Privacy Preserving Feature Selection for Sparse Linear Regression

Adi Akavia
University of Haifa
Haifa, Israel
adi.akavia@gmail.com

Ben Galili
Technion
Haifa, Israel
benga9@gmail.com

Hayim Shaul
IBM Research
Haifa, Israel
hayim.shaul@gmail.com

Mor Weiss
Bar-Ilan University
Ramat-Gan, Israel
mor.weiss@biu.ac.il

Zohar Yakhini
RUNI & Technion
Herzlia, Israel
zohar.yakhini@gmail.com

ABSTRACT

Privacy-Preserving Machine Learning (PPML) provides protocols for learning and statistical analysis of data that may be distributed amongst multiple data owners (e.g., hospitals that own proprietary healthcare data), while preserving data privacy. The PPML literature includes protocols for various learning methods, including ridge regression. Ridge regression controls the L_2 norm of the model, but does not aim to strictly reduce the number of non-zero coefficients, namely the L_0 *norm* of the model. Reducing the number of non-zero coefficients (a form of *feature selection*) is important for avoiding overfitting, and for reducing the cost of using learnt models in practice. In this work, we develop a first privacy-preserving protocol for *sparse linear regression* under L_0 constraints. The protocol addresses data contributed by several data owners (e.g., hospitals). Our protocol outsources the bulk of the computation to two non-colluding servers, using homomorphic encryption as a central tool. We provide a rigorous security proof for our protocol, where security is against semi-honest adversaries controlling any number of data owners and at most one server. We implemented our protocol, and evaluated performance with nearly a million samples and up to 40 features.

KEYWORDS

Privacy preserving machine learning, sparse linear regression, feature selection, secure multiparty computation, homomorphic encryption

1 INTRODUCTION

The Machine Learning (ML) revolution critically relies on large volumes of data to attain high confidence predictions. However, the massive amounts of data collected on individuals and organizations incur serious threats to security and privacy. From a legal perspective, privacy regulations such as the European General Data Protection Regulation (GDPR) and the California Customer Privacy Act (CCPA) aim at controlling these threats. From a technological perspective, Privacy Preserving Machine Learning (PPML) [44] attains ML utility without exposing the raw data,[1] e.g., by leveraging tools for secure computation [27, 32, 68]. Many PPML solutions focus on the *inference* phase, e.g., [11, 30, 33]. A growing body of literature also addresses *training*, covering a range of learning tasks and techniques, including training decision tree models, e.g., [44], as well as linear regression, logistic regression and neural networks models, e.g., [48]. Recently, attention has been drawn also to the task of privacy preserving *feature selection* [43], which –in the context of linear regression– is the focus of our work.

Feature selection [35] is the process of selecting a subset of informative features to be used for model training, while removing non-informative or redundant ones. Feature selection is important for avoiding overfitting, that is, for producing models that support high quality prediction on new data rather than models that essentially "memorize" the training data set while failing to generalize. Moreover, feature selection supports producing *sparse models* whose use for prediction in practice requires measuring only the few selected features. Sparsity is often a desired property, leading to significant cost savings when the model is repeatedly used for prediction, especially when feature extraction is expensive. For example, sparse models are highly desired in medical applications where feature extraction might involve medical interventions with associated financial and morbidity costs (see Section 3.1).

The focus of this work is on feature selection under L_0 *constraints*,[2] specifically for *wrapper methods*. In contrast, previous work on privacy preserving feature selection considered *only filter* [15, 43, 52] *and embedded methods* [1, 3, 4, 19, 26, 28, 40, 44, 45, 49, 61–63, 65, 67, 70, 70], but not *wrapper methods*. We next elaborate on the differences between these methods.

Selecting the subset of features of highest predictive power is computationally intractable [14]; nonetheless heuristic methods – which can be categorized into *filter*, *embedded* and *wrapper* methods – are widely employed in practice with great success. Filter methods assign a score to each feature in isolation, outputting the highest scoring features; they are typically very fast, but *often fail to select the best features* because they ignore relationships and dependencies between features. Embedded methods intrinsically incorporate feature selection into the model training process, e.g., in decision tree regression, as well as in ridge and Lasso regression; The latter two techniques penalize models according to their (L_2 and L_1 respectively) norms, favoring models with fewer high-weight features. However, they *do not necessarily return a sparse model*: the model often includes a tail of low-weight features (particularly in ridge regression). Wrapper methods select features in an iterative process, with a target ML algorithm in mind (e.g., linear regression). In each iteration: (1) a model is trained on the current subset of features, then (2) features are ranked according to an evaluation metric measuring their usefulness for that model, and (3) this ranking is used to specify the subset of features for the next iteration. Wrapper methods are considerably slower than filter and embedded

[1]Other PPML concerns include limiting the amount of information revealed by the output, e.g., using differential privacy in [56]; see a survey in [42].

[2]Concretely, an L_0 constraint determines the intended number of features in the model (e.g., dictated by hardware considerations and/or limitations, as in [51]). We then look for the best model that satisfies this constraint.

This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license visit https://creativecommons.org/licenses/by/4.0/ or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

Proceedings on Privacy Enhancing Technologies 2024(1), 300–313
© 2024 Copyright held by the owner/author(s).
https://doi.org/10.56553/popets-2024-0017

methods, due to the multiple rounds of model training, but *often lead to superior outcomes* for the target ML algorithm.

In summary, none of the aforementioned methods strictly dominates the others (see, e.g., a comparative study in [39]). Consequently, common practices often examine several methods in search of the best performing one (using cross-validation to avoid overfitting) or use a *hybrid* approach to combine the strengths of several methods. Examples for hybrid approaches include using a filter method to quickly remove many features, then selecting from the remaining ones using an embedded or wrapper method (see, e.g., [18, 58]); or, executing wrapper and embedded methods in parallel, each scoring the features, where the output includes the features with highest total score (see, e.g., [16, 17, 34]).

Thus, providing privacy preserving solutions to a wide variety of feature selection methods is essential to supporting versatile usecases.

1.1 Our Contribution

In this work we present *the first privacy-preserving feature selection solution in a wrapper method*, producing a *sparse regression model* under L_0 constraint (i.e., with exactly the specified number of features). Concretely:

Our protocol. We design the *Secure Iterative Ridge regression (SIR)* protocol: a new protocol that produces a *sparse ridge regression model*, over input data that is (horizontally)-partitioned among distrusting data-owners, and where the bulk of the computation is outsourced to two non-colluding servers (aka, *two-server model*). Horizontal partitioning means that each data owner contributes a subset of data samples. Our solution utilizes homomorphic encryption [27, 53] as a central tool. The wrapper method that we realized in a privacy preserving fashion is the commonly used *Recursive Feature Elimination (RFE)* algorithm, introduced in the highly influential paper [36]. RFE starts by considering all features, iteratively training a model on the current set of “surviving” features, and removing features of lowest weight (the number of removed features is a tunable parameter), until reaching the L_0 constraint. The ML model that we train at each iteration is a ridge regression model, so that at the termination of all iterations we obtain a sparse regression model. As a central new component we introduce a *scaled ridge regression* protocol, which may be of independent interest.

Privacy & Threat Model. Privacy holds against all semi-honest computationally-bounded adversaries controlling any number of data owners and at most one of the two servers. The security guarantee is that such adversaries cannot infer any information on the inputs of data owners that they do not control, except for what can be efficiently computed directly from the public parameters, and the inputs and outputs of corrupted parties.[3] Our security proof for SIR covers the case of overdetermined linear regression (more data samples than features and full rank). To demonstrate the necessity of the security measures implemented in SIR, we devise *explicit attacks* showing that simplified variants without these measures are insecure.

Complexity. The complexity of each data owner is proportional only to her input and output size (and polynomial in the security parameter). The two servers engage in a two-party protocol with round complexity logarithmic in the number of input features d, and complexity that is cubic in d (up to poly-logarithmic factors) and logarithmic in the number of input records n (and polynomial in the security parameter); the protocol includes homomorphic computations of multiplicative depth at most $O(\log d + \log\log n)$.

System and empirical evaluation. We implemented our protocol SIR and empirically evaluated its performance. Our system is generic and can be applied to any dataset with numerical features. As a concrete example, we ran experiments on a gene-expression dataset derived from TCGA [59]. The full data matrix taken from TCGA breast cancer data consists of gene expression profiles for 781 samples. Each profile, representing a human subject, consists of over $10K$ values. Our proof of concept data represents regressing the expression pattern of a target gene (a vector with 781 entries), based on arbitrarily selected 40 other genes. Furthermore, we artificially randomly partition the 781 instances amongst 10 data owners. We note that horizontal partition, as in our experiments, has interesting use cases in analyzing gene expression related data, e.g., in [6, 8, 29, 57]. In our experiments, each iteration removes 10% of the features, and the L_0 constraint is to produce a model consisting of 8 features.

We point out that the initial selection of 40 out of $10K$ features would typically be executed using filter methods (due to their fast runtime), which can be done using the prior art on privacy preserving filter methods [15, 43, 52]. We focus therefore on selecting the 8 out of 40 remaining features using SIR –our privacy preserving RFE– which is the contribution of our work.

Our experiments demonstrate that our system produces the desired model – i.e., the same model as produced in the clear. Running in the clear takes seconds while our privacy preserving system terminates in just under a day, using 134GB RAM.

Scalability. Our protocol scales favorably with the number of input records and data owners, as demonstrated by our empirical evaluation on up to 802,816 records and 1000 data owners, where a 512× growth in the number of records (respectively, 10× growth in the number of data owners) led to runtime increase by 10% (resp., 1%).

1.2 Comparison to Prior Work

Prior privacy preserving feature selection solutions were in filter or embedded methods, whereas ours is the first in a wrapper method (concretely, a privacy preserving RFE). Although none of these methods dominates the others, RFE was developed in [36] for the use case of gene expression data, where it excelled. Indeed, the mean square error (MSE) of the model produced by our system significantly outperforms the regression models produced by filter, ridge and Lasso (the baseline); see the full version [2] for details.

In terms of runtime, our system is expected to be slow, since, even in cleartext, wrapper methods are considerably slower than filter and embedded ones. This is also evident by our experiments, where the runtime of our system (on 40 features) is roughly 1 day, compared to runtimes between slightly under a minute, and a few hours, on various dataset sizes (with 20-120 features) reported in previous works [4, 15, 26, 28, 40, 43, 49, 52, 62, 63, 70, 70].

Producing better models (i.e., with smaller MSE) –as in SIR– is typically desired, even if it incurs a slower (but reasonable) training

[3] The public parameters consist of the number of input samples and features, data precision, model regularization parameter, and model sparsity.

runtime. This is because training a model occurs once, whereas the fruits of having a better MSE provide recurring benefits in each use of the model. Furthermore, a runtime of 1 day (and even more) seems to be reasonable in the context of gene-expression data, where the trained model is used to guide the manufacturing of a medical testing device (DNA chip), and our training time is insignificant when compared to the manufacturing pipeline.

Furthermore, our runtime can be significantly lowered by tuning the system parameters to remove a larger fraction of the features in each iteration, thus reducing the total number of iterations. In particular, when tuning parameters so that our system returns a model satisfying the L_0 constraint in a single iteration, our system terminates in only 1 hour (when executed on 20 features and 1000 data samples). This single-iteration parameter setting may be of independent interest, because it provides a privacy preserving truncated ridge model – i.e., a ridge regression model whose low weight features are truncated as to satisfy the L_0 constraint. This should be used with caution though, as our experiments indicate that there is a tradeoff between runtime and MSE – where performing more iterations typically yields a better MSE.

In summary, SIR expands over prior work in offering the first privacy preserving feature selection in a wrapper method. This widens the PPML toolset to include the commonly used RFE, which leads to favorable learning outcomes in some use cases of interest.

1.3 Paper Organization

We give an overview of our techniques, along with a comparison to techniques used in prior work, in Section 2. Preliminary definitions appear in Section 3; the problem statement in Section 4. The SIR protocol appear in Section 5. Our system and empirical evaluation appear in Section 6. Conclusions appear in Section 7.

2 OVERVIEW OF OUR TECHNIQUES

In this section we give an overview of our techniques. We present the high level overview of SIR in Section 2.1, elaborate on its key components in Sections 2.2-2.3, discuss our attacks in Section 2.4, and compare our techniques to prior work in Section 2.6.

2.1 IR and SIR

We first describe the (insecure) feature selection algorithm, called Iterated Ridge (IR); and then present our Secure Iterative Ridge regression (SIR) protocol.

Iterated Ridge (IR, cf. Figure 1) is an RFE algorithm for sparse ridge regression, analogous to the sparse logistic regression algorithm used in [25]. IR starts with all features, removing 10% of the features in each iteration, until reaching $2 \cdot s$ features, where s is the user-determined target number of features, then removes features one-by-one.[4] (This follows the paradigm of adjusting the learning rate to a smaller value when approaching the solution.) Selecting which features to remove in each iteration is done by solving ridge regression (see Section 3.1) on the surviving features to obtain an intermediate model, and removing the features whose weights (in absolute value) are in the bottom 10%.

[4]The fraction of features to be removed in each iteration, and the threshold determining when to transition to the one-by-one phase, are both user-definable hyper-parameters.

Secure Iterative Ridge (SIR, cf. Figure 2) is executed between m data owners $\mathrm{DO}_1, \ldots, \mathrm{DO}_m$ and two non-colluding servers $\mathcal{S}_1, \mathcal{S}_2$. The data owners' inputs are a horizontal partition of the data matrix $X \in \mathbb{R}^{n \times d}$ and the target vector $\vec{y} \in \mathbb{R}^n$; i.e., there is a partition $I_1, \ldots, I_m$ of $[n] = \{1, \ldots, n\}$ so that each data owner DO_j holds the restriction of X and $\vec{y}$ to rows with indices in I_j, denoted X^j and $\vec{y^j}$. Let N denote a sufficiently large integer (as determined by Equation 4). In SIR we use homomorphic computation over encrypted data, which translates into computing on the underlying cleartext values with arithmetic modulo N, i.e., in the ring $\mathbb{Z}_N$ (unless explicitly stated otherwise). The values in $X, \vec{y}$ are assumed to be normalized to $[-1, 1]$ (which is common in ML), and each data owner scales her data to integral values in $[-10^\ell, 10^\ell]$ for a common precision parameter ℓ. Moreover, N is set to be sufficiently large so that, despite computing modulo N, we are able to produce the same final output as if computing over the integers.

SIR is composed of four phases as detailed next.

Phase I. Setup & Upload: $\mathcal{S}_2$ generates a key pair (pk, sk) for a fully homomorphic encryption scheme, and publishes pk.[5] Each data owner DO_j encrypts (entry-by-entry) $A^j = (X^j)^T \cdot X^j$ and $\vec{b^j} = (X^j)^T \cdot \vec{y^j}$ under pk, and sends the ciphertexts to $\mathcal{S}_1$ who homomorphically combines them to obtain ciphertexts for:

$$A = \sum_j A^j + \lambda I \in \mathbb{R}^{d \times d} \text{ and } \vec{b} = \sum_j \vec{b^j} \in \mathbb{R}^d.$$

Phase II. Obliviously Permute Features: $\mathcal{S}_1$ and $\mathcal{S}_2$ engage in a 1-message protocol that concludes with $\mathcal{S}_1$ holding ciphertexts for

$$A_p = P^T \cdot A \cdot P \text{ and } \vec{b_p} = P^T \vec{b},$$

where P is a random $d \times d$ permutation matrix. Importantly, $\mathcal{S}_1$ and $\mathcal{S}_2$ have no information on P other than it being a random permutation of the features' indices $[d] = \{1, \ldots, d\}$. For this purpose P is sampled obliviously as follows. First, $\mathcal{S}_2$ samples a uniformly random $d \times d$ permutation matrix P_2, encrypts it, and sends the ciphertexts to $\mathcal{S}_1$. Next, $\mathcal{S}_1$ samples a uniformly random $d \times d$ permutation matrix P_1, and homomorphically computes $P = P_1 \cdot P_2$, A_p and $\vec{b_p}$.

Phase III. Privacy Preserving RFE (cf. Figure 2, Steps 2-3): At the onset of this phase, $\mathcal{S}_1$ initializes the set F of surviving features to be all features, i.e., $F = [d]$. Denote by $A_{p|F}$ and $\vec{b}_{p|F}$ the restriction of A_p and $\vec{b_p}$ to the surviving features F (i.e., include only rows of A_p and $\vec{b_p}$ whose indices are in F, and likewise for columns of A_p). While the number of features in F exceeds the L_0 constraint, features are removed as follows:

(1) *Scaled ridge (cf. Figure 3)*: First, $\mathcal{S}_1$ and $\mathcal{S}_2$ engage in a two-party protocol, at the conclusion of which $\mathcal{S}_1$ holds a ciphertext encrypting the *scaled* ridge regression model:

$$\vec{w}_{\text{scaled}} = \det(A_{p|F}) \cdot \left[(A_{p|F})^{-1} \cdot \vec{b}_{p|F} \right]$$

This is a scaling by factor $\det(A_{p|F})$ of the ridge regression model for $(A_{p|F}, \vec{b}_{p|F})$ which is $\vec{w} = (A_{p|F})^{-1} \cdot \vec{b}_{p|F}$. This

[5]More precisely, $\mathcal{S}_2$ produces two key pairs $-(pk_N, sk_N)$ and (pk_D, sk_D), with $d+1 \leq D \ll N-$ supporting homomorphic computation modulo N and D respectively. We primarily employ pk_N; using pk_D only briefly during ranking (Phase III, Part 2).

scaling is central for both the correctness and the efficiency of our protocol; see a detailed discussion in Section 2.2.
In detail, the scaled ridge protocol operates as follows. First, $\mathcal{S}_1$ homomorphically masks $A_{p|F}$ and $\vec{b}_{p|F}$, and sends their masked versions to $\mathcal{S}_2$, where these masked versions are: $A^{\text{masked}} = A_{p|F} \cdot R$ and $\vec{b}^{\text{masked}} = \vec{b}_{p|F} + A_{p|F} \cdot \vec{r}$ for uniformly random invertible matrix R and random vector r of the appropriate dimensions.[6] Next, $\mathcal{S}_2$ decrypts, computes $\det(A^{\text{masked}})$ and $\vec{w}^{\text{masked}}_{\text{scaled}} = \text{adj}(A^{\text{masked}}) \cdot \vec{b}^{\text{masked}}$ in the clear, encrypts them, and sends the ciphertexts to $\mathcal{S}_1$. Finally, $\mathcal{S}_1$ homomorphically unmasks to obtain ciphertexts for the scaled (un-masked) model, using the algebraic identity: $\vec{w}_{\text{scaled}} = R \cdot \vec{w}^{\text{masked}}_{\text{scaled}} + \det(A^{\text{masked}}) \cdot \vec{r}$. We refer the reader to the full version [2] for the proof of correctness.

(2) *Ranking*: Next, $\mathcal{S}_1$ and $\mathcal{S}_2$ engage in a two-party protocol, at the conclusion of which $\mathcal{S}_1$ holds a ranking –in cleartext– of the surviving features according to their weight in $\vec{w}_{\text{scaled}}$; see details in Section 2.3. Importantly, $\mathcal{S}_1$ does not know the actual weights, only their ordering. This does not violate privacy, because the features were randomly permuted; details are provided in the full version [2].

(3) *Removal:* Finally, $\mathcal{S}_1$ removes the (dynamically adjusted desired number of) lowest ranking features, and updates F, $A_{p|F}$ and $\vec{b}_{p|F}$ accordingly.

We elaborate on the scaled ridge regression and ranking sub-protocols in the subsections below.

Phase IV. Obliviously un-permute and rationally reconstruct: At the onset of this phase the servers hold a subset $F \subseteq [d]$ of feature indices, which satisfies the L_0 constraint, i.e., $|F| = s$. They now engage in a two-party protocol to produce a (cleartext) sparse ridge regression model whose non-zero weights are only on features in F.[7] First, $\mathcal{S}_1$ and $\mathcal{S}_2$ engage in a two-party sub-protocol, at the conclusion of which $\mathcal{S}_1$ holds ciphertexts for the ridge regression model $\vec{w}_p = (A_{p|F})^{-1} \cdot \vec{b}_{p|F}$ for the final set of features F. Second, $\mathcal{S}_1$ homomorphically un-permutes the features' order (where features that did not survive have weight zero), and sends this un-permuted encrypted model to $\mathcal{S}_2$ who decrypts it, and sends it in the clear to $\mathcal{S}_1$. That is, $\mathcal{S}_1$ now has $\vec{w} = A^{-1}_{|F} \cdot \vec{b}_{|F}$, where the inverse and product are computed in $\mathbb{Z}_N$. Finally, $\mathcal{S}_1$ maps $\vec{w}$ to the solution to ridge regression over the surviving features *when computed over the rationals* (rather than in $\mathbb{Z}_N$) via rational reconstruction [22, 64],[8] and outputs the resulting model.

2.2 Our Scaled Ridge Protocol

A central new component in our solution is a sub-protocol that we call *scaled ridge regression*, which may be of independent interest. Our scaled ridge builds on the following two observations.

[6]This masking was introduced in [28] in the context of privacy preserving ridge regression. However, unlike our work, they employ masking to produce a ridge model that is *un-scaled* and *in cleartext*; see details in Section 2.6.

[7]The ridge regression model is computed similarly to the protocol of [4], except for restricting the data to the features in F, and obliviously masking and un-permuting over an encrypted model; see Section 2.6.

[8]Rational reconstruction refers to the Lagrange-Gauss algorithm which allows one to recover a rational $q = r/s$ from its representation $q' = r \cdot s^{-1} \in \mathbb{Z}_N$ for sufficiently large N (in particular, this holds for N which satisfies Equation 4).

Observation 1: Ranking is invariant to scaling. Ranking indices of a vector $\vec{w}$ according to the magnitude of their associated values $w[i]$ (in absolute value) is invariant to scaling the vector by a positive factor. Therefore, instead of ranking features according to their magnitude in the ridge regression model, we can rank them according to any (non-zero) scaling of this model.

Observation 2: homomorphic ranking is considerably faster with our scaling. We show that homomorphic ranking is considerably faster when we scale the ridge regression model as we do. To explain why this is so, we first provide some background and point out a key complexity bottleneck in homomorphic ranking of the ridge regression model. We then explain how to eliminate this bottleneck using scaling.

Let $(X, \vec{y})$ be a data matrix and target vector in a ridge regression problem, and set $A = X^T \cdot X + \lambda I$ and $\vec{b} = X^T \cdot \vec{y}$. The ridge regression model is:

$$\vec{w} = A^{-1} \cdot \vec{b}$$

where the arithmetic in computing A, $\vec{b}$ and $\vec{w}$ is over the reals. In our privacy-preserving solution we compute an encrypted model $\vec{w}$ via homomorphic computation over encrypted A and $\vec{b}$, and where the underlying plaintext computation is conducted in a finite ring $\mathbb{Z}_N$ of integers modulo N (rather than over the reals). To ensure that the same model is obtained despite performing computations modulo N, two measures are required. First, N must be sufficiently large (cf. Equation 4) so that, e.g., $A \bmod N$ and $\vec{b} \bmod N$ are identical to A and $\vec{b}$. Second, rational reconstruction must be computed on each entry of the reduced model $A^{-1} \cdot \vec{b} \bmod N$ to map it to the model $\vec{w}$ computed over the reals. Unfortunately, *homomorphically computing rational reconstruction is a complexity bottleneck.*

Next, we show that scaling the model by the factor $\det(A)$ allows us to eliminate the need for rational reconstruction. Relying on the algebraic identity $A^{-1} = \frac{\text{adj}(A)}{\det(A)}$ (where $\text{adj}(A)$ is the adjugate matrix of A), we can see that the scaled model $\vec{w}_{\text{scaled}} = \det(A) \cdot A^{-1} \cdot \vec{b}$ is equal to

$$\vec{w}_{\text{scaled}} = \text{adj}(A) \cdot \vec{b}.$$

Since computing $\text{adj}(A)$ involves only multiplications and additions, then –when using a sufficiently large N, as we do– no wrap-around occurs when computing $\vec{w}_{\text{scaled}}$ modulo N. Namely, computing over $\mathbb{Z}_N$ produces an identical scaled model as when computing over the reals. This implies that we can directly rank the entries of $\vec{w}_{\text{scaled}} \bmod N$ *without executing rational reconstruction.* Namely, we are able to avoid performing rational reconstruction in each iteration, thus eliminating a key complexity bottleneck.

2.3 Our Ranking Protocol

We rank features in the scaled ridge model of the current iteration by their absolute value. This is done by ordering features according to their "size", measured as their distance from the nearest multiple of N. This measurement of the "size" of a feature can be thought of as treating values larger than $(N-1)/2$ as negative, and comparing the absolute value of the features. Denote the current scaled model by $\vec{w}_{\text{scaled}} = (z_1, \ldots, z_d) \in \mathbb{Z}_N^d$ s.t. it has non-zero values $z_i \neq 0$ only on indices i in the current surviving set of features $F \subseteq [d]$. The ranking is computed as follows. First, $\mathcal{S}_1$ masks each pairwise difference $z_i^2 - z_j^2 \bmod N$, with $i, j \in F$, by homomorphically adding to it a uniformly random integer $r_{i,j}$ (modulo N), and sends the masked differences to $\mathcal{S}_2$. Second, $\mathcal{S}_2$ decrypts each masked difference $z_i^2 - z_j^2 + r_{i,j} \bmod N$, converts it to a binary representation (in the clear), encrypts this bit-by-bit and sends the ciphertexts to $\mathcal{S}_1$. The encryption of this binary representation is under the key pair (pk_D, sk_D), generated by $\mathcal{S}_2$ during setup, whose plaintext modulus is a small integer D larger than d. Third, $\mathcal{S}_1$ homomorphically compares each encrypted masked pairwise difference $z_i^2 - z_j^2 + r_{i,j}$ with his cleartext $r_{i,j}$ (where both are in binary representation, and while accounting for all possible overflows), producing an encrypted outcome bit $b_{i,j}$ which is equal to 1 if-and-only-if $z_i^2 > z_j^2$ and zero otherwise. Then, for each i, $\mathcal{S}_1$ homomorphically sums up the bits $b_{i,j}$ for all j to obtain a ciphertext encrypting Ord_i, which is the number of indices j with magnitude z_j^2 smaller than z_i^2 (we assume all weights are distinct), and sends all these ciphertexts to $\mathcal{S}_2$. Fourth, $\mathcal{S}_2$ decrypts all ordinals, computes the set of surviving features according to $\mathrm{Ord}_1, \ldots, \mathrm{Ord}_d$ (i.e., the set of highest-ranking features), and sends the indicator vector of this set (in the clear) to $\mathcal{S}_1$.

Importantly, in order to support fast homomorphic summation of these encrypted results, we instructed $\mathcal{S}_2$ – when encrypting the weights in binary representation – to use a small plaintext modulus of size D, where $D > d$ is larger than the number of comparison results to be summed-up. Since $D > d$, then there is no overflow when computing $\mathrm{Ord}_1, \ldots, \mathrm{Ord}_d$, and so they are a permutation of $\{1, \ldots, d\}$ ranking the entries of $\vec{w}$ by their size. The Boolean operations computed during the aforementioned homomorphic comparison are then emulated in $\mathbb{Z}_D$ (defining, for $a, b \in \{0, 1\}$, $AND(a, b) = a \cdot b \bmod D$ and $XOR(a, b) = (a - b)^2 \bmod D$). This at most doubles the multiplicative depth of comparison, for the benefit of making the summation computation a linear function that requires no multiplications.

The above (simplified) description of the ranking protocol overlooked the following subtle point: in each iteration, the servers learn the ordering between the features in the current iteration's model. If *the same* permutation on features were to be used in all iterations, this knowledge from the execution of multiple iterations might reveal non-trivial information (similar to the issues which arise by performing scaled ridge on unpermuted features). To overcome this, we have $\mathcal{S}_1$ apply a *fresh* permutation on the Ord_i's before sending them to $\mathcal{S}_2$. The permutation is inverted at the end of this step, and does not affect the rest of the computation. Privacy is preserved because each iteration uses a fresh permutation for ranking (which is applied on top of the long-term permutation applied at the onset of the protocol). See the full version [2] for details.

2.4 Security Challenges and Attacks

The iterative training of intermediate models is inherent to wrapper methods (and does not occur in filter or embedded methods), and introduces several security challenges. Indeed, we show that revealing any of the following (which are revealed in standard IR) would violate privacy (see the full version [2] for further details):

- The order by which features are removed.
- The intermediate models.
- The scaling factor $\det(A)$ in a scaled ridge model.

This demonstrates the necessity of the security measures implemented in SIR. Hiding this information while maintaining efficiency is a key challenge in designing SIR.

2.5 Discussion: Model Inversion Attacks

Our protocol (SIR) guarantees that only the output model (and whatever can be efficiently inferred from it) is revealed. This is the standard security goal in secure computation; however, it leaves open the question of how much information is revealed by the output model, and how to reveal even less.

Prior works [23, 24, 69] have shown that learning the model may indeed reveal non-trivial information on private inputs of honest parties. Most relevantly, [24] showed, in the context of genomic data for personalized medicine using a *(non-sparse)* linear regression model, that access to the trained model can be abused to infer private genomic attributes about individuals in the training dataset. Moreover, [24] showed that differential privacy is not effective for guaranteeing privacy because it is at odds with utility: when setting the privacy parameters sufficiently high to prevent their attack, the produced model does not retain sufficient clinical efficacy.

SIR is likewise susceptible to model inversion attacks. This is because sparsity of the model in itself does *not* guarantee security against inversion attacks, since even *a single bit* of information can compromise privacy. For example, in the context of gene-expression data, certain genetic variants are more common in some ethnicities than in others, and so revealing whether a feature corresponding to such a genetic variant is selected in the model may reveal the ethnicity of the population represented in the training dataset. Concretely, we present a model inversion attack showing that corrupt data owners who inject maliciously crafted inputs (but otherwise follow the protocol) can infer from the output model non-trivial information on the inputs of the honest data owners. The attack holds even when the output is a 1-sparse model, i.e., it consists of one selected feature. At a high-level, corrupted data owners use "balanced" inputs, in which all features have the same correlation with the response vector. Thus, inputs provided by corrupted parties do not affect the output model, and any correlations in the inputs of the honest parties will determine which features are selected in the sparse output model. See the full version [2] for further details.

To reduce the privacy risk associated with revealing the model, our solution SIR offers a fine grained control on how, and to whom, the model is revealed. This can be obtained by applying the following minor changes in Phase IV: "Obliviously un-permute and rationally reconstruct" step.

Alternative 1: Revealing the output model in cleartext to all participants of the protocol (but nothing more). This is the outcome when executing the protocol as specified in Section 2.1.

Alternative 2: Revealing the model only to one designated party, denoted O, who may be an external party participating only in Phase IV, as follows. $\mathcal{S}_1$ sends the (encrypted and un-permuted) model to O (instead of $\mathcal{S}_2$); O homomorphically masks the model with a random additive mask, and sends the encrypted masked model to $\mathcal{S}_2$ who decrypts and returns the (cleartext) masked model to O; finally, O removes the masking and computes rational reconstruction to obtain the output model. This is motivated by scenarios where

$\mathcal{S}_1, \mathcal{S}_2$ are powerful servers to which computation is outsourced (say, Amazon AWS and Google Cloud) and where O is some health organization authorized to receive the computed model (say, the World Health Organization).

Alternative 3: Outputting the model only in encrypted form. In this case $\mathcal{S}_1$ simply publishes the encrypted model, instead of sending it to $\mathcal{S}_2$ for decryption. The model in this case is specified by a vector of d ciphertexts whose plaintext values are in $\mathbb{Z}_N$ and where at most s of them are non-zero (in order to hide both the indices of the selected features and their weight). Outputting an encrypted model is motivated by scenarios where the encrypted model is employed for privacy preserving inference using homomorphic computation, as follows. Given the encrypted model and a (possibly, encrypted) sample[9] homomorphically compute the inner product of the model and sample, homomorphically mask the resulting value in $\mathbb{Z}_N$ (using additive mask), and send the ciphertext to $\mathcal{S}_2$ for decryption; then unmask the returned plaintext and apply rational reconstruction to obtain the inference result as a rational number. See the full version [2] for further details. We note that in order to reduce the efficacy of model inversion attacks it is advised to enforce a security policy restricting the entities authorize to make inference queries, and limiting their number of allowed queries. Analyzing such security measures is beyond the scope of this work.

2.6 Comparison to Prior Techniques

The two-server approach for privacy-preserving ridge regression dates back to the work of Nikolenko et al. [49], who were also the first to propose using additive homomorphic encryption for merging the data from all data owners. We follow them in our Setup & Upload phase. Solving the linear system $(A, \vec{b})$ was done in [49] using garbled circuits to guarantee security. Giacomelli et al. [28] proposed instead to solve a masked linear system $(A^{\mathsf{masked}}, \vec{b}^{\mathsf{masked}})$, where $\mathcal{S}_1$ masks the systems using homomorphic addition, $\mathcal{S}_2$ decrypts, solves, and sends the solution (the model) –in cleartext– to $\mathcal{S}_1$, who unmasks and applies rational reconstruction in the clear. Importantly, in [28] the model is sent in the clear, and both the unmasking and the rational reconstruction are done in the clear. However, this is impossible in our setting: we cannot expose the intermediate models in the clear, because this compromises privacy as we show in our attacks. Moreover, we cannot simply have $\mathcal{S}_2$ send the model in encrypted form and have $\mathcal{S}_1$ process it homomorphically, because the rational reconstruction step requires many sequential steps, making it computationally expensive to compute homomorphically.

Blom et al. [10] proposed avoiding rational reconstruction by having $\mathcal{S}_2$ send $\mathrm{adj}(A^{\mathsf{masked}}) \cdot \vec{b}^{\mathsf{masked}}$ and $\det(A^{\mathsf{masked}})$ (in the clear), where $\mathcal{S}_1$ unmasks to obtain $\mathrm{adj}(A) \cdot \vec{b}$ and $\det(A)$ and then computes the model $A^{-1}\vec{b} = \mathrm{adj}(A)\vec{b}/\det(A)$ with *division over the reals.* However, we cannot follow their approach, because we cannot send these values in the clear (as this would violate privacy, as we show in our attacks). Moreover, we cannot send the values in encrypted form, and have $\mathcal{S}_1$ homomorphically compute the division, because computing division homomorphically (be it modulo N or over the reals) is computationally expensive and thus not a viable alternative. In fact, even in the final iteration where the model can be revealed, our attacks demonstrate that it still violates privacy to reveal the pair $(\mathrm{adj}(A)\vec{b}, \det(A))$ rather than only their ratio $\mathrm{adj}(A)\vec{b}/\det(A)$. We therefore cannot employ the approach of [10]. Nonetheless, their approach inspired us in proposing our scaled ridge regression protocol.

Our proposed scaled ridge regression offers a new formulation, which eliminates the need for both rational reconstruction and division. In our protocol, $\mathcal{S}_2$ sends –in *encrypted form*– the pair $\mathrm{adj}(A^{\mathsf{masked}}) \cdot \vec{b}^{\mathsf{masked}}$ and $\det(A^{\mathsf{masked}})$, which $\mathcal{S}_1$ homomorphically unmasks to obtain ciphertexts for the scaled model $w_{\mathsf{scaled}} = \mathrm{adj}(A) \cdot \vec{b}$. Next, $\mathcal{S}_1$ *homomorphically ranks the features of this scaled model, without computing any division or rational reconstruction.* This is novel to our work, and may be of independent interest, with potential usage in other privacy preserving solutions using ridge regression as a component in a larger computation.

The overall structure of our protocol is likewise novel to our work, including its components of obliviously permuting and unpermuting the features (Phases II and IV in SIR) as well as the iterative execution of privacy-preserving regression for the ranking and removal (Phase III in SIR). We note that this overall structure necessitates using a *fully* homomorphic encryption (FHE), e.g., BGV [13] or B/FV [12, 21], to support both homomorphic addition and multiplication with respect to our plaintext modulus N (cf. only additive homomorphism in [10, 28]). However, as observed in [4], existing FHE libraries (e.g., HElib [38] and SEAL [54]) only support plaintext modulus of size up to 64-bit, whereas our protocol requires much larger integers (1260-bit integer in our implementation). To resolve this issue we follow [4] who suggested using the Chinese Remainder Theorem to represent each integer modulo N as a tuple of integers modulo small(ish) primes as supported by these libraries.

3 PRELIMINARIES

Notation. Upper-case letters (e.g., M) denote matrices, and vector notation (e.g., $\vec{v}$) denotes vectors. We use boldface letters to denote ciphertexts (e.g., $\mathbf{M}, \vec{\mathbf{v}}$ for a matrix and vector, respectively). We use Greek letters to denote masked values.

For a vector $\vec{w}$ we denote the number of its non-zero entries by $\mathsf{nnz}(\vec{w})$, and call it *s-sparse* if $\mathsf{nnz}(\vec{w}) \leq s$. For a vector $\vec{v}$ (similarly, set S), we use $|\vec{v}|$ ($|S|$) to denote its length (size). For a matrix X, we use X_i to denote its i'th row, and X^T to denote its transpose. For a matrix A, we use $\mathrm{adj}(A)$, $\det(A)$ to denote the adjugate matrix and determinant of A, respectively. We note that for any pair A, B of matrices it holds that $\mathrm{adj}(A \cdot B) = \mathrm{adj}(B) \cdot \mathrm{adj}(A)$, and $\det(A \cdot B) = \det(A) \cdot \det(B)$. For natural $d, N \in \mathbb{N}$, we use $\mathrm{GL}(d, \mathbb{Z}_N)$ to denote the group of all invertible $d \times d$ matrices with entries in $\mathbb{Z}_N$.

For a real value x, we use $\mathrm{abs}(x)$ to denote its absolute value, and $\lfloor x \rceil$ to denote its nearest integer. We extend the notation to apply to vectors and matrices entry-by-entry. We say that $x \in \mathbb{R}$ has *precision* ℓ if x is given as a real number with ℓ digits after the decimal point (which could be 0). If x has precision ℓ, then by *scaling* x *to lie in* $\mathbb{Z}$ we mean multiplying x by 10^{ℓ}.

For $N \in \mathbb{N}$, $[N]$ denotes the set $\{1, 2, \ldots, N\}$, and $\mathbb{Z}_N$ denotes the ring of integers modulo N. In our protocols, elements of $\mathbb{Z}_N$

[9]The sample is a length d vector specifying the sample's value for each features. Note that we require specifying all d features, rather than only the selected s features, because in this scenario we do not reveal which features are selected to the model.

are represented using the integers $0, 1, \ldots, N-1$ (this is without loss of generality). We treat values in $\mathbb{Z}_N$ that are greater or equal to $N/2$ as negative. In particular, this allows us to define the "size" – alternatively, the "absolute value" – of a group element as its distance from the nearest multiple of N. For example, 1 is considered to be smaller than $N-2 \equiv -2 \mod N$.

For random variables R, R', $\mathsf{R} \approx \mathsf{R}'$ denotes that R, R' are computationally indistinguishable. $\mathsf{negl}(\kappa)$ denotes a function which is negligible in κ, and PPT is shorthand for Probabilistic Polynomial Time. We use the standard notion of computational indistinguishability (e.g., from [31]).

Fully Homomorphic Encryption (FHE) [27, 53] is an encryption scheme that allows computations to be performed over encrypted data ("homomorphic computation"), producing an encrypted version of the result. Computation is specified by an arithmetic circuit over a ring called the *plaintext ring*. Our protocol employs FHE as a black-box. See a tutorial in [37].

3.1 Sparse Linear Regression & Applications

Linear regression is an important and widely-used statistical tool for modeling the relationship between properties of data instances $\vec{x}_i \in \mathbb{R}^d$ (features) and an outcome $y_i \in \mathbb{R}$ (response) using a linear function $\hat{y_i} = \vec{x}_i\vec{w}$ (the feature vectors are augmented with an additional first entry set to 1, as is standard). Training a regression model, takes n data instances $(\vec{x}_i, y_i) \in \mathbb{R}^{d+1}$ and returns a model $\vec{w} \in \mathbb{R}^{d+1}$ that minimizes a loss function, e.g., the Mean-Square-Error (MSE):

$$\vec{w} = \operatorname*{argmin}_{\vec{u}\in\mathbb{R}^{d+1}} \|\vec{y} - X\vec{u}\|_2^2 \tag{1}$$

where the rows of the matrix X are the n (augmented) vectors $\vec{x}_i \in \mathbb{R}^{d+1}$.

As we will discuss below, regularizing the solution $\vec{w}$ to Equation 1 is often beneficial, leading to LASSO regression [9, 60] and ridge regression, both are special cases of controlling the norm of $\vec{w}$. Ridge regression [4, 28, 41] seeks to find

$$\vec{w} = \operatorname*{argmin}_{\vec{u}\in\mathbb{R}^{d+1}} \left(\|\vec{y} - X\vec{u}\|_2^2 + \lambda\|\vec{u}\|_2^2\right) \tag{2}$$

where notation is as above and $\lambda \geq 0$ is the regularization (hyper)-parameter. Lasso seeks to find: $\vec{w} = \operatorname{argmin}_{\vec{u}\in\mathbb{R}^{d+1}} \left(\|\vec{y} - X\vec{u}\|_2^2 + \lambda\|\vec{u}\|_1\right)$.

In certain cases it is desired (or even required) that the output model $\vec{w}$ be sparse. That is, we are seeking a model $\vec{w}$ with many zero coefficients. Even stronger –due to hardware limitations, for example– we would be seeking a model with a fixed number of features. The latter is called L_0 *sparsity*, and leads to the following optimization task:

$$\vec{w} = \operatorname*{argmin}_{\vec{u}\in\mathbb{R}^{d+1}, \mathsf{nnz}(\vec{u})\leq s} \left(\|\vec{y} - X\vec{u}\|_2^2 + \lambda\|\vec{u}\|_2^2\right) \tag{3}$$

where $\mathsf{nnz}(\vec{u})$ denotes the number of non-zero entries in $\vec{u}$, and $s \in \mathbb{N}$ is the sparsity (hyper)-parameter. This task is the one addressed in this work, and is referred to as *sparse linear regression.*

In typical datasets, learning sparse linear models is useful due to two main reasons. First, simpler models are preferred during the training stage to avoid overfitting [9, 55]. Lower complexity translates to lower degree of polynomial models and/or less features in the output model. The latter can be reduced to model sparsity. The second reason to prefer sparser models is due to practical considerations. In some cases, hardware limitations restrict the number of features which can be measured when using the prediction model in the execution phase - when used to predict values, y, for new instances. In other cases, using more features in the execution prediction model is more expensive. For example, if y represents tumor severity, it might be reasonable to assume that y can be expressed as a linear (or polynomial) combination of molecular genomic information, say gene expression levels, in X. However, we expect, from a biological perspective, most genes to minimally affect the prediction performance. That is, the biology will be driven by a small number of genes.[10] Therefore, most components of $\vec{w}$ can be zero so that an assay used in clinical practice, based on such a predictive model, can use less expensive hardware, quantifying the expression levels of fewer genes [7, 20, 25, 50, 66].

3.2 Feature Selection and Iterated Ridge

Feature selection is an essential component in computational modelling and in the practical application of models. It has therefore been an active and prolific field of research in various domains such as pattern recognition, machine learning, statistics and data mining [46, 47]. Clever selection of the set of features to be used for data modelling, and as part of the execution models derived from learning, has been shown to improve the performance of supervised and unsupervised learning. Reasons are discussed above, as well as in the literature [7, 20, 36, 51]

Feature selection methods can be classified into several types based on the employed techniques, as discussed in Section 1. In this work we focus on a variant of Recursive Feature Elimination [36], a wrapper approach. A detailed description of the approach, in the clear, follows. Our approach is an iterative one that starts with all features, and iteratively removes features. This is similar to [25], that developed a sparse logistic regression model using RFE. In each iteration we run ridge regression with $\lambda \geq 0$ [4, 55] to calculate the weights for all features considered. Then, we remove features with low weights (in absolute value). The algorithm operates in two phases. In the first phase, we remove a 0.1-fraction of features, whereas when the current number of features decreases below a (user-defined) threshold thr, we move to the second phase, in which we remove a single feature in each iteration (this latter phase is analogous to Backward Subset Selection). The choice of the actual value of thr and the choice of the fraction removed in the early stages can affect the computational complexity of the process. Moreover, they are hyper-parameters of the model and can be tuned by cross validation. The pseudo-code of this algorithm is given in Figure 1.

4 PROBLEM STATEMENT

We follow the security and threat model of [4], and parts of this section are taken almost verbatim from [4]. SIR guarantees computational security, in the passive setting, against a single server

[10]The human genome codes for roughly $30K$ genes and many more functional elements.

```
Input: A dataset $D \in \mathbb{R}^{n\times d}$ and a target vector $\vec{y} \in \mathbb{R}^n$ (where entries are normalized to the same scale), and parameters $s$, rej $\in (0,1)$, thr $\in [d]$.
Output: A set $\Omega_s$ of the $s \le d$ selected features, and a ridge regression model $\vec{w}_s$ on these features.
Steps:
  (1) Initialize $\Omega$ to be the set of all features, and $\vec{w} = \vec{1}$.
  (2) While $\mathsf{nnz}(\vec{w}) > \mathsf{thr}$:
      (a) $\vec{w} = LR(D, y, \Omega)$
      (b) $\pi = \mathsf{argsort}(\mathsf{abs}(\vec{w}))$
      (c) prefix($\pi$) = $\pi[0\colon \mathsf{rej} \cdot |\Omega|]$
      (d) $\Omega = \Omega \setminus \text{prefix}(\pi)$
  (3) While $\mathsf{nnz}(\vec{w}) > s$:
      (a) $\vec{w} = LR(D, y, \Omega)$
      (b) smallest = $\mathsf{argmin}(\mathsf{abs}(\vec{w}))$
      (c) $\Omega = \Omega \setminus \{\text{smallest}\}$
  (4) Let $\Omega_s = \Omega$ and $\vec{w}_s = LR(D, y, \Omega_s)$. Return $(\Omega_s, \vec{w}_s)$.
```

Figure 1: Iterated Ridge (IR). Notations: $LR(D, y, \Omega)$ **denotes the solution of the linear regression system given by** (D, y) **when using only features in** Ω**;** $\mathsf{nnz}(\vec{w})$ **denotes the number of non-zero elements in** $\vec{w}$**; the function** argsort **sorts the indices of an array according to the values it contains;** $\mathsf{abs}(\vec{w})$ **returns the absolute value of each entry of** $\vec{w}$**. The regularization parameter** λ **is implicit in this pseudo-code.**

colluding with a proper subset of the data owners. More specifically, we assume all parties, even corrupted ones, are PPT and follow the protocol (though corrupted parties will try to infer additional information). We guarantee correctness of the output, and privacy of the inputs, in this setting. Specifically, the only information revealed to the corrupted parties is the *leakage profile*, namely the information that is *explicitly* revealed by the protocol. In our protocols, the leakage profile consists of the output model $\vec{w}$, as well as the following public parameters: the number n of data instances; the number d of features; the precision ℓ; a sparsity parameter s; and a regularization parameter $\lambda \ge 0$. More formally, we consider k-privacy in the passive setting, for inputs X such that $A = X^T X + \lambda I$ is invertible in the ring $\mathbb{Z}_N$ (invertability is needed for IR correctness; and in our case – as in previous works [4, 28] – also for privacy). We note that the input is horizontally-partitioned between the data owners (i.e., data owners hold disjoint subsets of rows of $(X, \vec{y})$).

Terminology. Let Π be an $(m+2)$-party protocol executed between PPT data owners $\mathsf{DO}_1, \dots, \mathsf{DO}_m$ and PPT servers $\mathcal{S}_1, \mathcal{S}_2$. We assume that every pair of parties share a secure point-to-point channel, and that all parties share a broadcast channel. We also restrict attention to protocols in which all parties obtain the same output, and only the data owners have inputs. For inputs $x_1, \dots, x_m$ of $\mathsf{DO}_1, \dots, \mathsf{DO}_m$, we use $\Pi(x_1, \dots, x_m)$ to denote the random variable describing the output in a random execution of Π (the probability is over the randomness of *all* participating parties, including the servers). For $I \subset \{\mathsf{DO}_1, \dots, \mathsf{DO}_m, \mathcal{S}_1, \mathcal{S}_2\}$, the (joint) *view* of I in Π, denoted $\mathsf{V}_I^{\Pi}(x_1, \dots, x_m)$, is the random variable consisting of the inputs and randomness of all parties in I, as well as the messages they received from the honest parties in a random execution of Π with inputs $x_1, \dots, x_m$. We say a subset $I \subseteq \{\mathsf{DO}_1, \dots, \mathsf{DO}_m, \mathcal{S}_1, \mathcal{S}_2\}$ is *k-permissible* if it contains at most k data owners, and at most one of the servers.

Security notion. We consider standard computational security against a passive adversary (see, e.g., [31]), adapted to the setting of non-colluding servers as in [49]. Since optimal feature selection under L_0 (Equation 3) is NP hard in general [14], we focus on providing a secure variant of the *Iterated Ridge* heuristic approach (see Section 3.1). Specifically, we require correctness in the sense that the secure variant has the same output as the cleartext iterated ridge algorithm, and privacy in the sense that any k-permissible set I learns nothing except the leakage profile (which consists of the public parameters and the output model) and the inputs of the parties in I (and anything efficiently computable therefrom). We also offer the option of returning an encrypted model (cf. Section 2.5), in which case the output model is excluded from the leakage profile and the adversary learns nothing beyond the public parameters and the input of the parties in I (and anything efficiently computed therefrom). Following [28] we define correctness with respect to a subset $\mathcal{T}$ of inputs (where there is no correctness guarantee for inputs not in $\mathcal{T}$). Formally,

Definition 4.1 (Secure Iterated Ridge Implementation)**.** Let $m, k \in \mathbb{N}$, let κ be a security parameter, let $\mathcal{D}, \mathcal{R}$ be an arbitrary domain and range, let $f : \mathcal{D}^m \to \mathcal{R}$ be an iterated ridge algorithm (e.g., the algorithm of Figure 1), and let $\mathcal{T} \subseteq \mathcal{D}^m$. We say that an $(m+2)$-party protocol Π is a *secure iterated ridge implementation* of f with k-privacy for inputs in $\mathcal{T}$ with leakage profile $\mathcal{L}$ if:

(1) **Correctness:** there exists a negligible function $\mathsf{negl}(\kappa) : \mathbb{N} \to \mathbb{N}$ such that for all inputs $(x_1, \dots, x_m) \in \mathcal{T}$,
$$\Pr[\Pi(x_1, \dots, x_m) = f(x_1, \dots, x_m)] = 1 - \mathsf{negl}(\kappa)$$
where the probability is over the randomness of the parties.
(2) **Privacy:** for every k-permissible I there exists a PPT simulator Sim such that for every $(x_1, \dots, x_m) \in \mathcal{T}$:
$$\mathsf{V}_I^{\Pi}(x_1, \dots, x_m) \approx \mathsf{Sim}\left((x_j)_{\mathsf{DO}_j \in I}, \mathcal{L}\right).$$

5 SIR PROTOCOL

Our privacy preserving iterated ridge protocol SIR is specified in Figures 2-3 (with further details available in in the full version). See also an overview in Section 2; remarks on input encoding, parameter choice, and useful observations in Section 5.1. Our security and complexity analysis of SIR is summarized below. Complexity is stated in term of d and N where $\log N = \widetilde{O}(d \log n)$ (by Equation 4).

Theorem 5.1 (SIR analysis). *Let $m, n, d \in \mathbb{N}$, $X \in \mathbb{R}^{n\times d}$ and $\vec{y} \in \mathbb{R}^{n\times 1}$ s.t. X has full rank and $d \le n$. Then, the following holds when executing SIR on $(X, \vec{y})$ when horizontally partitioned amongst m data-owners:*

Security. *SIR (Figure 2) is a secure iterated ridge implementation of IR (Figure 1) with m-privacy.*

Complexity. *Let $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ be the homomorphic encryption scheme with which SIR is instantiated, and $\mathbb{Z}_N$ be the used plaintext ring, then:*

- *Each data owner runtime is dominated by the time to compute d^2 encryptions, and her communication complexity is dominated by transmitting d^2 ciphertexts (in one round).*
- *$\mathcal{S}_1$ runtime is dominated by the time to rank features, which entails homomorphically evaluating $O(\log d)$ circuits, each*

with $O(d^2 \log N) = \widetilde{O}(d^3 \log n)$ multiplication gates and of multiplicative depth $O(\log\log N) = O(\log d + \log\log n)$.[11]

- *$\mathcal{S}_2$ runtime complexity is dominated by the time to solve (in the clear) $O(\log d)$ linear systems of size $d \times d$.*
- *The communication of the two servers consists of $O(\log d)$ communication rounds, transmitting $O(d^2 \log N) = \widetilde{O}(d^3 \log n)$ ciphertexts in each round.*[12]

See the full version [2] for the proof.

5.1 Input, Parameters and Observations

Remarks clarifying some implementation details follow.

Remark 5.2 (Input representation and encoding.). We assume that the datasets entries are in the range $[-1, 1]$, given with ℓ-digit precision. The inputs are scaled to be in $\mathbb{Z}_N$ for a sufficiently large N (for the choice of N, see Remark 5.3). All subsequent computations in the protocol are performed in $\mathbb{Z}_N$ or in $\mathbb{Z}_D$ for some $D \geq d$. Note that D is much smaller than N. All inputs (and intermediate values generated during the computation) are encoded as in [4]. (We refer the interested reader to [4] for a detailed description of the encoding and its efficiency benefits.)

Remark 5.3 (On the choice of N). The plaintext ring $\mathbb{Z}_N$ should be sufficiently large to guarantee that all computations during the (scaled) ridge regression step emulate the corresponding computations over the reals (i.e., no overflows occur), as well as to allow for rational reconstruction to be performed on the output model at the end of the protocol. This can be guaranteed by using the same plaintext ring $\mathbb{Z}_{N_{\text{Gia}}}$ as [28]. However, for our selection protocol we will need the modulo N to be at least square that value, namely:

$$N = N_{\text{Gia}}^2 > \left(2d(d-1)^{\frac{d-1}{2}} 10^{4\ell d} (n^2 + \lambda)^{2d}\right)^2 \quad (4)$$

where n, d, ℓ, λ are as specified in Section 4.

Remark 5.4 (On the choice of the plaintext modulus D in SIR.). For efficiency reasons, we would like to avoid (when possible) performing computations in the ring $\mathbb{Z}_N$, since such computations would be heavy due to the size of N (as described above). Instead, in SIR we are able to use a smaller modulus D for some computations. We can make due with any $D \geq d$ which can be used as a plaintext modulus in the underlying FHE scheme.

Remark 5.5 (Dimension reduction and projection). Our protocol iteratively reduces the set F of current features (i.e., ones that will be part of the output model), which is done in two steps as follows. (1) reset the entries of $A, \vec{b}$ that correspond to entries in $[d] \setminus F$, by setting to zero the rows and columns of A (the entries of $\vec{b}$, respectively) that are indexed by $i \notin F$, resulting in a matrix A' and a vector $\vec{b}'$. (2) projecting $A', \vec{b}'$ to F by erasing the rows and columns of A' (entries of $\vec{b}$, respectively) indexed by $i \notin F$. We denote this operation by $\text{pjct}_F(\cdot)$ (this operation can be applied to a matrix or a vector), namely we compute $\text{pjct}_F(A')$, $\text{pjct}_F\left(\vec{b}'\right)$. Step (1) is obtained by multiplying with a *nullifier matrix* $\mathcal{N}_F$ which is defined as follows: $\mathcal{N}_F \in \mathbb{Z}_N^{d\times d}$ is obtained from the $d \times d$ identity matrix by resetting the diagonal entries in all rows indexed by $i \notin F$ (we omit d from the notation, since it is clear from the context). More specifically, we set $A' = \mathcal{N}_F \cdot A \cdot \mathcal{N}_F$ and $\vec{b}' = \mathcal{N}_F \cdot \vec{b}$. Notice that for any matrix X, multiplying by $\mathcal{N}_F$ from the left (right, respectively) rests the rows (columns, respectively) indexed by $i \notin F$, and similarly when multiplying a vector $\vec{v}$ by $\mathcal{N}_F$ from the left. Another operation which will be used in our protocols is an *expansion* from dimension F to dimension $[d]$. Specifically, we define $\text{expd}_F(\cdot)$ such that on input an $|F| \times |F|$ matrix X (a length-$|F|$ vector $\vec{v}$, respectively) returns the $d \times d$ matrix X' (length-d vector $\vec{v}'$, respectively) such that for every $i \notin F$ the ith row and column in X' (ith entry in $\vec{v}'$, respectively) is 0, and additionally $X = \text{pjct}_F(X')$, $\vec{v} = \text{pjct}_F(\vec{v}')$.

Remark 5.6 (Unique Entries in Intermediate Models). Our security analysis will rely on the assumption that for every intermediate model $\vec{z}_F$ computed in Step 3a of the SIR protocol (Figure 2), all entries are unique (i.e., if $i \neq j$ then $\vec{z}_{F,i} \neq \vec{z}_{F,j}$). This can be easily achieved as follows. First, when scaling the inputs in the setup phase, we incorporate $\log d$ additional "empty" least-significant bits. That is, instead of scaling an ℓ-precision real number by multiplying it by 10^{ℓ}, we multiply it by $10^{\ell+\log d}$. Then, at the end of each scaled ridge regression iteration (Figure 3) we replace the $\log d$ least-significant bits of $\vec{z}_{F,i}$ with the binary representation of i (this can be done because at this point the ciphertext is encrypted entry-by-entry).

Remark 5.7 (Emulating Boolean circuits using arithmetic circuits). Our protocols embed Boolean values into a larger ring $\mathbb{Z}_D$, and operate over these representations. This is done as follows. $a \wedge b$ is implemented by multiplying $a \cdot b$ in $\mathbb{Z}_D$. $a \oplus b$ is implementing by computing $(a - b)^2$ in $\mathbb{Z}_D$. This perfectly emulates AND and XOR whenever $a, b \in \mathbb{Z}_D \cap \{0, 1\}$. We negate a Boolean value c by computing $1 - c$ (where 1 is the identity of $\mathbb{Z}_D$).

6 SYSTEM AND EMPIRICAL EVALUATION

We implemented the SIR protocol into a system and ran experiments on real data to evaluate its concrete complexity and correctness (i.e., that output models match cleartext results). Furthermore, we compare the iterated ridge approach (as securely realized in SIR) to the filter and truncation approaches for feature selection, showing it significantly outperforms the latter (see details in the full version).

6.1 Data

The Cancer Genome Atlas (TCGA), a landmark cancer genomics program, molecularly characterized over 20,000 primary cancer and matched normal human tissue samples spanning 33 cancer types. The program integrates contributions from many researchers coming from diverse disciplines and from multiple institutions. The data spans genomic, epigenomic, transcriptomic, and proteomic data measured on the aforementioned samples. We used a small portion of this data for our experiments. Concretely we used randomly selected portions of one of the breast cancer transcriptomics data matrices. We start with a matrix with features normalized to lie in $[-1, 1]$ with 3-digit precision. The matrix has 781 rows (samples/instances) and $> 10K$ columns (features, representing genes

[11] We ignore addition gates, since additive homomorphism is much faster than multiplicative homomorphism in practice. We note that the masking step includes matrix multiplication, which has complexity cubic in d, but since it entails only additive homomorphism it is not accounted for in the complexity analysis.

[12] When using a homomorphic encryption that supports packing sl plaintext values in each ciphertext with support for single instruction multiple data (SIMD) computation, the time and communication complexity of the servers can be divided by sl.

Public parameter: an FHE scheme $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$, a security parameter κ, a regularization parameter λ, dimensions $n \times d$ of the input matrix, a plaintext ring size N satisfying the requirements of Remark 5.3, the number of data owners m, positive input sizes $n_1, n_2, \ldots, n_m > 0$ such that $\sum_{j=1}^{m} n_j = n$, a sparsity parameter $s < d$, a precision parameter ℓ, a sparsity parameter rej which determines the fraction of features that are removed in each iteration, and a threshold parameter thr which determines when the protocol starts to remove a single feature in each iteration. Additionally, let $D \geq d$ (see Remark 5.4).
Inputs: For every $j \in [m]$, the input of data owner DO_j consists of a data matrix $X^j \in \mathbb{R}^{n_j \times d}$ and a response vector $\vec{y}^j \in \mathbb{R}^{n_j}$. We denote by $(X|\vec{y})$ the combined input obtained from all $X^j, \vec{y}^j$. That is, $[n]$ is partitioned into m subsets $I_1, \ldots, I_m \subseteq \{1, \ldots, n\}$, and $X^j, \vec{y}^j$ is the restriction of $X, \vec{y}$ to the rows in I_j. (Here, X and $\vec{y}$ are scaled to lie in $\mathbb{Z}$, and then embedded in $\mathbb{Z}_N$ for a sufficiently large N, see Remark 5.3.) The servers $\mathcal{S}_1, \mathcal{S}_2$ have no input.
Output: all parties obtain as output an s-sparse model $\vec{w} \in \mathbb{R}^d$.
Steps:

1. **Setup:** The parties execute the setup protocol (described in the full version [2]) to obtain keys $(\mathsf{pk}_N, \mathsf{sk}_N)$ and $(\mathsf{pk}_D, \mathsf{sk}_D)$. Then, the parties execute the data uploading, merging and permuting protocols (described in the full version), and $\mathcal{S}_1$ obtains encryptions of the aggregated input matrix and response vector $A, \vec{b}$.
2. Let $F = [d]$ (i.e., initially F contains all features), and let $A_F = A, \vec{b}_F = \vec{b}$.
3. While $|F| > s$, do:
   - (a) **Ridge Regression Iteration:** $\mathcal{S}_1$ and $\mathcal{S}_2$ execute the scaled ridge protocol of Figure 3, where the input of $\mathcal{S}_1$ consists of F, encryptions $\mathbf{A}_F, \vec{\mathbf{b}}_F$ of $A_F, \vec{b}_F$, respectively, and the input of $\mathcal{S}_2$ is F, sk_N. The output of $\mathcal{S}_1$ is an entry-wise encryption $\vec{\mathbf{z}}_F$ of a vector $\vec{z}_F \in \mathbb{Z}_N^d$ under key pk_N.
   - (b) **Selecting features:**
     - **Large-set case:** If $|F| > \mathsf{thr}$ then set $k' = \lfloor \mathsf{rej} \cdot |F| \rfloor$. (Intuitively, when $|F| > \mathsf{thr}$ the set of current features is still sufficiently large that we can remove a subset of features in each iteration.)
     - **Small-set case:** Otherwise (i.e., $|F| \leq \mathsf{thr}$), set $k' = 1$. (In this case, the set of current features is small, so features should be removed one at a time.)
     - $\mathcal{S}_1$ and $\mathcal{S}_2$ execute the selection protocol (see the full version [2]) to find the smallest k' features. $\mathcal{S}_1$ has input $\vec{\mathbf{z}}_F$, $\mathcal{S}_2$ has input sk_N, and both parties have input $\mathsf{pk}_N, \mathsf{pk}_D, F, k'$. The output of both servers is the set S_{del} consisting of the k' features to be removed.
   - (c) **Compacting data for the next iteration:** Let $F^* = F \setminus S_{\mathsf{del}}$. Then $\mathcal{S}_1$ locally executes the compacting algorithm (described in the full version [2]), with input $F^*, \mathsf{pk}_N, \mathbf{A}$ and $\vec{\mathbf{b}}$ (encrypting A and $\vec{b}$, respectively). The output of $\mathcal{S}_1$ are updated (compacted) $\mathbf{A}_{F^*}, \vec{\mathbf{b}}_{F^*}$.
   - (d) set $F = F^*$.
4. **Output:** Parties execute the computing and unpermuting output protocol (described in the full version [2]) to obtain the s-sparse model $\vec{w}$.

Figure 2: SIR: Secure Iterated Ridge

The protocol uses the circuits $\mathsf{MatMultR}_M$, $\mathsf{VecMatMultR}_{\vec{r}}$, $\mathsf{MatVecMultL}_M$, Add, Sub, $\mathsf{VecScalerMult}_{\vec{r}}$ and $\mathsf{ScalerVecMult}_c$ of Section 5.1.
Public parameters: $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$, d, N, as in Figure 2.
Inputs from previous phases: the public encryption key pk_N, the set $F \subseteq [d]$ of feature indices that "survived" to the current iteration, and encryptions $\mathbf{A}_F, \vec{\mathbf{b}}_F$ of the matrix $A_F = \mathcal{N}_F \cdot A \cdot \mathcal{N}_F \in \mathbb{Z}_N^{d \times d}$ and vector $\vec{b}_F = \mathcal{N}_F \cdot \vec{b} \in \mathbb{Z}_N^d$ (see Remark 5.5 for the description of the nullifier matrix $\mathcal{N}_F$; and recall that intuitively, $A_F, \vec{b}_F$ are obtained from $A, \vec{b}$ by resetting the rows, columns, and entries for $i \notin F$). $\mathcal{S}_2$ additionally has as input the private decryption key sk_N.
Output: the output of $\mathcal{S}_1$ is an encryption $\vec{\mathbf{z}}$, under key pk_N, of $\vec{z} \in \mathbb{Z}_N^d$ such that $\vec{z} = \det\left(A'_F\right) \cdot \vec{w}_F$, where: (1) $A'_F = \mathsf{pjct}_F\left(A_F\right)$ is the projection of A_F to the indices in F; (2) $A'_F \cdot \vec{w}'_F = \mathsf{pjct}_F\left(\vec{b}_F\right)$, and additionally (3) $\vec{w}_F = \mathsf{expd}_F\left(\vec{w}'_F\right)$. (See Remark 5.5 for a description of $\mathsf{expd}_F(\cdot)$ and $\mathsf{pjct}_F(\cdot)$.) The other parties have no output.
Steps:

1. **Masks Generation:** $\mathcal{S}_1$ picks a random invertible matrix $R' \leftarrow \mathrm{GL}\left(|F|, \mathbb{Z}_N\right)$, and a random vector $\vec{r}' \leftarrow \mathbb{Z}_N^d$. Then, $\mathcal{S}_1$ computes $R = \mathsf{expd}_F\left(R'\right), \vec{r} = \mathcal{N}_F \cdot \vec{r}'$.
2. **Data masking:** $\mathcal{S}_1$ masks the data by homomorphically computing $\Gamma_F = A_F \cdot R$ and $\vec{\beta}_F = \vec{b}_F + A_F \cdot \vec{r}$ as follows:
   - $\mathbf{\Gamma}_F \leftarrow \mathsf{Eval}\left(\mathsf{pk}_N, \mathsf{MatMultR}_R, \mathbf{A}_F\right)$.
   - $\vec{\boldsymbol{t}} \leftarrow \mathsf{Eval}\left(\mathsf{pk}_N, \mathsf{VecMatMultR}_{\vec{r}}, \mathbf{A}_F\right)$.
   - $\vec{\boldsymbol{\beta}}_F \leftarrow \mathsf{Eval}\left(\mathsf{pk}_N, \mathsf{Add}, \vec{\mathbf{b}}_F, \vec{\boldsymbol{t}}\right)$.

   Then, $\mathcal{S}_1$ sends $\mathbf{\Gamma}_F, \vec{\boldsymbol{\beta}}_F$ to $\mathcal{S}_2$. (Notice that for every $i \notin F$, the ith row and column in Γ_F is $\vec{0}$, and similarly the ith entry of $\vec{\beta}_F$ is 0.)
3. **Decrypting Masked Data:** $\mathcal{S}_2$ decrypts $\Gamma_F = \mathsf{Dec}\left(\mathsf{sk}_N, \mathbf{\Gamma}_F\right)$ and $\vec{\beta}_F = \mathsf{Dec}\left(\mathsf{sk}_N, \vec{\boldsymbol{\beta}}_F\right)$. Then, $\mathcal{S}_1$ projects $\Gamma_F, \vec{\beta}_F$ to the indices in F by computing $\Gamma = \mathsf{pjct}_F\left(\Gamma_F\right) \in \mathbb{Z}_N^{|F| \times |F|}$ and $\vec{\beta} = \mathsf{pjct}_F\left(\vec{\beta}_F\right) \in \mathbb{Z}_N^{|F|}$.
4. **Masked learning:** $\mathcal{S}_2$ computes $\mathrm{adj}\left(\Gamma\right)$ and $\Delta = \det\left(\Gamma\right)$, as well as $\vec{\zeta} = \mathrm{adj}\left(\Gamma\right) \cdot \vec{\beta}$. Then, $\mathcal{S}_2$ computes $\vec{\zeta}_F = \mathsf{expd}_F\left(\zeta\right) \in \mathbb{Z}_N^d$. Finally, $\mathcal{S}_2$ encrypts $\vec{\boldsymbol{\zeta}}_F \leftarrow \mathsf{Enc}\left(\mathsf{pk}_N, \vec{\zeta}_F\right)$ and $\mathbf{\Delta} \leftarrow \mathsf{Enc}\left(\mathsf{pk}_N, \Delta\right)$, and sends $\vec{\zeta}_F, \mathbf{\Delta}$ to $\mathcal{S}_1$. (Intuitively, $\mathcal{S}_2$ solves the linear system $\Gamma \cdot \vec{\omega} = \vec{\beta}$ to obtain the masked model $\vec{\omega}$. Notice that $\vec{\zeta} = \det\left(\Gamma\right) \cdot \vec{\omega}$.)
5. **Unmasking:** $\mathcal{S}_1$ homomorphically computes $\vec{z} = \det\left(A'_F\right) \cdot \vec{w}_F$, where $A'_F = \mathsf{pjct}_F\left(A_F\right)$. This is done by performing the following:
   - $\vec{\boldsymbol{t}}^1 \leftarrow \mathsf{Eval}\left(\mathsf{pk}_N, \mathsf{MatVecMultL}_R, \vec{\boldsymbol{\zeta}}_F\right)$. (Notice that $\vec{t}^1 = R \cdot \vec{\zeta}_F$ so $\vec{t}^1 = \det\left(\Gamma\right) \cdot \left(\vec{w}_F + \vec{r}\right)$.)
   - $\vec{\boldsymbol{t}}^2 \leftarrow \mathsf{Eval}\left(\mathsf{pk}_N, \mathsf{VecScalerMult}_{\vec{r}}, \mathbf{\Delta}\right)$. (Notice that $\vec{t}^2 = \det\left(\Gamma\right) \cdot \vec{r}$.)
   - $\vec{\boldsymbol{t}}^3 \leftarrow \mathsf{Eval}\left(\mathsf{pk}_N, \mathsf{Sub}, \vec{\boldsymbol{t}}^1, \vec{\boldsymbol{t}}^2\right)$. (Notice that $\vec{t}^3 = \vec{t}^1 - \vec{t}^2 = \det\left(\Gamma\right) \cdot \vec{w}_F$.)
   - $\vec{\mathbf{z}} \leftarrow \mathsf{Eval}\left(\mathsf{pk}_N, \mathsf{ScalerVecMult}_{\det(R')^{-1}}, \vec{\boldsymbol{t}}^3\right)$. (Notice that $\vec{z} = \det\left(R'\right)^{-1} \cdot \vec{t}^3$, so $\vec{z} = \det\left(A'_F\right) \cdot \vec{w}_F$, because $\Gamma = A'_F \cdot R'$ so $\det\left(\Gamma\right) = \det\left(R'\right) \cdot \det\left(A'_F\right)$.)
6. $\mathcal{S}_1$ sets $\vec{\mathbf{z}}$ to be its output for the phase.

Figure 3: Scaled ridge regression (single iteration).

profiled). Each (i, j) entry of the matrix represents the expression level of gene j in sample i. We use $\mathcal{D}$ to denote this full TCGA breast cancer expression profiling matrix. The TCGA data also includes 781 TIL levels for this cohort, as part of additional data to support biological and clinical interpretation. TIL levels quantify tumor infiltrating immune cells in the (tumor) samples.

To run an experiment with d features, that is adequate for our current running time complexity, we chose to work with 4, 10, 40 and 100 features. To generate a dataset for a given d, we randomly (uniformly) selected d features (columns of $\mathcal{D}$) to be the columns of the data matrix X. We then set the target vector $\vec{y}$ to be the vector of TIL levels. To support *bias*, we add an extra column of 1's to X. When relevant, we evenly distributed the 781 samples between the data owners. As described in the protocol, each data owner computed $A = \lfloor 10^{\ell} X^T X \rceil$ and $\vec{b} = \lfloor 10^{\ell} X^T \vec{y} \rceil$, which also scaled and rounded the values. Note that in our experiments on d features, A and $\vec{b}$ are therefore of dimensions $(d+1) \times (d+1)$ and $(d+1)$, respectively. Solving for $\vec{y}$ continues in the same way as in Protocol 2 (computing adjugate and determinant of a $(d+1) \times (d+1)$ matrix), but computing the ranks of the features involves only d features because the bias is not treated as a selectable feature.

6.2 System Implementation Details

Ring size. To obtain correctness in SIR, the ring sizes needed for our experiments exceeded the sizes currently supported by FHE libraries. For example, for inputs with 40 features, we require a ring of size 1,260 bits, while current libraries support rings of size less than 64 bits. To support such large ring sizes, we encoded the input using the Chinese Remainder Theorem (CRT) with several distinct 30-bit primes, as was used in [4].

Permuting the input. In a preliminary step $\mathcal{S}_1$ and $\mathcal{S}_2$ participate in a protocol to permute the input: $A = (P_1 \cdot P_2)^T \cdot T^1 \cdot (P_1 \cdot P_2)$ (for T^1 an intermediate value), where P_1 and P_2 are random permutation matrices chosen by $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively. There are several ways to compute this step. One option is for $\mathcal{S}_1$ to perform all the computations using FHE (with multiplicative depth 2). Another option – which is the one used in our implementation – is to utilize a protocol between $\mathcal{S}_1$ and $\mathcal{S}_2$ that uses masking and requires only additive homomorphism. This step is executed only once at the onset of the protocol, and its running time is inconsequential compared to the total protocol runtime (e.g., 1,152 out of 84,690 seconds on 40 features, see Table 1).

Ranking weights. To decide which entries of $\vec{w}$ to remove in each iteration, we compute a full ranking over the entries. This involves comparing pairs of entries. Our experiments show that the majority of the running time is spent in this step. For example, the total real time of SIR when $d = 40$ was 84,690 seconds, while computing the ranks took 76,184 seconds (see Table 1). To Compare a single pair w_i, w_j of entries, $\mathcal{S}_1$ compares $w_i^2 - w_j^2 + r_{ij}$ to ranges received from $\mathcal{S}_2$, to determine whether $w_i^2 - w_j^2 \in [0, N/2]$ (and hence $w_i \geq w_j$). In our implementation $\mathcal{S}_1$ performed all $\binom{d}{2}$ comparisons. An alternative is to use an approach similar to Bitonic sort (or Batcher sort, see [5]). However, while a sorting approach performs only $\frac{d}{2}\binom{\lceil \log_2 d \rceil}{2}$ comparisons, it is less amenable to parallelization. Specifically, the sorting approach is faster when $\lceil \frac{1}{\text{CPUs}}\binom{d}{2} \rceil > \lceil \frac{d/2}{\text{CPUs}} \rceil \cdot \binom{\lceil \log_2 d \rceil}{2}$, so for the number of features and CPUs in our experiments the naive all-pairs approach was faster. We stress that to utilize the Bitonic sort alternative (which would be faster for larger d's), one only needs to implement the comparison step using Bitonic sort, making our protocol flexible, efficiently supporting both regimes of d.

Cryptographic Libraries. At a high level, the steps of our protocol can be categorized into two types with different characteristics:

- *Ranking steps.* Computing the ranks of features in $\vec{w}$, when the entries are given in binary. These steps involve a subcircuit of large depth for sorting d large numbers in binary representation, e.g. 1,260-bit numbers when $d = 40$. This requires a key that supports large-depth circuits, and possibly also bootstrapping. We note that the plaintext modulo needed by these steps is relatively small.
- *CRT steps* consist of all other steps, and operate over numbers that are represented using CRT. The CRT steps of the protocol require only *additive* homomorphism, but necessitate multiple (co-prime) plaintext moduli to implement the CRT. Preferably, these plaintext moduli should be as large as possible, because larger moduli mean less elements in the CRT encoding.

We used two different FHE libraries (with different schemes) to implement these two types of steps. For ranking we use BGV in HElib [38] 2.1.0, because it supports bootstrapping (unlike SEAL). For CRT steps we use B/FV in SEAL [54] 4.0, in which it is easier to configure keys for multiple plaintext moduli. Switching between the two libraries and schemes is done by $\mathcal{S}_2$, who generated the keys (pk_N, sk_N) and (pk_D, sk_D) in HElib and SEAL respectively. In detail, in HElib [38] 2.1.0 we initialize the keys while setting the plaintext modulo to 17^2, and the cyclotomic polynomial degree to 78,881. This resulted in ciphertexts with 7,000 slots, supporting a multiplicative depth of 28, as well as bootstrapping. In SEAL we initialize the keys while setting the cyclotomic polynomial degree to 8,192, a chain length of 7, chain bits of 25 and prime bits of 30. This resulted in ciphertexts with 4,096 slots that support additive homomorphism.

6.3 Evaluation Setup

We executed SIR (Protocol 2) on the data generated from TCGA described in Section 6.1, and measured performance of the data owners and servers. We executed two types of experiments: end-to-end and single-iteration experiments. In all experiments rej was set to 10% and the ring size N was determined by Equation 4.

End-to-end experiments measure performance when executing the entire SIR protocol, including the data encoding and all iterations, until producing a sparse model that selects s out of the d initial features from a dataset of n records distributed amogst m data owners. We ran end-to-end experiments on the following m, n, d, s parameters: (1) TCGA data with number of features (d, s) varying between $(4, 2)$, $(10, 4)$, $(40, 8)$ and $(m, n) = (10, 781)$; (2) Synthetic data with number of records n =1568, 6272, 12544, 50176, 100352, 200704, 401408, 802816 and $(m, d, s) = (10, 10, 2)$; (3) Synthetic data with number of data owners $m = 100, 200, \ldots, 1000$ and $(n, d, s) = (802816, 10, 2)$.

Single-iteration experiments measure performance when executing a single iteration (Protocol 2, Step 3, i.e., computing scaled ridge, ranking the features, and removing the features with the smallest

(d, s, N)	RAM	encrypt	permute	mask	unmask	decrypt	solve	pair diffs	ranges	ranks	total
$(4, 2, 2^{180})$	21	1 (×2)	11 (×84)	1 (×100)	5 (×78)	1 (×100)	7 (×70)	1 (×10)	7 (×4)	1510 (×5)	2142 (×4)
$(10, 4, 2^{360})$	43	80 (×2)	105 (×80)	3 (×100)	41 (×60)	11 (×87)	56 (×70)	2 (×24)	23 (×17)	4540 (×24)	5718 (×21)
$(40, 8, 2^{1260})$	134	80 (×2)	1152 (×52)	76 (×58)	1408 (×35)	152 (×92)	1584 (×40)	66 (×77)	336 (×65)	76184 (×79)	84690 (×74)

Table 1: Runtime (seconds), memory consumption (GB), and parallelization ratio (in parenthesis) in end-to-end SIR executions.

(d, N)	mask	unmask	decrypt	solve	pair diffs	ranges	ranks
$(5, 2^{210})$	2 (×29)	3 (×42)	1 (×7)	3 (×61)	1 (×3)	5 (×4)	806 (×10)
$(10, 2^{360})$	1 (×100)	20 (×53)	4 (×80)	24 (×64)	1 (×10)	11 (×25)	939 (×43)
$(15, 2^{510})$	3 (×57)	44 (×44)	1 (×130)	42 (×40)	1 (×23)	39 (×47)	2319 (×63)
$(20, 2^{660})$	5 (×46)	77 (×34)	4 (×100)	72 (×29)	1 (×51)	45 (×58)	4147 (×87)
$(25, 2^{810})$	5 (×56)	139 (×32)	4 (×86)	158 (×38)	1 (×97)	38 (×67)	7104 (×90)
$(30, 2^{960})$	6 (×64)	274 (×36)	26 (×91)	323 (×44)	2 (×65)	55 (×63)	13093 (×87)
$(35, 2^{1110})$	7 (×57)	371 (×29)	9 (×90)	394 (×32)	3 (×72)	57 (×63)	11728 (×95)
$(40, 2^{1260})$	9 (×58)	473 (×30)	24 (×91)	519 (×34)	5 (×72)	54 (×70)	19354 (×91)

Table 2: Runtime (seconds) in a single SIR iteration, and parallelization ratio (in parenthesis).

(d, N)	n	solve	ranks
$(10, 2^{360})$	1568	97	7407
$(10, 2^{420})$	6272	112	7391
$(10, 2^{450})$	12544	113	7373
$(10, 2^{480})$	50176	114	7367
$(10, 2^{510})$	100352	112	7312
$(10, 2^{540})$	200704	126	8166
$(10, 2^{540})$	401408	133	8209
$(10, 2^{570})$	802816	139	8048

Table 3: Runtime (seconds) of SIR on a growing number of records (n).

weight) on different values of d. Concretely, we take $5 \leq d \leq 40$ features, in increments of 5.

Hardware. In all experiments, we used a single virtual machine to simulate the data owners, $\mathcal{S}_1$ and $\mathcal{S}_2$. The virtual machine had 100 Xeon 2.7GHz CPUs and 900 GigaBytes of RAM. These are off-the-shelf standard CPUs. Nonetheless, each data owner was executed on a *single CPU*, to capture the prevalent usecase in which data owners are computationally significantly weaker than the servers.

What we measured. We report the total runtime, as well as the runtime of each sub-task in SIR: *encrypt*; *permute*; *mask* (Figure 3, Step 2); *unmask* (Figure 3, Step 5); *decrypt* (Figure 3, Step 3);[13] *solve* (Figure 3, Step 4);[14] *pair diff*; *ranges*; *ranks.* Furthermore, we report the *parallelization ratio*, which is the ratio between runtime when executing the computation on a single CPU (as measured by the operating system) vs. the runtime on our 100-cores system. Intuitively, this indicates the average number of CPUs that were busy performing the task, where a higher ratio means the task is more amenable to parallelization as it utilize more CPUs (the maximum being 100).

6.4 SIR Performance

Tables 1-2 summarize the performance exhibited in the end-to-end experiments and the single-iteration experiments, respectively. Table 3 presents SIR runtime on various database sizes.

Runtime. The total runtime is dominated by the time it takes to rank the weights. For example, ranking took 91% (94%) of the total runtime in our end-to-end (single-iteration) experiment on $d = 40$. Increasing the number of records by 512× (from 1568 to 802,816 records) led to runtime increase by less than 10% (from 7,504 to 8,187 seconds). Increasing the number of data owners by 10× (from 100 to 1000 data owners) affects only the time to homomorphically merge the data, that indeed increased by ten-fold (from 10 to 102 seconds); however this has only a minor influence on the overall runtime (which is dominated by the 8048 seconds to rank features), and so a ten-fold increase in the number of data owners led to an increase in the total runtime by roughly 1%.

RAM. The RAM usage of our system was significantly lower than the allocated RAM, ranging from 21GB to 134GB. Furthermore, our experiments indicate that the RAM requirement grows sub linearly in the measured values for d (e.g. from 43GB when $d = 10$ to 134GB when $d = 40$). This is because our ciphertexts had more slots than needed for our encodings (for small d's), so the total number of ciphertexts grew only mildly in d.

Parallelization. As d grows, most tasks become more parallelizeable. In particular, the ranking task –which dominates the bulk of the runtime– is "embarrassingly" parallelizable since we make all $\binom{d}{2}$ comparisons, which can be executed in parallel. We note that if we had an *unlimited number of CPUs*, the ranking runtime would essentially be the time of a single comparison. For example, when $(d, N) = (40, 2^{1260})$ (cf. Table 2 bottom row), our ranking utilized an average of 91 CPUs (the system had 100 CPUs), whereas having access to $\binom{d}{2}$ CPUs is expected to improved the ranking runtime by a factor of $\binom{40}{2}/91 \approx 8.6$, thus improving the total runtime by 6×.

We remark that an alternative way to compute the ranking would be to homomorphically evaluate an oblivious sorting algorithm (e.g., bitonic sort). Bitonic sort would require less comparisons – $O(d \log^2 d)$; but this type of sorting admits a circuit structure having $O(\log^2 d)$ layers, which is less amenable to parallelization. For 40 features and 100 CPUs, this alternative would have higher runtime.

7 CONCLUSIONS

We present a privacy-preserving multi-party protocol for running sparse linear regression in a federated learning setup, based on an iterated ridge framework. Our protocol enjoys rigorous security, and scales favorably with the number of records and data owners.

[13]Recall that we encoded the numbers using CRT; the time reported here includes the CRT decoding time.
[14]Here the computations are over $\mathbb{Z}_N$, where N is large (e.g., 1, 260-bits for 40 features).

Moreover, our protocol naturally gives a privacy-preserving ridge truncation protocol, which is less accurate (as we have shown), but simpler and faster, and therefore may be preferred in some cases.

The design of our protocol is based, amongst other consideration, on certain potential attacks that can be developed when partial or intermediate information is leaked. In particular, we show that revealing the order in which features are removed can be used to infer non-trivial information about the input data. We also extend this attack to other leakages such as revealing intermediate models or the determinant of the intermediate A matrices. SIR is susceptible to model inversion attacks (cf. Section 2.5); determining the exact extent to which such attacks are harmful, and devising measures to protect against them, are left for future work.

We mostly focus on the case $d \leq n$. In particular, our security proof addresses this case and furthermore requires the matrix A to be invertible. Our protocol can be adjusted for settings with $d > n$ by combining SIR with a faster learning method (e.g., filter) to first partially reduce the number of features. One can similarly perform a preparatory step to handle the case in which X is not full rank.

ACKNOWLEDGMENTS

We thank the anonymous POPETs reviewers for their helpful comments. The first author was supported in part by the Israel Science Foundation grant 3380/19, and the Israel National Cyber Directorate via Haifa, BIU and Tel-Aviv Cyber Centers. The results described in this manuscript are partially based upon data generated by the TCGA Research Network: https://www.cancer.gov/tcga.

REFERENCES

[1] Mark Abspoel, Daniel Escudero, and Nikolaj Volgushev. 2021. Secure training of decision trees with continuous attributes. *Proc. Priv. Enhancing Technol.* 2021, 1 (2021), 167–187.

[2] Adi Akavia, Ben Galili, Hayim Shaul, Mor Weiss, and Zohar Yakhini. 2023. Privacy Preserving Feature Selection for Sparse Linear Regression. *IACR Cryptol. ePrint Arch.* 2023, 1254 (2023). https://eprint.iacr.org/2023/1354

[3] Adi Akavia, Max Leibovich, Yehezkel S Resheff, Roey Ron, Moni Shahar, and Margarita Vald. 2022. Privacy-preserving decision trees training and prediction. *ACM Transactions on Privacy and Security* 25, 3 (2022), 1–30.

[4] Adi Akavia, Hayim Shaul, Mor Weiss, and Zohar Yakhini. 2019. Linear-Regression on Packed Encrypted Data in the Two-Server Model. In *WAHC@CCS'19*. ACM, 21–32.

[5] Selim G. Akl. 2011. Bitonic Sort. In *Encyclopedia of Parallel Computing*. Springer, 139–146.

[6] Miriam Ragle Aure, Israel Steinfeld, Lars Oliver Baumbusch, Knut Liestøl, Doron Lipson, Sandra Nyberg, Bjørn Naume, Kristine Kleivi Sahlberg, Vessela N Kristensen, Anne-Lise Børresen-Dale, et al. 2013. Identifying in-trans process associated genes in breast cancer by integrated analysis of copy number and expression data. *PloS one* 8, 1 (2013), e53014.

[7] Amir Ben-Dor, Nir Friedman, and Zohar Yakhini. 2001. Class discovery in gene expression data. In *RECOMB'21*. 31–38.

[8] Shay Ben-Elazar, Miriam Ragle Aure, Kristin Jonsdottir, Suvi-Katri Leivonen, Vessela N Kristensen, Emiel AM Janssen, Kristine Kleivi Sahlberg, Ole Christian Lingjærde, and Zohar Yakhini. 2021. miRNA normalization enables joint analysis of several datasets to increase sensitivity and to reveal novel miRNAs differentially expressed in breast cancer. *PLoS computational biology* 17, 2 (2021), e1008608.

[9] Christopher M. Bishop. 2007. *Pattern recognition and machine learning*. Springer.

[10] Frank Blom, Niek J. Bouman, Berry Schoenmakers, and Niels de Vreede. 2021. Efficient Secure Ridge Regression from Randomized Gaussian Elimination. In *CSCML'21 (Lecture Notes in Computer Science, Vol. 12716)*. Springer, 301–316.

[11] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. 2015. Machine Learning Classification over Encrypted Data. In *NDSS'15*. The Internet Society.

[12] Zvika Brakerski. 2012. Fully homomorphic encryption without modulus switching from classical GapSVP. In *Annual Cryptology Conference*. Springer, 868–886.

[13] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2014. (Leveled) fully homomorphic encryption without bootstrapping. *TOCT* 6, 3 (2014), 1–36.

[14] Thomas M Cover and Jan M Van Campenhout. 1977. On the possible orderings in the measurement selection problem. *Transactions on SMC* 7, 9 (1977), 657–661.

[15] Jack L. H. Crawford, Craig Gentry, Shai Halevi, Daniel Platt, and Victor Shoup. 2018. Doing Real Work with FHE: The Case of Logistic Regression. In *WAHC@CCS'18*. ACM, 1–12.

[16] Jose Cruz, Wilson Mamani, Christian Romero, and Ferdinand Pineda. 2021. Selection of Characteristics by Hybrid Method: RFE, Ridge, Lasso, and Bayesian for the Power Forecast for a Photovoltaic System. *SN Computer Science* 2, 3 (2021), 1–14.

[17] Houjiao Dai, Minhua Lu, Bingsheng Huang, Mimi Tang, Tiantian Pang, Bing Liao, Huasong Cai, Mengqi Huang, Yongjin Zhou, Xin Chen, et al. 2021. Considerable effects of imaging sequences, feature extraction, feature selection, and classifiers on radiomics-based prediction of microvascular invasion in hepatocellular carcinoma using magnetic resonance imaging. *Quantitative imaging in medicine and surgery* 11, 5 (2021), 1836.

[18] Sanmay Das. 2001. Filters, wrappers and a boosting-based hybrid for feature selection. In *Icml*, Vol. 1. Citeseer, 74–81.

[19] Sebastiaan De Hoogh, Berry Schoenmakers, Ping Chen, and Harm op den Akker. 2014. Practical secure decision tree learning in a teletreatment application. In *FC'14*. Springer, 179–194.

[20] David L Donoho, Yaakov Tsaig, Iddo Drori, and Jean-Luc Starck. 2012. Sparse solution of underdetermined systems of linear equations by stagewise orthogonal matching pursuit. *IEEE transactions on Information Theory* 58, 2 (2012), 1094–1121.

[21] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive* (2012).

[22] Pierre-Alain Fouque, Jacques Stern, and Jan-Geert Wackers. 2002. CryptoComputing with Rationals. In *FC'02*. 136–146.

[23] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. 1322–1333.

[24] Matthew Fredrikson, Eric Lantz, Somesh Jha, Simon Lin, David Page, and Thomas Ristenpart. 2014. Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*. 17–32.

[25] Ben Galili, Xavier Tekpli, Vessela N Kristensen, and Zohar Yakhini. 2021. Efficient gene expression signature for a breast cancer immuno-subtype. *Plos one* 16, 1 (2021), e0245215.

[26] Adrià Gascón, Phillipp Schoppmann, Borja Balle, Mariana Raykova, Jack Doerner, Samee Zahur, and David Evans. 2017. Privacy-Preserving Distributed Linear Regression on High-Dimensional Data. *Proc. Priv. Enhancing Technol.* 2017, 4 (2017), 345–364.

[27] Craig Gentry. 2009. *A fully homomorphic encryption scheme*. Stanford university.

[28] Irene Giacomelli, Somesh Jha, Marc Joye, C. David Page, and Kyonghwan Yoon. 2018. Privacy-Preserving Ridge Regression with only Linearly-Homomorphic Encryption. In *ACNS'18 (Lecture Notes in Computer Science, Vol. 10892)*. Springer, 243–261.

[29] Sean M Gibbons, Claire Duvallet, and Eric J Alm. 2018. Correcting for batch effects in case-control microbiome studies. *PLoS computational biology* 14, 4 (2018), e1006102.

[30] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin E. Lauter, Michael Naehrig, and John Wernsing. 2016. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In *ICML'16*. 201–210.

[31] Oded Goldreich. 2004. *The Foundations of Cryptography - Volume 2: Basic Applications*. Cambridge University Press.

[32] Oded Goldreich, Silvio Micali, and Avi Wigderson. 2019. How to play any mental game, or a completeness theorem for protocols with honest majority. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*. 307–328.

[33] Thore Graepel, Kristin Lauter, and Michael Naehrig. 2013. ML Confidential: Machine Learning on Encrypted Data. In *ICISC'12*. Springer-Verlag, Berlin, Heidelberg, 1–21.

[34] Madhuri Gupta and Bharat Gupta. 2021. A novel gene expression test method of minimizing breast cancer risk in reduced cost and time by improving SVM-RFE gene selection method combined with LASSO. *Journal of Integrative Bioinformatics* 18, 2 (2021), 139–153.

[35] Isabelle Guyon and André Elisseeff. 2003. An introduction to variable and feature selection. *Journal of machine learning research* 3 (2003), 1157–1182.

[36] Isabelle Guyon, Jason Weston, Stephen Barnhill, and Vladimir Vapnik. 2002. Gene selection for cancer classification using support vector machines. *Machine learning* 46, 1 (2002), 389–422.

[37] Shai Halevi. 2017. Homomorphic encryption. In *Tutorials on the Foundations of Cryptography*. Springer, 219–276.

[38] Shai Halevi and Victor Shoup. 2014. Algorithms in helib. In *Annual Cryptology Conference*. Springer, 554–571.

[39] Trevor Hastie, Robert Tibshirani, and Ryan Tibshirani. 2020. Best subset, forward stepwise or lasso? Analysis and recommendations based on extensive comparisons. *Statist. Sci.* 35, 4 (2020), 579–592.

[40] Shengshan Hu, Qian Wang, Jingjun Wang, Sherman SM Chow, and Qin Zou. 2016. Securing fast learning! Ridge regression over encrypted big data. In *2016 IEEE Trustcom/BigDataSE/ISPA*. 19–26.

[41] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2013. *An introduction to statistical learning*. Vol. 112. Springer.

[42] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. 2021. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning* 14, 1–2 (2021), 1–210.

[43] Xiling Li, Rafael Dowsley, and Martine De Cock. 2021. Privacy-preserving feature selection with secure multiparty computation. In *ICML'21*. PMLR, 6326–6336.

[44] Yehuda Lindell and Benny Pinkas. 2000. Privacy preserving data mining. In *CRYPTO'00*. Springer, 36–54.

[45] Lin Liu, Rongmao Chen, Ximeng Liu, Jinshu Su, and Linbo Qiao. 2020. Towards practical privacy-preserving decision tree training and evaluation in the cloud. *IEEE Transactions on Information Forensics and Security* 15 (2020), 2914–2929.

[46] Alan Miller. 2002. *Subset selection in regression*. CRC Press.

[47] Pabitra Mitra, CA Murthy, and Sankar K. Pal. 2002. Unsupervised feature selection using feature similarity. *IEEE transactions on pattern analysis and machine intelligence* 24, 3 (2002), 301–312.

[48] Payman Mohassel and Yupeng Zhang. 2017. SecureML: A System for Scalable Privacy-Preserving Machine Learning. In *S&P'17*. 19–38.

[49] Valeria Nikolaenko, Udi Weinsberg, Stratis Ioannidis, Marc Joye, Dan Boneh, and Nina Taft. 2013. Privacy-Preserving Ridge Regression on Hundreds of Millions of Records. In *S&P'13*. 334–348.

[50] In Ja Park, Yun Suk Yu, Bilal Mustafa, Jin Young Park, Yong Bae Seo, Gun-Do Kim, Jinpyo Kim, Chang Min Kim, Hyun Deok Noh, Seung-Mo Hong, Yeon Wook Kim, Mi-Ju Kim, Adnan Ahmad Ansari, Luigi Buonaguro, Sung-Min Ahn, and Chang-Sik Yu. 2020. A Nine-Gene Signature for Predicting the Response to Preoperative Chemoradiotherapy in Patients with Locally Advanced Rectal Cancer. *Cancers* 12, 4 (March 2020), 800.

[51] Yonatan Peleg, Shai Shefer, Leon Anavy, Alexandra Chudnovsky, Alvaro Israel, Alexander Golberg, and Zohar Yakhini. 2019. Sparse NIR optimization method (SNIRO) to quantify analyte composition with visible (VIS)/near infrared (NIR) spectroscopy (350 nm-2500 nm). *Analytica Chimica Acta* 1051 (2019), 32–40.

[52] Vanishree Rao, Yunhui Long, Hoda Eldardiry, Shantanu Rane, Ryan Rossi, and Frank Torres. 2019. Secure Two-Party Feature Selection. *arXiv preprint arXiv:1901.00832* (2019).

[53] Ronald L. Rivest, Len Adleman, and Michael L. Dertouzos. 1978. On data banks and privacy homomorphisms.

[54] SEAL 2022. Microsoft SEAL (release 4.0). https://github.com/Microsoft/SEAL. Microsoft Research, Redmond, WA..

[55] Shai Shalev-Shwartz and Shai Ben-David. 2014. *Understanding machine learning: From theory to algorithms*. Cambridge university press.

[56] Reza Shokri and Vitaly Shmatikov. 2015. Privacy-preserving deep learning. In *CCS'15*. 1310–1321.

[57] Andrew H Sims, Graeme J Smethurst, Yvonne Hey, Michal J Okoniewski, Stuart D Pepper, Anthony Howell, Crispin J Miller, and Robert B Clarke. 2008. The removal of multiplicative, systematic bias allows integration of breast cancer gene expression datasets–improving meta-analysis and prediction of prognosis. *BMC medical genomics* 1, 1 (2008), 1–14.

[58] Saúl Solorio-Fernández, J Ariel Carrasco-Ochoa, and José Fco Martínez-Trinidad. 2016. A new hybrid filter–wrapper feature selection method for clustering based on ranking. *Neurocomputing* 214 (2016), 866–880.

[59] TCGA. n.d.. The Cancer Genome Atlas Program. https://www.cancer.gov/tcga, accessed 16.8.2022.

[60] Robert Tibshirani. 1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)* 58, 1 (1996), 267–288.

[61] Stacey Truex, Nathalie Baracaldo, Ali Anwar, Thomas Steinke, Heiko Ludwig, Rui Zhang, and Yi Zhou. 2019. A hybrid approach to privacy-preserving federated learning. In *AISec Workshop*. 1–11.

[62] Marie Beth van Egmond, Gabriele Spini, Onno van der Galien, Arne IJpma, Thijs Veugen, Wessel Kraaij, Alex Sangers, Thomas Rooijakkers, Peter Langenkamp, Bart Kamphorst, et al. 2021. Privacy-preserving dataset combination and Lasso regression for healthcare predictions. *BMC medical informatics and decision making* 21, 1 (2021), 1–16.

[63] Thijs Veugen, Bart Kamphorst, Natasja van de L'Isle, and Marie Beth van Egmond. 2021. Privacy-Preserving Coupling of Vertically-Partitioned Databases and Subsequent Training with Gradient Descent. In *CSCML*. Springer, 38–51.

[64] Paul S. Wang, M. J. T. Guy, and James H. Davenport. 1982. P-adic reconstruction of rational numbers. *ACM SIGSAM Bulletin* 16, 2 (1982), 2–3.

[65] David J. Wu, Tony Feng, Michael Naehrig, and Kristin E. Lauter. 2016. Privately Evaluating Decision Trees and Random Forests. *Proc. Priv. Enhancing Technol.* 2016, 4 (2016), 335–355.

[66] Mengwei Wu, Xiaobin Li, Taiping Zhang, Ziwen Liu, and Yupei Zhao. 2019. Identification of a Nine-Gene Signature and Establishment of a Prognostic Nomogram Predicting Overall Survival of Pancreatic Cancer. *Frontiers in Oncology* 9 (Sept. 2019).

[67] Yuncheng Wu, Shaofeng Cai, Xiaokui Xiao, Gang Chen, and Beng Chin Ooi. 2020. Privacy Preserving Vertical Federated Learning for Tree-based Models. *Proc. VLDB Endow.* 13, 11 (2020), 2090–2103.

[68] Andrew C Yao. 1982. Protocols for secure computations. In *FOCS'82*. IEEE, 160–164.

[69] Yuheng Zhang, Ruoxi Jia, Hengzhi Pei, Wenxiao Wang, Bo Li, and Dawn Song. 2020. The secret revealer: Generative model-inversion attacks against deep neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 253–261.

[70] Wenting Zheng, Raluca Ada Popa, Joseph E Gonzalez, and Ion Stoica. 2019. Helen: Maliciously secure coopetitive learning for linear models. In *S&P'19*. 724–738.