SIR: Privacy Preserving Feature Selection for Sparse Linear Regression

***Abstract*—Privacy-Preserving Machine Learning (PPML) provides protocols for learning and statistical analysis of data that is distributed amongst multiple data owners (e.g., hospitals that own proprietary healthcare data), while preserving data privacy. PPML literature includes protocols for various learning methods, such as neural networks, decision trees and logistic regression, as well as linear regression, including ridge regression. Ridge 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 to avoid overfitting, and reduce 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 as above). Our protocol outsources the bulk of the computation to two non-colluding servers, using homomorphic encryption as a central tool. Security is against semi-honest adversaries controlling any number of data owners and at most one server. We implemented our protocol, and evaluated its performance for data with hundreds of samples and up to 40 features. Our PPML protocol runs in just under 24 hours for these parameters, producing the same model as obtained in cleartext.

1. Introduction

The Machine Learning (ML) revolution critically relies on large volumes of data for attaining 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) [1] leverages tools for secure computation [2], [3], [4] to attain ML utility without exposing the raw data.[1] Many PPML solutions focus on the inference phase, e.g., [7], [8], [9]. A growing body of literature also addresses training, covering a range of learning tasks and techniques, including training decision tree models, e.g., [1], as well as linear regression, logistic regression and neural networks models, e.g., [10]. In linear regression, when working in the regime where there are more features than data samples, as well as in other use cases, it is crucial to include feature selection as part of the training phase. In linear regression the data is given as a matrix X with n rows (samples/instances) and d columns (features), as well as an n dimensional response/target vector.

Feature selection [11] 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 to avoid overfitting. (That is, to produce 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 features extraction might involve medical interventions with associated financial and morbidity costs (see Section 2.1). Selecting the features of highest predictive power is computationally intractable [12]; nonetheless heuristic methods are widely employed in practice with great success. In this work we focus our attention on feature selection under L_0 constraints. That is, first set the intended number of features in the model (as determined by hardware considerations, for example [13]), and then develop the best model that satisfies this constraint.

Feature selection methods are sometimes categorized into filter, embedded and wrapper methods, described next. *Filter methods* assign scores to each feature in isolation. One can then truncate all features with scores below some threshold or select the desired number of best (one dimensional, in isolation) performance features. While typically very fast, filter methods often fail to select the best features, because they ignore feature relationships and dependencies. *Embedded methods* refer to learning techniques that intrinsically incorporate feature selection in the model training process, as with decision tree regression, as

1. Other PPML concerns include limiting the amount of information revealed by the output, e.g., using differential privacy in [5]; see a survey in [6] This is beyond the scope of our paper.

well as Ridge and Lasso followed by a truncation step to satisfy the L_0 constraint. *Wrapper methods* select features in an iterative process. One such approach works with the target ML algorithm in mind, evaluating subsets of features according to performance of the model they would yield. For example, when the target is to train a linear/polynomial regression model, on the selected features, the performance of each subset of features can be measured by the Mean Square Error (MSE) of the regression model when trained on these features (using cross-validation to avoid overfitting).[2]

Recursive Feature Elimination (RFE) [14] is a commonly used and highly influential wrapper method for feature selection. RFE starts by considering all features, and iteratively removes features until reaching the L_0 constraint. In each iteration, the features to be removed are chosen by (1) training an intermediate model on the surviving features (those that have not yet been removed), (2) ranking the features according to their weight in this intermediate model, and (3) removing the (desired, dynamically adjusted, number of) lowest ranking features. We note that wrapper methods are computationally intensive since they perform training in each iteration. Importantly, wrapper methods typically lead to a better selection than filter methods since they consider the predictive power of combining *several* features (rather than considering each feature in isolation). Other important wrapper feature selection approaches include FSS and BSS [15], [16], [17].

Privacy preserving feature selection, in regression as well as in other ML tasks, aims to select features without revealing any additional information on the data used during selection. Prior art and literature proposed privacy-preserving feature selection solutions that either use filter methods [18], [19], [20] (with scoring based on correlation coefficients, chi-square test and Gini impurity, respectively), or embedded methods as (implicitly) implied in works on privately learning decision tree models [1], [21], [22], [23], [24], [25], [26], [27] and Lasso regression [28], [29], [30], [31], [32]. A specific relevant line of work addresses privacy-preserving Ridge regression [28], [33], [34], [35], [36], [37]. Ridge regression (see Section 2.2) favors models with few high-weight features and a long tail of low-weight features. A natural approach to turning these into L_0-sparse models is to truncate the weights vector. In principle, this truncation can be done in a privacy-preserving fashion; we call this approach the *truncation* algorithm for feature selection.

1.1. Our Contribution

In this work we present *the first privacy-preserving feature selection under L_0 constraints, producing a sparse regression model.* Concretely:

Our protocol. We design the *Secure Iterative Ridge Regression (SIR)* protocol: a new protocol that produces a *sparse ridge regression model* via RFE, 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*). Horizontally partitioned means that each data owner contributes a subset of the rows of the data matrix X. Our solution utilizes homomorphic encryption as a central tool.

Privacy & Threat Model. Security 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 samples/instances than features and full rank).

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 polynomial in the security parameter); the protocol includes homomorphic computations of multiplicative depth up to $O(\log d)$.

System and empirical evaluation. To demonstrate the usability of SIR in practice, we implemented our protocol 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 [38]. 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 $> 10K$ values. Our proof of concept data represents regressing the expression pattern (a vector with 781 entries) of a target gene, based on arbitrarily selected 40 other genes. Furthermore, we artificially randomly partition the 781 instances amongst 10 data owners. Our focus is on designing a *secure* variant of iterated ridge regression (and *not* devising new cleartext ML algorithms or discovering new biology). Our choice of horizontally partitioning the data into 10 data owners was similarly aimed to demonstrate the applicability of our solution. We note that both horizontal and vertical partition would be interesting use cases in analyzing gene expression related data [39], [40], [41], [42]. In our experiments, each iteration removes 10% of the features, and the L_0

2. Cross validation means evaluating model performance on fresh data that was not used during training. MSE is the (squared) Euclidean distance between the prediction and the target. See Section 2.1.

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

constraint is 8 features. 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 PPML system terminates in just under a day, using 134GB RAM. The model produced by our system significantly outperforms the models produced by filter or truncation algorithms, which may be considered straw-man feature selection approaches that can be implemented in a privacy-preserving manner (see Section 5).

In summary – we developed a privacy preserving sparse linear regression protocol, proved its cryptographic adequacy, analyzed its complexity, and demonstrated its utility on actual data.

1.2. Overview of our Techniques

We now present a high-level overview of our protocol, SIR, and of our system implementing it, while focusing on the challenges we faced and how we resolved them. We start by describing the baseline (insecure) feature selection algorithm – that we call IR – which SIR securely realizes.

Iterated ridge (IR). The RFE algorithm for sparse ridge regression is analogous to the sparse logistic regression algorithm used in [43]. The algorithm (Figure 1) 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. (This follows the paradigm of adjusting the learning rate to a smaller value when approaching the solution.) Selecting the features to be removed at each iteration is done solving ridge regression (see Section 2.1) on the surviving features to obtain an intermediate model, and removing the features whose weights (in absolute value) are in the bottom 10%. 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.

Our starting point for securely realizing IR is the work of [36], [37] on privacy-preserving ridge regression. In these protocols, data owners encrypt their data using Linearly Homomorphic Encryption (LHE), and upload the encrypted data to server $\mathcal{S}_1$. The encryption public key is provided by a second server $\mathcal{S}_2$. The two servers then engage in a secure 2-party protocol, at the conclusion of which they learn the ridge regression model in the clear.

To securely realize IR, a natural idea is to compute, in each iteration, the intermediate ridge regression model on the surviving features, using the protocols of [36], [37]. Unfortunately, this does not work, since there is an inherent tension between privacy, and the iterative nature of IR which computes and employs partial information to obtain sparse prediction models. (This should be contrasted with non-iterative solutions, which are much easier to implement securely, but whose prediction accuracy is worse.) In particular, Iterated ridge reveals the order in which features are removed, as well as intermediate models computed in the iterations. As we demonstrate in Section 8.1, revealing this information violates security, *even if the final model is public*.

Thus, we need to hide the intermediate models, and the order in which features are removed. We do so by designing a *Scaled* Ridge Regression protocol which builds on the protocols of [36], [37] but does *not* reveal the intermediate model in the clear, at any stage of the communication between $\mathcal{S}_1$ and $\mathcal{S}_2$; see next paragraph. We combine this with a protocol for securely ranking the features of intermediate models, which we devise. In both protocols, obtaining good concrete efficiency necessitated several delicate design choices.

Our scaled ridge regression protocol (Figure 3). The three main design choices shaping the privacy-preserving ridge regression component of our SIR protocol are as follows.

Design choice 1: producing an encrypted model. We produce an *encrypted model* (rather than in cleartext), encrypted under fully homomorphic encryption, so that features can be ranked while the model is encrypted.

However, doing so in the straightforward way is not satisfactory. The problem is that the protocols of [36], [37] require post-processing the cleartext model – concretely, computing *rational reconstruction*[4] – which results in poor concrete efficiency when computed over an *encrypted* model.

To better explain the problem let us first recall the protocols of [36], [37] which, roughly, operate as follows. First, $\mathcal{S}_1$ homomorphically combines the encrypted data that he received from all data owners into an encrypted $d \times d$ matrix A and an encrypted length-d vector $\vec{b}$ (where d is the number of features), so that the ridge regression model is the solution $\vec{\omega}$ to the linear system $A\vec{\omega} = \vec{b}$. Next $\mathcal{S}_1$ homomorphically masks the encrypted A and $\vec{b}$, and sends their masked versions A' and $\vec{b'}$ to $\mathcal{S}_2$. Third, $\mathcal{S}_2$ decrypts and solves the linear system $A'\vec{\omega'} = \vec{b'}$ in the clear and sends the cleartext masked solution $\vec{\omega'}$ to $\mathcal{S}_1$. Finally $\mathcal{S}_1$ unmasks and post-processes the model to obtain the ridge regression model $\vec{\omega}$ in the clear.

Importantly, although the ridge regression model should be the solution – *over the reals* – to the aforementioned linear system, [36], [37] embed all values and all computations in the ring $\mathbb{Z}_N$, for a large positive integer N. This embedding is crucial for the masking step to successfully hide A and $\vec{b}$, because the wrap-around modulo N guarantees that A' and $\vec{b'}$ are uniformly random (subject to A being invertable). To produce a correct model despite this embedding,

4. Rational reconstruction to refer 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). See [44], [45] for further details.

[36], [37] have two techniques. First, they embed the plaintext data in a large finite ring, concretely, choosing a sufficiently large N as to exclude overflows when computing over the unmasked data. Second, when $\mathcal{S}_1$ receives from $\mathcal{S}_2$ the solution ω' to the masked linear system in $\mathbb{Z}_N$, he does not only unmask it, but rather also applies rational reconstruction to map it to the solution over the reals.

For our purpose, it is *essential* that we use the solution to the linear system over *the reals* (rather than the model over $\mathbb{Z}_N$, prior to applying the rational reconstruction). This is because we use the intermediate model learned in each iteration to *rank* features by their weights, to determine which features to remove. This ranking cannot be computed from the initial model, because the magnitude of $\mathbb{Z}_N$-elements is meaningless in our context due to the periodic nature of the ring $\mathbb{Z}_N$. Thus, using this initial model would be devastating for our ranking step. Therefore, in the context of *iterated* executions of the ridge regression step, there is an tension between efficiency and the need to perform rational reconstruction (to obtain correctness).

Design choice 2: eliminating the need for rational reconstruction. To remedy this issue, our second design choice is to produce an altered model that preserves the same features' ranking, while *eliminating the need for rational reconstruction*. We do so by using the following trick, which is inspired by [46]. Instead of computing the model $\vec{\omega} = A^{-1} \cdot \vec{b}$, we use the fact that $A^{-1} = \text{Adj}\,(A) \,/\, \det\,(A)$, and compute (in encrypted form) a *scaled* version of the model: $\vec{\omega^*} = \text{Adj}\,(A) \cdot \vec{b}$. Note that the formula for computing the adjugate $\text{Adj}(A)$ involves only multiplications and additions, and so – when using a sufficiently large N, as we do – no wrap-around occurs, meaning that the orderings in $\mathbb{Z}_N$ and $\mathbb{Q}$ are identical, and so no rational reconstruc Importantly, the ranking of the weights is *identical*, regardless of whether we use $\vec{\omega}$ or its scaled version $\vec{\omega^*}$. Namely, using the scaled model $\vec{\omega^*}$ (instead of $\vec{\omega}$) allows $\mathcal{S}_1$ to homomorphically rank the features while they are still embedded in $\mathbb{Z}_N$, without performing rational reconstruction.

Design choice 3: obliviously permuting the features. Finally, as noted above our analysis of Section 8.1 shows that to obtain security, even the meta-data of which features are removed in each iteration must remain hidden. To address this issue, we perform ranking over *obliviously permuted features*, i.e., permuting the order of features using a random permutation which is *unknown* to any of the parties. Consequently, the protocol can reveal the (permuted) indices without disclosing any information on the (unpermuted) indices.

To compute the oblivious permutation, each server $\mathcal{S}_i$ $(i = 1, 2)$ samples a uniformly random permutation matrix P_i; $\mathcal{S}_2$ encrypts his P_2 and send the ciphertexts to $\mathcal{S}_1$; $\mathcal{S}_1$ homomorphically computes $P = P_1 \cdot P_2$ and employs it to homomorphically permute – according to P – the encrypted input aggregated from all data owners. Concretely, for the matrix A and vector $\vec{b}$ aggregating the input from all data-owners (in encrypted form), $\mathcal{S}_1$ homomorphically computes $P^T \cdot A \cdot P$ which permutes both the rows and columns of A according to P; and homomorphically computes $P^T \cdot \vec{b}$ which permutes $\vec{b}$'s rows according to P. (We remark that when referring to $A, \vec{b}$ in the discussion of design choices 1-2 above, we were actually referring to the *permuted* matrix and vector. Similarly, below we use $A, \vec{b}$ when referring to their *permuted* versions.) The result is a randomly permuted order of features where neither $\mathcal{S}_1$ nor $\mathcal{S}_2$ know the permutation.

Our ranking of encrypted features (Figure 7). Having described our modifications to the privacy preserving ridge regression stage of each iteration, we turn to discussing how we homomorphically rank the (encrypted) weights of the intermediate model.

Comparing two large integers. A key step during ranking is comparing pairs of encrypted weights, which are represented as integers modulo N (rather than in binary – or any other small radix – representation). Comparisons in $\mathbb{Z}_N$ for large N is notoriously inefficient, and even more so since our modulus N is particularly large; for example, our end-to-end experiments required $N = 2^{1,260}$.[5]

We resolve this issue by utilizing additional interaction between the servers, specifically, devising a 2-message protocol for securely comparing two values a and b in absolute value, or equivalently testing whether the difference of their squared values $c = a^2 - b^2$ is positive. To reduce the round complexity, we employ this protocol to compare all weights *in parallel*. The goal of this sub-protocol is to “translate” the differences c into their binary representation, and then use the latter to efficiently homomorphically rank features. The protocol begins as follows. First, $\mathcal{S}_1$ masks the difference c by homomorphically adding to it a uniformly random integer r (modulo N), and sends the masked difference to $\mathcal{S}_2$. Next, $\mathcal{S}_2$ decrypts the masked difference, computes its binary representation B_c (in the clear), encrypts B_c bit by bit – using a different modulus, see details below when discussing ranking the weights – and sends the ciphertexts to $\mathcal{S}_1$. What remains is for $\mathcal{S}_1$ to homomorphically unmask the differences and compare them to 0. (As noted above, the comparisons would indeed be reasonably fast, because they are performed over binary representations.)

Unfortunately, unmasking the differences is problematic, as we now explain. Unmasking involves

5. Another challenge is that existing libraries for implementing fully homomorphic encryption schemes support up to 64-bit integers, where we require much larger integers. To resolve this issue we follow [37] who suggested using the Chinese Remainder Theorem to represent each integer modulo N as a tuple of integers modulo small(ish) primes. Concretely, primes of size up to 64-bits, so that existing FHE libraries support encrypting the integers in the tuple.

homomorphically computing the difference between two integers which are encrypted bit-by-bit, by emulating subtraction of integers using a Boolean circuit. This would be computationally expensive to compute homomorphically, particularly so since N is so large in our settings.

Instead, we use the following trick. We observe that $c > 0$ if-and-only-if $c + r > r$. Now recall that $\mathcal{S}_1$ has received from $\mathcal{S}_2$ ciphertexts for $c + r$ in binary representation; and moreover, $\mathcal{S}_1$ chose the mask r himself, so he can efficiently compute its binary representation. Therefore, $\mathcal{S}_1$ can avoid the aforementioned unmasking step, and instead directly homomorphically compare the encrypted $c + r$ with his cleartext r, where both are given in binary representation. We stress that when comparing between $c+r, r$, $\mathcal{S}_1$ needs to also account for possible overflows, for which we devise a dedicated sub-algorithm; see Section 4 for details.

Ranking the encrypted weights. Once we have securely compared all pairs of weights, we compute the rank of each weight by summing up the (encrypted) results from its comparison with all other weights, which gives its rank (i.e., the number of weights smaller than it).[6]

Importantly, in order to support fast homomorphic summation of these encrypted results, we instruct $\mathcal{S}_2$ – 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. The Boolean operations computed during the aforementioned homomorphic comparison are then emulated in $\mathbb{Z}_D$ (defining, for $a, b \in \{0, 1\}$, $AND(a, b) = a \cdot b \mod D$ and $XOR(a, b) = (a - b)^2 \mod D$). This at most doubles the multiplicative depth of comparison, for the benefit of making the summation computation a linear function that requires no multiplication.

Paper organization. Preliminary definitions appear in Section 2; the problem statement in Section 3; the SIR protocol in Section 4; our system and empirical evaluation in Section 5; and conclusions in Section 6.

2. Preliminaries

Notations. We use upper-case letters (e.g., X) to denote matrices, and vector notation (e.g., $\vec{v}$) to denote vectors. We use boldface letters to denote ciphertexts (e.g., $\mathbf{A}$ for a matrix and $\vec{\mathbf{b}}$ for a vector). We use Greek letters to denote masked values (see, e.g., Figure 6).

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 $\mathsf{Adj}(A), \det(A)$ to denote the adjugate matrix and determinant of A, respectively. We note that for any pair A, B of matrices it holds that $\mathsf{Adj}(A \cdot B) = \mathsf{Adj}(B) \cdot \mathsf{Adj}(A)$, 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 $\mathsf{abs}(x)$ to denote its absolute value, and $\lfloor x \rceil$ to denote its nearest integer. We extend the notation to apply on vectors and matrices by entries-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. Without loss of generality, $\mathbb{Z}_N$ elements in our protocols are represented using the integers $0, 1, \ldots, N - 1$. 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 $N - 2 \equiv -2 \mod N$.

If R, R' are random variables then $\mathsf{R} \approx \mathsf{R}'$ denotes they are computationally indistinguishable. We use $\mathsf{negl}(\kappa)$ to denote a function which is negligible in κ. We use the standard notion of computational indistinguishability (e.g., from [47]). We use PPT as shorthand for Probabilistic Polynomial Time.

Fully Homomorphic Encryption (FHE) [4], [48] 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 [49], and Section 7.

2.1. Sparse Linear Regression and Motivating Application Scenarios

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):

$$\vec{w} = \underset{\vec{u} \in \mathbb{R}^{d+1}}{\operatorname{argmin}} \|\vec{y} - X\vec{u}\|_2^2 \quad (1)$$

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

6. We note that we assume that all weights are distinct.

regression [50], [51] and ridge regression, both are special cases of controlling the norm of $\vec{w}$. Ridge regression [36], [37], [52] seeks to find

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

where notation is as above and $\lambda \geq 0$ is the regularization (hyper)-parameter.

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:

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

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 [51], [53]. 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.[7] 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 [43], [54], [55], [56], [57].

2.2. 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 [58], [59]. 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 [13], [14], [54], [55]

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 [14], 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 [43], that developed a sparse logistic regression model using RFE. In each iteration we run ridge regression with $\lambda = 1$ [37], [53] 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 BSS). 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 pseudocode of this algorithm is given in Figure 1.

```
Iterative Regression (IR) Algorithm
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 $s \leq d$ 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 nnz$(\vec{w}) >$ thr:
     a) $\vec{w} = LR(D, y, \Omega)$
     b) $\pi$ = argsort(abs$(\vec{w})$)
     c) prefix($\pi$) = $\pi$[0: rej $\cdot |\Omega|$]
     d) $\Omega = \Omega \setminus$ prefix($\pi$)
  3) While nnz$(\vec{w}) > s$:
     a) $\vec{w} = LR(D, y, \Omega)$
     b) smallest = argmin(abs$(\vec{w})$)
     c) $\Omega = \Omega \setminus \{$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; abs$(\vec{w})$ returns the absolute value of each entry of $\vec{w}$. The regularization parameter λ is implicit in this pseudocode.

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

3. Problem Statement

We follow the security and threat model of [37] which guarantee computational security in the passive setting, against a single server 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 $\lambda \geq 0$. 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 $\mathbb{Z}_N$ (the same assumption was made by previous works [36], [37]). 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 $\mathsf{DO}_1, \ldots, \mathsf{DO}_m$ 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 $x_1, \ldots, x_m$ of $\mathsf{DO}_1, \ldots, \mathsf{DO}_m$, we use $\Pi(x_1, \ldots, x_m)$ 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 every party $P \in \{\mathsf{DO}_1, \ldots, \mathsf{DO}_m, \mathcal{S}_1, \mathcal{S}_2\}$, the *view* of P in Π, denoted $\mathsf{V}_P^\Pi(x_1, \ldots, x_m)$, is the random variable consisting of the input and randomness of P, as well as the messages P received from the other parties in a random execution of Π with inputs $x_1, \ldots, x_m$. We say a subset $I \subseteq \{\mathsf{DO}_1, \ldots, \mathsf{DO}_m, \mathcal{S}_1, \mathcal{S}_2\}$ 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., [47]), adapted to the setting of non-colluding servers as in [33]. Since optimal feature selection under L_0 (Equation 3) is NP hard in general [12], we focus on providing a secure variant of the *Iterated Ridge* heuristic approach (see Section 2.1). 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 also includes the output model) and the inputs of the parties in I (and anything efficiently computable therefrom). Following [36] we define correctness with respect to a subset of inputs (where there is no correctness guarantee for inputs not in $\mathcal{T}$). Formally,

Definition 3.1 (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-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 $(x_1, \ldots, x_m) \in \mathcal{T}$,

$$\Pr[\Pi(x_1, \ldots, x_m) = f(x_1, \ldots, x_m)] = 1 - \mathsf{negl}(\kappa)$$

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 $(x_1, \ldots, x_m) \in \mathcal{T}$:

$$\mathsf{V}_I^\Pi(x_1, \ldots, x_m) \approx \mathsf{Sim}\left((x_j)_{\mathsf{DO}_j \in I}, \mathcal{L}\right).$$

4. SIR Protocol

4.1. Protocol Overview

In this section we describe our protocol, which we call SIR. The protocol itself is described in Figures 2-9.

Overview. SIR is a multi-party computation protocol which employs an FHE scheme $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ as a building block. It is executed between m data owners $\mathsf{DO}_1, \ldots, \mathsf{DO}_m$, and two servers $\mathcal{S}_1, \mathcal{S}_2$, and operates in three phases. The *Setup* phase establishes cryptographic keys to be used during the execution. During the *Data Uploading and Merging* phase, each data owner uploads its (encrypted) data to the servers by sending a single message to $\mathcal{S}_1$, who then merges the contributions from all data owners. The *Learning* phase is an interactive protocol between the servers (with no involvement of the data owners) whose outcome is the (public) model computed by the learning algorithm, which is then sent to all data owners. The learning phase itself consists of several sub-phases, outlined below. Throughout the protocol, each of the servers has its own designated role: $\mathcal{S}_1$ combines and masks the (encrypted) data contributed by the data owners; whereas $\mathcal{S}_2$ holds the decryption keys and executes the learning algorithm on the unencrypted, but masked, data. We now provide more details on each phase, on the roles of the parties, and on how the input is represented.

Input representation and encoding. We assume every entry in the input datasets is in the range $[-1, 1]$, given with ℓ-digit precision. The inputs are normalized to lie in $\mathbb{Z}_N$ for a sufficiently large N (for the choice of N, see Remark 4.1). All subsequent computations in the protocol are performed in $\mathbb{Z}_N$ or in $\mathbb{Z}_D$ for some $D \geq d$. All inputs (and intermediate values generated during the computation) are encoded using the encoding of [37]. (We refer the interested reader to [37] for a detailed description of the encoding and its efficiency benefits.)

The Setup Phase (Figure 5) consists of $\mathcal{S}_2$ generating encryption keys for the FHE scheme, and publishing the public keys. $\mathcal{S}_2$ additionally picks a random permutation matrix which is sends to $\mathcal{S}_1$. This permutation matrix is used to permute the data during the data uploading and merging phase.

The Data Uploading and Merging Phase (Figures 5 and 6). Recall from Section 3 that we assume the input is horizontally partitioned between the data owners. That is, let $X \in \mathbb{R}^{n\times d}, \vec{y} \in \mathbb{R}^{n\times 1}$ denote the data matrix and response vector, respectively, then there exists a partition $I_1, \ldots, I_m$ of $[n]$ such that DO_j holds $X^j = X_{I_j} = (X_k : k \in I_j)$ (where X_k denotes the k'th row of X) and $\vec{y}^j = \vec{y}_{I_j} = (y_k)_{k\in I_j}$. These inputs are scaled and embedded into $\mathbb{Z}_N$ for a sufficiently large N (which is specified in Remark 4.1). For every $1 \leq j \leq m$, let $A^j = \left(X^j\right)^T \cdot X^j \in \mathbb{Z}_N^{d\times d}$, and $\vec{b}^j = \left(X^j\right)^T \cdot \vec{y}^j \in \mathbb{Z}_N^d$. Each DO_j locally computes $A^j, \vec{b}^j$ from his local inputs, and sends encryptions of $A^j, \vec{b}^j$ to $\mathcal{S}_1$. Then, $\mathcal{S}_1$ homomorphically combines the data contributed by all data owners, and permutes the federated data. Specifically, following [33], [36], we combine the data to obtain $A = X^T X + \lambda I$ (where λ is a regularization parameter, see Section 2, and I is the identity matrix), and $\vec{b} = X^T\vec{y}$, by summing the contributions of all data owners as follows:

$$A = \sum_{j=1}^{m} A_j + \lambda I \quad \text{and} \quad \vec{b} = \sum_{j=1}^{m} \vec{b}_j.$$

Finally, $\mathcal{S}_1$ homomorphically permutes the data by computing $P^T AP$ and $P^T\vec{b}$ for a random permutation matrix P. This computation in effect applies the random permutation matrix P to the columns of the federated data matrix X and the entries of the response vector $\vec{y}$ (i.e., it permutes the features). As discussed in Sections 8 and 8.1 below, P must remain hidden from all parties. Therefore, it is computed as $P = P_1 \cdot P_2$, where $P_i, i = 1, 2$ is a secret random permutation matrix chosen by $\mathcal{S}_i$.

The Learning Phase is an iterative protocol between the servers, which repeatedly reduces the set F of features until it has the desired size s. The final model output by the protocol is an s-sparse model whose only non-zero entries correspond to the features that "survived" all iterations. Throughout the protocol execution, $\mathcal{S}_1$ maintains encryptions $\mathbf{A}_F, \vec{\mathbf{b}}_F$ of the restrictions $A_F, \vec{b}_F$ of $A, \vec{b}$ to the indices in the current set F of features. (More specifically, A_F is obtained from A by replacing all rows and columns indexed by $i \notin F$ with the all-0 vector, and $\vec{b}_F$ is obtained from $\vec{b}$ by replacing all entries indexed by $i \notin F$ with 0.) Each iteration consists of several sub-phases, as described next.

The *Scaled Ridge Regression Phase* (Figure 3) consists of a single iteration of a ridge regression protocol. The protocol combines ideas from the protocols of [36], [37], [46]. More specifically, it executes ridge regression on the current $A_F, \vec{b}_F$, but computes a *scaled* outcome as described below. The starting point is the size of the set $F \subseteq [d]$ of features that "survived" the previous iterations, and (secret) $A_F \in \mathbb{Z}_N^{|F|\times|F|}$, $\vec{b}_F \in \mathbb{Z}_N^{|F|}$ which are the restriction of *the permuted versions of* $A, \vec{b}$ described above to the set F of indices. During this phase, $\mathcal{S}_1$ masks $A_F, \vec{b}_F$ as follows: it chooses a random invertible $R' \in \mathbb{Z}_N^{|F|\times|F|}$, and a random $\vec{r}' \in \mathbb{Z}_N^{|F|}$, and expands R' ($\vec{r}'$, respectively) to $R \in \mathbb{Z}_N^{d\times d}$ ($\vec{r} \in \mathbb{Z}_N^d$, respectively) by setting to 0 the ith row and column (ith entry, respectively) for every $i \notin F$. Then, $\mathcal{S}_1$ homomorphically computes encryptions of $\Gamma_F = A_F \cdot R$ and $\vec{\beta}_F = \vec{b}_F + A_F \cdot \vec{r}$. (Throughout this phase we use Greek letters to denote masked values.) These encryptions are sent to $\mathcal{S}_2$, who decrypts them, projects them to $\Gamma \in \mathbb{Z}_N^{|F|\times|F|}, \vec{\beta} \in \mathbb{Z}_N^{|F|}$ by erasing the rows, columns and entries indexed by $i \notin F$, and solves the linear system $\Gamma \cdot \vec{\omega}'_F = \vec{\beta}$ to obtain a masked model $\vec{\omega}'_F$. The model is then expanded to $\vec{\omega}_F \in \mathbb{Z}_N^d$ by setting all entries indexed by $i \notin F$ to 0. (We note that this masked learning step – over unpermuted $A, \vec{b}$ – was used in [36], [37], but they apply only a single ridge regression step.)

However, we cannot perform this regression step directly, due to the iterative nature of our protocol. Specifically, the model learned in each regression step is then used to *order* the features (according to their weights in $\vec{\omega}_F$), and this ordering is used to determine which features to remove (see detailed discussion below). The ordering cannot be computed in $\mathbb{Z}_N$, but rather requires that we "translate" the entries of $\vec{\omega}_F$ back to the rationals by, e.g., performing rational reconstruction [44], [45].[8] Since rational reconstruction is expensive, to improve efficiency we avoid the need to perform it using the following trick, which is inspired by [46]. Instead of computing the model explicitly as $\vec{\omega}'_F = \Gamma^{-1} \cdot \vec{\beta}$, we use the fact that $\Gamma^{-1} = \mathrm{Adj}\left(\Gamma\right) / \det\left(\Gamma\right)$ and have $\mathcal{S}_2$ compute a scaled version $\vec{\zeta} = \det\left(\Gamma\right) \cdot \vec{\omega}'_F$ of the output model (see Step 4 in Figure 3), which is then expanded as above to obtain $\vec{\zeta}_F \in \mathbb{Z}_N^d$. $\mathcal{S}_2$ then sends encryptions of $\vec{\zeta}_F$ and $\det\left(\Gamma\right)$ to $\mathcal{S}_1$, who homomorphically unmasks $\vec{\zeta}_F$ to obtain $\vec{z}_F = \det\left(A'_F\right) \cdot \vec{w}_F$, where $A'_F \cdot \vec{w}'_F = \vec{b}'_F$, and $A'_F, \vec{w}'_F, \vec{b}'_F$ are the projections of $A_F, \vec{w}_F, \vec{b}_F$

8. We use the term "rational reconstruction" to refer 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).

(respectively) to the indices in F. Concretely, $\vec{z}_F$ is homomorphically computed as $\vec{z}_F = (\det(R'))^{-1} \cdot \left(R \cdot \vec{\zeta}_F - \det(\Gamma) \cdot \vec{r}\right)$ (see Step 5 in Figure 3). Using the scaled model $\vec{z}_F$ instead of $\vec{w}_F$ allows us to compute the ordering of features *while they are still represented as elements of* $\mathbb{Z}_N$, thus eliminating the need to perform rational reconstruction.[9]

The *Feature Selection Phase* (Figure 7) computes an updated set $F^* \subset F$ of features that "survived" the current iteration. This is done by removing the indices of the smallest features in the current solution $\vec{z}_F$, where the "size" of a feature is measured as its 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. The number of features that are removed depends on the size of F – for large $|F|$ several features can be removed at once, whereas for small $|F|$ features are removed one-by-one (to improve accuracy).

To remove the k smallest features, for a given k, $\mathcal{S}_1$ computes the complete ordering of the coordinates $z_{F,1}, \ldots z_{F,|F|}$ of $\vec{z}_F$. The ordering can be computed by comparing $z_i^2 - z_j^2$ to 0 for every $i, j \in F, i \neq j$ (indeed, $|z_i| > |z_j|$ if and only if $z_i^2 - z_j^2 > 0$, so this does indeed determine which of z_i, z_j is closest to a multiple of N).[10] $\mathcal{S}_1$ can homomorphically compute $\alpha_{i,j} := z_i^2 - z_j^2$, but homomorphically checking whether this quantity is positive is expensive. Instead, we use $\mathcal{S}_2$ to determine whether $\alpha_{i,j} > 0$. To preserve privacy, we cannot simply send the encryption of $\alpha_{i,j}$ to $\mathcal{S}_2$, and instead we send a masked difference $\alpha'_{i,j} := \alpha_{i,j} + r_{i,j}$, for a random $r_{i,j} \leftarrow \mathbb{Z}_N$. Notice that masking the difference $\alpha_{i,j}$ effectively erases (from $\mathcal{S}_2$'s point of view) all information regarding whether $\alpha_{i,j} > 0$, so $\mathcal{S}_2$ cannot simply perform this check. Instead, it will compute values that will facilitate $\mathcal{S}_1$ in homomorphically checking whether $\alpha_{i,j}$ is "positive" (i.e., $0 < \alpha_{i,j} \leq (N-1)/2$). Specifically, $\mathcal{S}_2$ will use $\alpha'_{i,j}$ to compute the range two extremal values $\mathsf{rangeMin}_{i,j}, \mathsf{rangeMax}_{i,j}$ of $r_{i,j}$, which $\mathcal{S}_1$ will use in the following way. If $\alpha'_{i,j}$ is "negative" (i.e., $(N-1)/2 < \alpha'_{i,j} < N$) then $\alpha_{i,j}$ is positive if and only if $r_{i,j} \in \left[\mathsf{rangeMin}_{i,j}, \mathsf{rangeMax}_{i,j}\right)$ over $\mathbb{Z}$. (Intuitively, in this case $r_{i,j}$ should be sufficiently large to cause $\alpha'_{i,j}$ to cross the $(N-1)/2$ threshold, but cannot be "too large", as that will cause a wrap-around and result in $\alpha'_{i,j} \leq (N-1)/2 \mod N$, because $r_{i,j} \in \mathbb{Z}_N$.) Otherwise, $\alpha'_{i,j}$ is non-negative (meaning $0 \leq \alpha'_{i,j} \leq (N-1)/2$), in which case $\alpha_{i,j}$ is positive if and only if $r_{i,j} < \mathsf{rangeMin}_{i,j}$ (so that $\alpha_{i,j} + r_{i,j} \leq (N-1)/2$ over $\mathbb{Z}$) or $\mathsf{rangeMax}_{i,j} \leq r_{i,j}$ (meaning adding $r_{i,j}$ caused a wrap-around resulting in $\alpha'_{i,j} \leq (N-1)/2 \mod N$.) To allow $\mathcal{S}_1$ to distinguish between the two cases, $\mathcal{S}_2$ computes an indicator $\mathsf{Negative}_{i,j}$ of whether $\alpha'_{i,j}$ is non-negative or not.

Given encryptions of $\mathsf{rangeMin}_{i,j}, \mathsf{rangeMax}_{i,j}$ and $\mathsf{Negative}_{i,j}$, $\mathcal{S}_1$ can homomorphically compute the vector of orderings. Then, it randomly permutes this vector, and sends the permuted vector to $\mathcal{S}_2$, who decrypts it and computes the (permuted) set F^* which is obtained from (the permuted) F by removing the k smallest coordinates. $\mathcal{S}_2$ then sends the permuted F^* to $\mathcal{S}_1$ (in the clear), and $\mathcal{S}_1$ unpermutes it to obtain the set F^* of surviving features, which it also reveals to $\mathcal{S}_2$. (We stress that by "$\mathcal{S}_1$ unpermutes F^*" we mean that $\mathcal{S}_1$ inverts the ad-hoc permutation applied during the feature selection phase. However, F^* is still permuted according to the permutation P which was applied to the input during the data uploading and merging phase. As described below, This permutation P will only be inverted once the final output model is computed.)

The Compacting Phase (Figure 8) restricts A ($\vec{b}$, respectively) to the set F^* of surviving features, by setting to 0 all rows and columns (entries, respectively) indexed by $i \notin F^*$. This computation is performed locally (homomorphically) by $\mathcal{S}_1$.

The Computing and Unpermuting Output Phase (Figure 9) is executed once the feature set has been sufficiently reduced, i.e., $|F| \leq s$. At the onset of this phase, $\mathcal{S}_1$ holds encryptions $\mathbf{A}_F, \vec{\mathbf{b}}_F$ of $A_F, \vec{b}_F$, i.e., of restrictions of $A, \vec{b}$ to the final set F of features. $\mathcal{S}_1$ and $\mathcal{S}_2$ then engage in a final execution of the scaled ridge regression protocol, which results in $\mathcal{S}_1$ obtaining an encryption $\vec{\mathbf{z}}_F$ of $\vec{z}_F = \det(A'_F) \cdot \vec{w}_F$, where $A'_F \cdot \vec{w}'_F = \vec{b}'_F$, and $A'_F, \vec{w}'_F, \vec{b}'_F$ are the projections of $A_F, \vec{w}_F, \vec{b}_F$ (respectively) to the indices in F. In particular, $\vec{z}_F$ is a *scaled* version of $\vec{w}_F$, which is a *permuted* version of the desired output model. $\mathcal{S}_1$ additionally obtains an encryption of Δ^{-1}, where $\Delta = \det(A'_F \cdot R')$. $\mathcal{S}_1$ can then homomrphically compute an encryption of $\vec{w}_F$, using the fact that $\det(A'_F)^{-1} = \Delta^{-1} \cdot \det(R')^{-1}$, and since $\mathcal{S}_1$ knows R'. The last remaining step is to unpermute the entries of $\vec{w}_F$. $\mathcal{S}_1$ homomorphically unpermutes $\vec{w}_F$ by multiplying it with an encryption $\mathbf{P}$ of P. Roughly (and somewhat inaccurately, see Lemma 8.4 for the formal argument) this gives an encryption $\vec{\mathbf{w}}$ of the unpermuted model $\vec{w}$ because

$$P \cdot \vec{w} = P \cdot A^{-1} \cdot \vec{b} = P \cdot \left(P^T \left(X^T X + \lambda I\right) P\right) \cdot \left(P^T X^T \vec{y}\right)$$

and $P^{-1} = P^T$ because P is a permutation matrix. Finally, $\mathcal{S}_1$ sends $\vec{\mathbf{w}}$ to $\mathcal{S}_2$, who decrypts it to obtain $\vec{w}$ (we note tht the recovered model is in $\mathbb{Z}_N$, and the corresponding model over the rationals is computed using rational reconstruction [44], [45]), and publishes it to all parties.

9. Indeed, since N is chosen to be sufficiently large (see Remark 4.1 below), additions and multiplications do not cause a wrap-around, meaning the orderings in $\mathbb{Z}_N$ and $\mathbb{Q}$ are identical. Since we avoid dividing by $\det\left(A'_F\right)$ (alternatively, multiplying by the inverse $\left(A'_F\right)^{-1}$), computing $\vec{z}_F$ involves only additions and multiplications.

10. In fact, if suffices to compute this difference for $i < j$; however traversing over all pairs $i \neq j$ simplifies the description of the protocol.

4.2. Parameters and Observations

Before describing our protocols, a few remarks are in order. Specifically, we explain how the moduli N, D are chosen, and make a few observations which are used in our protocols.

Remark 4.1 (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 $\mathbb{Z}_{N_{\mathsf{Gia}}}$ as [36]. However, for our selection protocol (Figure 7) we will need the modulo N to be at least square that value, namely:

$$N = N_{\mathsf{Gia}}^2 > \left(2d(d-1)^{\frac{d-1}{2}} 10^{4\ell d}(n^2+\lambda)^{2d}\right)^2 = 4d^2(d-1)^{d-1}10^{8\ell d}(n^2+\lambda)^{4d} \quad (4)$$

where n, d, ℓ, λ are as specified in Section 3.

Remark 4.2 (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 given the size of N (as described above) such computations would be heavy. Instead, in SIR we are able to use a smaller modulus D. We can make due with any $D \geq d$ which can be used as a plaintext modulus in the underlying FHE scheme.

Remark 4.3 (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 $\mathsf{pjct}_F(\cdot)$ (this operation can be applied to a matrix or a vector), namely we compute $\mathsf{pjct}_F(A'), \mathsf{pjct}_F\left(\vec{b}'\right)$. 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 $\mathsf{expd}_F(\cdot)$ 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 $X = \mathsf{pjct}_F(X'), \vec{v} = \mathsf{pjct}_F(\vec{v}')$.

Remark 4.4 (Unique Entries in Intermediate Models)**.** Our security analysis will rely on the assumption that for every intermediate model $\vec{z}$ 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 (Figure 5), 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).

4.3. Notations and Computations for Homomorphic Evaluations

The operations and notations described below are used during homomorphic evaluations in SIR.

Adding and multiplying matrices. Let $m \in \mathbb{N}$, and let $A_1, \ldots, A_m$ be matrices of the same dimensions over some ring G.

- *Scaled addition:* For a parameter $\lambda > 0$, Add_λ denotes the circuit that on input $A_1, \ldots, A_m$ outputs the matrix $\sum_{i\in[m]} A_i + \lambda I$.
- *Matrix multiplication:* $\mathsf{MatMult}$ denotes the circuit that on input A_1, A_2 outputs $A_1 \cdot A_2$.
- *Matrix by vector multiplication:* for a vector $\vec{v}$, $\mathsf{MatVecMult}$ denotes the circuit that on input A_1 and $\vec{v}$ outputs $A_1 \cdot \vec{v}$.
- *Multiplying by a public matrix:* for $d \in \mathbb{N}$, and a matrix R of dimensions $d \times d$, $\mathsf{MatMultR}_R$ ($\mathsf{MatMultL}_R$, respectively) denotes the circuit that on input a matrix M of dimensions $d \times d$ outputs the product $M \cdot R$ ($R \cdot M$, respectively).
- *Adding a public matrix:* for $d \in \mathbb{N}$, and a matrix R of dimensions $d \times d$, MatAdd_R denotes the circuit that on input a matrix M of dimensions $d \times d$ outputs the sum $M + R$.
- *Multiplying by a public vector:* for $d \in \mathbb{N}$, and a length-d vector $\vec{r}$, $\mathsf{VecMatMultR}_{\vec{r}}$ denotes the circuit that on input a matrix M of dimensions $d \times d$ outputs the product $M \cdot \vec{r}$.

Adding and multiplying vectors. Let $\vec{v}_1, \ldots, \vec{v}_m$ be vectors of the same length over some ring G.

- *Addition:* Add denotes the circuit that on input $\vec{v}_1, \ldots, \vec{v}_m$ outputs the vector $\sum_{i\in[m]} \vec{v}_i$.
- *Subtraction:* Sub denotes the circuit that on input $\vec{v}_1, \vec{v}_2$ outputs $\vec{v}_1 - \vec{v}_2$.

- *Multiplying by a public matrix:* for $d \in \mathbb{N}$, and a matrix R of dimensions $d \times d$, $\mathsf{MatVecMultL}_R$ denotes the circuit that on input a length-d vector $\vec{v}$ outputs the product $R \cdot \vec{v}$.
- *Adding a public vector:* for a vector $\vec{r}$, $\mathsf{VecAdd}_{\vec{r}}$ is the circuit that on input $\vec{v}_1$ outputs $\vec{v}_1 + \vec{r}$.
- *Multiplying a scaler by a public vector:* for a vector $\vec{v}$, $\mathsf{VecScalerMult}_{\vec{v}}$ denotes the circuit that on input a scaler c outputs $c \cdot \vec{v}$.
- *Multiplying by a public scaler:* for a scaler c, $\mathsf{ScalerVecMult}_c$ denotes the circuit that on input a vector $\vec{v}$ outputs $c \cdot \vec{v}$.
- *Multiplying by secret scalers:* $\mathsf{ScalerVecMult}$ denotes the circuit that on input a vector $\vec{v}$ and scalers $c_1, \ldots, c_k$ (for some $k \in \mathbb{N}$), outputs $c_1 \cdot \ldots \cdot c_k \cdot \vec{v}$ (i.e., $c_1 \cdot \ldots \cdot c_k$ multiplies each coordinate of $\vec{v}$).
- *Adding and subtracting public vectors in binary representation:* for vectors $\vec{v}, \vec{r} \in \{0,1\}^k$ for some k, $\mathsf{VecBinAdd}_{\vec{r}}, \mathsf{VecBinSub}_{\vec{r}}$ denote the circuits that on input $\vec{v}$ output $\vec{r} + \vec{v}$ and $\vec{r} - \vec{v}$, respectively, where the operations are computed using 2's complement. We allow $\mathsf{VecBinAdd}_{\vec{r}}, \mathsf{VecBinSub}_{\vec{r}}$ to be executed on vectors in $\mathbb{Z}_D^k$ for some $D \in \mathbb{N}$ (and emulating boolean operations using operations in $\mathbb{Z}_D$, see Remark 4.5 below), but there is no guarantee on the output when the entries of $\vec{v}, \vec{r}$ are not bits.

Adding and multiplying scalers. Let c, c' be a pair of values from the same domain (e.g., from $\mathbb{Z}_N$), and let $\vec{c}, \vec{c}'$ denote their binary representation.

- The circuit Square on input c outputs c^2.
- The circuits $\mathsf{AddNums}, \mathsf{SubNums}, \mathsf{MultNums}$ take c, c' as input, and output $c + c', c - c'$ and $c \cdot c'$, respectively.
- The circuit $\mathsf{AddConst}_{c'}$ takes c as input and outputs $c + c'$.
- The circuit $\mathsf{BinCompareR}_{c'}$ takes $\vec{c}$ as input, and returns 1 if $c \leq c'$, otherwise it returns 0. Similarly, $\mathsf{BinCompareL}_{c'}$ takes $\vec{c}$ as input and returns 1 if $c' < c$, otherwise it returns 0. This is done using a standard Boolean circuit for comparing a pair of binary strings.
- The circuit Neg on input $c \in \{0,1\}$ outputs its complement (i.e., 1 if $c = 0$, and 0 otherwise).

Remark 4.5 (Emulating Boolean circuits using arithmetic circuits)**.** Our protocols embed binary values into a larger ring $\mathbb{Z}_D$, and operate over these representations. Therefore, we need to emulate the Boolean circuits described above (i.e., $\mathsf{BinCompareL}_{c'}$ and $\mathsf{BinCompareR}_{c'}$) using arithmetic circuits over $\mathbb{Z}_D$. 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 $a, b \in \mathbb{Z}_D \cap \{0,1\}$. The Neg circuit on input c is emulated by computing $1-c$ (where 1 is the identity of $\mathbb{Z}_D$). This perfectly emulates the Neg circuit when $c \in \{0,1\}$.

4.4. Analysis of SIR

Our security and complexity analysis of SIR is summarized below. Complexity is stated in term of d and N where $\log N = \tilde{O}(d \log n)$ (by Equation 4). The proof, and a more detailed complexity analysis, appear in Sections 8 and 9.

Theorem 4.6 (SIR analysis)**.** *Let $m, n, d \in \mathbb{N}$, $X \in \mathbb{R}^{n \times d}$ and $y \in \mathbb{R}^{n \times 1}$ s.t. X has full rank and $d \leq n$. Then, the following holds when executing SIR on (X, y) when horizontally partitioned amongst m data-owners:*

Security. *SIR (Figure 5) 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, that entails homomorphically evaluating $O(\log d)$ circuits of size $O(d^2 \log N)$ and multiplicative depth $O(\log \log N)$.*[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 $O(d^2 \log N)$ ciphertexts in each round.*[12]

5. System Implementation Details and Experimental Performance Evaluation

We implemented the SIR protocol into a system and ran experiments on real data to evaluate its concrete complexity and to see 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.

5.1. Data

The Cancer Genome Atlas (TCGA), a landmark cancer genomics program, molecularly characterized

11. We note that the complexity of the masking step includes matrix multiplication, which is cubic in d, but since it entails only additive homomorphism it is much faster in practice.

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.

SIR Protocol

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 4.1, 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 4.2).

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 4.1.) The servers $\mathcal{S}_1, \mathcal{S}_2$ have no input.

Output: all parties obtain as output an s-sparse model $\vec{w} \in \mathbb{Z}^n_{\mathbb{R}}$.

Steps:

1) **Setup:** The parties execute the setup protocol of Figure 5 to obtain keys $(\mathsf{pk}_N, \mathsf{sk}_N)$ and $(\mathsf{pk}_D, \mathsf{sk}_D)$. Then, the parties execute the data uploading and merging protocol of Figure 5, and $\mathcal{S}_1$ obtains encryptions of $A, \vec{b}$.
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 of Figure 7 for finding 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 $\mathcal{S}_1$ executes the compacting algorithm of Figure 8, 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 of Figure 9 to obtain the s-sparse model $\vec{w}$.

Figure 2: SIR: Secure Iterated Ridge

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 these 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 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 $4, 10$ and 40 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 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.

5.2. Iterated Ridge Outperforms Filter and Truncation Approaches

To quantify the advantage of iterative ridge over the simpler truncation and filtering approaches we ran experiments in the clear. Data was generated as above. The 781 samples are $80/20$ split into training and test. In each such dataset we ran IR on the training data to

The Scaled Ridge Regression Phase

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 4.3.

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 4.3 for the description of the nullifier matrix $\mathcal{N}_F$, and recall that $A_F, \vec{b}_F$ 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 $\vec{z} = \det(A'_F) \cdot \vec{w}_F$, where: (1) $A'_F = \mathsf{pjct}_F(A_F)$ 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) $\vec{w}_F = \mathsf{expd}_F(\vec{w}'_F)$. (See Remark 4.3 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 $R' \leftarrow \mathrm{GL}(|F|, \mathbb{Z}_N)$, and a random vector $\vec{r}' \leftarrow \mathbb{Z}_N^d$. Then, $\mathcal{S}_1$ computes $R = \mathsf{expd}_F(R'), \vec{r} = \mathcal{N}_F \cdot \vec{r}'$.
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:
   - $\mathbf{\Gamma}_F \leftarrow \mathsf{Eval}(\mathsf{pk}_N, \mathsf{MatMultR}_R, \mathbf{A}_F)$.
   - $\vec{\boldsymbol{t}} \leftarrow \mathsf{Eval}(\mathsf{pk}_N, \mathsf{VecMatMultR}_{\vec{r}}, \mathbf{A}_F)$.
   - $\vec{\boldsymbol{\beta}}_F \leftarrow \mathsf{Eval}\left(\mathsf{pk}_N, \mathsf{Add}, \vec{\mathbf{b}}_F, \vec{\boldsymbol{t}}\right)$.

   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 $\Gamma_F = \mathsf{Dec}(\mathsf{sk}_N, \mathbf{\Gamma}_F)$ 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 $\Gamma = \mathsf{pjct}_F(\Gamma_F) \in \mathbb{Z}_N^{|F|\times|F|}$ and $\vec{\beta} = \mathsf{pjct}_F\left(\vec{\beta}_F\right) \in \mathbb{Z}_N^{|F|}$.
4) **Masked learning:** $\mathcal{S}_2$ computes $\mathrm{Adj}(\Gamma)$ and $\Delta = \det(\Gamma)$, as well as $\vec{\zeta} = \mathrm{Adj}(\Gamma) \cdot \vec{\beta}$. Then, $\mathcal{S}_2$ computes $\vec{\zeta}_F = \mathsf{expd}_F(\zeta) \in \mathbb{Z}_N^d$. Finally, $\mathcal{S}_2$ encrypts $\vec{\boldsymbol{\zeta}}_F \leftarrow \mathsf{Enc}\left(\mathsf{pk}_N, \vec{\zeta}_F\right)$ and $\mathbf{\Delta} \leftarrow \mathsf{Enc}(\mathsf{pk}_N, \Delta)$, 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 $\vec{\zeta} = \det(\Gamma) \cdot \vec{\omega}$.)
5) **Unmasking:** $\mathcal{S}_1$ homomorphically computes $\vec{z} = \det(A'_F) \cdot \vec{w}_F$, where $A'_F = \mathsf{pjct}_F(A_F)$. 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, as explained in the proof of Lemma 8.4, $\vec{t^1} = \det(\Gamma) \cdot (\vec{w}_F + \vec{r})$.)
   - $\vec{\boldsymbol{t}^2} \leftarrow \mathsf{Eval}(\mathsf{pk}_N, \mathsf{VecScalerMult}_{\vec{r}}, \mathbf{\Delta})$. (Notice that $\vec{t^2} = \det(\Gamma) \cdot \vec{r}$.)
   - $\vec{\boldsymbol{t}^3} \leftarrow \mathsf{Eval}\left(\mathsf{pk}_N, \mathsf{Sub}, \vec{\boldsymbol{t}^1}, \vec{\boldsymbol{t}^2}\right)$. (Notice that $\vec{t^3} = \vec{t^1} - \vec{t^2} = \det(\Gamma) \cdot \vec{w}_F$.)
   - $\vec{\mathbf{z}} \leftarrow \mathsf{Eval}\left(\mathsf{pk}_N, \mathsf{ScalerVecMult}_{\det(R')^{-1}}, \vec{\boldsymbol{t}^3}\right)$. (Notice that $\vec{z} = \det(R')^{-1} \cdot \vec{t^3}$, so $\vec{z} = \det(A'_F) \cdot \vec{w}_F$, because $\Gamma = A'_F \cdot R'$ so $\det(\Gamma) = \det(R') \cdot \det(A'_F)$.)
6) $\mathcal{S}_1$ sets $\vec{\mathbf{z}}$ to be its output for the phase.

Figure 3: Learning Phase: single scaled ridge regression iteration.

obtain a selection of 8 features. We also select 8 features by truncating the output of the first ridge iteration, as well as by selecting the top 8 correlated (Pearson in the training) features. In each one of these cases we then obtain the actual model (coefficients) and apply it to the test data. In Figure 4 we compare the resulting MSEs for IR and the truncation process, as obtained in the test data. The histogram represents the observed results for the percentage difference

$$\Delta = \frac{\mathsf{MSE(truncate)} - \mathsf{MSE(IR)}}{\mathsf{MSE(IR)}} \cdot 100 \qquad (5)$$

We clearly observe a significant performance advantage for IR. Similar results are obtained for the comparison to the filter approach.

5.3. System Implementation Details

Ring size. To obtain correctness in SIR, the ring sizes needed in 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 with several (different) 30-bit primes, as was used in [37].

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, see Figure 6), where P_1 and P_2 are random permutation matrices chosen by $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively.

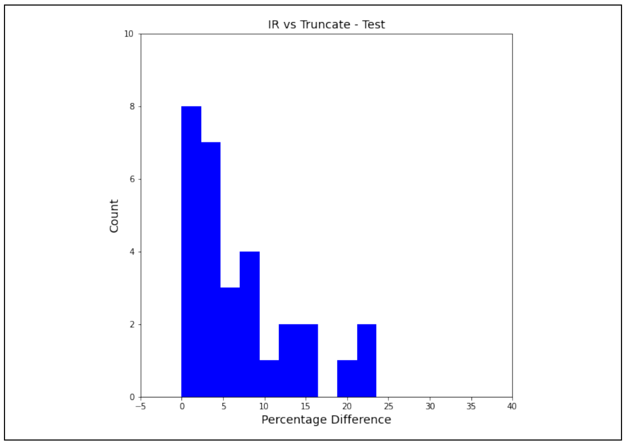

Figure 4: The IR approach significantly outperforms the truncation approach in terms of the resulting MSE on the test data. The results pertain to datasets generated, for $d = 10, 40, 100$, from a breast cancer gene expression data matrix from TCGA [38]. See text for x-axis definition.

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 of 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., with 40 features, this step took 1,152 seconds out of 84,690 seconds. See Table 1).

Ranking weights. To decide which weights of w to remove in each iteration, we compute a full ranking over the weights (see Figure 7). This involves comparing pairs of weights. 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 weights, $\mathcal{S}_1$ 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 $w_i > w_j$). 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 [60]). However, while a sorting approach perform only $\frac{d\lceil\log d\rceil^2}{2}$ comparisons, it is less amenable to parallelization. Specifically, the sorting approach is faster when $\lceil\frac{d^2}{\text{CPUs}}\rceil > \lceil\frac{d}{\text{CPUs}}\rceil \cdot \lceil\log_2 d\rceil^2$, 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 w, when the weights are given in binary. These steps involve a sub-circuit 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 *additively*-HE, 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 the ranking steps we used the BGV implementation of HElib [61] 2.1.0. To initialize the keys, we set the plaintext modulo to 17^2, and the cyclotomic polynoimal degree to $78,881$. This resulted in ciphertexts with $7,000$ slots, supporting a multiplicative depth of 28, as well as bootstrapping. For the CRT steps we used the BFV implementation of Seal [62] 4.0. To initialize the keys, we set the cyclotomic polynomial degree to $8,192$, a chain length of 7, chain bits of 25 and prime bits of 30. This resulted in a ciphertext with 4,096 slots that supports additive homomorphism.

We preferred not to use Seal for the ranking steps because it does not support bootstrapping. Additionally, we preferred not to use HElib for the CRT steps because it is easier to configure multiple keys for CRT with Seal.

Switching between the two libraries (and schemes) was done by $\mathcal{S}_2$. Specifically, after $\mathcal{S}_2$ decrypts the (masked) differences $w_i - w_j$ (for all $i \neq j$), it encrypted the ranges using the appropriate FHE scheme (see Step 2 in Figure 7).

5.4. Evaluation Setup

We executed SIR (Protocol 2) on the data generated from TCGA as described above, 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 d as specified in 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. We ran end-to-end experiments on the following (d, s) values: $(4, 2)$, $(10, 4)$, $(40, 8)$.

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 smallest weight features) 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 10 data owners, as well as $\mathcal{S}_1$ 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. The sub-tasks correspond to the following steps in SIR: *encrypt* (Figure 5, Step 2); *permute* (Figure 6, Step 3); *mask* (Figure 3, Step 2); *unmask* (Figure 3, Step 5); *decrypt* (Figure 3, Step 3);[13] *solve* (Figure 3, Step 4);[14] *pair diff* (Figure 7, Step 1); *ranges* (Figure 7, Step 2); *ranks* (Figure 7, Step 3).

Furthermore, we report how amenable to parallelization each task is, reported as *ratio* in Tables 1-2, which is defined to be the ratio $\frac{\text{CPU time}}{\text{real time}}$ between the runtime on a single CPU (*CPU time*) and in our 100 CPU hardware (*real time*).

5.5. SIR Performance

Tables 1-2 summarize the performance exhibited in the end-to-end experiments and the single iteration experiments, respectively.

Runtime. Our experiments show that the total runtime is dominated by the time it takes to rank the weights. For example, in our end-to-end experiments on $d = 40$, ranking took 91% of the total runtime ($76,184$ out of $83,538$ seconds). In our single iteration experiment on $d = 40$, ranking took 94% of the total runtime.

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. Our results show that as d grows, most tasks become more parallelizeable. In particular, the ranking task (which dominates the bulk of the runtime) is highly parallelizable. For example, the CPU time for computing the ranks grew from $1,510 \times 5 = 7,550$ when $d = 4$ to $76,184 \times 79 = 6,018,536$ when $d = 40$. Namely, the increase in runtime was relatively small (from $1,510$ seconds to $76,184$ seconds) due to a larger parallelization ratio (79 instead of 5).

We note that ranking is embarrassingly parallelizable since we make all $\binom{d}{2}$ comparisons, which can be executed in parallel. Another alternative for implementing the ranking would have been to homomorphically evaluate an oblivious sorting algorithm (e.g., bitonic sort) with a fewer comparisons in total. 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 solution would have higher runtime.

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).

6. Conclusions

We develop and analyze a privacy preserving multi-party protocol for running sparse linear regression in a federated learning setup, based on an iterated ridge framework. 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.

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.

We note that the protocol design is based, amongst other consideration, on certain potential attacks that can be developed when partial information or intermediate information is leaked. In particular, we show in Section 8.1 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.

Our protocol enjoys rigorous security (complete security proofs are provided in Section 8). In terms of complexity, the runtime of SIR doesn't strongly depend on the number of samples or data owners. In our experiments, SIR selects 8 out of 40 features in just under a day. Future work will be invested in more efficient implementations, to make the system more usable.

Acknowledgements. The results described in this manuscript are partially based upon data generated by the TCGA Research Network: https://www.cancer.gov/tcga.

References

[1] Y. Lindell and B. Pinkas, "Privacy preserving data mining," in *CRYPTO'00*. Springer, 2000, pp. 36–54.

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

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

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

[2] A. C. Yao, "Protocols for secure computations," in *FOCS'82*. IEEE, 1982, pp. 160–164.

[3] O. Goldreich, S. Micali, and A. Wigderson, "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*, 2019, pp. 307–328.

[4] C. Gentry, *A fully homomorphic encryption scheme*. Stanford university, 2009.

[5] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *CCS'15*, 2015, pp. 1310–1321.

[6] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings *et al.*, "Advances and open problems in federated learning," *Foundations and Trends® in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[7] T. Graepel, K. Lauter, and M. Naehrig, "Ml confidential: Machine learning on encrypted data," in *ICISC'12*. Berlin, Heidelberg: Springer-Verlag, 2013, pp. 1–21.

[8] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser, "Machine learning classification over encrypted data," in *NDSS'15*. The Internet Society, 2015.

[9] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. E. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *ICML'16*, 2016, pp. 201–210.

[10] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *S&P'17*, 2017, pp. 19–38.

[11] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of machine learning research*, vol. 3, pp. 1157–1182, 2003.

[12] T. M. Cover and J. M. Van Campenhout, "On the possible orderings in the measurement selection problem," *Transactions on SMC*, vol. 7, no. 9, pp. 657–661, 1977.

[13] Y. Peleg, S. Shefer, L. Anavy, A. Chudnovsky, A. Israel, A. Golberg, and Z. Yakhini, "Sparse NIR optimization method (SNIRO) to quantify analyte composition with visible (VIS)/near infrared (NIR) spectroscopy (350 nm-2500 nm)," *Analytica Chimica Acta*, vol. 1051, pp. 32–40, 2019.

[14] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, "Gene selection for cancer classification using support vector machines," *Machine learning*, vol. 46, no. 1, pp. 389–422, 2002.

[15] M. A. Efroymson, "Multiple regression analysis," *Mathematical methods for digital computers*, pp. 191–203, 1960.

[16] R. Hocking, "The analysis and selection of variables in linear regression," *Biometrica*, vol. 32, pp. 1–49, 1976.

[17] N. R. Draper and H. Smith, *Applied regression analysis*. John Wiley & Sons, 1998, vol. 326.

[18] J. L. H. Crawford, C. Gentry, S. Halevi, D. Platt, and V. Shoup, "Doing real work with FHE: the case of logistic regression," in *WAHC@CCS'18*. ACM, 2018, pp. 1–12.

[19] V. Rao, Y. Long, H. Eldardiry, S. Rane, R. Rossi, and F. Torres, "Secure two-party feature selection," *arXiv preprint arXiv:1901.00832*, 2019.

[20] X. Li, R. Dowsley, and M. De Cock, "Privacy-preserving feature selection with secure multiparty computation," in *ICML'21*. PMLR, 2021, pp. 6326–6336.

[21] S. De Hoogh, B. Schoenmakers, P. Chen, and H. op den Akker, "Practical secure decision tree learning in a teletreatment application," in *FC'14*. Springer, 2014, pp. 179–194.

[22] D. J. Wu, T. Feng, M. Naehrig, and K. E. Lauter, "Privately evaluating decision trees and random forests," *Proc. Priv. Enhancing Technol.*, vol. 2016, no. 4, pp. 335–355, 2016.

[23] S. Truex, N. Baracaldo, A. Anwar, T. Steinke, H. Ludwig, R. Zhang, and Y. Zhou, "A hybrid approach to privacy-preserving federated learning," in *AISec Workshop*, 2019, pp. 1–11.

[24] A. Akavia, M. Leibovich, Y. S. Resheff, R. Ron, M. Shahar, and M. Vald, "Privacy-preserving decision trees training and prediction," in *ECML PKDD 2020, Proceedings, Part I*, ser. Lecture Notes in Computer Science, vol. 12457. Springer, 2020, pp. 145–161.

[25] L. Liu, R. Chen, X. Liu, J. Su, and L. Qiao, "Towards practical privacy-preserving decision tree training and evaluation in the cloud," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 2914–2929, 2020.

[26] Y. Wu, S. Cai, X. Xiao, G. Chen, and B. C. Ooi, "Privacy preserving vertical federated learning for tree-based models," *arXiv preprint arXiv:2008.06170*, 2020.

[27] M. Abspoel, D. Escudero, and N. Volgushev, "Secure training of decision trees with continuous attributes." *Proc. Priv. Enhancing Technol.*, vol. 2021, no. 1, pp. 167–187, 2021.

[28] W. Zheng, R. A. Popa, J. E. Gonzalez, and I. Stoica, "Helen: Maliciously secure coopetitive learning for linear models," in *S&P'19*, 2019, pp. 724–738.

[29] M. B. van Egmond, G. Spini, O. van der Galien, A. IJpma, T. Veugen, W. Kraaij, A. Sangers, T. Rooijakkers, P. Langenkamp, B. Kamphorst *et al.*, "Privacy-preserving dataset combination and Lasso regression for healthcare predictions," *BMC medical informatics and decision making*, vol. 21, no. 1, pp. 1–16, 2021.

[30] Z. Liu and R. Zhang, "Privacy preserving collaborative machine learning," *EAI Endorsed Trans. Security Safety*, vol. 8, no. 28, p. e3, 2021.

[31] T. Veugen, B. Kamphorst, N. v. d. L'Isle, and M. B. v. Egmond, "Privacy-preserving coupling of vertically-partitioned databases and subsequent training with gradient descent," in *CSCML*. Springer, 2021, pp. 38–51.

[32] M. De Cock, R. Dowsley, A. C. Nascimento, D. Railsback, J. Shen, and A. Todoki, "High performance logistic regression for privacy-preserving genome analysis," *BMC Medical Genomics*, vol. 14, no. 1, pp. 1–18, 2021.

[33] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft, "Privacy-preserving ridge regression on hundreds of millions of records," in *S&P'13*, 2013, pp. 334–348.

[34] S. Hu, Q. Wang, J. Wang, S. S. Chow, and Q. Zou, "Securing fast learning! Ridge regression over encrypted big data," in *2016 IEEE Trustcom/BigDataSE/ISPA*, 2016, pp. 19–26.

[35] A. Gascón, P. Schoppmann, B. Balle, M. Raykova, J. Doerner, S. Zahur, and D. Evans, "Privacy-preserving distributed linear regression on high-dimensional data." *Proc. Priv. Enhancing Technol.*, vol. 2017, no. 4, pp. 345–364, 2017.

[36] I. Giacomelli, S. Jha, M. Joye, C. D. Page, and K. Yoon, "Privacy-preserving ridge regression with only linearly-homomorphic encryption," in *ACNS'18*, ser. Lecture Notes in Computer Science, vol. 10892. Springer, 2018, pp. 243–261.

[37] A. Akavia, H. Shaul, M. Weiss, and Z. Yakhini, "Linear-regression on packed encrypted data in the two-server model," in *WAHC@CCS'19*. ACM, 2019, pp. 21–32.

[38] The TCGA Research Network, https://www.cancer.gov/tcga, accessed 2022-08-16.

[39] M. R. Aure, I. Steinfeld, L. O. Baumbusch, K. Liestøl, D. Lipson, S. Nyberg, B. Naume, K. K. Sahlberg, V. N. Kristensen, A.-L. Børresen-Dale *et al.*, "Identifying in-trans process associated genes in breast cancer by integrated analysis of copy number and expression data," *PloS one*, vol. 8, no. 1, p. e53014, 2013.

[40] S. Ben-Elazar, M. R. Aure, K. Jonsdottir, S.-K. Leivonen, V. N. Kristensen, E. A. Janssen, K. Kleivi Sahlberg, O. C. Lingjærde, and Z. Yakhini, "miRNA normalization enables joint analysis of several datasets to increase sensitivity and to reveal novel miRNAs differentially expressed in breast cancer," *PLoS computational biology*, vol. 17, no. 2, p. e1008608, 2021.

[41] A. H. Sims, G. J. Smethurst, Y. Hey, M. J. Okoniewski, S. D. Pepper, A. Howell, C. J. Miller, and R. B. Clarke, "The removal of multiplicative, systematic bias allows integration of breast cancer gene expression datasets–improving meta-analysis and prediction of prognosis," *BMC medical genomics*, vol. 1, no. 1, pp. 1–14, 2008.

[42] S. M. Gibbons, C. Duvallet, and E. J. Alm, "Correcting for batch effects in case-control microbiome studies," *PLoS computational biology*, vol. 14, no. 4, p. e1006102, 2018.

[43] B. Galili, X. Tekpli, V. N. Kristensen, and Z. Yakhini, "Efficient gene expression signature for a breast cancer immuno-subtype," *Plos one*, vol. 16, no. 1, p. e0245215, 2021.

[44] P. S. Wang, M. J. T. Guy, and J. H. Davenport, "P-adic reconstruction of rational numbers," *ACM SIGSAM Bulletin*, vol. 16, no. 2, pp. 2–3, 1982.

[45] P. Fouque, J. Stern, and J. Wackers, "Cryptocomputing with rationals," in *FC'02*, 2002, pp. 136–146.

[46] F. Blom, N. J. Bouman, B. Schoenmakers, and N. de Vreede, "Efficient secure ridge regression from randomized Gaussian elimination," in *CSCML'21*, ser. Lecture Notes in Computer Science, vol. 12716. Springer, 2021, pp. 301–316.

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

[48] R. L. Rivest, L. Adleman, and M. L. Dertouzos, "On data banks and privacy homomorphisms," 1978.

[49] S. Halevi, "Homomorphic encryption," in *Tutorials on the Foundations of Cryptography*. Springer, 2017, pp. 219–276.

[50] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.

[51] C. M. Bishop, *Pattern recognition and machine learning*, ser. Information science and statistics. Springer, 2007.

[52] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*. Springer, 2013, vol. 112.

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

[54] D. L. Donoho, Y. Tsaig, I. Drori, and J.-L. Starck, "Sparse solution of underdetermined systems of linear equations by stagewise orthogonal matching pursuit," *IEEE transactions on Information Theory*, vol. 58, no. 2, pp. 1094–1121, 2012.

[55] A. Ben-Dor, N. Friedman, and Z. Yakhini, "Class discovery in gene expression data," in *RECOMB'21*, 2001, pp. 31–38.

[56] I. J. Park, Y. S. Yu, B. Mustafa, J. Y. Park, Y. B. Seo, G.-D. Kim, J. Kim, C. M. Kim, H. D. Noh, S.-M. Hong, Y. W. Kim, M.-J. Kim, A. A. Ansari, L. Buonaguro, S.-M. Ahn, and C.-S. Yu, "A nine-gene signature for predicting the response to preoperative chemoradiotherapy in patients with locally advanced rectal cancer," *Cancers*, vol. 12, no. 4, p. 800, Mar. 2020.

[57] M. Wu, X. Li, T. Zhang, Z. Liu, and Y. Zhao, "Identification of a nine-gene signature and establishment of a prognostic nomogram predicting overall survival of pancreatic cancer," *Frontiers in Oncology*, vol. 9, Sep. 2019.

[58] A. Miller, *Subset selection in regression*. CRC Press, 2002.

[59] P. Mitra, C. Murthy, and S. K. Pal, "Unsupervised feature selection using feature similarity," *IEEE transactions on pattern analysis and machine intelligence*, vol. 24, no. 3, pp. 301–312, 2002.

[60] S. G. Akl, "Bitonic sort," in *Encyclopedia of Parallel Computing*. Springer, 2011, pp. 139–146.

[61] S. Halevi and V. Shoup, "Algorithms in helib," in *Annual Cryptology Conference*. Springer, 2014, pp. 554–571.

[62] "Microsoft SEAL (release 4.0)," https://github.com/Microsoft/SEAL, Mar. 2022, microsoft Research, Redmond, WA.

[63] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," in *ITCS'12*. ACM, 2012, pp. 309–325.

[64] X. Jiang, M. Kim, K. Lauter, and Y. Song, "Secure outsourced matrix computation and application to neural networks," in *CCS'18*, 2018, p. 1209–1222.

[65] E. Aharoni, A. Adir, M. Baruch, N. Drucker, G. Ezov, A. Farkash, L. Greenberg, R. Masalha, G. Moshkowich, D. Murik, H. Shaul, and O. Soceanu, "Helayers: A tile tensors framework for large neural networks on encrypted data." *Proc. Priv. Enhancing Technol. To appear.*, vol. 2023, 2023. [Online]. Available: https://arxiv.org/abs/2011.01805

The Setup Phase

Public parameters: $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$, κ, n, d, N, D as in Figure 2.
Input: the parties have no private inputs.
Output: encryption keys $(\mathsf{pk}_N, \mathsf{sk}_N)$ and $(\mathsf{pk}_D, \mathsf{sk}_D)$ for $\mathcal{S}_2$, and public keys $\mathsf{pk}_N, \mathsf{pk}_D$ for all other parties. The output of $\mathcal{S}_1$ additionally includes encryptions $\mathbf{P}_2, \mathbf{P}_2^T$ of P_2, P_2^T (respectively) for a random permutation matrix $P_2 \in \mathbb{Z}_N^{d\times d}$.
Steps: $\mathcal{S}_2$ performs the following:

1) Generates encryption keys $(\mathsf{pk}_N, \mathsf{sk}_N) \leftarrow \mathsf{KeyGen}(1^\kappa, N)$ and $(\mathsf{pk}_D, \mathsf{sk}_D) \leftarrow \mathsf{KeyGen}(1^\kappa, D)$.
2) Picks a random permutation matrix $P_2 \in \mathbb{Z}_N^{d\times d}$, and encrypts $\mathbf{P}_2 \leftarrow \mathsf{Enc}(\mathsf{pk}_N, P_2)$, and $\mathbf{P}_2^T \leftarrow \mathsf{Enc}(\mathsf{pk}_N, P_2^T)$.
3) Sends $\mathbf{P}_2$ and $\mathbf{P}_2^T$ to $\mathcal{S}_1$, and publishes $\mathsf{pk}_N, \mathsf{pk}_D$.

The Data Uploading and Merging Phase

Public parameters: $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$, κ n, d, N, ℓ, $n_1, \ldots, n_m$, and m as in Figure 2.
Input from previous phase: all parties take as input the public encryption key pk_N.
Input: for every $j \in [m]$, the input of data owner DO_j consists of a matrix $X^j \in \mathbb{R}^{n_j \times d}$, and a vector $\vec{y}^j \in \mathbb{R}^{n_j}$, given with precision ℓ.
Output for the next phase: the output of $\mathcal{S}_1$ are encryptions $\mathbf{A}, \vec{\mathbf{b}}$ under key pk_N of a matrix $A = P_2^T P_1^T \cdot \left(X^T \cdot X + \lambda I\right) \cdot P_1 P_2 \in \mathbb{Z}_N^{d\times d}$ and a vector $\vec{b} = P_2^T P_1^T \cdot X^T \cdot \vec{y} \in \mathbb{Z}_N^d$, respectively, where P_1 is a random permutation matrix. The other parties have no output.
Steps: for every $1 \le j \le m$, DO_j does the following:

1) **Data Representation:** Scales its inputs $X^j, \vec{y}^j$ to have entries in $\mathbb{Z}$,and then embeds them in $\mathbb{Z}_N$. Then, DO_j computes $A^j = \left(X^j\right)^T \cdot X^j$, and $\vec{b}^j = \left(X^j\right)^T \cdot \vec{y}^j$.
2) **Data Encryption:** Encrypts $\mathbf{A}^j \leftarrow \mathsf{Enc}\left(\mathsf{pk}_N, A^j\right)$ and $\vec{\mathbf{b}}^j \leftarrow \mathsf{Enc}\left(\mathsf{pk}_N, \vec{b}^j\right)$, and sends $\mathbf{A}^j, \vec{\mathbf{b}}^j$ to $\mathcal{S}_1$.
3) **Data Merging and Permuting:** $\mathcal{S}_1$ executes the data merging and permuting algorithm of Figure 6, to obtain $\mathbf{A}, \vec{\mathbf{b}}$.

Figure 5: Setup and Data Uploading Phases.

Data Merging and Permuting Step

The algorithm is executed locally by $\mathcal{S}_1$ and uses the circuits Add_λ, Add, $\mathsf{MatMultL}_M$, $\mathsf{MatMultR}_M$, $\mathsf{MatMult}$ and $\mathsf{MatVecMult}$ of Section 4.3.
Public parameters: $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$, κ, m, n, d, N, λ, as in Figure 2.
Inputs from previous phase: the public encryption key pk_N, and, for every $j \in [m]$, encryptions $\mathbf{A}^j, \vec{\mathbf{b}}^j$ of $A^j \in \mathbb{Z}_N^{d\times d}, \vec{b}^j \in \mathbb{Z}_N^d$, as well as encryptions $\mathbf{P}_2, \mathbf{P}_2^T$ of P_2, P_2^T (respectively) under key pk_N.
Output: encryptions $\mathbf{A}$, $\vec{\mathbf{b}}$ of $A = P_2^T P_1^T \left(\sum_{j\in[m]} A^j + \lambda I\right) P_1 P_2$ and $\vec{b} = P_2^T P_1^T \sum_{j\in[m]} \vec{b}^j$, respectively, under key pk_N, where $P_1 \in \mathbb{Z}_N^{d\times d}$ is a random permutation matrix. (Notice that A is a permuted version of $\sum_{j\in[m]} A^j + \lambda I$, and $\vec{b}$ is the corresponding representation of $X^T\vec{y}$.)
Steps: $\mathcal{S}_1$ performs the following:

1) **Data Merging:** Homomorphically computes $T^1 = \sum_{j\in[m]} A^j + \lambda I$ ($I \in \mathbb{Z}_N^{d\times d}$ is the identity matrix) and $\vec{t} = \sum_{j\in[m]} \vec{b}^j$ by computing $\boldsymbol{T}^1 \leftarrow \mathsf{Eval}\left(\mathsf{pk}_N, \mathsf{Add}_\lambda, \mathbf{A}^1, \ldots, \mathbf{A}^m\right)$, and $\vec{\boldsymbol{t}} \leftarrow \mathsf{Eval}\left(\mathsf{pk}_N, \mathsf{Add}, \vec{\mathbf{b}}^1, \ldots, \vec{\mathbf{b}}^m\right)$. (We note that $T^1 = X^T \cdot X + \lambda I$, and $\vec{t} = X^T \cdot \vec{y}$) .)
2) **Permutation Generation:** Picks a random permutation matrix $P_1 \in \mathbb{Z}_N^{d\times d}$, and homomorphically computes $P = P_1 \cdot P_2$ and P^T by performing $\mathbf{P} \leftarrow \mathsf{Eval}\left(\mathsf{pk}_N, \mathsf{MatMultL}_{P_1}, \mathbf{P}_2\right)$ and $\mathbf{P}^T \leftarrow \mathsf{Eval}\left(\mathsf{pk}_N, \mathsf{MatMultR}_{P_1^T}, \mathbf{P}_2^T\right)$.
3) **Data Permuting:** Homomorphically computes $A = P^T \cdot T^1 \cdot P$ and $\vec{b} = P^T \cdot \vec{t}$ as follows: $\boldsymbol{T}^2 \leftarrow \mathsf{Eval}\left(\mathsf{pk}_N, \mathsf{MatMult}, \mathbf{P}^T, \boldsymbol{T}^1\right)$. $\mathbf{A} \leftarrow \mathsf{Eval}\left(\mathsf{pk}_N, \mathsf{MatMult}, \boldsymbol{T}^2, \mathbf{P}\right)$. $\vec{\mathbf{b}} \leftarrow \mathsf{Eval}\left(\mathsf{pk}_N, \mathsf{MatVecMult}, \mathbf{P}^T, \vec{\boldsymbol{t}}\right)$.

Figure 6: Data Merging and Permuting Step (prepares input for iterations).

Appendix

7. Additional Preliminaries

7.1. FHE

Fully-Homomorphic Encryption (FHE) is a public-key encryption scheme $\mathcal{E} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ that allows one to compute "under the hood" of the encryption (without the secret decryption key). We use the scheme as a black-box, and only make the assumption that during key generation, one can choose the *plaintext space* by specifying an $N \in \mathbb{N}$, so that homomorphic operations are performed modulo N. (We note that this is the case in many FHE candidates, e.g. [63].) This assumption is captured by incorporating the plaintext modulus N explicitly into the syntax of the scheme. We now formally define FHEs. The algorithms are required to satisfy the following three properties: (1) standard decryption correctness. (2) FHE correctness, in the sense that the ciphertext obtained by evaluating a circuit C (using Eval) on a set of ciphertexts

Selection of Smallest Features Phase

The protocol uses the circuits Square, $\mathsf{SubNums}$, $\mathsf{AddConst}_c$, $\mathsf{BinCompareR}_c$, $\mathsf{BinCompareL}_c$, Neg, $\mathsf{MultNums}$ and $\mathsf{AddNums}$ of Section 4.3. All computations are in $\mathbb{Z}_D$.

Public parameter: $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ and N, D as in Figure 2. We denote $l = \lceil \log N \rceil$.

Inputs from previous phases: both servers take as input the set $F \subseteq [d]$ of the currently used feature indices, and public encryption keys $\mathsf{pk}_N, \mathsf{pk}_D$, as well as a value k such that $1 \le k < |F|$ which determines the size of the output set S_{del}. $\mathcal{S}_1$ additionally has as input a vector $\vec{\mathbf{z}}$, which is an entry-wise encryption of a vector $\vec{z} \in \mathbb{Z}_N^d$, where $z_i < \sqrt{N}$ for every $i \in [d]$ (here and below, z_i denotes the i'th entry of $\vec{z}$). $\mathcal{S}_2$ additionally has as input the private decryption key sk_N.

Output: the output of $\mathcal{S}_1, \mathcal{S}_2$ is a subset $S_{\mathsf{del}} \subseteq [|F|]$ of size k which contains the indices of the smallest entries in $\vec{z}$.

Steps:

1) **Compute masked differences:** $serv_1$ sends to $\mathcal{S}_2$ the values $\{\mathbf{df}_{i,j}\}_{i,j \in F, i \neq j}$ which are computed as follows:
   - For every $i \in F$, homomorphically computes the value $z_i' := z_i^2$ by computing $\mathbf{z}_i' \leftarrow \mathsf{Eval}\left(\mathsf{pk}_N, \mathsf{Square}, \mathbf{z}_i\right)$.
   - For every $i, j \in F$ such that $i \neq j$, picks a random $r_{i,j} \leftarrow \mathbb{Z}_N$, and computes $\mathbf{t}_{i,j} \leftarrow \mathsf{Eval}\left(\mathsf{pk}_N, \mathsf{SubNums}, \mathbf{z}_i', \mathbf{z}_j'\right)$ and $\mathbf{df}_{i,j} \leftarrow \mathsf{Eval}\left(\mathsf{pk}_N, \mathsf{AddConst}_{r_{i,j}}, \mathbf{t}_{i,j}\right)$. ($\mathsf{df}_{i,j}$ is the difference $z_i^2 - z_j^2$, masked with the random $r_{i,j}$.)
2) **Compute non-negative ranges of masked differences:** $\mathcal{S}_2$ performs the following for every $i, j \in F, i \neq j$:
   - Decrypts $\mathsf{df}_{i,j} = \mathsf{Dec}\left(\mathsf{sk}_N, \mathbf{df}_{i,j}\right)$.
   - If $\mathsf{df}_{i,j} < \frac{N-1}{2} + 1$ then set $\mathsf{rangeMin}'_{i,j} = \mathsf{df}_{i,j}$, $\mathsf{rangeMax}'_{i,j} = \mathsf{df}_{i,j} + \frac{N-1}{2} + 1$ (the sum is computed over the integers), and $\mathsf{Negative}_{i,j} = 0$.
   - Else, set $\mathsf{rangeMin}'_{i,j} = \mathsf{df}_{i,j} - \frac{N-1}{2}$ (computed over the integers), $\mathsf{rangeMax}'_{i,j} = \mathsf{df}_{i,j} + 1$, and $\mathsf{Negative}_{i,j} = 1$.
   - Let $\overrightarrow{\mathsf{rangeMin}}_{i,j}, \overrightarrow{\mathsf{rangeMax}}_{i,j}$ denote the binary representation of $\mathsf{rangeMin}'_{i,j}$ and $\mathsf{rangeMax}'_{i,j}$ as length-l strings, respectively. $\mathcal{S}_2$ encrypts $\overrightarrow{\mathbf{rangeMin}}_{i,j} \leftarrow \mathsf{Enc}\left(\mathsf{pk}_D, \overrightarrow{\mathsf{rangeMin}}_{i,j}\right)$, $\overrightarrow{\mathbf{rangeMax}}_{i,j} \leftarrow \mathsf{Enc}\left(\mathsf{pk}_D, \overrightarrow{\mathsf{rangeMax}}_{i,j}\right)$, and $\mathbf{Negative}_{i,j} \leftarrow \mathsf{Enc}\left(\mathsf{pk}_D, \mathsf{Negative}_{i,j}\right)$.

   Then, $\mathcal{S}_2$ sends $\left\{\overrightarrow{\mathbf{rangeMin}}_{i,j}, \overrightarrow{\mathbf{rangeMax}}_{i,j}, \mathbf{Negative}_{i,j}\right\}_{i,j \in F, i \neq j}$ to $\mathcal{S}_1$.
3) **Compute ordering:** $\mathcal{S}_1$ homomorphically computes the ordering of $z_1, \ldots, z_d$, where the order of index i is the number of indices $j \in F$ such that $z_i < z_j$ (as elements of $\mathbb{Z}_N$). Specifically, $\mathcal{S}_1$ performs the following:
   - For every $i \in F$, encrypts $\mathbf{Ord}_i \leftarrow \mathsf{Enc}\left(\mathsf{pk}_D, 0\right)$. ($\mathsf{Ord}_i$ is the location of index i in the ordering; intuitively, it is initialized to 0 and will be increased for every $j \neq i$ such that $z_j \le z_i$.)
   - For every $i, j \in F, i \neq j$, let $\vec{r}_{i,j}$ denote the length-l bit string representing $r_{i,j}$, then $\mathcal{S}_1$ checks whether $r_{i,j} \in \left[\mathsf{rangeMin}_{i,j}, \mathsf{rangeMax}_{i,j}\right)$. This is done homomorphically as follows.
     - $\mathcal{S}_1$ homomorphically computes two indicators: $\mathbf{IsBigger}_{i,j} \leftarrow \mathsf{Eval}\left(\mathsf{pk}_D, \mathsf{BinCompareR}_{\vec{r}_{i,j}}, \overrightarrow{\mathbf{rangeMin}}_{i,j}\right)$, $\mathbf{IsSmaller}_{i,j} \leftarrow \mathsf{Eval}\left(\mathsf{pk}_D, \mathsf{BinCompareL}_{\vec{r}_{i,j}}, \overrightarrow{\mathbf{rangeMax}}_{i,j}\right)$ (Intuitively, $\mathsf{IsBigger}_{i,j} = \mathsf{IsSmaller}_{i,j} = 1$ if and only if $\mathsf{rangeMin}_{i,j} \le r_{i,j} < \mathsf{rangeMax}_{i,j}$.)
     - Next, $\mathcal{S}_1$ increases Ord_i by

       $$\mathsf{ost}_{i,j} := \mathsf{IsBigger}_{i,j} \cdot \mathsf{IsSmaller}_{i,j} \cdot \mathsf{Negative}_{i,j} + \left(1 - \mathsf{IsBigger}_{i,j} \cdot \mathsf{IsSmaller}_{i,j}\right) \cdot \left(1 - \mathsf{Negative}_{i,j}\right).$$

       (Intuitively, $\mathsf{ost}_{i,j} \in \{0,1\}$ is 1 if and only if $z_i > z_j$, which happens if either: (1) $z_i^2 - z_j^2 + r_{i,j}$ is "negative", which is denoted by $\mathsf{Negative}_{i,j} = 1$, and then $z_i^2 - z_j^2$ is "positive" if and only if $\mathsf{rangeMin}_{i,j} \le r_{i,j} < \mathsf{rangeMax}_{i,j}$, in which case the first summand is 1; or (2) $z_i^2 - z_j^2 + r_{i,j}$ is "non-negative", in which case $\mathsf{Negative}_{i,j} = 0$, and additionally $r_{i,j} < \mathsf{rangeMin}_{i,j}$ or $\mathsf{rangeMax}_{i,j} \le r_{i,j}$, i.e., the second summand is 1.) This is done by homomorphically computing the following values:

       a) The negations $\mathsf{notInRange}_{i,j}$ and $\mathsf{NotNegative}_{i,j}$ of $\mathsf{IsBigger}_{i,j} \cdot \mathsf{IsSmaller}_{i,j}$ and $\mathsf{Negative}_{i,j}$ (respectively), computed as $\mathbf{inRange}_{i,j} \leftarrow \mathsf{Eval}\left(\mathsf{pk}_D, \mathsf{MultNums}, \mathbf{IsBigger}_{i,j}, \mathbf{IsSmaller}_{i,j}\right)$, $\mathbf{notInRange}_{i,j} \leftarrow \mathsf{Eval}\left(\mathsf{pk}_D, \mathsf{Neg}, \mathbf{inRange}_{i,j}\right)$,$\mathbf{NotNegative}_{i,j} \leftarrow \mathsf{Eval}\left(\mathsf{pk}_D, \mathsf{Neg}, \mathbf{Negative}_{i,j}\right)$
       b) $\boldsymbol{t}_{i,j}^1 \leftarrow \mathsf{Eval}\left(\mathsf{pk}_D, \mathsf{MultNums}, \mathbf{IsBigger}_{i,j}, \mathbf{IsSmaller}_{i,j}\right)$. (Then $t_{i,j}^1 = \mathsf{IsBigger}_{i,j} \cdot \mathsf{IsSmaller}_{i,j}$.)
       c) $\boldsymbol{t}_{i,j}^2 \leftarrow \mathsf{Eval}\left(\mathsf{pk}_D, \mathsf{MultNums}, \boldsymbol{t}_{i,j}^1, \mathbf{Negative}_{i,j}\right)$. (Then $t_{i,j}^2 = \mathsf{IsBigger}_{i,j} \cdot \mathsf{IsSmaller}_{i,j} \cdot \mathsf{Negative}_{i,j}$.)
       d) $\boldsymbol{t}_{i,j}^3 \leftarrow \mathsf{Eval}\left(\mathsf{pk}_D, \mathsf{MultNums}, \mathbf{notInRange}_{i,j}, \mathbf{NotNegative}_{i,j}\right)$. (Then $t_{i,j}^3 = \left(1 - \mathsf{IsBigger}_{i,j} \cdot \mathsf{IsSmaller}_{i,j}\right) \cdot \left(1 - \mathsf{Negative}_{i,j}\right)$.)
       e) $\boldsymbol{t}_{i,j}^4 \leftarrow \mathsf{Eval}\left(\mathsf{pk}_D, \mathsf{AddNums}, \boldsymbol{t}_{i,j}^2, \boldsymbol{t}_{i,j}^3\right)$. (Then $t_{i,j}^4 = \mathsf{ost}_{i,j}$.)
       f) $\mathbf{Ord}_i \leftarrow \mathsf{Eval}\left(\mathsf{pk}_D, \mathsf{AddNums}, \mathbf{Ord}_i, \boldsymbol{t}_{i,j}^4\right)$. (This increases Ord_i by $\mathsf{ost}_{i,j}$.)
4) **Permuting ordinals:** $\mathcal{S}_1$ draws a random permutation Π on F, and computes the vector $\overrightarrow{\mathbf{Ord}}^{\Pi} \in \mathbb{Z}_D^{|F|}$ such that $\overrightarrow{\mathbf{Ord}}_i^{\Pi} = \mathbf{Ord}_{\Pi(i)}$ for every $i \in F$. Then, $\mathcal{S}_1$ sends $\overrightarrow{\mathbf{Ord}}^{\Pi}$ to $\mathcal{S}_2$.
5) **Computing k smallest features:** for every $i \in F$, $\mathcal{S}_2$ decrypts $\overrightarrow{\mathsf{Ord}}_i^{\Pi} = \mathsf{Dec}\left(\mathsf{sk}_D, \overrightarrow{\mathbf{Ord}}_i^{\Pi}\right)$, and generates a vector $\chi^{\Pi} \in \{0,1\}^d$ by setting $\chi_i^{\Pi} = 1$ if $\overrightarrow{\mathsf{Ord}}_i^{\Pi} < k$, otherwise setting $\chi_i^{\Pi} = 0$. $\mathcal{S}_2$ sends $\vec{\chi}^{\Pi} := \left(\chi_1^{\Pi}, \ldots, \chi_{|F|}^{\Pi}\right)$ to $\mathcal{S}_1$.
6) **Output:** S_1 computes $S_{\mathsf{del}} = \left\{i : \vec{\chi}_{\Pi(i)}^{\Pi} = 1\right\}$ and sends it to $\mathcal{S}_2$. Both servers set S_{del} to be their output for the phase.

Figure 7: Learning Phase: selecting which features to remove in the current iteration.

Compacting Phase

The algorithm is executed locally by $\mathcal{S}_1$ and uses the circuits $\mathsf{MatMultR}_M$ and $\mathsf{MatVecMultL}_M$ of Section 4.3.

Public parameter: $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ and N as in Figure 2.

Inputs from previous phases: a set F^* of the indices of features that survived the current iteration, the public encryption key pk_N, and encryptions $\mathbf{A}, \vec{\mathbf{b}}$ of $A \in \mathbb{Z}_N^{d\times d}$ and $\vec{b} \in \mathbb{Z}_N^d$, respectively.

Output: encryptions $\mathbf{A}_{F^*}, \vec{\mathbf{b}}_{F^*}$ of $A_{F^*} = \mathsf{pjct}_{F^*}(A) \in \mathbb{Z}_N^{d\times d}$ and $\vec{b}_{F^*} = \mathsf{pjct}_{F^*}\left(\vec{b}\right) \in \mathbb{Z}_N^d$ (i.e., the projection of $A, \vec{b}$ to $[F^*]$).

Steps:

1) **Compacting:** $\mathcal{S}_1$ compacts the data by homomorphically computing $A_{F^*} = \mathcal{N}_{F^*} \cdot A \cdot \mathcal{N}_{F^*}$ and $\vec{b}_{F^*} = \mathcal{N}_{F^*} \cdot \vec{b}$, by performing:
   - $\mathbf{T} \leftarrow \mathsf{Eval}(\mathsf{pk}_N, \mathsf{MatMultR}_{\mathcal{N}_{F^*}}, \mathbf{A})$. (Notice that $\mathbf{T}$ encrypts $A \cdot \mathcal{N}_{F^*}$.)
   - $\mathbf{A}_{F^*} \leftarrow \mathsf{Eval}(\mathsf{pk}_N, \mathsf{MatMultL}_{\mathcal{N}_{F^*}}, \mathbf{T})$.
   - $\vec{\mathbf{b}}_{F^*} \leftarrow \mathsf{Eval}\left(\mathsf{pk}_N, \mathsf{MatVecMultL}_{\mathcal{N}_{F^*}}, \vec{\mathbf{b}}\right)$.

   (Notice that for every $i \notin F^*$, the ith row and column in A_{F^*} are $\vec{0}$, and all other entries are identical to the corresponding entry in A. Similarly, for every $i \notin F^*$ the ith entry of $\vec{b}_{F^*}$ is 0, otherwise it equals the corresponding entry in $\vec{b}$.)
2) **Output:** $\mathcal{S}_1$ outputs $\mathbf{A}_{F^*}, \vec{\mathbf{b}}_{F^*}$.

Figure 8: Learning Phase: Compacting current data.

decrypts to the value that would have been obtained by evaulating C directly on the underlying messages. (3) Computational indistinguishability, namely for every κ, N, and every $\mathsf{msg} \in \mathbb{Z}_N$, the joint distribution of pk (i.e., a public key randomly generated by KG) and $\mathsf{c} \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathsf{msg})$ is computationally indistinguishable from the joint distribution of pk and $\mathsf{c}_0 \leftarrow \mathsf{Enc}(\mathsf{pk}, 0)$. This is formalized in the following definition.

Definition 7.1 (FHE)**.** A *Fully-Homomorphic Encryption (FHE)* scheme $\mathcal{E} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ consists of four algorithms where $\mathsf{KG}, \mathsf{Enc}$ and Eval are PPT algorithms, and Dec is (deterministic) polynomial time. The algorithms have the following syntax:

- $\mathsf{KG}(1^\kappa, N)$ takes as input a security parameter κ, and an $N \in \mathbb{N}$. It outputs a pair of public and secret keys $(\mathsf{pk}, \mathsf{sk})$. We assume without loss of generality that pk includes N in its description.
- $\mathsf{Enc}(\mathsf{pk}, \mathsf{msg})$ takes as input a public key pk, and a message $\mathsf{msg} \in \mathbb{Z}_N$, and outputs a ciphertext c.
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{c})$ takes as input a secret decryption key sk, and a ciphertext c, and outputs a plaintext message msg'.
- $\mathsf{Eval}(\mathsf{pk}, C, \mathsf{c}_1, \ldots, \mathsf{c}_k)$ takes as input a public key pk, a circuit $C : \mathbb{Z}_N^k \to \mathbb{Z}_N^l$ for some $l, k \in \mathbb{N}$, and k ciphertexts $\mathsf{c}_1, \ldots, \mathsf{c}_k$, and outputs l ciphertexts $(\mathsf{c}'_1, \ldots, \mathsf{c}'_l)$.

The scheme is required to satisfy the following semantic properties.

- **Correctness.** For every natural N, every security parameter κ, and every message $\mathsf{msg} \in \mathbb{Z}_N$:
$$\Pr\left[\mathsf{msg} = \mathsf{msg}' : \begin{array}{rl} (\mathsf{pk}, \mathsf{sk}) & \leftarrow \mathsf{KG}(1^\kappa, N) \\ \mathsf{c} & \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathsf{msg}) \\ \mathsf{msg}' & = \mathsf{Dec}(\mathsf{sk}, \mathsf{c}) \end{array}\right]$$
is at least $1 - \mathsf{negl}(\kappa)$, where the probability is over the randomness of KG and Enc.
- **FHE Correctness.** For every natural N, every security parameter κ, every $k, l \in \mathbb{N}$, every arithmetic circuit $C : \mathbb{Z}_N^k \to \mathbb{Z}_N^l$, and every $\mathsf{msg}_1, \ldots, \mathsf{msg}_k \in \mathbb{Z}_N$:
$$\Pr\left[\mathsf{msg} = \mathsf{msg}' : \begin{array}{rl} (\mathsf{pk}, \mathsf{sk}) & \leftarrow \mathsf{KG}(1^\kappa, N) \\ \mathsf{c}_i & \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathsf{msg}_i),\ \forall i \in [k] \\ \mathsf{c} & \leftarrow \mathsf{Eval}(\mathsf{pk}, C, (\mathsf{c}_1, \ldots, \mathsf{c}_k)) \\ \mathsf{msg} & = \mathsf{Dec}(\mathsf{sk}, \mathsf{c}) \end{array}\right]$$
is at least $1 - \mathsf{negl}(\kappa)$, where $\mathsf{msg}' = C(\mathsf{msg}_1, \ldots, \mathsf{msg}_l)$, and the probability is over the randomness of $\mathsf{KG}, \mathsf{Enc}$ and Eval.
- **Computational security (non-uniform distinguishers).** For every κ, all public parameters params, and every $\mathsf{msg} \in Q$, the following distributions are computationally indistinguishable by non-uniform distinguishers:
  - Sample $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KG}(1^\kappa, \mathsf{params})$, and $\mathsf{c} \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathsf{msg})$.[15] Output $(\mathsf{pk}, \mathsf{c})$.
  - Sample $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KG}(1^\kappa, \mathsf{params})$, and $\mathsf{c} \leftarrow \mathsf{Enc}(\mathsf{pk}, 0)$. Output $(\mathsf{pk}, \mathsf{c})$.

Remark 7.2. Though we define FHE schemes as encrypting a single ring element, we also consider FHE schemes encrypting vectors or matrices of ring elements, namely Enc might take as input a vector or matrix of ring elements, and Dec might take as input a vector or matrix of ciphertexts (each encrypting a field element). See Section 5 for further details.

15. See Remark 7.2 below about simultaneously encrypting multiple field elements.

Computing and Unpermuting Output Phase

The protocol uses the circuits $\mathsf{ScalerVecMult}$ and $\mathsf{MatVecMult}$ of Section 4.3.

Public parameter: $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$, n, d, m, s, N as in Figure 2.

Inputs from previous phases: the input of $\mathcal{S}_1$ consists of the public encryption key pk_N, encryptions $\mathbf{A}_F, \vec{\mathbf{b}}_F$ of $A_F, \vec{b}_F$ (respectively), the set F of the features that survived to this iteration, and an encryption $\mathbf{P}$ of a permutation matrix P. The input of $\mathcal{S}_2$ consists of F and the keys $\mathsf{pk}_N, \mathsf{sk}_N$. The data owners have no input.

Output: the output of all parties is an s-sparse model $\vec{w}$.

Steps:

1) **Learning step:**
   - $\mathcal{S}_1$ and $\mathcal{S}_2$ execute the scaled ridge protocol of Figure 3 (where the input of $\mathcal{S}_1$ is $F, \mathsf{pk}_N, \mathbf{A}_F, \vec{\mathbf{b}}_F$, and the input of $\mathcal{S}_2$ is F, sk_N) with the following changes:
     - The output of $\mathcal{S}_1$ is an *entry-by-entry* encryption $\vec{\mathbf{z}}_F$ of a vector $\vec{z}_F \in \mathbb{Z}_N^d$.
     - Let $\Delta = \det(\Gamma)$ denote the determinant which $\mathcal{S}_2$ computes in Step 4 of Figure 3. Then $\mathcal{S}_2$ encrypts $\mathbf{\Delta^{-1}} \leftarrow \mathsf{Enc}\left(\mathsf{pk}_N, \Delta^{-1}\right)$, and sends $\mathbf{\Delta^{-1}}$ to $\mathcal{S}_1$.
   - Recall that $\vec{z}_F = \mathsf{expd}_F\left(\det\left(A'_F\right) \cdot \vec{w}'_F\right)$, where $A'_F = \mathsf{pjct}_F\left(A_F\right) \in \mathbb{Z}_N^{|F| \times |F|}$ and $\vec{w}'_F \in \mathbb{Z}_N^{|F|}$ were defined in Figure 3. $\mathcal{S}_1$ homomorphically computes $\vec{w}_F = \mathsf{expd}_F\left(\vec{w}'_F\right)$ as follows:
     - let $D = \det\left(A'_F\right)$, then $\mathcal{S}_1$ homomorphically computes D^{-1} by computing $\mathbf{D}^{-1} \leftarrow \mathsf{Eval}\left(\mathsf{pk}_N, \mathsf{ScalerVecMult}_{(\det(R'))^{-1}}, \mathbf{\Delta^{-1}}\right)$. (This step computes D^{-1} because
     $$\Delta = \det(\Gamma) = \det\left(A'_F \cdot R'\right) = \det\left(R'\right) \cdot \det\left(A'_F\right).)$$
     - homomorphically computes $\vec{\mathbf{w}}_F \leftarrow \mathsf{Eval}\left(\mathsf{pk}_N, \mathsf{ScalerVecMult}, \vec{\mathbf{z}}_F, \mathbf{D}^{-1}\right)$. (Notice that $\vec{w}_F = \mathsf{expd}_F\left(\vec{w}'_F\right)$, where $A'_F \cdot \vec{w}'_F = \mathsf{pjct}_F\left(\vec{b}_F\right)$.)
2) **Unpermuting:** $\mathcal{S}_1$ homomorphically unpermutes $\vec{w}_F$ to obtain $\vec{w}^{\mathsf{int}} = P \cdot \vec{w}_F$ by computing $\vec{\mathbf{w}}^{\mathsf{int}} \leftarrow \mathsf{Eval}\left(\mathsf{pk}_N, \mathsf{MatVecMult}, \mathbf{P}, \vec{w}_F\right)$, and sends $\vec{\mathbf{w}}^{\mathsf{int}}$ to $\mathcal{S}_2$. (Notice that $\vec{w}_F$ contains the correct weights, but they are permuted according to P^T. Multiplying by $P = \left(P^T\right)^{-1}$ thus inverts the permutation.)
3) **Decrypting the output:** $\mathcal{S}_2$ decrypts $\vec{w}^{\mathsf{int}} = \mathsf{Dec}\left(\mathsf{sk}_N, \vec{\mathbf{w}}^{\mathsf{int}}\right)$, recovers from it the model $\vec{w} \in \mathbb{Q}^d$ using rational reconstruction [44], [45], and sends $\vec{w}$ to all parties.

Figure 9: Learning Phase: computing and unmasking the output

8. Security Analysis

In this section we analyze the security of our protocol SIR. We first set some notation.

Notation 8.1. For natural N, d, $\lambda \geq 0$, and $\emptyset \neq F_k \subset \ldots \subset F_1 \subseteq [d]$, let $\mathcal{T}_{\mathsf{Inv},N,d,\lambda,F_1,\ldots,F_k}$ denote the subset of $\mathbb{R}^{n\times d} \times \mathbb{R}^{n\times 1}$ consisting of all $(X, \vec{y})$ such that:

- $A = X^T X + \lambda I$ is invertible in $\mathbb{Z}_N^{d\times d}$.
- For every $1 \leq j \leq k$, $\mathsf{pjct}_{F_j}(A)$ is invertible in $\mathbb{Z}_N^{|F_j|\times|F_j|}$.

Remark 8.2 (On the types of inputs for which SIR is secure)**.** As noted in Section 1, SIR is secure when $d \leq n$ and X is invertible (the same assumption is also made in previous works [36], [37]). More specifically, we only consider security for inputs in $\mathcal{T}_{\mathsf{Inv},N,d,\lambda,F_1,\ldots,F_k}$ (where λ, d and N are as defined in Section 3, and $F_1, \ldots F_k$ are the sets of features that survive iterations $1, \ldots, k$, respectively; notice that the sizes of these sets depend only on the public parameters of SIR). This is similar to the security guarantee of Giacomelli et al. [36], who only consider security for inputs in $\mathcal{T}_{\mathsf{Inv},N,d,\lambda,[d]}$. Our assumption that $(X, \vec{y}) \in \mathcal{T}_{\mathsf{Inv},N,d,\lambda,F_1,\ldots,F_k}$ is a natural extension of the assumption of [36] (that $(X, \vec{y}) \in \mathcal{T}_{\mathsf{Inv},N,d,\lambda,[d]}$) to our setting of iterated computation. We stress that when $d \leq n$, it suffices to assume that X is inevitable (i.e., has degree d). Indeed, in this case X_F – the restriction of X to the rows and columns in $F \subseteq [d]$ – also has full degree (i.e., degree $|F|$), and consequently the matrices $A_F = (X_F)^T X_F$ used in the ridge regression iteration all have full degree. As noted in Section 1, for the case $d > n$ SIR can be combined with another learning method (e.g., filter) to reduce the number of features to n.

Theorem 8.3 (SIR Security)**.** *Let $d < n \in \mathbb{N}$. Then SIR is an iterated ridge implementation of the algorithm of Figure 1 with m-privacy for any $\lambda \geq 0$, and any $N \in \mathbb{N}$ that satisfies Equation 4, assuming the input matrix X is invertible.*

Proof of Theorem 8.3. Correctness Follows from Lemma 8.4 below. As for privacy, let $I \subseteq [m]$ denote the subset of corrupted data owners, and we consider three cases. First, assume that $\mathcal{S}_1$ is corrupted. Then privacy follows from Lemma 8.5 below. Second, assume that $\mathcal{S}_2$ is corrupt, then privacy follows from Lemma 8.6 below. Third, assume that both servers are honest. Since the servers have no input, and the output is public, simulatability follows immediately from simulatability when one of the servers is corrupted. □

The next lemma states that the protocol is correct.

Lemma 8.4 (Correctness)**.** *Protocol SIR is correct. That is, for every input $X, \vec{y}$, the output of SIR is identical to its non-secure version Iterated ridge of Figure 1, except with negligible probability $\mathsf{negl}(\kappa)$.*

Proof. We will prove a stronger claim: we will prove that if the FHE scheme has *perfect* correctness, then the outputs of SIR and Iterated ridge are *identical*. That is, we will prove that the only source of error is the correctness error of the FHE scheme (introduced by either the Eval or the Dec algorithms). This is indeed stronger than the statement of Lemma 8.4, because the correctness error of the FHE scheme is negligible. Therefore, for the remainder of the proof we assume that the FHE scheme is perfectly correct. Consequently, it suffices to prove correctness of a revised version of the protocol in which all values are given in the clear, and all computations are executed directly (not using homomorphic evaluation). This "unencrypted" version of SIR (Figure 2) differs from the non-secure Iterated ridge algorithm (Figure 1) in the following points:[16]

1) **Correctness of computing over a finite ring:** Iterated ridge performs all computations in $\mathbb{R}$, whereas SIR works in finite rings $\mathbb{Z}_N$ and $\mathbb{Z}_D$, and then uses rational reconstruction to recover a sparse model in $\mathbb{Q}^d$.
2) **Correctness of working over permuted inputs:** In SIR, the data used for learning consists of *permuted* versions $A, \vec{b}$ of the merged data $\sum_{j\in[m]} A^j + \lambda I, \sum_{j\in[m]} \vec{b}^j$, whereas in Iterated ridge $A, \vec{b}$ are used directly.
3) **Correctness of the scaled ridge phase:** SIR performs every ridge regression step (Figure 3) over masked data, computes the model $\vec{w} = A^{-1} \cdot \vec{b}$ using the identity $A^{-1} = \det(A) \cdot \mathrm{Adj}(A)$, and outputs a model scaled by $\det(A)$. Moreover, the values Γ, β used for learning are obtained from a $d \times d$ matrix and a length-d vector, which are projected to the set F of surviving feature indices. Iterated ridge, on the other hand, performs this learning phase in the clear (over unmasked data), computes A^{-1} explicitly, and outputs the actual model (instead of a scaled one as in SIR). Moreover, in Iterated ridge the values $A, \vec{b}$ used to perform the learning are the projected matrix and vector obtained from the previous iteration.
4) **Correctness of the selection phase:** The set S_{del} of smallest features that are removed in each iteration are computed from the full set $[d]$ using *masked differences of squares* (i.e., $z_i^2 - z_j^2 + r_{i,j}$), whereas in Iterated ridge it is computed directly from the set of surviving features using the absolute value (represented in $\mathbb{R}$). Also, in SIR these are computed over a permuted ordering, whereas in Iterated ridge it is computed directly on the ordering.

16. We note that this comparison is in terms of the *operations performed* in each of the protocols. Thus, the fact that the computation in SIR is divided between two servers, and in Iterated ridge it is performed by a single server, is irrelevant for us, as we assume that the communication channels do not induce errors (this can be guaranteed using error-correcting codes).

5) **Correctness of the compaction phase:** The data compaction phase in SIR resets the entries of $A, \vec{b}$ corresponding to features that did not survive the iteration, whereas in Iterated ridge these features are removed by projecting $A, \vec{b}$ to the set of indices of surviving features.
6) **Correctness of the final output:** The final learning phase in Iterated ridge simply consists of performing another linear regression step which outputs the final sparse model. SIR uses a dedicated output phase, in which it performs masked ridge regression to obtain a scaled model, rescales it to the actual (permuted) output model, and finally unpermutes the output model.

We now argue that each of these steps preserves correctness (with probability 1) and therefore the protocol itself, which combines all steps, is also correct.

Computing over a finite ring (Item 1): the learning phase computations in SIR are performed in $\mathbb{Z}_N$, whereas $\mathbb{Z}_D$ is only used to compute the ordering between the features. N is chosen such that no overflow will occur (see Remark 4.1), guaranteeing that all computations in $\mathbb{Z}_N$ perfectly emulate computations in $\mathbb{R}$. Moreover, since $D \geq d$, and the ordering contains values between 0 and $d-1$, then overflows do not occur in the computations in $\mathbb{Z}_D$. Moreover, correctness of the rational reconstruction algorithm was proven in [36] (and holds here because N is sufficiently large, see Remark 4.1).

Using permuted inputs (Item 2): permuting $A, \vec{b}$ is equivalent to permuting the columns of the X^j's and $\vec{y}^j$'s. (This holds because of the manner in which we apply the permutation, specifically by multiplying A by P^T from the right and by P from the left.) Put differently, the permutation simply permutes the features, as well as their weights in the output model. Other than that, the computation remains unchanged and, in particular, the computed models (in all iterations) have the "right" weights, but they are permuted according to the same permutation as the input. Following the final learning phase, we unpermute the learned model (by applying the inverse permutation $P = \left(P^T\right)^{-1}$, see Figure 9), thus the permutation does not affect correctness.

Correctness of the scaled ridge phase: Item 3 does not affect correctness due to the following. First, computing A^{-1} as $\det(A) \cdot \mathrm{Adj}(A)$ is based on a mathematical identity. Second, the fact that the output model is scaled by $\det(A)$ does not change the ordering (in absolute value) between the model's entries, and therefore does not affect the following stages of the iteration (in which the model is used to order the features and removes the ones with smallest weights – in absolute value). Next, we prove that the output is the correct (scaled) model $\det\left(A'_F\right) \cdot \vec{w}_F$, even though computations are performed over masked $A, \vec{b}$, and even though the inputs to this phase were *not* projected to $|F|$ (recall that in $A_F, \vec{b}_F$ the rows, columns and entries corresponding to indices not in F were set to 0, but *were not deleted* as they are in Iterated ridge). Referring to Figure 3, the (scaled, encrypted) model which $\mathcal{S}_2$ sends to $\mathcal{S}_1$ is $\vec{\zeta}_F = \mathsf{expd}_F\left(\mathsf{Adj}\left(\Gamma\right) \cdot \vec{\beta}\right)$ (see Remark 4.3 for a description of $\mathsf{expd}_F\left(\cdot\right)$). $\mathcal{S}_1$ then removes the masking by computing

$$\begin{aligned}&\left(\det\left(R'\right)\right)^{-1} \cdot \left[R \cdot \vec{\zeta}_F - \det\left(\Gamma\right) \cdot \vec{r}\right]\\ &= \left(\det\left(R'\right)\right)^{-1} \cdot \left[R \cdot \mathsf{expd}_F\left(\mathsf{Adj}\left(\Gamma\right) \cdot \vec{\beta}\right) - \det\left(\Gamma\right) \cdot \vec{r}\right].\end{aligned}$$

We now analyze each of these components separately. We first make the following observation:

$$\begin{aligned}&\forall F \subseteq [d], \forall B, C \in \mathbb{Z}_N^{d\times d} \ :\\ &\quad \mathsf{pjct}_F\left(\mathcal{N}_F B \mathcal{N}_F \cdot \mathcal{N}_F C \mathcal{N}_F\right) = \mathsf{pjct}_F\left(B\right) \cdot \mathsf{pjct}_F\left(C\right)\end{aligned} \tag{6}$$

and

$$\begin{aligned}&\forall F \subseteq [d], \forall B \in \mathbb{Z}_N^{d\times d}, \forall \vec{v} \in \mathbb{Z}_N^{|F|} \ :\\ &\quad \mathsf{pjct}_F\left(\mathcal{N}_F B \mathcal{N}_F \cdot \mathsf{expd}_F\left(\vec{v}\right)\right) = \mathsf{pjct}_F\left(B\right) \cdot \vec{v}\end{aligned} \tag{7}$$

Using Equation 6, we have:

$$\begin{aligned}\Gamma &= \mathsf{pjct}_F\left(\Gamma_F\right) = \mathsf{pjct}_F\left(A_F \cdot R\right)\\ &=^{(1)} \mathsf{pjct}_F\left(\mathcal{N}_F A \mathcal{N}_F \cdot \mathcal{N}_F R \mathcal{N}_F\right)\\ &=^{(2)} \mathsf{pjct}_F\left(A\right) \cdot \mathsf{pjct}_F\left(R\right) = A'_F \cdot R'\end{aligned}$$

where the equality denoted by (1) follows from the definition of A and from the fact that $R = \mathsf{expd}_F\left(R'\right)$ (so $R = \mathcal{N}_F R \mathcal{N}_F$), the equality denoted by (2) used Equation 6, and the rightmost equality follows from the definitions of A'_F and R' (see Figure 3).

Therefore,

$$\begin{aligned}\mathsf{Adj}\left(\Gamma\right) &= \det\left(\Gamma\right) \cdot \Gamma^{-1}\\ &= \det\left(\Gamma\right) \cdot \left(A'_F \cdot R'\right)^{-1}\\ &= \det\left(\Gamma\right) \cdot \left(R'\right)^{-1} \cdot \left(A'_F\right)^{-1}\end{aligned}$$

Next, we observe that:

$$\begin{aligned}&\forall F \subseteq [d], \forall \vec{v}, \vec{u} \in \mathbb{Z}_N^d \ :\\ &\quad \mathsf{pjct}_F\left(\vec{v} + \vec{u}\right) = \mathsf{pjct}_F\left(\vec{v}\right) + \mathsf{pjct}_F\left(\vec{u}\right)\end{aligned} \tag{8}$$

which implies that

$$\begin{aligned}\vec{\beta} &=^{(1)} \mathsf{pjct}_F\left(\vec{\beta}_F\right)\\ &=^{(2)} \mathsf{pjct}_F\left(\vec{b}_F + A_F \cdot \vec{r}\right)\\ &=^{(3)} \mathsf{pjct}_F\left(\vec{b}_F\right) + \mathsf{pjct}_F\left(A_F \cdot \vec{r}\right)\\ &=^{(4)} \mathsf{pjct}_F\left(\vec{b}_F\right) + \mathsf{pjct}_F\left(A\right) \cdot \vec{r}'\\ &=^{(5)} \mathsf{pjct}_F\left(\vec{b}_F\right) + A'_F \cdot \vec{r}'\end{aligned}$$

where (1) is by the definition of $\vec{\beta}$, (2) is by the definition of $\vec{\beta}_F$, (3) follows from Equation 8, (4) follows from Equation 7 because $A_F = \mathcal{N}_F A \mathcal{N}_F$ and

$\vec{r} = \mathsf{expd}_F(\vec{r}')$, and (5) follows from the definition of A'_F.

Putting the two together, we have

$$\begin{aligned}\vec{\zeta} &= \mathsf{Adj}(\Gamma)\cdot\vec{\beta}\\ &= \det(\Gamma)\cdot(R')^{-1}\cdot(A'_F)^{-1}\cdot\left[\mathsf{pjct}_F\left(\vec{b}_F\right)+A'_F\cdot\vec{r}'\right]\\ &= \det(\Gamma)\cdot(R')^{-1}\cdot\left[\vec{w}'_F+\vec{r}'\right].\end{aligned} \tag{9}$$

where the right-most equality follows from the definition of $\vec{w}'_F$ (see Figure 3).

Next, we notice that

$$\begin{aligned}&\forall F\subseteq[d], \forall\vec{v},\vec{u}\in\mathbb{Z}_N^{|F|} \ :\\ &\quad \mathsf{expd}_F(\vec{v}+\vec{u}) = \mathsf{expd}_F(\vec{v})+\mathsf{expd}_F(\vec{u})\end{aligned} \tag{10}$$

Therefore,

$$\begin{aligned}\vec{\zeta}_F &=^{(1)} \mathsf{expd}_F\left(\vec{\zeta}\right)\\ &=^{(2)} \mathsf{expd}_F\left(\det(\Gamma)\cdot(R')^{-1}\cdot[\vec{w}'_F+\vec{r}']\right)\\ &= \det(\Gamma)\cdot\mathsf{expd}_F\left((R')^{-1}\cdot[\vec{w}'_F+\vec{r}']\right)\\ &=^{(3)} \det(\Gamma)\cdot\mathsf{expd}_F\left((R')^{-1}\cdot\vec{w}'_F\right)\\ &\quad+\det(\Gamma)\cdot\mathsf{expd}_F\left((R')^{-1}\cdot\vec{r}'\right).\end{aligned}$$

where (1) follows from the definition of ζ, (2) follows from Equation 9, and (3) follows from Equation 10.

Next, notice that

$$\begin{aligned}&\forall F\subseteq[d], \forall B\in\mathbb{Z}_N^{|F|\times|F|}, \forall\vec{v}\in\mathbb{Z}_N^{|F|} \ :\\ &\quad \mathsf{expd}_F(B\cdot\vec{v}) = \mathsf{expd}_F(B)\cdot\mathsf{expd}_F(\vec{v})\end{aligned} \tag{11}$$

which implies that

$$\begin{aligned}\vec{\zeta}_F &=^{(1)} \det(\Gamma)\cdot\mathsf{expd}_F\left((R')^{-1}\cdot\vec{w}'_F\right)\\ &\quad+\det(\Gamma)\cdot\mathsf{expd}_F\left((R')^{-1}\cdot\vec{r}'\right)\\ &=^{(2)} \det(\Gamma)\cdot\mathsf{expd}_F\left((R')^{-1}\right)\cdot\mathsf{expd}_F(\vec{w}'_F)\\ &\quad+\det(\Gamma)\cdot\mathsf{expd}_F\left((R')^{-1}\right)\cdot\mathsf{expd}_F(\vec{r}')\\ &=^{(3)} \det(\Gamma)\cdot\mathsf{expd}_F\left((R')^{-1}\right)\cdot[\vec{w}_F+\vec{r}]\end{aligned}$$

where (1) was established above, (2) follows from Equation 11, and (3) follows from the definition of $\vec{w}'_F$ and $\vec{r}$.

Moreover, notice that

$$\begin{aligned}&\forall F\subseteq[d], \forall B,C\in\mathbb{Z}_N^{|F|\times|F|} \ :\\ &\quad \mathsf{expd}_F(B\cdot C) = \mathsf{expd}_F(B)\cdot\mathsf{expd}_F(C)\end{aligned} \tag{12}$$

in particular, this implies that

$$\begin{aligned}R\cdot\mathsf{expd}_F\left((R')^{-1}\right) &=^{(1)} \mathsf{expd}_F(R')\cdot\mathsf{expd}_F\left((R')^{-1}\right)\\ &=^{(2)} \mathsf{expd}_F\left(R'\cdot(R')^{-1}\right)\\ &= \mathsf{expd}_F\left(I_{|F|}\right)\end{aligned} \tag{13}$$

where (1) follows from the definition of R, (2) follows from Equation 12, and $I_{|F|}$ denotes the $|F|\times|F|$ identity matrix. Consequently, we have that

$$\begin{aligned}R\cdot\vec{\zeta}_F &=^{(1)} R\cdot\det(\Gamma)\cdot\mathsf{expd}_F\left((R')^{-1}\right)\cdot[\vec{w}_F+\vec{r}]\\ &=^{(2)} \det(\Gamma)\cdot\mathsf{expd}_F\left(I_{|F|}\times|F|\right)\cdot[\vec{w}_F+\vec{r}]\\ &=^{(3)} \det(\Gamma)\cdot[\vec{w}_F+\vec{r}]\end{aligned}$$

where (1) follows from substituting $\vec{\zeta}_F$ with the value computed above, (2) follows from Equation 13, and (3) follows from the fact that $\vec{w}_F, \vec{r}$ have 0 in every index $i\notin F$.

Finally, notice that

$$\det(\Gamma) = \det(A'_F\cdot R') = \det(R')\cdot\det(A'_F) \tag{14}$$

where we used the fact, noted above, that $\Gamma = A'_F\cdot R'$.

Putting it together, we have that

$$\begin{aligned}&(\det(R'))^{-1}\cdot\left[R\cdot\vec{\zeta}_F-\det(\Gamma)\cdot\vec{R}\right]\\ &=^{(1)} (\det(R'))^{-1}\cdot[\det(\Gamma)\cdot(\vec{w}_F+\vec{r})-\det(\Gamma)\cdot\vec{r}]\\ &= (\det(R'))^{-1}\cdot\det(\Gamma)\cdot\vec{w}_F\\ &=^{(2)} (\det(R'))^{-1}\cdot\det(R')\cdot\det(A'_F)\cdot\vec{w}_F\\ &= (A'_F)\cdot\vec{w}_F\end{aligned}$$

where (1) follows by substituting $R\cdot\vec{\zeta}_F$ with the value computed above, and (2) follows from Equation 14. This concludes the correctness proof for the scaled ridge phase.

Correctness of the selection phase (Item 4): first, the fact that selection in SIR is computed from the set $[d]$ instead of the current set F of features does not affect correctness, since $\mathcal{S}_1$ only computes the ordinals of indices $i\in F$ (see Figure 7). This effectively means that the ordinals are computed over the restriction of the model $\vec{z}_F$ to the set F of features that survived to the current iteration, exactly as in $\mathsf{Iterated\ ridge}$. Moreover, $\mathcal{S}_2$ identifies the correct features to be removed (i.e., the ones with smallest weights) regardless of the fact that the weights are permuted, because applying a permutation between the weights does not affect the ordering between them. Once the smallest features are identified, $\mathcal{S}_1$ inverts the permutation, so the resultant set S_{del} of features to be removed is identical to the set computed in Figure 1, conditioned on the ordinals being correctly computed. It therefore remains to show that the ordinals are correctly computed.

We now show that Ord_i is the number of z_j's which are smaller than z_i. Recall that the "size" of z_i is measured as its distance from the nearest multiple of N, and so it measures the *absolute value* of z_i, when the elements of $\mathbb{Z}_N$ are represented using the integers $\left\{-\frac{N-1}{2},\ldots,-1,0,1,\ldots,\frac{N-1}{2}\right\}$. Since the elements of $\mathbb{Z}_N$ are represented using the elements $0,1,\ldots,N-1$, this corresponds to interpreting values $0 < x \leq (N-1)/2$ as "positive", and values

$(N-1)/2 < x < N$ as "negative". We will show that $\mathsf{Ord}_i = |\{z_j : |z_j| < |z_i|\}|$. Notice that $|z_j| < |z_i|$ if and only if $z_j^2 < z_i^2$, because N was chosen such that $z_i, z_j < \sqrt{N}$ (i.e., $z_i^2, z_j^2 < N$), meaning no wrap around occurs when squaring. Therefore, $|z_j| < |z_i|$ if and only if $z_i^2 - z_j^2 > 0$. Thus, computing the ordinals can be done using the differences of squares, namely $\mathsf{Ord}_i = \left|\{j : z_i^2 - z_j^2 > 0\}\right|$. It remains to show that the computation using the *masked* differences of squares is correct. Fix a pair $i, j \in F, i \neq j$, fix an $r_{i,j} \in \mathbb{Z}_N$, and denote $\alpha_{i,j} := z_i^2 - z_j^2$ and $\alpha'_{i,j} := \alpha_{i,j} + r_{i,j}$. We consider four cases, based on whether $\alpha'_{i,j} < \frac{N-1}{2} + 1$ (i.e., it is "nonnegative" according to the aforementioned interpretation of positive and negative values in $\mathbb{Z}_N$) or not, and whether a wrap-around occurred when computing $\alpha'_{i,j}$ from $\alpha_{i,j}$ and $r_{i,j}$. In all cases, we consider two subcases: (1) $\alpha_{i,j}$ is non-negative, i.e., it is positive (recall that we assume all weights z_i are unique, so $\alpha_{i,j} \neq 0$), in which case we show that $\mathsf{ost}_{i,j} = 1$; and (2) $\alpha_{i,j}$ is negative, in which case we show that $\mathsf{ost}_{i,j} = 0$. In the following, all computations are over $\mathbb{Z}$.

Case I: $\alpha'_{i,j} < \frac{N-1}{2} + 1$ and no wrap-around occurred. In this case, $\mathsf{Negative}_{i,j} = 0$ and $\mathsf{rangeMin}'_{i,j} = \alpha'_{i,j}$, $\mathsf{rangeMax}'_{i,j} = \alpha'_{i,j} + \frac{N-1}{2} + 1$. Additionally, $\alpha'_{i,j} = r_{i,j} + \alpha_{i,j}$ because no wrap-around occurred. First, if $\alpha_{i,j}$ is positive then $0 < \alpha_{i,j} < \frac{N+1}{2} + 1$, which implies that

$$r_{i,j} <^{(2)} r_{i,j} + \alpha_{i,j} = \alpha'_{i,j} <^{(1)} r_{i,j} + \frac{N-1}{2} + 1.$$

Since $\alpha'_{i,j} < \frac{N-1}{2} + 1$ by the assumption of case I, inequality (1) always holds (because $r_{i,j} \geq 0$). Inequality (2) implies that $r_{i,j} < \alpha'_{i,j} = \mathsf{rangeMin}_{i,j}$, so $\mathsf{IsBigger}_{i,j} = 0$, and $\mathsf{ost}_{i,j} = 1$ (because the second summand in its definition in Figure 7 is 1) as required. If, on the other hand, $\alpha_{i,j}$ is negative then $(N-1)/2 < \alpha_{i,j}$ so

$$\frac{N-1}{2} \leq r_{i,j} + \frac{N-1}{2} < r_{i,j} + \alpha_{i,j} = \alpha'_{i,j}$$

(the left-most inequality holds because $r_{i,j} \geq 0$) which contradicts the case assumption that $\alpha_{i,j} \leq (N-1)/2$. Therefore, if $\alpha_{i,j}$ is negative and no wrap-around occurred, then it cannot be the case that $\alpha'_{i,j} < \frac{N}{2} + 1$.

Case II: $\alpha'_{i,j} < \frac{N-1}{2} + 1$ and a wrap-around occurred. In this case, $\mathsf{Negative}_{i,j} = 0$ and $\mathsf{rangeMin}'_{i,j} = \alpha'_{i,j}$, $\mathsf{rangeMax}'_{i,j} = \alpha'_{i,j} + \frac{N-1}{2} + 1$ as in case I, but $\alpha'_{i,j} = r_{i,j} + \alpha_{i,j} - N$ because a wrap-around occurred. (This is indeed the only other possibility since $r_{i,j}, \alpha_{i,j} < N$.) First, if $\alpha_{i,j}$ is positive, i.e., $0 < \alpha_{i,j} < \frac{N+1}{2} + 1$, then

$$\begin{aligned} r_{i,j} - N &<^{(1)} r_{i,j} + \alpha_{i,j} - N = \alpha'_{i,j} \\ &<^{(2)} r_{i,j} + \frac{N-1}{2} + 1 - N \\ &= r_{i,j} - \frac{N-1}{2}. \end{aligned}$$

Since $\alpha'_{i,j}$ is positive, then inequality (1) always holds over $\mathbb{Z}$. Inequality (2) implies that $\alpha'_{i,j} + \frac{N-1}{2} < r_{i,j}$, meaning $\mathsf{rangeMax}_{i,j} = \alpha'_{i,j} + \frac{N-1}{2} + 1 \leq r_{i,j}$, so $\mathsf{IsSmaller}_{i,j} = 0$, and consequently $\mathsf{ost}_{i,j} = 1$. If, on the other hand, $\alpha_{i,j}$ is negative then $(N-1)/2 < \alpha_{i,j} < N$ so

$$\begin{aligned} r_{i,j} - \frac{N-1}{2} &\leq r_{i,j} + \frac{N-1}{2} - N \\ &<^{(1)} r_{i,j} + \alpha_{i,j} - N = \alpha'_{i,j} \\ &<^{(2)} r_{i,j} + N - N = r_{i,j}. \end{aligned}$$

Then inequality (1) implies that $r_{i,j} < \alpha'_{i,j} + \frac{N-1}{2} < \mathsf{rangeMax}_{i,j}$ so $\mathsf{IsSmaller}_{i,j} = 1$, and inequality (2) implies that $\mathsf{rangeMin}_{i,j} = \alpha'_{i,j} < r_{i,j}$ so $\mathsf{IsBigger}_{i,j} = 1$. Together with the fact that $\mathsf{Negative}_{i,j} = 0$, this implies that $\mathsf{ost}_{i,j} = 0$.

Case III: $\alpha'_{i,j} \geq \frac{N-1}{2} + 1$ and no wrap-around occurred. In this case, $\mathsf{Negative}_{i,j} = 1$ and $\mathsf{rangeMin}'_{i,j} = \alpha'_{i,j} - \frac{N-1}{2}$, $\mathsf{rangeMax}'_{i,j} = \alpha'_{i,j} + 1$. Additionally, $\alpha'_{i,j} = r_{i,j} + \alpha_{i,j}$ because no wrap-around occurred. First, if $\alpha_{i,j}$ is positive then $0 < \alpha_{i,j} < \frac{N+1}{2} + 1$, which implies as in case I that

$$r_{i,j} <^{(1)} \alpha'_{i,j} <^{(1)} r_{i,j} + \frac{N-1}{2} + 1.$$

Inequality (1) implies that $r_{i,j} < \alpha'_{i,j} < \mathsf{rangeMax}_{i,j}$ and so $\mathsf{IsSmaller} = 1$. Inequality (2) guarantees that $\mathsf{rangeMin}_{i,j} - 1 = \alpha'_{i,j} - \frac{N-1}{2} - 1 < r_{i,j}$ so $\mathsf{rangeMin}_{i,j} \leq r_{i,j}$, meaning $\mathsf{IsBigger}_{i,j} = 1$. Combined with the fact that $\mathsf{Negative}_{i,j} = 1$, we get that $\mathsf{ost}_{i,j} = 1$ (because of the first summand in its definition in Figure 7). If, on the other hand, $\alpha_{i,j}$ is negative then $(N-1)/2 < \alpha_{i,j}$ so

$$r_{i,j} + \frac{N-1}{2} < r_{i,j} + \alpha_{i,j} = \alpha'_{i,j}$$

namely $\mathsf{rangeMin}_{i,j} = \alpha'_{i,j} - \frac{N-1}{2} > r_{i,j}$, so $\mathsf{IsBigger}_{i,j} = 0$. Combined with the fact that $\mathsf{Negative}_{i,j} = 1$ this implies that $\mathsf{ost}_{i,j} = 0$.

Case IV: $\alpha'_{i,j} \geq \frac{N-1}{2} + 1$ and a wrap-around occurred. In this case, $\mathsf{Negative}_{i,j} = 1$ and $\mathsf{rangeMin}'_{i,j} = \alpha'_{i,j} - \frac{N-1}{2}$, $\mathsf{rangeMax}'_{i,j} = \alpha'_{i,j} + 1$ as in case III, but $\alpha'_{i,j} = r_{i,j} + \alpha_{i,j} - N$ because a wrap-around occurred. We show first that in this case, $\alpha_{i,j}$ cannot be positive. Indeed, since $r_{i,j} < N$, if $\alpha_{i,j}$ were positive, i.e., $\alpha_{i,j} \leq \frac{N-1}{2}$, then it would hold that

$$\alpha'_{i,j} = r_{i,j} + \alpha_{i,j} - N < N + \frac{N-1}{2} - N = \frac{N-1}{2}$$

which contradicts the case assumption that $\alpha'_{i,j} \geq \frac{N-1}{2} + 1$. It remains therefore to show that in case IV necessarily $\mathsf{IsBigger}_{i,j} = 0$ or $\mathsf{IsSmaller}_{i,j} = 0$ (this will imply that $\mathsf{ost}_{i,j} = 0$ because $\mathsf{Negative}_{i,j} = 1$).

We show that $\mathsf{IsSmaller}_{i,j} = 0$. Indeed, since $\alpha_{i,j} < N$ then

$$\mathsf{rangeMax}_{i,j} - 1 = \alpha'_{i,j} = r_{i,j} + \alpha_{i,j} - N < r_{i,j} + N - N = r_{i,j}$$

namely $\mathsf{rangeMax}_{i,j} \leq r_{i,j}$ and so $\mathsf{IsSmaller}_{i,j} = 0$.

Correctness of the compaction phase (Item 5): this phase only affects the computations using the compacted $A, \vec{b}$, namely the scaled ridge phase and the output phase. We show when discussing these phases, that resetting the entries corresponding to features that were removed has the same effect as projecting $A, \vec{b}$ to the set of surviving features, so Item 5 does not affect correctness.

Correctness of the final output (item 6): we claim that the final output of SIR is correct (i.e., identical to that of $\mathsf{Iterated\ ridge}$). Indeed, as described in the analysis of Item 3 above, the scalde ridge phase outputs a scaled model $\vec{z}_F = \det(A'_F) \cdot \vec{w}_F$ (where $\vec{w}_F$ is a permuted version of the actual output model). $\mathcal{S}_1$ additionally obtains an encryption of

$$\begin{aligned}\Delta^{-1} &= (\det(\Gamma))^{-1} \\ &=^{(1)} (\det(A'_F) \cdot \det(R'))^{-1} \\ &= \det(A'_F)^{-1} \cdot \det(R')^{-1}\end{aligned}$$

(where (1) follows from Equation 14) from which $\mathcal{S}_1$ computes $\det(A'_F)^{-1}$ (this is possible because $\mathcal{S}_1$ knows R'). $\mathcal{S}_1$ then recovers $\vec{w}_F$ by computing $\det(A_F)^{-1} \cdot \vec{z}_F$. We will show that $\vec{w}_F$ contains the correct weights (as in the output model of Figure 1), but they are permuted according to P. This will imply the correctness of the output model $\vec{w}^{\mathsf{int}}$ because it is obtained by multiplying $\vec{w}_F$ with $P^T = P^{-1}$, which inverts the permutation (and the correctness of the final output model $\vec{w}$ then follows from the correctness of the rational reconstruction [44], [45]).

To see why $\vec{w}_F$ contains the correct weights, but permuted according to P, recall that

$$\begin{aligned}\vec{w}_F = \mathsf{expd}_F(\vec{w}'_F) &= \mathsf{expd}_F\left((A'_F)^{-1} \cdot \mathsf{pjct}_F\left(\vec{b}_F\right)\right) \\ &=^{(1)} \mathsf{expd}_F\left((A'_F)^{-1}\right) \cdot \mathsf{expd}_F\left(\mathsf{pjct}_F\left(\vec{b}_F\right)\right) \\ &= \mathsf{expd}_F\left((\mathsf{pjct}_F(A))^{-1}\right) \cdot \vec{b}_F\end{aligned}$$

where (1) follows from Equation 11. Moreover, by the definition of A and $\vec{b}$ we have:

$$\begin{aligned}A = P^T\left(X^T X + \lambda I\right) P &= P^T X^T X P + P^T \cdot \lambda I \cdot P \\ &=^{(1)} (XP)^T \cdot XP + \lambda I\end{aligned}$$

(where (1) uses the fact that $P^T = P^{-1}$), and

$$\vec{b} = P^T X^T \vec{y} = (XP)^T \cdot \vec{y}.$$

Therefore, performing the learning over $A, \vec{b}$ (which are permuted versions of $X^T X + \lambda I$ and $X^T \vec{y}$, respectively) is equivalent to performing the learning over *unpermuted* $A', \vec{b}'$ that were obtained from a *permuted* input $X' = XP$, namely in which the columns (i.e., features) of X were permuted. Therefore, the correctness of each ridge regression step guarntees that the output model $\vec{w}_F$ – which contains the weights of these features – is permuted according to the same permutation P, as we set out to prove. □

The next lemma states that SIR is secure against an adversary corrupting $\mathcal{S}_1$ and an arbitrary subset of data owners. Intuitively, throughout the protocol execution the only unencrypted information which the adversary sees are the subsets S_{del} of features to be removed, which it obtains from $\mathcal{S}_2$ in Step 3b of Figure 2. Thus, security follows from the fact (which we prove below) that these sets are perfectly simulatable because of the random permutation $P = P_1 \cdot P_2$ used to permute the data.

Lemma 8.5 (Privacy when $\mathcal{S}_1$ is corrupted)**.** *Let $I \subseteq [m]$, let $\lambda, d \in \mathbb{N}$, and let $N \in \mathbb{N}$ which satisfies Equation 4. Then SIR is private against an adversary corrupting I and $\mathcal{S}_1$.*

Proof. To prove the lemma, we describe a simulator Sim, whose input consists of the public parameters of the protocol (i.e., $\kappa, \lambda, N, D, n_1, \ldots, n_m, d, s, \ell, \mathsf{rej}, \mathsf{thr}$), the inputs $\left\{\left(X^j, \vec{y}^j\right)\right\}_{j \in I}$ of the corrupted data owners, and the output model $\vec{w} \in \mathbb{Q}^d$. We note that the number of iterations performed by SIR, and the number of features that should be removed in each iteration (i.e., the size of the set S_{del}), depend (deterministically) only on the public parameters d, s, rej and thr. Therefore, these values can be computed by Sim. Moreover, we note that if $I = [m]$ then Sim can trivially emulate the entire execution of the protocol, because it is given the inputs $X^j, \vec{y}^j$ of all data owners. Therefore, we assume that $I \subset [m]$.

Sim operates as follows.

- Initializes a set $F_0 := [d]$, and $f_0 = d$. (Intuitively, F_l and f_l will denote the set and number of features, respectively, which survived the l'th iteration.)
- During setup (Step 1 in Figure 2), Sim honestly generates encryption keys $(\mathsf{pk}_N, \mathsf{sk}_N), (\mathsf{pk}_D, \mathsf{sk}_D)$, and random encryptions $\mathbf{P}_2 \leftarrow \mathsf{Enc}(\mathsf{pk}_N, 0_d)$ and $\mathbf{P}_2^T \leftarrow \mathsf{Enc}(\mathsf{pk}_N, 0_d)$, where 0_d denotes the $d \times d$ all-zeros matrix. Additionally, for every honest DO_j, it generates a random encryption $\mathbf{A}^j \leftarrow \mathsf{Enc}(\mathsf{pk}_N, 0_d)$, and $\vec{\mathbf{b}}^j \leftarrow \mathsf{Enc}\left(\mathsf{pk}_N, 0^d\right)$ where 0^d denotes the length-d all-zeros vector.
- In the l'th iteration (Step 3 in Figure 2), Sim simulates the encryptions which $\mathcal{S}_1$ is given, as follows:
  - Simulates the messages received by $\mathcal{S}_1$ during the execution

of the ridge regression phase (Figure 3) by generating a random encryption $\boldsymbol{\zeta}^l \leftarrow \mathsf{Enc}\left(\mathsf{pk}_N, 0^d\right)$, and $\boldsymbol{\Delta}^l \leftarrow \mathsf{Enc}\left(\mathsf{pk}_N, 0\right)$.
  - Simulates the ciphertexts received by $\mathcal{S}_1$ during the execution of the selection phase (Figure 7) by setting $\mathsf{IsBigger}_{i,j}, \mathsf{IsSmaller}_{i,j}, \mathsf{Negative}_{i,j}$ to be random encryptions of 0 under pk_D, for every $i, j \in F_l, i \neq j$.
  - Simulates the set S^l_{del} of features to be removed in the l'th iteration (computed in Step 3b of Figure 2) as follows. Let F_{l-1} denote the set of features that survived to the l'th iteration, and let v_l denote the number of features that should be removed in the l'th iteration (as noted above, Sim can compute v_l from the public parameters). Then Sim picks a random subset $S^l_{\mathsf{del}} \subseteq F_{l-1}$ of size $\left|S^l_{\mathsf{del}}\right| = v_l$. Then, it sets $F_l = F_{l-1} \setminus S^l_{\mathsf{del}}$, and $f_l = f_{l-1} - v_l$. It then samples a random permutation π on $[d]$ and permutes the indicator vector of S^l_{del} according to π, to obtain a vector $\vec{\chi}^{\pi}$.

- During the output phase (Step 4 in Figure 2), Sim sets $\vec{\boldsymbol{\zeta}}$ to be d random encryptions of 0 under pk_N, where s is the sparsity parameter. It also generates a random encryption $\boldsymbol{\Delta}^{-1} \leftarrow \mathsf{Enc}\left(\mathsf{pk}_N, 0\right)$.

Sim outputs the view V_S consisting of $\mathsf{pk}_N, \mathsf{pk}_D$, the public parameters of the protocol, the inputs of the corrupted data owners, all ciphertexts sent to $\mathcal{S}_1$ throughout the simulation, the permuted indicator vectors $\vec{\chi}^{\pi,l}$ of all sets S^l_{del} generated throughout the simulation, and the output model $\vec{w}$.

We now prove that the simulated and real views are computationally indistinguishable through a sequence of hybrids. We note that since the encryptions keys are identically distributed in the real and simulated views, it suffices to prove indistinguishability conditioned on them. The same holds for the random coins of $\mathcal{S}_1$.

$\mathcal{H}_0$: This is the real view V_R.

$\mathcal{H}_1$: In $\mathcal{H}_1$, we replace all encryptions in V_R with encryptions of 0 (under the appropriate keys). We show that $\mathcal{H}_0 \approx \mathcal{H}_1$ follows from the security of the encryption scheme.

Indeed, let t denote the number of ciphertexts in V_R, then we define a sequence of hybrids $\mathcal{H}_0 = \mathcal{H}^0_0, \mathcal{H}^1_0, \ldots, \mathcal{H}^t_0 = \mathcal{H}_1$, where in the h'th hybrid we replace the first h encrypted values from their value in V_R to 0. Then if $\mathcal{H}_0, \mathcal{H}_1$ are not computationally close, then there exists a pair of adjacent hybrids $\mathcal{H}^h_0, \mathcal{H}^{h+1}_0$ which are not computationally close. That is, there exists a distinguisher $\mathcal{D}$ that distinguishes between them with non-negligible probability. We use $\mathcal{D}$ to break the security of the FHE scheme against non-uniform distinguishers.

We define a (non-uniform) distinguisher $\mathcal{D}'$ that has the inputs hard-wired into it. We will use the fact that the intermediate matrices $A, \vec{b}$, and the intermediate scaled models $\vec{z}$, are determined deterministically by these inputs. $\mathcal{D}'$ operates as follows. Let v denote the value of the $h+1$'th ciphertext. Notice that v is either determined by the inputs, or computable from the inputs by a random masking and permutation that can be sampled by $\mathcal{D}'$. $\mathcal{D}'$ sends $v, 0$ to its distinguisher as the values on which it wants to be tested, and obtains from its challenger an encryption c of either v or 0, as well as pk_N or pk_D (depending on v, and whether it is encrypted using pk_N or pk_D in V_R). It honestly generates the secret and public encryption keys for the other pair (i.e., if it got pk_D then it generates $(\mathsf{pk}_N, \mathsf{sk}_N)$, and vice versa). Then, it uses c to generate the entire hybrid distribution: it computes $A^j, \vec{b^j}$ as they are computed by the data owners, as well as $A, \vec{b}$ and their intermediate compacted versions. It also computes the intermediate models $\vec{z^l}$, their masked version $\vec{\zeta^l}$ and the masked determinants Δ^l, the values $\left\{\mathsf{IsBigger}^l_{i,j}, \mathsf{IsSmaller}^l_{i,j}, \mathsf{Negative}^l_{i,j}\right\}_{i,j\in F, i\neq j}$ generated during the selection phase in the l'th iteration, the inverse Δ^{-1} of the determinant in the final iteration, and all the subsets S^l_{del}. It honestly encrypts the values that appear encrypted in V_R using pk_N and pk_D, and then replaces the first h ciphertexts with encryptions of 0 (under the appropriate key pk_N or pk_D). For the $h+1$'th ciphertext, it uses c. Then, it runs $\mathcal{D}$ and forwards its guess to the challenger of $\mathcal{D}'$. Notice that $\mathcal{D}'$ perfectly emulates $\mathcal{H}^h_0$ (if c encrypts v) or $\mathcal{H}^{h+1}_0$ (if c encrypts 0) for $\mathcal{D}$, so $\mathcal{D}'$ obtains the same non-negligible distinguishing advantage as $\mathcal{D}$.

Notice that $\mathcal{H}_1$ contains no information about the permutation P (because it contains no information about P_2).

$\mathcal{H}_2$: In $\mathcal{H}_2$, we generate the sets S^l_{del} as follows. First, we compute *unpermuted* versions $A', \vec{b}'$ of $\sum_{j=1}^m \left(X^j\right)^T \cdot X^j + \lambda I, \sum_{j=1}^m \left(X^j\right)^T \cdot \vec{y^j}$, respectively. Then, we iteratively compute the intermediate scaled models $\vec{z^l}$ from the unpermuted

$A', \vec{b}'$ in Step 3a, and compact the unpermuted versions in Step 3c. For each iteration l, once $\vec{z}^l$ has been computed, we compute the set S^l_{del} directly from it in Step 3b. Finally, we apply the permutation P to the S^l_{del}'s, and compute the $\vec{\chi}^{\pi,l}$ from them (using a freshly sampled permutation π as described in the simulation). In summary, a sample from $\mathcal{H}_2$ is generated by sampling according to $\mathcal{H}_1$, then recomputing the S^l_{del}'s to be consistent with P.

We claim that $\mathcal{H}_1 \equiv \mathcal{H}_2$. This holds because the permutation P does not affect the learned models $\vec{z}^l$ except for permuting their coordinates. Indeed, assume that a ridge regression step is applied to some permuted inputs $H, \vec{g}$, and the output model is $\vec{v}$. Let $H', \vec{g}'$ denote the unpermuted inputs, and $\vec{v}'$ denote the output model of the ridge regression step on $H', \vec{g}'$. Then $\vec{v}$ is exactly the vector obtained by applying P to $\vec{v}'$ (here we also use the fact that the ridge regression step is deterministic, and consists of solving a linear equation). Since the S^l_{del}'s are deterministically determined by the $\vec{z}^l$'s, this implies that $\mathcal{H}_1 \equiv \mathcal{H}_2$.

$\mathcal{H}_3$: $\mathcal{H}_3$ is identical to $\mathcal{H}_2$, except for the way in which the S^l_{del}'s are permuted. In $\mathcal{H}_3$, we set $P_0 = P$. Additionally, for every iteration l, we choose a permutation P_l which is uniformly random and independent subject to the constraint that P_l agrees with P_{l-1} on $\cup_{t=1}^{l-1} S^t_{\text{del}}$. Each S^l_{del} is permuted according to P_l before it is revealed to $\mathcal{S}_1$ through the permuted indicator vector $\vec{\chi}^{\pi,l}$. (Intuitively, in each iteration we re-sample the images of the coordinates that were not yet revealed to $\mathcal{S}_1$.)

We claim that $\mathcal{H}_3 \equiv \mathcal{H}_2$. Indeed, notice that in $\mathcal{H}_2$ the permutation P could have been sampled "lazily", where in each iteration l we only sample the images of the coordinates in S^l_{del}. (This is because the ciphertexts in $\mathcal{H}_2$ no longer carry any information on P, so the only information revealed on P is from the sets S^l_{del}. However, the images of the other coordinates are not used in the l'th iteration, therefore their sampling can be deferred to the next iterations). An alternative (and identical) method is to sample an entire permutation in the first iteration, and then consistently resample the images of the coordinates that were not previously used (where by "consistently resample" we mean that the images that were already revealed in previous iterations remain unchanged). This resampling is exactly how the permutations are sampled in $\mathcal{H}_3$.

$\mathcal{H}_4$: In $\mathcal{H}_4$, after computing the intermediate models $\vec{z}^l$'s, we define vectors $\vec{u}^l$ as follows: for every $1 \le t \le d$, if $z^l_t = 0$ then $u^l_t = 0$. For the remaining coordinates, let $z^l_{t_1}, \ldots, z^l_{t_{f_{l-1}}}$ denote the non-0 coordinates of $\vec{z}^l$, ordered from the smallest to the largest (i.e., $z^l_{t_1} < \ldots < z^l_{t_{f_{l-1}}}$). Here, we also use the assumption that all entries of $\vec{z}^l$ are unique; see Remark 4.4. Then for every $1 \le g \le f_{l-1}$ we set $u^l_{t_g} = g$. The S^l_{del}'s are then computed from the $\vec{u}^l$'s instead of the $\vec{z}^l$'s (and permuted as in $\mathcal{H}_3$).

Then $\mathcal{H}_3 \equiv \mathcal{H}_4$ because S^l_{del} depends only on *the ordering* between the entries of $\vec{z}^l$, and not on their actual values.

$\mathcal{H}_5$: This is simulated view which the simulator outputs.

We claim that $\mathcal{H}_4 \equiv \mathcal{H}_5$. The only difference between the hybrids is in how the S^l_{del}'s are chosen: in $\mathcal{H}_4$ they are computed according to the actual order of the coordinates in the intermediate model $\vec{z}^l$, whereas in $\mathcal{H}_5$ they are chosen as a random subset of the "surviving" coordinates. However, since each S^l_{del} in $\mathcal{H}_4$ is permuted using the permutation P_l, which is uniformly random and independent conditioned on being identical to P_{l-1} on $\cup_{t=1}^{l-1} S^t_{\text{del}}$, S^l_{del} is also uniformly random in $\mathcal{H}_4$.

□

The next lemma states that SIR is secure against an adversary corrupting $\mathcal{S}_2$ and an arbitrary subset of data owners. Its proof will use observations from the proof of Lemma 8.5 (regarding how to simulate the sets S^l_{del}), and an observation of [36], that the masking perfectly hides the data matrix and response vectors (see Lemma 8.7 below). We note that Giacomelli et al. [36] proved this for a single execution of ridge regression, and we show that it extends to the multiple iterations of SIR as well, due to the per-iteration masking.

Lemma 8.6 (Privacy when $\mathcal{S}_2$ is corrupted)**.** *Let $I \subseteq [m]$, let $\lambda, d \in \mathbb{N}$, and let $N \in \mathbb{N}$ which satisfies Equation 4. Then SIR is private against an adversary corrupting I and $\mathcal{S}_2$, for input matrices X which are invertible.*

The proof of Lemma 8.6 will use the following Lemma from [36, Lemma 1].

Lemma 8.7 (Lemma 1 from [36])**.** *Let $N, d \in \mathbb{N}$, and let $A \in GL(d, \mathbb{Z}_N)$ and $\vec{b} \in \mathbb{Z}_N^d$. Then* *the random variable defined by picking a random $R \leftarrow GL(d, \mathbb{Z}_N)$ and a random $\vec{r} \leftarrow \mathbb{Z}_N^d$ and outputting $\left(A \cdot R, \vec{b} + A \cdot \vec{r}\right)$ is uniformly distributed over $GL(d, \mathbb{Z}_N) \times \mathbb{Z}_N^d$.*

Roughly (and somewhat inaccurately), we will use Lemma 8.7 to claim that the masked matrices Γ_F, and masked vectors $\vec{\beta}_F$, which $\mathcal{S}_2$ obtains in each ridge regression step, are distributed uniformly over $GL(|F|, \mathbb{Z}_N) \times \mathbb{Z}_N^{|F|}$. This, however, does not follow directly from Lemma 8.7, because in SIR the masked $\Gamma_F, \vec{\beta}_F$ are extended versions of such a matrix and vector. In the following lemma, we use Lemma 8.7 to show that $\Gamma_F, \vec{\beta}_F$ in SIR are distributed as extensions of uniformly random values in $GL(|F|, \mathbb{Z}_N)$ and $\mathbb{Z}_N^{|F|}$, respectively.

Lemma 8.8. *Let $N, d \in \mathbb{N}$, let $F \subseteq [d]$ of size $f = |F|$, and let $A' \in GL(f, \mathbb{Z}_N)$ and $\vec{b} \in \mathbb{Z}_N^f$. Then the random variable defined by picking a random $R' \leftarrow GL(f, \mathbb{Z}_N)$ and a random $\vec{r}' \leftarrow \mathbb{Z}_N^f$ and outputting*

$$\left(\mathsf{expd}_F(A') \cdot \mathsf{expd}_F(R'), \mathsf{expd}_F\left(\vec{b}\right) + \mathsf{expd}_F(A') \cdot \mathsf{expd}_F(\vec{r}')\right)$$

is uniformly distributed over $\mathsf{expd}_F(GL(f, \mathbb{Z}_N)) \times \mathsf{expd}_F\left(\mathbb{Z}_N^f\right)$.

Proof. Let $M := \mathsf{expd}_F(A') \cdot \mathsf{expd}_F(R')$, and $\vec{v} = \mathsf{expd}_F\left(\vec{b}\right) + \mathsf{expd}_F(A') \cdot \mathsf{expd}_F(\vec{r}')$. We need to prove that $M = \mathsf{expd}_F(M'), \vec{v} = \mathsf{expd}_F(\vec{v}')$ for $M' \in \mathbb{Z}_N^{f \times f}, \vec{v}' \in \mathbb{Z}_N^f$ where $(M', \vec{v}')$ is uniformly distributed in $GL(f, \mathbb{Z}_N) \times \mathbb{Z}_N^f$. Notice first that by Equation 11, we have:

$$\vec{v} = \mathsf{expd}_F\left(\vec{b}\right) + \mathsf{expd}_F(A') \cdot \mathsf{expd}_F(\vec{r}')$$

$$= \mathsf{expd}_F\left(\vec{b}\right) + \mathsf{expd}_F(A' \cdot \vec{r}')$$

which by Equation 10 implies that $\vec{v} = \mathsf{expd}_F\left(\vec{b} + A' \cdot \vec{r}'\right)$. Second, notice that by Equation 12, it holds that $M = \mathsf{expd}_F(A') \cdot \mathsf{expd}_F(R') = \mathsf{expd}_F(A' \cdot R')$. Therefore, it remains to prove that $\left(A' \cdot R', \vec{b} + A' \cdot \vec{r}'\right)$ is uniformly distributed over $GL(f, \mathbb{Z}_N) \times \mathbb{Z}_N^f$. This follows directly from Lemma 8.7 by setting $d := f$. □

Proof of Lemma 8.6. To prove the lemma, we describe a simulator Sim. As in the proof of Lemma 8.5, the input of Sim consists of the public parameters of the protocol (i.e., $\kappa, \lambda, N, D, n_1, \ldots, n_m, d, s, \ell, \mathsf{rej}, \mathsf{thr}$), the inputs $\left\{\left(X^j, \vec{y}^j\right)\right\}_{j \in I}$ of the corrupted data owners, and the output model $\vec{w} \in \mathbb{Q}^d$. We assume without loss of generality that $I \subset [m]$. Also, similar to the proof of Lemma 8.5, Sim can compute the number of iterations performed by SIR, and the number of features that should be removed in each iteration (i.e., the size of the set S_{del}), since these depend (deterministically) only on the public parameters.

Sim operates as follows.

1) Initializes a set $F_0 := [d]$, and $f_0 = d$. (Intuitively, F_l and f_l will denote the set and number of features, respectively, which survived the l'th iteration.)
2) During setup (Step 1 in Figure 2), Sim uses the randomness of $\mathcal{S}_2$ to determine the encryption keys $(\mathsf{pk}_N, \mathsf{sk}_N), (\mathsf{pk}_D, \mathsf{sk}_D)$.
3) In the l'th iteration (Step 3 in Figure 2), Sim simulates the encryptions which $\mathcal{S}_2$ receives, as follows:
   a) Messages during ridge regression phase (Figure 3): generates a random invertible matrix $\Gamma^{l\prime} \leftarrow \mathrm{GL}(f_{l-1}, \mathbb{Z}_N)$ and a random vector $\vec{\beta}^{l\prime} \leftarrow \mathbb{Z}_N^{f_{l-1}}$, sets $\Gamma^l = \mathsf{expd}_{F_{l-1}}\left(\Gamma^{l\prime}\right), \vec{\beta}^l = \mathsf{expd}_{F_{l-1}}\left(\vec{\beta}^{l\prime}\right)$, and randomly encrypts them as $\mathbf{\Gamma}^l \leftarrow \mathsf{Enc}\left(\mathsf{pk}_N, \Gamma^l\right)$, and $\vec{\boldsymbol{\beta}}^l \leftarrow \mathsf{Enc}\left(\mathsf{pk}_N, \vec{\beta}^l\right)$.
   b) Messages during the feature selection phase (Figure 7): for every $i, j \in F_{l-1}, i \neq j$, generates a random $\mathsf{df}_{i,j}^l \leftarrow \mathbb{Z}_N$, and encrypts $\mathbf{df}_{i,j}^l \leftarrow \mathsf{Enc}\left(\mathsf{pk}_N, \mathsf{df}_{i,j}^l\right)$. Additionally, picks a random permutation Π_l on $\{0, 1, \ldots, f_{l-1} - 1\}$ (recall that f_{l-1} is the number of features that survived to the beginning of the l'th iteration), computes the vector $\vec{\mathsf{Ord}}^l$ obtained by applying Π_l on $(0, 1, \ldots, f_{l-1} - 1)^T$, and randomly encrypts $\vec{\mathbf{Ord}}^l \leftarrow \mathsf{Enc}\left(\mathsf{pk}_D, \vec{\mathsf{Ord}}^l\right)$.
   c) Simulates the set S_{del}^l of features that were removed in the l'th iteration (computed as part of the selection phase in Step 3b in Figure 2) as follows. Let v_l denote the number of features that should be removed in the l'th iteration (as noted above, Sim can compute v_l from the public parameters), and recall that F_{l-1} is the set of features that survived to the beginning of the l'th iteration. Then Sim picks a random subset $S_{\mathsf{del}}^l \subseteq F_{l-1}$ of size $\left|S_{\mathsf{del}}^l\right| = v_l$. Then, it sets $F_l = F_{l-1} \setminus S_{\mathsf{del}}^l$, and $f_l = f_{l-1} - v_l$.
4) During the output phase (Step 4 in Figure 2), Sim generates encryptions of masked matrix and vector $\Gamma^l, \vec{\beta}^l$ as described in Step 3a above. Additionally, it uses the model $\vec{w} \in \mathbb{Q}^d$ which it obtained as input, to determine the model $\vec{w}' \in \mathbb{Z}_N^d$ which corresponds to $\vec{w}$ (through the rational reconstruction),[17] and randomly encrypts $\vec{\mathbf{w}}' \leftarrow \mathsf{Enc}(\mathsf{pk}_N, \vec{w}')$.

Sim outputs the view V_S consisting of the randomness of $\mathcal{S}_2$ (which determines $\mathsf{pk}_N, \mathsf{pk}_D$), the public parameters of the protocol, the inputs of the corrupted

17. More specifically, Sim does the following for every coordinate $1 \le t \le d$: let $\vec{w}_t = p/q$, then $\vec{w}'_t := pq^{-1} \mod N$.

data owners, all ciphertexts sent to $\mathcal{S}_2$ throughout the simulation, all the sets S^l_{del} generated throughout the simulation, and the output model $\vec{w}$.

We now prove that the simulated and real views are identically distributed through a sequence of hybrids.

$\mathcal{H}_0$**:** This is the real view V_R.

$\mathcal{H}_1$**:** In $\mathcal{H}_1$, we replace the values $\mathsf{df}^l_{i,j}$ for $i,j \in F_{l-1}, i \neq j$ generated in all iterations (in the selection phase of Figure 7) with uniformly random values in $\mathbb{Z}_N$.
We claim that $\mathcal{H}_0 \equiv \mathcal{H}_1$. Indeed, this follows from the perfect security of the One-Time Pad (OTP). Notice that the masking $r_{i,j}$ used to create $\mathsf{df}_{i,j}$ in Figure 7 effectively OTP-encrypts these values – i.e., $(z^l_i)^2 - (z^l_j)^2$ – with fresh keys, since the computations (i.e., adding the masks) are performed in $\mathbb{Z}_N$. More specifically, let t denote the number of such values $\mathsf{df}^l_{i,j}$ in V_R, then we define a sequence of hybrids $\mathcal{H}_0 = \mathcal{H}^0_0, \mathcal{H}^1_0, \ldots, \mathcal{H}^t_0 = \mathcal{H}_1$, where in the h'th hybrid we replace the first h values from their value in V_R to random values as described above. Then each pair of adjacent hybrids are identically distributed from the security of OTP (in particular, even non-uniform distinguishers cannot distinguish between the hybrids, which is what we need in this case to deduce indistinguishability of the hybrids $\mathcal{H}_0, \mathcal{H}_1$ from the indistinguishability of a single OTP ciphertext), and consequently $\mathcal{H}_0 \equiv \mathcal{H}_1$.

$\mathcal{H}_2$**:** In $\mathcal{H}_2$, for every iteration l we replace the pair $\left(\Gamma^l, \beta^l\right)$ generated in the ridge regression phase of Figure 3 with extensions of a uniformly random matrix in $\mathrm{GL}\left(f_{l-1}, \mathbb{Z}_N\right)$ and a uniformly random vector in $\mathbb{Z}_N^{f_{l-1}}$ (respectively). That is, in each iteration we choose $\vec{v} \leftarrow \mathbb{Z}_N^{f_{l-1}}$ and $M \leftarrow GL\left(f_{l-1}, \mathbb{Z}_N\right)$ (these values are chosen independently of each other, and of all other values), and set $\vec{b}^l := \mathsf{expd}_F\left(\vec{v}\right)$ and $\Gamma^l = \mathsf{expd}_F\left(M\right)$.
Then $\mathcal{H}_1 \equiv \mathcal{H}_2$ by Lemma 8.8. Indeed, Lemma 8.8 guarantees that $\left(\Gamma^l, \beta^l\right)$ are uniformly distributed in $\mathsf{expd}_F\left(\mathrm{GL}\left(f_{l-1}, \mathbb{Z}_N\right)\right) \times \mathsf{expd}_F\left(\mathbb{Z}_N^{f_{l-1}}\right)$. (Here, we use the fact that all intermediate matrices A_F are invertible, which holds because X is invertible and so $(X, \vec{y}) \in \mathcal{T}_{\mathsf{Inv},N,d,\lambda,F_1,\ldots,F_k}$, see Remark 8.2.) In particular, this indistinguishability holds for non-uniform distinguishers so similar to the previous set of hybrids, we can define a sequence of hybrids starting from $\mathcal{H}_1$ and ending with $\mathcal{H}_2$ in which we replace the pairs $\left(\Gamma^l, \beta^l\right)$ one at a time. Then each pair of consecutive hybrids are indistinguishable by Lemma 8.8 and consequently $\mathcal{H}_1 \equiv \mathcal{H}_2$.

$\mathcal{H}_3$**:** $\mathcal{H}_3$ is identical to $\mathcal{H}_2$, except for the way in which the permuted ordinal vectors $\vec{\mathsf{Ord}}^{\Pi_l,l}$ are computed. In $\mathcal{H}_3$, for every iteration l we pick a random permutation $\hat{\Pi}_l$ and set $\vec{\mathsf{Ord}}^{\Pi_l,k}$ to be the vector obtained by applying $\hat{\Pi}_l$ to $(0,1,\ldots,f_{l-1}-1)^T$.
We claim that $\mathcal{H}_3 \equiv \mathcal{H}_2$. Indeed, the intermediate model $\vec{z}^l$ computed in the l'th iteration has f_{l-1} *unique* coordinates (see Remark 4.4). Therefore, the *unpermuted* ordinal vector $\vec{\mathsf{Ord}}^l$ computed in the l'th iteration is a length-f_{l-1} vector that contain each value in $\{0,1,\ldots,f_{l-1}-1\}$ exactly once. Put differently, it is a permutated version of the vector $(0,1,\ldots,f_{l-1}-1)^T$, obtained by applying some permutation Π'_l. Applying the permutation Π_l to $\vec{\mathsf{Ord}}^l$ is therefore equivalent to applying the permutation $\Pi'_l \circ \Pi_l$ on the vector $(0,1,\ldots,f_{l-1}-1)^T$. Since the set of permutations over $\{0,1,\ldots,f_l-1\}$ is a group, and Π_l is uniformly random, then $\Pi'_l \circ \Pi_l$ is a uniformly random permutation, namely it is distributed identically to the permutation $\hat{\Pi}_l$ used in $\mathcal{H}_3$. In summary, both $\mathcal{H}_2$ and $\mathcal{H}_3$ generate the ordinal vector $\vec{\mathsf{Ord}}^{\Pi_l,l}$ by applying a random permutation to the vector $(0,1,\ldots,f_{l-1}-1)^T$. Moreover, in all other respects the hybrids are identical. Therefore $\mathcal{H}_3 \equiv \mathcal{H}_2$.

Notice that $\mathcal{H}_3$ contains no information about the permutation P (because it contains no information about P_1, since we have already replaced all values - such as $A, \vec{b}$ - that depend on it). Moreover, it contains no information about the intermediate $A_F, \vec{b}_F$ (except for what can be deduced from the inputs of the corrupted parties).

The remaining hybrids are defined similarly to $\mathcal{H}_3, \mathcal{H}_4$ and $\mathcal{H}_5$ in the proof of Lemma 8.6, and the proof of indistinguishability is also similar:

$\mathcal{H}_4$**:** $\mathcal{H}_4$ is identical to $\mathcal{H}_3$, except for the way in which the S^l_{del}'s are computed. Specifically, in $\mathcal{H}_4$, we change the way in which the S^l_{del}'s are computed. We set $P_0 = P$; and for every iteration l, we choose a permutation P_l which is uniformly random and independent subject to the constraint that P_l agrees with P_{l-1} on $\cup_{t=1}^{l-1} S^t_{\mathsf{del}}$. Each S^l_{del} is permuted according to P_l before it is used to compute F_l in $\mathcal{H}_4$. (Intuitively, in each iteration we re-sample the images of the coordinates that were not yet used to compute S^l_{del}'s which were revealed to $\mathcal{S}_2$.) We claim that $\mathcal{H}_4 \equiv \mathcal{H}_3$. Indeed, neither of the hybrids contain any information about P. Moreover, as in the proof of Lemma 8.5, the permutation P could have been sampled "lazily" in $\mathcal{H}_3$, and is therefore sampled identically to $\mathcal{H}_4$.

$\mathcal{H}_5$: $\mathcal{H}_5$ is identical to $\mathcal{H}_4$ in the proof of Lemma 8.5, and the proof of indistinguishable from the previous hybrid is identical to the proof in Lemma 8.5. (Here, we also rely on the fact that the hybrids contain no information about $A_F, \vec{b}_F$.)

$\mathcal{H}_6$: This is simulated view output by the simulator.

We claim that $\mathcal{H}_6 \equiv \mathcal{H}_5$. There are two differences between the hybrids: (1) how the encrypted model $\vec{w}$ is computed; and (2) how S^l_{del}'s are sampled. Regarding (2), these two methods of sampling S^l_{del} induce identical distributions, as proven in Lemma 8.5 (in the proof that $\mathcal{H}_4 \equiv \mathcal{H}_5$). As for (1), in $\mathcal{H}_5$, $\vec{w}$ is computed from the previously-calculated intermediate models, whereas in $\mathcal{H}_6$ it is reconstructed from the output model in $\mathbb{Q}^d$. However, since the output model in $\mathbb{Q}^d$ is obtained from $\vec{w}$ through rational reconstruction, the correctness of the rational reconstruction algorithm guarantees that the same model is obtained in both hybrids.

□

8.1. On the Necessity of Hiding Partial Information

In this section we explain the cryptographic design choices made in our protocols. Specifically, we explain why certain operations in our protocols are masked, or performed under the hood of the encryption (consequently increasing the round and/or communication complexity of the resultant protocol), by showing that performing these operations on cleartext (and unmasked) data would violate privacy.

We describe several "toy" attacks which illustrate the insecurities of performing the iterated ridge regression protocol (whose scaled version appears in Figure 3) on cleartext, unpermuted, or unmasked data. For the sake of clarity, we present *minimal, concrete* examples that capture the main ideas underlying the attacks. These toy examples help illustrate situations which arise in practice, when performing ridge regression on actual data (see discussion below).

An "attack" on the security of the scheme (i.e., violating Definition 3.1) consists of a pair of inputs for the data owners, for which the inputs and outputs of some permissible adversary (i.e., one which corrupts at most one server and a subset of the data owners) in the protocols are identical, but the adversary's views of the protocol executions are different. In all attacks we consider the setting with 4 or 5 data records, with a single corrupt server (i.e., all data owners are honest). In this case, the adversary has no input, and its output is the (final) output model. Moreover, for simplicity of the examples, we set the regularization parameter to $\lambda = 0$, thus $A = X^T X$. Furthermore, we analyze the attack over $\mathbb{R}$ (and not over $\mathbb{Z}_N$). Analyzing the attacks over $\mathbb{R}$ suffices, because N is chosen such that the computation in $\mathbb{Z}_N$ perfectly emulates the computation over $\mathbb{R}$ (i.e., throughout the computation all values are scaled to be integers, and there are no overflows).

The necessity of permuting the data.. Our protocols operate over *permuted* federated data (see Step 3 in Figure 6). As we show in Section 8, permuting the data hides the order in which features are discarded in the iterative learning process. This is necessary for privacy, since the order might reveal non-trivial information about the inputs, as we now show. Specifically, we will show a pair of inputs for which the resultant model is identical, but the order in which columns are removed in the iterations is different.

The high-level idea of the attack is to begin with $X, \vec{y}$ in which one column is not correlated with $\vec{y}$, the second is somewhat correlated with $\vec{y}$, and the third is highly correlated with $\vec{y}$, where we are looking for a 1-sparse solution. Thus, the first column will be removed first, and the second column will be removed next. This can be used to construct a pair of inputs $X^{(0)}, \vec{y}^{(0)}$ and $X^{(1)}, \vec{y}^{(1)}$ for which the order in which columns are removed is different, by switching the locations of the first and second columns. We first present the explicit example, then discuss the intuition underlying it, and possible extensions.

THE EXAMPLE. Consider the following data matrix and response vector:

$$X = \begin{pmatrix} 1 & 1 & 10 \\ -1 & 2 & 11 \\ 5 & 1 & 12 \\ 1 & 1 & 1 \end{pmatrix}, \quad \vec{y} = \begin{pmatrix} 21 \\ 23 \\ 25 \\ 3 \end{pmatrix}.$$

Then the iterated ridge protocol is instantiated with inputs

$$A = X^T X = \begin{pmatrix} 28 & 5 & 60 \\ 5 & 7 & 45 \\ 60 & 45 & 366 \end{pmatrix}, \quad \vec{b} = X^T \vec{y} = \begin{pmatrix} 126 \\ 95 \\ 766 \end{pmatrix}$$

The solution to the linear system $A\vec{w} = \vec{b}$ is

$$\vec{w} = \begin{pmatrix} 157/1281 \\ 970/1281 \\ 2536/1281 \end{pmatrix}$$

Therefore, from the correctness of the iterated ridge step (Figure 3, see analysis in Section 8), the first column of A will be removed in the first iteration. Thus, the second iteration is initialized with inputs[18]

$$A' = \begin{pmatrix} 7 & 45 \\ 45 & 366 \end{pmatrix}, \quad \vec{b}' = \begin{pmatrix} 95 \\ 766 \end{pmatrix}$$

And the solution to $A'\vec{w}' = \vec{b}'$ is

$$\vec{w}' = \begin{pmatrix} 100/179 \\ 1087/537 \end{pmatrix}$$

So the second column will be removed in the second iteration (from the correctness of the iterated ridge step). Thus, for $X, \vec{y}$ the order in which columns are removed is 1,2 (and column 3 "survives" the iterations). Consider now switching the first and second columns of X, i.e., running the protocol with inputs

$$X^{(1)} = \begin{pmatrix} 1 & 1 & 10 \\ 2 & -1 & 11 \\ 1 & 5 & 12 \\ 1 & 1 & 1 \end{pmatrix}, \quad \vec{y}^{(1)} = \begin{pmatrix} 21 \\ 23 \\ 25 \\ 3 \end{pmatrix}.$$

In this case, all calculations will be symmetric to those with $X, \vec{y}$, except that the role of columns 1,2 is reversed. In particular, column 2 will be removed in the first iteration, and column 1 will be removed in the second iteration. Consequently, if the protocol is executed over unpermuted data, then the set of surviving features (computed in Step 3c in Figure 2) will differ when executing over $X, \vec{y}$, and $X^{(1)}, \vec{y}^{(1)}$, and so the adversary's view is different, even though its output (the final 1-sparse model $w = (0 \;\; 0 \;\; 766/366)^T$) is the same.

DISCUSSION: INTUITION AND EXTENSIONS. We now describe the intuition underlying the example. The starting point is a set of 4 linear equations in 3 variables, with an *exact* solution, in which each of the columns of the matrix defining the linear system has a unique "role" (i.e., there is no symmetry between the columns). Specifically, consider the following system:

$$\begin{pmatrix} 1 & 1 & 10 \\ -1 & 1 & 11 \\ 5 & 1 & 12 \\ 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} 21 \\ 23 \\ 25 \\ 3 \end{pmatrix}.$$

The solution of the system is $w_1 = 0, w_2 = 1, w_3 = 2$. Intuitively, this is because the first column is uncorrelated with the result, and the "weight" of the third column (i.e., its effect on the solution) is double that of the second column. Thus, when searching for a 1-sparse solution with these inputs, the first column will be removed in the first iteration, and the second column will be removed next. Therefore, the system given above already gives an example in which the order of removing columns reveals non-trivial information.

18. Here, we assume that *only* the first feature is removed in the first iteration. This will indeed be the case for this toy example, since the cutoff point in Step 3b of Figure 2 will be computed with $|F| \leq \texttt{thr}$ since the number of features is small.

To show that such information leakage is possible even when no exact solution exist, we add "noise" to the system, by changing the second entry on the second row from 1 to 2, thus obtaining the $X, \vec{y}$ described above.

We note that for simplicity, we chose to present a case in which the two inputs $X^{(0)}, X^{(1)}$ have the same columns (in a different order). However, the example generalizes to cases in which $X^{(0)}, X^{(1)}$ do not share any column, as long as the order of correlations between the columns and the response vector are different in the two inputs. We additionally note that the example can be generalized to matrices of higher dimensions, where instead of considering single columns of X, we divide the columns into 3 sets: a set that is highly correlated with $\vec{y}$ (representing column 3 in the toy example), a set that is somewhat correlated with $\vec{y}$ (representing column 2 in the toy example), and a set that is not correlated with $\vec{y}$ (representing column 1 in the toy example).

Such types of inputs naturally arise when learning is performed over real data. Indeed, X and $X^{(1)}$ can represent records from two different sub-populations $\mathcal{P}_1, \mathcal{P}_2$ with different characteristics. Specifically, the second feature (i.e., column) might be highly correlated with the response vector in $\mathcal{P}_1$ but not in $\mathcal{P}_2$, whereas the first feature (i.e., column) is highly correlated with the response vector in $\mathcal{P}_2$ (but not in $\mathcal{P}_1$). Thus, the information leakage described through this toy example can be used to infer from which of the sub-populations the data was taken.

The necessity of hiding the intermediate models.. Our protocols hide the models computed during intermediate iterations of the ridge regressions protocol (Figure 3). We now show that this is necessary for privacy, as intermediate models might reveal non-trivial information about the inputs. Specifically, we will show a pair of inputs for which the final model, and the order in which columns are removed, are identical, but the intermediate models are not. In particular, this shows that revealing the intermediate models reveals information *beyond* what is revealed by the order in which features are removed.

The high-level idea of the attack is to consider pairs $X^{(0)}, \vec{y}^{(0)}$ and $X^{(1)}, \vec{y}^{(1)}$ of inputs for which there exist $(s+1)$-sparse solutions (that are not s-sparse) that differ only in the value of the smallest coordinate which is present in the $(s+1)$-sparse solution. Since we are looking for an s-sparse solution, this coordinate will not appear in the final model, which will therefore be identical in both cases. More specifically, we can even take $X^{(0)}, X^{(1)}$ to be identical. We proceed to describe the example.

THE EXAMPLE. Consider the following data matrix and response vector:

$$X = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}, \quad \vec{y}^{(0)} = \begin{pmatrix} 6 \\ 3 \\ 7 \\ 10 \\ 9 \end{pmatrix}$$

where we are looking for a 3-sparse solution. Then the data merging step (Figure 6) outputs a permuted version of

$$A = X^T X = \begin{pmatrix} 4 & 3 & 3 & 1 \\ 3 & 4 & 2 & 1 \\ 3 & 2 & 3 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

and

$$\vec{b}^{(0)} = X^T \vec{y}^{(0)} = \begin{pmatrix} 32 \\ 29 \\ 25 \\ 10 \end{pmatrix}$$

where for simplicity of the description we assume the permutation is the identity (so the permuted versions are identical to $A, \vec{b}^{(0)}$). This is without loss of generality, since applying a general permutation will only permute the coordinates of the intermediate models, but will not affect their values. Therefore, the first iteration of the scaled iterated ridge regression phase (Figure 3) is run on inputs $A, \vec{b}^{(0)}$, where $\mathcal{S}_2$ executes the learning algorithm over masked versions of $A, \vec{b}^{(0)}$. Again, for simplicity of the description we assume the masking is trivial (i.e., R is the identity and $\vec{r} = \vec{0}$), so that learning is performed directly on $A, \vec{b}^{(0)}$. This again is without loss of generality since from the correctness of the protocol (see Section 8), learning on masked data returns the "correct" model, which would have been computed if learning had been performed directly on unmasked data. In this case, $\mathcal{S}_2$ computes the scaled model $\vec{z}^{(0)}$ as:

$$\begin{aligned} &\vec{z}^{(0)} = \mathrm{Adj}\,(A) \cdot \vec{b}^{(0)} \\ &= \begin{pmatrix} 5 & -2 & -4 & 1 \\ -2 & 2 & 1 & -1 \\ -4 & 1 & 5 & -2 \\ 1 & -1 & -2 & 5 \end{pmatrix} \cdot \begin{pmatrix} 32 \\ 29 \\ 25 \\ 10 \end{pmatrix} = \begin{pmatrix} 12 \\ 9 \\ 6 \\ 3 \end{pmatrix} \end{aligned} \tag{15}$$

Notice that the corresponding model is

$$\vec{w}^{(0)} = (\det(A))^{-1} \cdot \vec{z}^{(0)} = \begin{pmatrix} 4 \\ 3 \\ 2 \\ 1 \end{pmatrix}.$$

The last column of A will be removed in the first iteration,[19] which will also be the last iteration since we are looking for a 3-sparse solution. The final model will be

$$\vec{w} = \begin{pmatrix} 4 \\ 3 \\ 2 \\ 0 \end{pmatrix}.$$

19. We note that though the computation is performed over the binary representation of $\mathtt{abs}\left(\vec{z}^{(0)}\right)$, it returns the same result as would have been computed directly on $\vec{z}^{(0)}$ because the computation is over $\mathbb{R}$ and all coordinates are positive.

Consider now the inputs $X, \vec{y}^{(1)}$, where

$$\vec{y}^{(1)} = \begin{pmatrix} 6 \\ 3 \\ 7 \\ 9.5 \\ 9 \end{pmatrix}.$$

Again, we assume without loss of generality that the permutation and masking are trivial, and so in the first iteration of the scaled iterated ridge regression phase, $\mathcal{S}_2$ computes the scaled model $\vec{z}^{(1)} = \mathrm{Adj}\,(A) \cdot \vec{b}^{(1)}$, where

$$\vec{b}^{(1)} = X^T \vec{y}^{(1)} = \begin{pmatrix} 31.5 \\ 28.5 \\ 24.5 \\ 9.5 \end{pmatrix}.$$

Then

$$\vec{z}^{(1)} = \begin{pmatrix} 12 \\ 9 \\ 6 \\ 1.5 \end{pmatrix},$$

and notice that the corresponding unscaled model is

$$\vec{w}^{(1)} = \begin{pmatrix} 4 \\ 3 \\ 2 \\ 0.5 \end{pmatrix}.$$

Again, there will be a single iteration and the final model will be $\vec{w}$, but the intermediate models are different: $\vec{z}^{(0)} \neq \vec{z}^{(1)}$ and consequently also $\vec{w}^{(0)} \neq \vec{w}^{(1)}$.

The necessity of hiding $\det(A)$.. Our protocols hide the determinant $\det(A)$ used to un-scale the final model (see Step 1 in Figure 9). We now show this is necessary for privacy, as $\det(A)$ might reveal non-trivial information about the inputs. Specifically, we will show a pair of inputs for which the (unscaled) final and intermediate models, and the order in which columns are removed, are identical, but $\det(A)$ is not. In particular, this shows that revealing $\det(A)$ reveals information *beyond* what is revealed by the order in which features are removed, *and* the intermediate models.

The high-level idea of the attack is conceptually simple: we consider a pair $X^{(0)}, \vec{y}^{(0)}$ of inputs, and a "scaled" version $X^{(1)} = 2X^{(0)}, \vec{y}^{(1)} = 2\vec{y}^{(0)}$. Since the data and response vector are scaled by the same scaler, the resultant models will be the same (throughout the execution of the protocol), but $\det(A)$ will be different. We proceed to explicitly describe the example.

THE EXAMPLE. For $X^{(0)}, \vec{y}^{(0)}$ we consider the same inputs as in the previous example (which showed the necessity of hiding the intermediate models). As in the previous example, we consider the case that parties are looking for a 3-sparse solution, and make the same simplifying assumptions (namely, that the permutation and masking are trivial). Consequently, all computations will be identical. In particular, the scaled model $\vec{z}^{(0)}$, the unscaled model $\vec{w}^{(0)}$, the final model $\vec{w}$, and $\det\left(A^{(0)}\right) = \det\left(\left(X^{(0)}\right)^T \cdot X^{(0)}\right)$ satisfy:

$$\vec{z}^{(0)} = \begin{pmatrix} 12 \\ 9 \\ 6 \\ 3 \end{pmatrix}, \quad \vec{w}^{(0)} = \begin{pmatrix} 4 \\ 3 \\ 2 \\ 1 \end{pmatrix},$$

$$\vec{w} = \begin{pmatrix} 4 \\ 3 \\ 2 \\ 0 \end{pmatrix}, \quad \det\left(A^{(0)}\right) = 3.$$

Now, consider the inputs $X^{(1)} = 2X^{(0)}, \vec{y}^{(1)} = 2\vec{y}^{(0)}$. Then

$$A^{(1)} = \left(X^{(1)}\right)^T X^{(1)} = \begin{pmatrix} 16 & 12 & 12 & 4 \\ 12 & 16 & 8 & 4 \\ 12 & 8 & 12 & 4 \\ 4 & 4 & 4 & 4 \end{pmatrix} = 4A^{(0)},$$

and

$$\vec{b}^{(1)} = \left(X^{(1)}\right)^T \vec{y}^{(1)} = \begin{pmatrix} 128 \\ 116 \\ 100 \\ 40 \end{pmatrix} = 4\vec{b}^{(0)}$$

Therefore, in the first iteration of the scaled iterated ridge regression phase (Figure 3), $\mathcal{S}_2$ computes the scaled model $\vec{z}^{(1)}$ as:

$$\begin{aligned} \vec{z}^{(1)} &= \text{Adj}\left(A^{(1)}\right) \cdot \vec{b}^{(1)} \\ &= \begin{pmatrix} 320 & -128 & -256 & 64 \\ -128 & 128 & 64 & -64 \\ -256 & 64 & 320 & -128 \\ 64 & -64 & -128 & 320 \end{pmatrix} \cdot \begin{pmatrix} 128 \\ 116 \\ 100 \\ 40 \end{pmatrix} \\ &= 64\text{Adj}\left(A^{(0)}\right) \cdot 4\vec{b}^{(0)} = 256\vec{z}^{(0)} = \begin{pmatrix} 3072 \\ 2304 \\ 1536 \\ 768 \end{pmatrix} \end{aligned} \tag{16}$$

Notice that the corresponding model is

$$\vec{w}^{(1)} = \left(\det\left(A^{(1)}\right)\right)^{-1} \cdot \vec{z}^{(1)} = \frac{1}{768} \cdot \vec{z}^{(1)} = \begin{pmatrix} 4 \\ 3 \\ 2 \\ 1 \end{pmatrix}.$$

The last column of A will again be removed in the first iteration, which will also be the last iteration since we are looking for a 3-sparse solution. The final model will be $\vec{w}$ as in the previous case. In summary, the (unscaled) intermediate and final models are the same, as well as the order in which columns are removed. However, $\det\left(A^{(0)}\right) = 3 \neq 768 = \det\left(A^{(1)}\right)$.

9. Complexity Analysis

In this section we analyze the performance of SIR, and in particular the number of iterations and communication rounds, its runtime and communication complexity.

Communication is counted in the number of ciphertexts exchanged, and $\mathcal{S}_1$ performs all operations over ciphertexts. The CRT is implemented using $O(\log N)$ primes. Ciphertext size is polynomial in the security parameter. We denote $\tau = 1 - \mathsf{rej}$. The analysis is for $\mathsf{rej}, \tau = O(1)$, as is the case in our experiments.

The following corollary summarizing the main complexity measures of SIR.

Corollary 9.1. *Let $d, \mathsf{thr}, \ell, m, n \in \mathbb{N}$ and $\lambda \geq 0$ be as in Figure 2. Then SIR has the following complexity properties:*

- *To execute Step 1 of SIR (the data merging and permuting step, Figure 6), $\mathcal{S}_1$ evaluates a circuit (consisting solely of addition gates) of size*
$$O(md^2\lceil\frac{d^2}{\mathsf{sl}}\rceil(\ell + \log(n^2+\lambda) + \log d))$$
- *To execute Steps 3-4 of SIR, $\mathcal{S}_1$ evaluates a circuit of size*
$$O\left(d(d+\mathsf{thr}^2)\lceil\frac{d}{\mathsf{sl}}\rceil(\ell + \log(n^2+\lambda) + \log d)\right)$$
and multiplicative depth
$$O\left(\log d + \log(\ell + \log(n^2+\lambda))\right).$$
- *The communication complexity between $\mathcal{S}_1$ and $\mathcal{S}_2$ throughout the execution is*
$$O\left(d \cdot (d+\mathsf{thr}^2) \cdot \lceil\frac{d}{\mathsf{sl}}\rceil \cdot (\ell + \log(n^2+\lambda) + \log d)\right)$$
ciphertexts.
- *There are $O(\log d + \mathsf{thr})$ rounds of communication.*

To prove the corollary, we analyze the number of iterations, and the complexity of the different steps in SIR. We first bound the number of iterations when executing SIR.

Lemma 9.2 (Number of iterations)**.** *Let $d > \mathsf{thr} > s > 0$ be parameters as in Figure 2. Then SIR performs $O(\log d + \mathsf{thr})$ iterations, and the number of features at the onest of each of these iterations is: $d, d \cdot \tau, d \cdot \tau^2, \ldots, d \cdot \tau^r, d \cdot \tau^r - 1, \ldots s$, where $\tau = 1 - \mathsf{rej}$ and $r = \lceil \log_\tau \frac{\mathsf{thr}}{d} \rceil$.*

The proof is straight forward:

Proof. Let d_i be the number of features at the onset of the i-th iteration, where $d = d_1$ is the number of features in the input to SIR. Then the i-th iteration removes $d_i \cdot \mathsf{rej}$ features if $d_i > \mathsf{thr}$ and it removes a single feature otherwise. Since $\tau = 1 - \mathsf{rej}$, the number of features at the onset of each iteration is $d, d \cdot \tau, d \cdot \tau^2, \ldots, d\tau^r, d \cdot \tau^r - 1, \ldots s+1$, where $r = \lceil \log_\tau \frac{\mathsf{thr}}{d} \rceil$. □

We next turn to analyze the data owner complexity.

Lemma 9.3 (Data owner complexity)**.** *Let $n_j, d, N \in \mathbb{N}$ be parameters as in Figure 2, let $\mathsf{sl} \in \mathbb{N}$ be the number of slots in a ciphertext, and let $X^j \in \mathbb{R}^{n_j \times d}$, and $\vec{y}^j \in \mathbb{R}^{n_j}$ be the input of data owner DO_j. Then the running time of DO_j in SIR is $O(d^2 \cdot n_j)$, and shes sends $O(d\lceil\frac{d^2}{\mathsf{sl}}\rceil \log N)$ ciphertexts.*

Proof. We note that $\mathsf{DO}_j, j \in [m]$ participates only in the data uploading and merging phase (Figure 6) where she computes $A^j = (X^j)^T \cdot X^j$ and $\vec{b}^j = (X^j)^T \cdot \vec{y}^j$. The dominant part is computing and uploading A^j. Computing A^j is done in plaintext and takes $O(d^2 \cdot n_j)$ time. Since we use the encoding and the packing of [37], A is encoded using d rotated copies. These d rotated copies of A consist of $O\left(d \cdot \lceil\frac{d^2}{\mathsf{sl}}\rceil\right)$ elements in total. Since each element is CRT-encoded using $O(\log N)$ ciphertexts, the total communication is $O(d\lceil\frac{d^2}{\mathsf{sl}}\rceil \log N)$ ciphertexts. □

We note that using a different encoding (e.g. as described in [64], [65]) can reduce the time and communication requirement from the data owner on the expense of increasing the complexity for $\mathcal{S}_1$.

Lemma 9.4 (Data preparation)**.** *Let $d, N, m \in \mathbb{N}$ be parameters as in Figure 2. Then during the data merging and permuting step (Figure 6), $\mathcal{S}_1$ performs $O(md\lceil\frac{d^2}{\mathsf{sl}}\rceil \log N)$ operations over ciphertexts.*

Proof. The dominant part is merging the A^j's into A, and permuting A. Computing $A = \sum_{j=1}^m A^j$ requires $O(md\lceil\frac{d^2}{\mathsf{sl}}\rceil)$ operations on elements of A. Permuting A is done by $O(1)$ matrix multiplications where each matrix multiplication takes $O(d\lceil\frac{d^2}{\mathsf{sl}}\rceil)$ operations on elements of A. (This is because A has $\lceil\frac{d^2}{\mathsf{sl}}\rceil$ elements, and is represented using d rotated copies, since we use the encoding of [37].)

Each element of A is CRT-encoded with $O(\log N)$ different plaintext moduli. This yields a total running time of $O(d\lceil\frac{d^2}{\mathsf{sl}}\rceil \log N)$. □

Lemma 9.5. *(Single SIR iteration) Let d_i be the number of features at the onset of the i-th iteration, N be as in Figure 2, and sl be the number of slots in a ciphertext. Then in the i-th iteration:*

- *$\mathcal{S}_1$ evalutes a circuit of size $O(d_i\lceil\frac{d_i}{\mathsf{sl}}\rceil \log N + d_i^2(\log\log N + \lceil\frac{\log N}{\mathsf{sl}}\rceil))$ and depth $O(\log\log N)$.*
- *The communication complexity between $\mathcal{S}_1$ and $\mathcal{S}_2$ is $O(d_i^2\lceil\frac{\log N}{\mathsf{sl}}\rceil + d_i\lceil\frac{d_i}{\mathsf{sl}}\rceil \cdot \log N)$.*
- *There are $O(1)$ rounds of communication.*

Proof. A single SIR iteration is composed of a scaled ridge regression step, in which a vector w that satisfies $Aw = \vec{b}$ (for the given $A, \vec{b}$) is computed, and of a ranking step where the elements of w are ranked.

By [37, Thm. 5], the ridge step can be implemented using a circuit of size $O(d_i\lceil\frac{d_i^2}{\mathsf{sl}}\rceil \cdot \log N)$ with no multiplications (i.e., the multiplicative depth is 0), and with a single communication round, communicating $O(\lceil\frac{d_i^2}{\mathsf{sl}}\rceil \log N)$ ciphertexts.

To rank the (squared) elements of w (Figure 7), SIR first computes the masked differences $\Delta_{i,j} = w_i^2 - w_j^2 + r_{i,j}$ for all $i > j$, where $r_{i,j}$ is chosen uniformly at random. Next, $\mathcal{S}_1$ and $\mathcal{S}_2$ communicate to compute ranges in binary representation. $\mathcal{S}_1$ then compares these ranges to $r_{i,j}$ to determine whether $w_i^2 > w_j^2$ for all i, j, and then computes the ranks. We now analyze: (i) computing $\Delta_{i,j}$, for $i > j$, (ii) computing the ranges and (iii) ranking the elements of w.

Computing $\Delta_{i,j}$. The elements of w are packed in the slots of a ciphertext C (if $d_i > \mathsf{sl}$ this is simulated using $\lceil\frac{d_i}{\mathsf{sl}}\rceil$ ciphertexts). Subtracting a rotation of C by j slots ($C - \mathsf{Rotate}(C, j)$) yields $\Delta_{i,i+j}$, for $i = 1, 2, \ldots, d_i - j$. Repeating this for $j = 1, 2, \ldots, d_i - 1$ gives $\Delta_{i,j}$ for all $i > j$. This step is performed by $\mathcal{S}_1$ using a circuit of size $O(d_i\lceil\frac{d_i}{\mathsf{sl}}\rceil \log N)$ and multiplicative depth 0. (There are d_i vectors $\mathsf{Rotate}(C, j)$, each represented using $\lceil\frac{d_i}{\mathsf{sl}}\rceil \log N$ ciphertexts.)

Computing ranges. To compute the ranges for a pair i, j, $\mathcal{S}_1$ sends $\Delta_{i,j}$ to $\mathcal{S}_2$. $\mathcal{S}_2$ then decrypts $\Delta_{i,j}$ (which are *masked* differences), computes the ranges – in binary representation – and sends the encryptions of the ranges to $\mathcal{S}_1$. Each value in these ranges can be represented with $\lceil\log_2 N\rceil$ bits (padded with 0s if needed). The bits are encoded in the slots of a single ciphertext (where if $\log N > \mathsf{sl}$ then this is simulated using $\lceil\frac{\log N}{\mathsf{sl}}\rceil$ ciphertexts). Therefore, computing the ranges can be done with a single communication round in which $O(d_i\lceil\frac{d_i}{\mathsf{sl}}\rceil \log N + d_i^2\lceil\frac{\log N}{\mathsf{sl}}\rceil)$ ciphertexts are communicated. (The first term is due to communicating the $(\Delta_{i,j})_{i<j}$ from $\mathcal{S}_1$ to $\mathcal{S}_2$, where the second is due to communicating the ranges.)

Computing ranks. To compute the ranks, $\mathcal{S}_1$ computes an indicator $\chi_{i,j}$ of whether $w_i^2 < w_j^2$, namely $\chi_{i,j} = 1$ if $w_i^2 < w_j^2$, otherwise $\chi_{i,j} = 0$. $\chi_{i,j}$ is computed by comparing the appropriate ranges to $r_{i,j}$. This comparison is done using a circuit of size $O(\log\log N + \lceil\frac{\log N}{\mathsf{sl}}\rceil)$ and depth $O(\log\log N)$. Computing the ranks of all d_i features is done by computing all indicators $\chi_{i,j}$ and summing $\mathsf{rank}_i = \sum_{j\neq i}\chi_{i,j}$. Therefore, the entire circuit has size $O(d_i^2(\log\log N + \lceil\frac{\log N}{\mathsf{sl}}\rceil))$ and depth $O(\log\log N)$.

Putting everything together gives the bounds specified in the theorem statement. □

We note that an alternative method for computing the ranks is to use a sorting network (e.g., bitonic or Batcher sort). In this case, the ranking protocol (which would replace parts of the protocol of Figure 7) can be done using a circuit of size $O(d^2 \log^2 d \log\log N)$ and depth $O(\log^2 d \log\log N)$.

Theorem 9.6 (SIR analysis)**.** *Let $d, s, N, \mathsf{rej}, \mathsf{thr}$ be as in Figure 2. Then SIR has the following complexity properties:*

- *To execute Step 1 of SIR (the data merging and permuting step, Figure 6), $\mathcal{S}_1$ evaluates a circuit (consisting solely of addition gates) of size*
$$O(md\lceil\frac{d^2}{\mathsf{sl}}\rceil \log N)$$
- *To execute Steps 3-4 of SIR, $\mathcal{S}_1$ evaluates a circuit of size*
$$O((d + \mathsf{thr}^2)\lceil\frac{d}{\mathsf{sl}}\rceil \log N + O\left((d^2 + \mathsf{thr}^3)(\log\log N + \lceil\frac{\log N}{\mathsf{sl}}\rceil)\right)$$
and multiplicative depth $O(\log\log N)$.
- *The communication complexity between $\mathcal{S}_1$ and $\mathcal{S}_2$ throughout the execution is*
$$O\left((d + \mathsf{thr}^2)\lceil\frac{d}{\mathsf{sl}}\rceil \log N + (d^2 + \mathsf{thr}^3)\lceil\frac{\log N}{\mathsf{sl}}\rceil\right)$$
ciphertexts.
- *There are $O(\log d + \mathsf{thr})$ rounds of communication.*

Proof. The complexity of the circuit which $\mathcal{S}_1$ evaluates in Step 1 of SIR is described in Lemma 9.4. Let $I = O(\log d)$ denote the number of iterations in the first phase of SIR (i.e., when each iteration removes a rej-fraction of features). By Lemma 9.2, the number of features at the onset of the iterations of SIR are: $d, d\cdot\tau, d\cdot\tau^2, \ldots, d\cdot\tau^r, d\cdot\tau^r - 1, \ldots s$, where $\tau = 1 - \mathsf{rej}$, and $d\tau^r = O(\mathsf{thr})$. Substituting these numbers in the formula describing the complexity of a single iteration in Lemma 9.5, we have that the size of the entire circuit (evaluating all iterations) is
$$O\left(\sum_{i=1}^{I}(d\tau^i\lceil\frac{d\tau^i}{\mathsf{sl}}\rceil \log N + d^2\tau^{2i}(\log\log N + \lceil\frac{\log N}{\mathsf{sl}}\rceil))\right.$$
$$\left.+ \sum_{\delta=s}^{\mathsf{thr}}(\delta\lceil\frac{\delta}{\mathsf{sl}}\rceil \log N + \delta^2(\log\log N + \lceil\frac{\log N}{\mathsf{sl}}\rceil))\right.$$

rearranging, we get

$$O\Bigg(d\log N\cdot\sum_{i=1}^{I}\tau^i\lceil\frac{d\tau^i}{\mathsf{sl}}\rceil+d^2(\log\log N+\lceil\frac{\log N}{\mathsf{sl}}\rceil)\cdot\sum_{i=1}^{I}\tau^{2i}$$
$$+\log N\sum_{\delta=s}^{\mathsf{thr}}\delta\lceil\frac{\delta}{\mathsf{sl}}\rceil+(\log\log N+\lceil\frac{\log N}{\mathsf{sl}}\rceil)\sum_{\delta=s}^{\mathsf{thr}}\delta^2\Bigg)$$
$$\le O\left(d\log N\cdot\sum_{i=1}^{\infty}\tau^i\lceil\frac{d\tau^i}{\mathsf{sl}}\rceil\right)$$
$$+O\left(d^2(\log\log N+\lceil\frac{\log N}{\mathsf{sl}}\rceil)\cdot\sum_{i=1}^{\infty}\tau^{2i}\right)$$
$$+\log N\sum_{\delta=s}^{\mathsf{thr}}\mathsf{thr}\lceil\frac{\mathsf{thr}}{\mathsf{sl}}\rceil+(\log\log N+\lceil\frac{\log N}{\mathsf{sl}}\rceil)\sum_{\delta=s}^{\mathsf{thr}}\mathsf{thr}^2$$

using the formula for an infinite geometric sum, we get

$$O(d\log N\cdot\lceil\frac{d}{\mathsf{sl}}\rceil+d^2(\log\log N+\lceil\frac{\log N}{\mathsf{sl}}\rceil))$$
$$+\log N\cdot\mathsf{thr}^2\lceil\frac{\mathsf{thr}}{\mathsf{sl}}\rceil+(\log\log N+\lceil\frac{\log N}{\mathsf{sl}}\rceil)\cdot\mathsf{thr}^3$$
$$\le O((d+\mathsf{thr}^2)\lceil\frac{d}{\mathsf{sl}}\rceil\log N$$
$$+O\left((d^2+\mathsf{thr}^3)\left(\log\log N+\lceil\frac{\log N}{\mathsf{sl}}\rceil\right)\right)$$

where in the last inequality we use the fact that $d > \mathsf{thr}$.

Since every iteration starts with fresh ciphertexts (because in each iteration $\mathcal{S}_2$ generates fresh ciphertexts) then circuit depth is $O(\log\log N)$. The analysis of the communication complexity is similar to the analysis for the circuit size, and yields

$$O((d+\mathsf{thr}^2)\lceil\frac{d}{\mathsf{sl}}\rceil\log N+(d^2+\mathsf{thr}^3)\lceil\frac{\log N}{\mathsf{sl}}\rceil).$$

□

Corollary 9.1 now follows by plugging-in the value of N from Equation 4:

Proof of Corollary 9.1. By Equation 4 we have $\log N = O\left(d(\ell+\log(n^2+\lambda)+\log d)\right)$. Plugging this into Theorem 9.6, immediately gives the stated size of the circuit computed by $\mathcal{S}_1$ in Step 1 of SIR. Using the fact that $d > \mathsf{thr}$, $x \ge \log x$ for every $x > 0$, and $\lceil\frac{d(\ell+\log(n^2+\lambda)+\log d)}{\mathsf{sl}}\rceil < \lceil\frac{d}{\mathsf{sl}}\rceil(\ell+\log(n^2+\lambda)+\log d)$, we get that $\mathcal{S}_1$ evaluates a circuit of size

$$O\left(d\cdot(d+\mathsf{thr}^2)\cdot\lceil\frac{d}{\mathsf{sl}}\rceil\cdot\left(\ell+\log(n^2+\lambda)+\log d\right)\right)$$

and depth

$$O\left(\log d+\log(\ell+\log(n^2+\lambda))\right)$$

and the communication between $\mathcal{S}_1$ and $\mathcal{S}_2$ consists of

$$O\left(d\cdot(d+\mathsf{thr}^2)\cdot\lceil\frac{d}{\mathsf{sl}}\rceil\cdot\left(\ell+\log(n^2+\lambda)+\log d\right)\right)$$

ciphertexts.

□