. Lindell and B. Pinkas, "Privacy preserving data mining," *J. Cryptology*, vol. 15, no. 3, 2002.

[54] C. Dwork, "Differential privacy," in *Automata, Languages and Programming (ICALP)*. Springer, 2006.

[55] C. Dwork and J. Lei, "Differential privacy and robust statistics," in *ACM STOC*, 2009.

[56] F. McSherry and I. Mironov, "Differentially private recommender systems: Building privacy into the Netflix prize contenders," in *ACM KDD*, 2009.