Privacy Preserving Feature Selection for Sparse Linear Regression

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

Ben Galili
Technion
Haifa, Israel
benga9@gmail.com

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

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

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

ABSTRACT

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

KEYWORDS

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

1 INTRODUCTION

The Machine Learning (ML) revolution critically relies on large volumes of data to attain high confidence predictions. However, the massive amounts of data collected on individuals and organizations incur serious threats to security and privacy. From a legal perspective, privacy regulations such as the European General Data Protection Regulation (GDPR) and the California Customer Privacy Act (CCPA) aim at controlling these threats. From a technological perspective, Privacy Preserving Machine Learning (PPML) [45] attains ML utility without exposing the raw data,[1] e.g., by leveraging tools for secure computation [28, 33, 69]. Many PPML solutions focus on the *inference* phase, e.g., [11, 31, 34]. A growing body of literature also addresses *training*, covering a range of learning tasks and techniques, including training decision tree models, e.g., [45], as well as linear regression, logistic regression and neural networks models, e.g., [49]. Recently, attention has been drawn also to the task of privacy preserving *feature selection* [44], which –in the context of linear regression– is the focus of our work.

Feature selection [36] is the process of selecting a subset of informative features to be used for model training, while removing non-informative or redundant ones. Feature selection is important for avoiding overfitting, that is, for producing models that support high quality prediction on new data rather than models that essentially "memorize" the training data set while failing to generalize. Moreover, feature selection supports producing *sparse models* whose use for prediction in practice requires measuring only the few selected features. Sparsity is often a desired property, leading to significant cost savings when the model is repeatedly used for prediction, especially when feature extraction is expensive. For example, sparse models are highly desired in medical applications where feature extraction might involve medical interventions with associated financial and morbidity costs (see Section 3.1).

The focus of this work is on feature selection under L_0 *constraints*,[2] specifically for *wrapper methods*. In contrast, previous work on privacy preserving feature selection considered *only filter* [16, 44, 53] *and embedded methods* [1, 3, 4, 20, 27, 29, 40, 45, 46, 50, 62–64, 66, 68, 71, 71], but not *wrapper methods*. We next elaborate on the differences between these methods.

Selecting the subset of features of highest predictive power is computationally intractable [15]; nonetheless heuristic methods – which can be categorized into *filter*, *embedded* and *wrapper* methods – are widely employed in practice with great success. Filter methods assign a score to each feature in isolation, outputting the highest scoring features; they are typically very fast, but *often fail to select the best features* because they ignore relationships and dependencies between features. Embedded methods intrinsically incorporate feature selection into the model training process, e.g., in decision tree regression, as well as in ridge and Lasso regression; The latter two techniques penalize models according to their (L_2 and L_1 respectively) norms, favoring models with fewer high-weight features. However, they *do not necessarily return a sparse model*: the model often includes a tail of low-weight features (particularly in ridge regression). Wrapper methods select features in an iterative process, with a target ML algorithm in mind (e.g., linear regression). In each iteration: (1) a model is trained on the current subset of features, then (2) features are ranked according to an evaluation metric measuring their usefulness for that model, and (3) this ranking is used to specify the subset of features for the next iteration. Wrapper methods are considerably slower than filter and embedded methods, due to the multiple rounds of model training, but *often lead to superior outcomes* for the target ML algorithm.

In summary, none of the aforementioned methods strictly dominates the others (see, e.g., a comparative study in [39]).

[1]Other PPML concerns include limiting the amount of information revealed by the output, e.g., using differential privacy in [57]; see a survey in [43].

[2]Concretely, an L_0 constraint determines the intended number of features in the model (e.g., dictated by hardware considerations and/or limitations, as in [52]). We then look for the best model that satisfies this constraint.

Consequently, common practices often examine several methods in search of the best performing one (using cross-validation to avoid overfitting) or use a *hybrid* approach to combine the strengths of several methods. Examples for hybrid approaches include using a filter method to quickly remove many features, then selecting from the remaining ones using an embedded or wrapper method (see, e.g., [19, 59]); or, executing wrapper and embedded methods in parallel, each scoring the features, where the output includes the features with highest total score (see, e.g., [17, 18, 35]).

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

1.1 Our Contribution

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

Our protocol. We design the *Secure Iterative Ridge regression (SIR)* protocol: a new protocol that produces a *sparse ridge regression model*, over input data that is (horizontally)-partitioned among distrusting data-owners, and where the bulk of the computation is outsourced to two non-colluding servers (aka, *two-server model*). Horizontal partitioning means that each data owner contributes a subset of data samples. Our solution utilizes homomorphic encryption [28, 54] as a central tool. The wrapper method that we realized in a privacy preserving fashion is the commonly used *Recursive Feature Elimination (RFE)* algorithm, introduced in the highly influential paper [37]. RFE starts by considering all features, iteratively training a model on the current set of "surviving" features, and removing features of lowest weight (the number of removed features is a tunable parameter), until reaching the L_0 constraint. The ML model that we train at each iteration is a ridge regression model, so that at the termination of all iterations we obtain a sparse regression model. As a central new component we introduce a *scaled ridge regression* protocol, which may be of independent interest.

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

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

System and empirical evaluation. We implemented our protocol SIR and empirically evaluated its performance. Our system is generic and can be applied to any dataset with numerical features. As a concrete example, we ran experiments on a gene-expression dataset derived from TCGA [60]. The full data matrix taken from TCGA breast cancer data consists of gene expression profiles for 781 samples. Each profile, representing a human subject, consists of over $10K$ values. Our proof of concept data represents regressing the expression pattern of a target gene (a vector with 781 entries), based on arbitrarily selected 40 other genes. Furthermore, we artificially randomly partition the 781 instances amongst 10 data owners. We note that horizontal partition, as in our experiments, has interesting use cases in analyzing gene expression related data, e.g., in [6, 8, 30, 58]. In our experiments, each iteration removes 10% of the features, and the L_0 constraint is to produce a model consisting of 8 features.

We point out that the initial selection of 40 out of $10K$ features would typically be executed using filter methods (due to their fast runtime), which can be done using the prior art on privacy preserving filter methods [16, 44, 53]. We focus therefore on selecting the 8 out of 40 remaining features using SIR –our privacy preserving RFE– which is the contribution of our work.

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

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

1.2 Comparison to Prior Work

Prior privacy preserving feature selection solutions were in filter or embedded methods, whereas ours is the first in a wrapper method (concretely, a privacy preserving RFE). Although none of these methods dominates the others, RFE was developed in [37] for the use case of gene expression data, where it excelled. Indeed, the mean square error (MSE) of the model produced by our system significantly outperforms the regression models produced by filter, ridge and Lasso (the baseline); see Section 10.2.

In terms of runtime, our system is expected to be slow, since, even in cleartext, wrapper methods are considerably slower than filter and embedded ones. This is also evident by our experiments, where the runtime of our system (on 40 features) is roughly 1 day, compared to runtimes between slightly under a minute, and a few hours, on various dataset sizes (with 20-120 features) reported in previous works [4, 16, 27, 29, 40, 44, 50, 53, 63, 64, 71, 71].

Producing better models (i.e., with smaller MSE) –as in SIR– is typically desired, even if it incurs a slower (but reasonable) training runtime. This is because training a model occurs once, whereas the fruits of having a better MSE provide recurring benefits in

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

each use of the model. Furthermore, a runtime of 1 day (and even more) seems to be reasonable in the context of gene-expression data, where the trained model is used to guide the manufacturing of a medical testing device (DNA chip), and our training time is insignificant when compared to the manufacturing pipeline.

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

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

1.3 Paper Organization

We give an overview of our techniques, along with a comparison to techniques used in prior work, in Section 2. Preliminary definitions appear in Section 3; the problem statement in Section 4. The SIR protocol appear in Sections 5-6; our attacks (on simplified variants of SIR, demonstrating the necessity of SIR's various components) in Section 7; and the security and complexity analysis of SIR in Sections 8-9 respectively. Our system and empirical evaluation appear in Section 10. Conclusions appear in Section 11.

2 OVERVIEW OF OUR TECHNIQUES

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

2.1 IR and SIR

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

Iterated Ridge (IR, cf. Figure 1) is an RFE algorithm for sparse ridge regression, analogous to the sparse logistic regression algorithm used in [26]. IR starts with all features, removing 10% of the features in each iteration, until reaching $2 \cdot s$ features, where s is the user-determined target number of features, then removes features one-by-one.[4] (This follows the paradigm of adjusting the learning rate to a smaller value when approaching the solution.) Selecting which features to remove in each iteration is done by solving ridge regression (see Section 3.1) on the surviving features to obtain an intermediate model, and removing the features whose weights (in absolute value) are in the bottom 10%.

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

SIR is composed of four phases as detailed next.

Phase I. Setup & Upload (cf. Figures 4-5 and Figure 6 Step 1): $\mathcal{S}_2$ generates a key pair (pk, sk) for a fully homomorphic encryption scheme, and publishes pk.[5] Each data owner DO_j encrypts (entry-by-entry) $A^j = (X^j)^T \cdot X^j$ and $\vec{b^j} = (X^j)^T \cdot \vec{y^j}$ under pk, and sends the ciphertexts to $\mathcal{S}_1$ who homomorphically combines them to obtain ciphertexts for:

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

Phase II. Obliviously Permute Features (cf. Figure 6 Steps 2-3): $\mathcal{S}_1$ and $\mathcal{S}_2$ engage in a 1-message protocol that concludes with $\mathcal{S}_1$ holding ciphertexts for

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

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

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

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

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

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

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

This is a scaling by factor $\det(A_{p|F})$ of the ridge regression model for $(A_{p|F}, \vec{b}_{p|F})$ which is $\vec{w} = (A_{p|F})^{-1} \cdot \vec{b}_{p|F}$. This scaling is central for both the correctness and the efficiency of our protocol; see a detailed discussion in Section 2.2.

In detail, the scaled ridge protocol operates as follows. First, $\mathcal{S}_1$ homomorphically masks $A_{p|F}$ and $\vec{b}_{p|F}$, and sends their masked versions to $\mathcal{S}_2$, where these masked versions are: $A^{\text{masked}} = A_{p|F} \cdot R$ and $\vec{b}^{\text{masked}} = \vec{b}_{p|F} + A_{p|F} \cdot \vec{r}$ for uniformly random invertible matrix R and random vector r of the appropriate dimensions.[6] Next, $\mathcal{S}_2$ decrypts, computes $\det(A^{\text{masked}})$ and $\vec{w}^{\text{masked}}_{\text{scaled}} = \text{adj}(A^{\text{masked}}) \cdot \vec{b}^{\text{masked}}$ in the clear, encrypts them, and sends the ciphertexts to $\mathcal{S}_1$. Finally, $\mathcal{S}_1$ homomorphically unmasks to obtain ciphertexts for the scaled (un-masked) model, using the algebraic identity: $\vec{w}_{\text{scaled}} = R \cdot \vec{w}^{\text{masked}}_{\text{scaled}} + \det(A^{\text{masked}}) \cdot \vec{r}$. We refer the reader to our analysis in Section 8 for the proof of correctness.

(2) *Ranking (cf. Figure 9, Steps 1-4)*: Next, $\mathcal{S}_1$ and $\mathcal{S}_2$ engage in a two-party protocol, at the conclusion of which $\mathcal{S}_1$ holds a ranking –in cleartext– of the surviving features according to their weight in $\vec{w}_{\text{scaled}}$; see details in Section 2.3. Importantly, $\mathcal{S}_1$ does not know the actual weights, only their ordering. This does not violate privacy, because the features were randomly permuted; details are provided in our privacy analysis in Section 8.

(3) *Removal (cf. Figure 9, Steps 5-6):* Finally, $\mathcal{S}_1$ removes the (dynamically adjusted desired number of) lowest ranking features, and updates F, $A_{p|F}$ and $\vec{b}_{p|F}$ accordingly.

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

Phase IV. Obliviously un-permute and rationally reconstruct (cf. Figure 8): At the onset of this phase the servers hold a subset $F \subseteq [d]$ of feature indices, which satisfies the L_0 constraint, i.e., $|F| = s$. They now engage in a two-party protocol to produce a (cleartext) sparse ridge regression model whose non-zero weights are only on features in F.[7] First, $\mathcal{S}_1$ and $\mathcal{S}_2$ engage in a two-party sub-protocol, at the conclusion of which $\mathcal{S}_1$ holds ciphertexts for the ridge regression model $\vec{w}_p = (A_{p|F})^{-1} \cdot \vec{b}_{p|F}$ for the final set of features F. Second, $\mathcal{S}_1$ homomorphically un-permutes the features' order (where features that did not survive have weight zero), and sends this un-permuted encrypted model to $\mathcal{S}_2$ who decrypts it, and sends it in the clear to $\mathcal{S}_1$. That is, $\mathcal{S}_1$ now has $\vec{w} = A^{-1}_{|F} \cdot \vec{b}_{|F}$, where the inverse and product are computed in $\mathbb{Z}_N$. Finally, $\mathcal{S}_1$ maps $\vec{w}$ to the solution to ridge regression over the surviving features *when computed over the rationals* (rather than in $\mathbb{Z}_N$) via rational reconstruction [23, 65],[8] and outputs the resulting model.

[6]This masking was introduced in [29] in the context of privacy preserving ridge regression. However, unlike our work, they employ masking to produce a ridge model that is *un-scaled* and *in cleartext*; see details in Section 2.6.

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

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

2.2 Our Scaled Ridge Protocol

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

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

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

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

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

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

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

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

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

2.3 Our Ranking Protocol

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

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

The above (simplified) description of the ranking protocol overlooked the following subtle point: in each iteration, the servers learn the ordering between the features in the current iteration's model. If *the same* permutation on features were to be used in all iterations, this knowledge from the execution of multiple iterations might reveal non-trivial information (similar to the issues which arise by performing scaled ridge on unpermuted features, see Section 7). To overcome this, we have $\mathcal{S}_1$ apply a *fresh* permutation on the Ord_i's before sending them to $\mathcal{S}_2$. The permutation is inverted at the end of this step, and does not affect the rest of the computation. Privacy is preserved because each iteration uses a fresh permutation for ranking (which is applied on top of the long-term permutation applied at the onset of the protocol). See Section 8 for details.

2.4 Security Challenges and Attacks

The iterative training of intermediate models is inherent to wrapper methods (and does not occur in filter or embedded methods), and introduces several security challenges. Indeed, we show that revealing any of the following (which are revealed in standard IR) would violate privacy (see details in Section 7):

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

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

2.5 Discussion: Model Inversion Attacks

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

Prior works [24, 25, 70] have shown that learning the model may indeed reveal non-trivial information on private inputs of honest parties. Most relevantly, [25] showed, in the context of genomic data for personalized medicine using a *(non-sparse)* linear regression model, that access to the trained model can be abused to infer private genomic attributes about individuals in the training dataset. Moreover, [25] showed that differential privacy is not effective for guaranteeing privacy because it is at odds with utility: when setting the privacy parameters sufficiently high to prevent their attack, the produced model does not retain sufficient clinical efficacy.

SIR is likewise susceptible to model inversion attacks. This is because sparsity of the model in itself does *not* guarantee security against inversion attacks, since even *a single bit* of information can compromise privacy. For example, in the context of gene-expression data, certain genetic variants are more common in some ethnicities than in others, and so revealing whether a feature corresponding to such a genetic variant is selected in the model may reveal the ethnicity of the population represented in the training dataset. Concretely, we present a model inversion attack showing that corrupt data owners who inject maliciously crafted inputs (but otherwise follow the protocol) can infer from the output model non-trivial information on the inputs of the honest data owners. The attack holds even when the output is a 1-sparse model, i.e., it consists of one selected feature. At a high-level, corrupted data owners use "balanced" inputs, in which all features have the same correlation with the response vector. Thus, inputs provided by corrupted parties do not affect the output model, and any correlations in the inputs of the honest parties will determine which features are selected in the sparse output model. The formal details are provided in Section 7.

To reduce the privacy risk associated with revealing the model, our solution SIR offers a fine grained control on how, and to whom, the model is revealed. This can be obtained by applying the following minor changes in Phase IV: "Obliviously un-permute and rationally reconstruct" step(cf. Section 2.1 and Figure 8 Step 3).

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

Alternative 2: Revealing the model only to one designated party, denoted O, who may be an external party participating only in

Phase IV, as follows. $\mathcal{S}_1$ sends the (encrypted and un-permuted) model to O (instead of $\mathcal{S}_2$); O homomorphically masks the model with a random additive mask (using Add from Table 1), and sends the encrypted masked model to $\mathcal{S}_2$ who decrypts and returns the (cleartext) masked model to O; finally, O removes the masking and computes rational reconstruction to obtain the output model. This is motivated by scenarios where $\mathcal{S}_1, \mathcal{S}_2$ are powerful servers to which computation is outsourced (say, Amazon AWS and Google Cloud) and where O is some health organization authorized to receive the computed model (say, the World Health Organization).

Alternative 3: Outputting the model only in encrypted form. In this case $\mathcal{S}_1$ simply publishes the encrypted model, instead of sending it to $\mathcal{S}_2$ for decryption (i.e., remove Step 3 in Figure 8). The model in this case is specified by a vector of d ciphertexts whose plaintext values are in $\mathbb{Z}_N$ and where at most s of them are non-zero (in order to hide both the indices of the selected features and their weight). Outputting an encrypted model is motivated by scenarios where the encrypted model is employed for privacy preserving inference using homomorphic computation, as follows. Given the encrypted model and a (possibly, encrypted) sample[9] homomorphically compute the inner product of the model and sample, homomorphically mask the resulting value in $\mathbb{Z}_N$ (using additive mask), and send the ciphertext to $\mathcal{S}_2$ for decryption; then unmask the returned plaintext and apply rational reconstruction to obtain the inference result as a rational number. See Remark 8.9 for further details. We note that in order to reduce the efficacy of model inversion attacks it is advised to enforce a security policy restricting the entities authorize to make inference queries, and limiting their number of allowed queries. Analyzing such security measures is beyond the scope of this work.

2.6 Comparison to Prior Techniques

The two-server approach for privacy-preserving ridge regression dates back to the work of Nikolenko et al. [50], who were also the first to propose using additive homomorphic encryption for merging the data from all data owners. We follow them in our Setup & Upload phase. Solving the linear system $(A, \vec{b})$ was done in [50] using garbled circuits to guarantee security. Giacomelli et al. [29] proposed instead to solve a masked linear system $(A^{\text{masked}}, \vec{b}^{\text{masked}})$, where $\mathcal{S}_1$ masks the systems using homomorphic addition, $\mathcal{S}_2$ decrypts, solves, and sends the solution (the model) –in cleartext– to $\mathcal{S}_1$, who unmasks and applies rational reconstruction in the clear. Importantly, in [29] the model is sent in the clear, and both the unmasking and the rational reconstruction are done in the clear. However, this is impossible in our setting: we cannot expose the intermediate models in the clear, because this compromises privacy as we show in our attacks. Moreover, we cannot simply have $\mathcal{S}_2$ send the model in encrypted form and have $\mathcal{S}_1$ process it homomorphically, because the rational reconstruction step requires many sequential steps, making it computationally expensive to compute homomorphically.

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

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

The overall structure of our protocol is likewise novel to our work, including its components of obliviously permuting and un-permuting the features (Phases II and IV in SIR) as well as the iterative execution of privacy-preserving regression for the ranking and removal (Phase III in SIR). We note that this overall structure necessitates using a *fully* homomorphic encryption (FHE), e.g., BGV [14] or B/FV [12, 22], to support both homomorphic addition and multiplication with respect to our plaintext modulus N (cf. only additive homomorphism in [10, 29]). However, as observed in [4], existing FHE libraries (e.g., HElib [38] and SEAL [55]) only support plaintext modulus of size up to 64-bit, whereas our protocol requires much larger integers (1260-bit integer in our implementation). To resolve this issue we follow [4] who suggested using the Chinese Remainder Theorem to represent each integer modulo N as a tuple of integers modulo small(ish) primes as supported by these libraries.

3 PRELIMINARIES

Notation. Upper-case letters (e.g., M) denote matrices, and vector notation (e.g., $\vec{v}$) denotes vectors. We use boldface letters to denote ciphertexts (e.g., $\mathbf{M}, \vec{\mathbf{v}}$ for a matrix and vector, respectively). We use Greek letters to denote masked values (see, e.g., Figure 6).

For a vector $\vec{w}$ we denote the number of its non-zero entries by $\text{nnz}(\vec{w})$, and call it *s-sparse* if $\text{nnz}(\vec{w}) \leq s$. 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 $\text{adj}(A)$, $\det(A)$ to denote the adjugate matrix and determinant of A, respectively. We note that for any pair A, B of matrices it holds that $\text{adj}(A \cdot B) = \text{adj}(B) \cdot \text{adj}(A)$, and $\det(A \cdot B) = \det(A) \cdot \det(B)$. For natural $d, N \in \mathbb{N}$, we use $\text{GL}(d, \mathbb{Z}_N)$ to denote the group of all invertible $d \times d$ matrices with entries in $\mathbb{Z}_N$.

For a real value x, we use $\text{abs}(x)$ to denote its absolute value, and $\lfloor x \rceil$ to denote its nearest integer. We extend the notation to

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

apply to vectors and matrices entry-by-entry. We say that $x \in \mathbb{R}$ has *precision* ℓ if x is given as a real number with ℓ digits after the decimal point (which could be 0). If x has precision ℓ, then by *scaling* x *to lie in* $\mathbb{Z}$ we mean multiplying x by 10^{ℓ}.

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

For random variables R, R', $\mathsf{R} \approx \mathsf{R}'$ denotes that R, R' are computationally indistinguishable. $\mathsf{negl}(\kappa)$ denotes a function which is negligible in κ, and PPT is shorthand for Probabilistic Polynomial Time. We use the standard notion of computational indistinguishability (e.g., from [32]).

3.1 Sparse Linear Regression & Applications

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

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

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

As we will discuss below, regularizing the solution $\vec{w}$ to Equation 1 is often beneficial, leading to LASSO regression [9, 61] and ridge regression, both are special cases of controlling the norm of $\vec{w}$. Ridge regression [4, 29, 41] seeks to find

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

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

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

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

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

In typical datasets, learning sparse linear models is useful due to two main reasons. First, simpler models are preferred during the training stage to avoid overfitting [9, 56]. Lower complexity translates to lower degree of polynomial models and/or less features in the output model. The latter can be reduced to model sparsity. The second reason to prefer sparser models is due to practical considerations. In some cases, hardware limitations restrict the number of features which can be measured when using the prediction model in the execution phase - when used to predict values, y, for new instances. In other cases, using more features in the execution prediction model is more expensive. For example, if y represents tumor severity, it might be reasonable to assume that y can be expressed as a linear (or polynomial) combination of molecular genomic information, say gene expression levels, in X. However, we expect, from a biological perspective, most genes to minimally affect the prediction performance. That is, the biology will be driven by a small number of genes.[10] Therefore, most components of $\vec{w}$ can be zero so that an assay used in clinical practice, based on such a predictive model, can use less expensive hardware, quantifying the expression levels of fewer genes [7, 21, 26, 51, 67].

3.2 Feature Selection and Iterated Ridge

Feature selection is an essential component in computational modelling and in the practical application of models. It has therefore been an active and prolific field of research in various domains such as pattern recognition, machine learning, statistics and data mining [47, 48]. Clever selection of the set of features to be used for data modelling, and as part of the execution models derived from learning, has been shown to improve the performance of supervised and unsupervised learning. Reasons are discussed above, as well as in the literature [7, 21, 37, 52]

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 [37], 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 [26], that developed a sparse logistic regression model using RFE. In each iteration we run ridge regression with $\lambda \geq 0$ [4, 56] to calculate the weights for all features considered. Then, we remove features with low weights (in absolute value). The algorithm operates in two phases. In the first phase, we remove a 0.1-fraction of features, whereas when the current number of features decreases below a (user-defined) threshold thr, we move to the second phase, in which we remove a single feature in each iteration (this latter phase is analogous to Backward Subset Selection). The choice of the actual value of thr and the choice of the fraction removed in the early stages can affect the computational complexity of the process. Moreover, they are hyper-parameters of the model and can be tuned by cross validation. The pseudo-code of this algorithm is given in Figure 1.

3.3 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

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

Input: A dataset $D \in \mathbb{R}^{n \times d}$ and a target vector $\vec{y} \in \mathbb{R}^n$ (where entries are normalized to the same scale), and parameters s, rej $\in (0,1)$, thr $\in [d]$.
Output: A set Ω_s of the $s \le d$ selected features, and a ridge regression model $\vec{w}_s$ on these features.
Steps:

```
(1) Initialize $\Omega$ to be the set of all features, and $\vec{w} = \vec{1}$.
(2) While $\mathrm{nnz}(\vec{w}) > \mathrm{thr}$:
    (a) $\vec{w} = LR(D, y, \Omega)$
    (b) $\pi = \mathrm{argsort}(\mathrm{abs}(\vec{w}))$
    (c) $\mathrm{prefix}(\pi) = \pi[0\colon \mathrm{rej} \cdot |\Omega|]$
    (d) $\Omega = \Omega \setminus \mathrm{prefix}(\pi)$
(3) While $\mathrm{nnz}(\vec{w}) > s$:
    (a) $\vec{w} = LR(D, y, \Omega)$
    (b) $\mathrm{smallest} = \mathrm{argmin}(\mathrm{abs}(\vec{w}))$
    (c) $\Omega = \Omega \setminus \{\mathrm{smallest}\}$
(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; $\mathrm{nnz}(\vec{w})$ 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; $\mathrm{abs}(\vec{w})$ returns the absolute value of each entry of $\vec{w}$. The regularization parameter λ is implicit in this pseudo-code.

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. [13].) 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 decrypts to the value that would have been obtained by evaluating 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 3.1 (FHE). A *Fully-Homomorphic Encryption (FHE)* scheme $\mathcal{E} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ consists of four algorithms where KG, 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 $\left(\mathsf{c}'_1, \ldots, \mathsf{c}'_l\right)$.

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 $\mathsf{msg}' = C\left(\mathsf{msg}_1, \ldots, \mathsf{msg}_l\right)$, and the probability is over the randomness of KG, 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 $\mathsf{c} \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathsf{msg})$.[11] 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 3.2. 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 10 for further details.

4 PROBLEM STATEMENT

We follow the security and threat model of [4], and parts of this section are taken almost verbatim from [4]. SIR guarantees computational security, in the passive setting, against a single server colluding with a proper subset of the data owners. More specifically, we assume all parties, even corrupted ones, are PPT and follow the protocol (though corrupted parties will try to infer additional information). We guarantee correctness of the output, and privacy of the inputs, in this setting. Specifically, the only information revealed to the corrupted parties is the *leakage profile*, namely the information that is *explicitly* revealed by the protocol. In our protocols, the leakage profile consists of the output model $\vec{w}$, as well as the following public parameters: the number n of data instances; the number d of features; the precision ℓ; a sparsity parameter s; and a regularization parameter $\lambda \ge 0$. More formally,

[11]See Remark 3.2 below about simultaneously encrypting multiple field elements.

we consider k-privacy in the passive setting, for inputs X such that $A = X^T X + \lambda I$ is invertible in the ring $\mathbb{Z}_N$ (invertability is needed for IR correctness; and in our case – as in previous works [4, 29] – also for privacy). We note that the input is horizontally-partitioned between the data owners (i.e., data owners hold disjoint subsets of rows of $(X, \vec{y})$).

Terminology. Let Π be an $(m+2)$-party protocol executed between PPT data owners $\mathsf{DO}_1, \ldots, \mathsf{DO}_m$ 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 $I \subset \{\mathsf{DO}_1, \ldots, \mathsf{DO}_m, \mathcal{S}_1, \mathcal{S}_2\}$, the (joint) *view* of I in Π, denoted $\mathsf{V}_I^{\Pi}(x_1, \ldots, x_m)$, is the random variable consisting of the inputs and randomness of all parties in I, as well as the messages they received from the honest parties in a random execution of Π with inputs $x_1, \ldots, x_m$. 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., [32]), adapted to the setting of non-colluding servers as in [50]. Since optimal feature selection under L_0 (Equation 3) is NP hard in general [15], we focus on providing a secure variant of the *Iterated Ridge* heuristic approach (see Section 3.1). Specifically, we require correctness in the sense that the secure variant has the same output as the cleartext iterated ridge algorithm, and privacy in the sense that any k-permissible set I learns nothing except the leakage profile (which consists of the public parameters and the output model) and the inputs of the parties in I (and anything efficiently computable therefrom). We also offer the option of returning an encrypted model (cf. Section 2.5), in which case the output model is excluded from the leakage profile and the adversary learns nothing beyond the public parameters and the input of the parties in I (and anything efficiently computed therefrom). Following [29] we define correctness with respect to a subset $\mathcal{T}$ of inputs (where there is no correctness guarantee for inputs not in $\mathcal{T}$). Formally,

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

(1) **Correctness:** there exists a negligible function $\mathsf{negl}(\kappa) : \mathbb{N} \to \mathbb{N}$ such that for all inputs $(x_1, \ldots, x_m) \in \mathcal{T}$,

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

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

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

5 SIR PROTOCOL

Our privacy preserving iterated ridge protocol SIR is specified in Figures 2-3 (with further details available in the figures in Section 6). See also an overview in Section 2; remarks on input encoding, parameter choice, and useful observations in Section 5.1; and notations for simple sub-circuits we use in the homomorphic evaluation in Table 1. Our security and complexity analysis of SIR is summarized below. Complexity is stated in term of d and N where $\log N = \widetilde{O}(d \log n)$ (by Equation 4).

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

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

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

- *Each data owner runtime is dominated by the time to compute d^2 encryptions, and her communication complexity is dominated by transmitting d^2 ciphertexts (in one round).*
- *$\mathcal{S}_1$ runtime is dominated by the time to rank features, which entails homomorphically evaluating $O(\log d)$ circuits, each with $O(d^2 \log N) = \widetilde{O}(d^3 \log n)$ multiplication gates and of multiplicative depth $O(\log \log N) = O(\log d + \log \log n)$.*[12]
- *$\mathcal{S}_2$ runtime complexity is dominated by the time to solve (in the clear) $O(\log d)$ linear systems of size $d \times d$.*
- *The communication of the two servers consists of $O(\log d)$ communication rounds, transmitting $O(d^2 \log N) = \widetilde{O}(d^3 \log n)$ ciphertexts in each round.*[13]

Proof. The proof is provided in Sections 8 and 9. □

5.1 Input, Parameters and Observations

The notations and operations used in SIR are described in Table 1. Remarks clarifying some implementation details follow.

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

Remark 5.3 (On the choice of N)**.** The plaintext ring $\mathbb{Z}_N$ should be sufficiently large to guarantee that all computations during the (scaled) ridge regression step emulate the corresponding computations over the reals (i.e., no overflows occur), as well as

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

[13]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.

to allow for rational reconstruction to be performed on the output model at the end of the protocol. This can be guaranteed by using the same plaintext ring $\mathbb{Z}_{N_{\text{Gia}}}$ as [29]. However, for our selection protocol (Figure 9) we will need the modulo N to be at least square that value, namely:

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

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

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

Remark 5.5 (Dimension reduction and projection)**.** Our protocol iteratively reduces the set F of current features (i.e., ones that will be part of the output model), which is done in two steps as follows. (1) reset the entries of $A, \vec{b}$ that correspond to entries in $[d] \setminus F$, by setting to zero the rows and columns of A (the entries of $\vec{b}$, respectively) that are indexed by $i \notin F$, resulting in a matrix A' and a vector $\vec{b}'$. (2) projecting $A', \vec{b}'$ to F by erasing the rows and columns of A' (entries of $\vec{b}$, respectively) indexed by $i \notin F$. We denote this operation by $\mathsf{pjct}_F(\cdot)$ (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 5.6 (Unique Entries in Intermediate Models)**.** Our security analysis will rely on the assumption that for every intermediate model $\vec{z}_F$ computed in Step 3a of the SIR protocol (Figure 2), all entries are unique (i.e., if $i \neq j$ then $\vec{z}_{F,i} \neq \vec{z}_{F,j}$). This can be easily achieved as follows. First, when scaling the inputs in the setup phase (Figure 4), we incorporate $\log d$ additional "empty" least-significant bits. That is, instead of scaling an ℓ-precision real number by multiplying it by 10^{ℓ}, we multiply it by $10^{\ell+\log d}$. Then, at the end of each scaled ridge regression iteration (Figure 3) we replace the $\log d$ least-significant bits of $\vec{z}_{F,i}$ with the binary representation of i (this can be done because at this point the ciphertext is encrypted entry-by-entry).

Remark 5.7 (Emulating Boolean circuits using arithmetic circuits)**.** Our protocols embed binary values into a larger ring $\mathbb{Z}_D$, and operate over these representations. Therefore, we need to emulate the Boolean circuits $\mathsf{BinCompareL}_{c'}$ 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 $c \in \{0,1\}$.

6 SIR PROTOCOL: FULL DESCRIPTION

In this section we provide the full description of all sub-protocols used as part of SIR.

Public parameters: $\mathcal{E}$, 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 $\mathsf{pk}_N, \mathsf{pk}_D$.

Figure 4: Setup

Public parameters: $\mathcal{E}$, 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 $A = P_2^T P_1^T \cdot (X^T \cdot X + \lambda I) \cdot P_1 P_2 \in \mathbb{Z}_N^{d\times d}$ 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 $1 \leq j \leq m$, 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 $A^j = (X^j)^T \cdot X^j$, and $\vec{b}^j = (X^j)^T \cdot \vec{y}^j$.
2. **Data Encryption:** Encrypts $\mathbf{A}^j \leftarrow \mathsf{Enc}(\mathsf{pk}_N, A^j)$ 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: Data Uploading

7 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

Operation Type	Circuit Name	Inputs	Constants	output
Scaled matrix addition	Add_λ	matrices $A_1, \ldots, A_m$	$\lambda > 0$	$\sum_{i\in[m]} A_i + \lambda I$
Matrix multiplication	$\mathsf{MatMult}$	matrices A_1, A_2	—	$A_1 \cdot A_2$
Matrix by vector multiplication	$\mathsf{MatVecMult}$	matrix A_1 and vector $\vec{v}$	—	$A_1 \cdot \vec{v}$
Matrix right multiplication (w. plaintext R)	$\mathsf{MatMultR}_R$	$d \times d$-size matrix M	$d \times d$-size matrix R	$M \cdot R$
Matrix left multiplication (w. plaintext L)	$\mathsf{MatMultL}_L$	$d \times d$-size matrix M	$d \times d$-size matrix L	$L \cdot M$
Addition with plaintext matrix	MatAdd_R	$d \times d$-size matrix M	$d \times d$-size matrix R	$M + R$
Multiplication with plaintext vector	$\mathsf{VecMatMultR}_{\vec{r}}$	$d \times d$-size matrix M	length-d vector $\vec{r}$	$M \cdot \vec{r}$
Vector addition	Add	vectors $\vec{v}_1, \ldots, \vec{v}_m$	—	$\sum_{i\in[m]} \vec{v}_i$
Vector subtraction	Sub	vectors $\vec{v}_1, \vec{v}_2$	—	$\vec{v}_1 - \vec{v}_2$
Vector multiplication w. plaintext matrix	$\mathsf{MatVecMultL}_R$	length-d vector $\vec{v}$	$d \times d$ matrix R	$R \cdot \vec{v}$
Vector addition w. plaintext vector	$\mathsf{VecAdd}_{\vec{r}}$	vector $\vec{v}$	vector $\vec{r}$	$\vec{v} + \vec{r}$
Vector multiplication w. plaintext scalar	$\mathsf{ScalerVecMult}_c$	vector $\vec{v}$	scalar c	$c \cdot \vec{v}$
Vector multiplication w. secret scalars	$\mathsf{ScalerVecMult}$	vector $\vec{v}$ and scalars $c_1, \ldots, c_k$	—	$c_1 \cdot \ldots \cdot c_k \cdot \vec{v}$
Vector addition (binary representation)	$\mathsf{VecBinAdd}_{\vec{r}}$	vector $\vec{v} \in \{0,1\}^k$	vector $\vec{r} \in \{0,1\}^k$	$\vec{r} + \vec{v}$ (computed using 2's complement)
Vector subtraction (binary representation)	$\mathsf{VecBinSub}_{\vec{r}}$	vector $\vec{v} \in \{0,1\}^k$	vector $\vec{r} \in \{0,1\}^k$	$\vec{r} - \vec{v}$ (computed using 2's complement)
Scalar multiplication w. plaintext vector	$\mathsf{VecScalerMult}_{\vec{v}}$	scalar c'	vector $\vec{v}$	$c' \cdot \vec{v}$
Scalar squaring	Square	scalar c'	—	$(c')^2$
Scalar addition	$\mathsf{AddNums}$	scalars c', c''	—	$c' + c''$
Scalar subtraction	$\mathsf{SubNums}$	scalars c', c''	—	$c' - c''$
Scalar multiplication	$\mathsf{MultNums}$	scalars c', c''	—	$c' \cdot c''$
Scalar addition w. plaintext scalar	$\mathsf{AddConst}_c$	scalar c'	scalar c	$c + c'$
Scalar smaller than	$\mathsf{BinCompareR}_c$	$\vec{c}' \in \{0,1\}^*$	scalar c	1 if $c' \leq c$, otherwise 0
Scalar greater than	$\mathsf{BinCompareL}_c$	$\vec{c}' \in \{0,1\}^*$	scalar c	1 if $c < c'$, otherwise 0
Scalar negation	Neg	scalar $c' \in \{0,1\}$	—	c''s complement (i.e., 1 if $c = 0'$, 0 otherwise)

Table 1: Sub-circuits used for homomorphic computation in SIR. In the table, $k, d, m \in \mathbb{N}$; $A_1, \ldots, A_m$ are matrices of the same dimensions over some ring G; $\vec{v}, \vec{v}_1, \ldots, \vec{v}_m$ are vectors of the same length over G; c', c'' are a pair of values from the same domain (e.g., from $\mathbb{Z}_N$), and $\vec{c}', \vec{c}''$ denote their binary representation. In the "vector multiplication with secret scalars" operation, $c_1 \cdot \ldots \cdot c_k$ multiplies each coordinate of $\vec{v}$. In the "vector addition/subtraction (binary representation)" operations, 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 5.7), but there is no guarantee on the output when the entries of $\vec{v}, \vec{r}$ are not bits. The "scalar greater/smaller than" operations are computed using standard Boolean circuits for comparing a pair of binary strings.

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. In the end of the section we also describe an attack showing that SIR is susceptible to inversion attacks.

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 4.1) 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

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

1. **Setup:** The parties execute the setup protocol of Figure 4 to obtain keys $(\mathsf{pk}_N, \mathsf{sk}_N)$ and $(\mathsf{pk}_D, \mathsf{sk}_D)$. Then, the parties execute the data uploading, merging and permuting protocols of Figures 5-6, and $\mathcal{S}_1$ obtains encryptions of the aggregated input matrix and response vector $A, \vec{b}$.
2. Let $F = [d]$ (i.e., initially F contains all features), and let $A_F = A, \vec{b}_F = \vec{b}$.
3. While $|F| > s$, do:
   (a) **Ridge Regression Iteration:** $\mathcal{S}_1$ and $\mathcal{S}_2$ execute the scaled ridge protocol of Figure 3, where the input of $\mathcal{S}_1$ consists of F, encryptions $\mathbf{A}_F, \vec{\mathbf{b}}_F$ of $A_F, \vec{b}_F$, respectively, and the input of $\mathcal{S}_2$ is F, sk_N. The output of $\mathcal{S}_1$ is an entry-wise encryption $\vec{\mathbf{z}}_F$ of a vector $\vec{z}_F \in \mathbb{Z}_N^d$ under key pk_N.
   (b) **Selecting features:**
   - **Large-set case:** If $|F| > \mathsf{thr}$ then set $k' = \lfloor \mathsf{rej} \cdot |F| \rfloor$. (Intuitively, when $|F| > \mathsf{thr}$ the set of current features is still sufficiently large that we can remove a subset of features in each iteration.)
   - **Small-set case:** Otherwise (i.e., $|F| \leq \mathsf{thr}$), set $k' = 1$. (In this case, the set of current features is small, so features should be removed one at a time.)
   - $\mathcal{S}_1$ and $\mathcal{S}_2$ execute the selection protocol of Figure 9 to find the smallest k' features. $\mathcal{S}_1$ has input $\vec{\mathbf{z}}_F$, $\mathcal{S}_2$ has input sk_N, and both parties have input $\mathsf{pk}_N, \mathsf{pk}_D, F, k'$. The output of both servers is the set S_{del} consisting of the k' features to be removed.

   (c) **Compacting data for the next iteration:** Let $F^* = F \setminus S_{\mathsf{del}}$. Then $\mathcal{S}_1$ locally executes the compacting algorithm of Figure 7, 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 8 to obtain the s-sparse model $\vec{w}$.

Figure 2: SIR: Secure Iterated Ridge

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

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

1. **Masks Generation:** $\mathcal{S}_1$ picks a random invertible matrix $R' \leftarrow GL(|F|, \mathbb{Z}_N)$, 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 $\mathsf{adj}(\Gamma)$ and $\Delta = \det(\Gamma)$, as well as $\vec{\zeta} = \mathsf{adj}(\Gamma) \cdot \vec{\beta}$. 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 $\vec{z} = \det\left(A'_F\right) \cdot \vec{w}_F$, 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 $\vec{z} = \det\left(A'_F\right) \cdot \vec{w}_F$, because $\Gamma = A'_F \cdot R'$ so $\det(\Gamma) = \det(R') \cdot \det\left(A'_F\right)$.)
6. $\mathcal{S}_1$ sets $\vec{\mathbf{z}}$ to be its output for the phase.

Figure 3: Scaled ridge regression (single iteration).

The algorithm is executed locally by $\mathcal{S}_1$, using the circuits Add_λ, Add, $\mathsf{MatMultL}_M$, $\mathsf{MatMultR}_M$, MatMult and MatVecMult of Section 5.1.
Public parameters: $\mathcal{E}$, 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

The algorithm is executed locally by $\mathcal{S}_1$ and uses the circuits $\mathsf{MatMultR}_M$ and $\mathsf{MatVecMultL}_M$ of Section 5.1.
Public parameter: $\mathcal{E}$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:
   - $\mathbf{T} \leftarrow \mathsf{Eval}\left(\mathsf{pk}_N, \mathsf{MatMultR}_{\mathcal{N}_{F^*}}, \mathbf{A}\right)$.(Notice that $\mathbf{T}$ encrypts $A \cdot \mathcal{N}_{F^*}$.)
   - $\mathbf{A}_{F^*} \leftarrow \mathsf{Eval}\left(\mathsf{pk}_N, \mathsf{MatMultL}_{\mathcal{N}_{F^*}}, \mathbf{T}\right)$.
   - $\vec{\mathbf{b}}_{F^*} \leftarrow \mathsf{Eval}\left(\mathsf{pk}_N, \mathsf{MatVecMultL}_{\mathcal{N}_{F^*}}, \vec{\mathbf{b}}\right)$.

   (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 7: Compacting current data.

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

The protocol uses the circuits ScalerVecMult and MatVecMult of Section 5.1.
Public parameter: $\mathcal{E}$, 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 $A'_F = \mathsf{pjct}_F(A_F) \in \mathbb{Z}_N^{|F|\times|F|}$ 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 [23, 65], and sends $\vec{w}$ to all parties.

Figure 8: Computing and unmasking the output

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 protocol uses the circuits Square, SubNums, $\mathsf{AddConst}_c$, $\mathsf{BinCompareR}_c$, $\mathsf{BinCompareL}_c$, Neg, MultNums and AddNums of Section 5.1. 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 $\left\{\mathbf{df}_{i,j}\right\}_{i,j \in F, i \ne j}$ 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 $i \ne j$, 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 $i, j \in F, i \ne j$:
- 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 $\left\{\overrightarrow{\mathbf{rangeMin}}_{i,j}, \overrightarrow{\mathbf{rangeMax}}_{i,j}, \mathbf{Negative}_{i,j}\right\}_{i,j \in F, i \ne j}$ 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 $j \ne i$ such that $z_j \le z_i$.)
- For every $i, j \in F, i \ne j$, 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) $\boldsymbol{t}^1_{i,j} \leftarrow \mathsf{Eval}\left(\mathsf{pk}_D, \mathsf{MultNums}, \mathbf{IsBigger}_{i,j}, \mathbf{IsSmaller}_{i,j}\right)$. (Then $t^1_{i,j} = \mathsf{IsBigger}_{i,j} \cdot \mathsf{IsSmaller}_{i,j}$.)
    (c) $\boldsymbol{t}^2_{i,j} \leftarrow \mathsf{Eval}\left(\mathsf{pk}_D, \mathsf{MultNums}, \boldsymbol{t}^1_{i,j}, \mathbf{Negative}_{i,j}\right)$. (Then $t^2_{i,j} = \mathsf{IsBigger}_{i,j} \cdot \mathsf{IsSmaller}_{i,j} \cdot \mathsf{Negative}_{i,j}$.)
    (d) $\boldsymbol{t}^3_{i,j} \leftarrow \mathsf{Eval}\left(\mathsf{pk}_D, \mathsf{MultNums}, \mathbf{notInRange}_{i,j}, \mathbf{NotNegative}_{i,j}\right)$. (Then $t^3_{i,j} = \left(1 - \mathsf{IsBigger}_{i,j} \cdot \mathsf{IsSmaller}_{i,j}\right) \cdot \left(1 - \mathsf{Negative}_{i,j}\right)$.)
    (e) $\boldsymbol{t}^4_{i,j} \leftarrow \mathsf{Eval}\left(\mathsf{pk}_D, \mathsf{AddNums}, \boldsymbol{t}^2_{i,j}, \boldsymbol{t}^3_{i,j}\right)$. (Then $t^4_{i,j} = \mathsf{ost}_{i,j}$.)
    (f) $\mathbf{Ord}_i \leftarrow \mathsf{Eval}\left(\mathsf{pk}_D, \mathsf{AddNums}, \mathbf{Ord}_i, \boldsymbol{t}^4_{i,j}\right)$. (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 $\overrightarrow{\mathbf{Ord}}^{\Pi}_i = \mathbf{Ord}_{\Pi(i)}$ 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 $\overrightarrow{\mathsf{Ord}}^{\Pi}_i = \mathsf{Dec}\left(\mathsf{sk}_D, \overrightarrow{\mathbf{Ord}}^{\Pi}_i\right)$, and generates a vector $\chi^{\Pi} \in \{0,1\}^d$ by setting $\chi^{\Pi}_i = 1$ if $\overrightarrow{\mathsf{Ord}}^{\Pi}_i < k$, otherwise setting $\chi^{\Pi}_i = 0$. $\mathcal{S}_2$ sends $\vec{\chi}^{\Pi} := \left(\chi^{\Pi}_1, \ldots, \chi^{\Pi}_{|F|}\right)$ to $\mathcal{S}_1$.

(6) **Output:** S_1 computes $S_{\mathsf{del}} = \left\{i : \vec{\chi}^{\Pi}_{\Pi(i)} = 1\right\}$ and sends it to $\mathcal{S}_2$. Both servers set S_{del} to be their output for the phase.

Figure 9: Selecting which features to remove in the current iteration.

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[14]

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

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. 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

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

[14]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 $|F| \le \text{thr}$ since the number of features is small.

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:

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

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,[15] which will also be the last iteration since we are looking for a 3-sparse solution. The final model will be

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

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 8). 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:

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

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

[15]We note that though the computation is performed over the binary representation of 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.

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:

$$\begin{aligned} \vec{z}^{(1)} &= \operatorname{adj}\left(A^{(1)}\right) \cdot \vec{b}^{(1)} \\ &= \begin{pmatrix} 320 & -128 & -256 & 64 \\ -128 & 128 & 64 & -64 \\ -256 & 64 & 320 & -128 \\ 64 & -64 & -128 & 320 \end{pmatrix} \cdot \begin{pmatrix} 128 \\ 116 \\ 100 \\ 40 \end{pmatrix} \\ &= 64\operatorname{adj}\left(A^{(0)}\right) \cdot 4\vec{b}^{(0)} = 256\vec{z}^{(0)} = \begin{pmatrix} 3072 \\ 2304 \\ 1536 \\ 768 \end{pmatrix} \end{aligned} \tag{6}$$

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, $\det\left(A^{(0)}\right) = 3 \neq 768 = \det\left(A^{(1)}\right)$.

SIR is susceptible to inversion attacks. We conclude this section by showing that SIR is susceptible to inversion attacks. More specifically, we show that the output model may be used to infer non trivial information on the inputs of parties, even when *all parties follow the protocol* (i.e., are semi-honest). In particular, such attacks might be exploited by corrupted parties, who may choose to use different inputs (but otherwise follow the protocol) in order to learn non-trivial information about the inputs of the honest parties.

The high-level idea of the attack is for corrupted parties to use "balanced" inputs, in which all features have the same correlation with the response vector. Thus, the inputs provided by corrupted parties will not affect the output model, and any correlation in the output model will then point to correlations in the inputs of honest parties. A concrete example follows.

We consider a setting with two data owners $\mathsf{DO}_{\mathsf{Alice}}$ and $\mathsf{DO}_{\mathsf{Bob}}$, each holding 3 data records with 3 features. $\mathsf{DO}_{\mathsf{Bob}}$ will be corrupt, and will try to infer information about the inputs of $\mathsf{DO}_{\mathsf{Alice}}$. We note that unlike the previous attacks of the section, we cannot directly consider the final data matrix X (on which SIR will be executed), but rather we define the data matrices of $\mathsf{DO}_{\mathsf{Alice}}$, $\mathsf{DO}_{\mathsf{Bob}}$ and use them to define X and the response vector. (The reason for this difference is that previous attacks considered only a corrupted server, so leaking information about the inputs of *any* of the data owners was problematic; whereas here we are trying to leak on the input of a *specific* data owner.)

The Example. Consider the following data matrix and response vector for $\mathsf{DO}_{\mathsf{Bob}}$:

$$X^{\mathsf{Bob}} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad \vec{y}^{\mathsf{Bob}} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

and two possible inputs for $\mathsf{DO}_{\mathsf{Alice}}$:

$$X^{\mathsf{Alice},0} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad \vec{y}^{\mathsf{Alice},0} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

and

$$X^{\mathsf{Alice},1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad \vec{y}^{\mathsf{Alice},1} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

Consider executing sparse ridge regression with $\lambda = 0$, where we are looking for a 1-sparse solution. Then

$$A^{\mathsf{Bob}} := \left(X^{\mathsf{Bob}}\right)^T \cdot X^{\mathsf{Bob}} = I$$

(here, I is the unit matrix), and

$$\vec{b}^{\mathsf{Bob}} := \left(X^{\mathsf{Bob}}\right)^T \cdot \vec{y}^{\mathsf{Bob}} = \vec{y}^{\mathsf{Bob}}.$$

In the first execution, when $\mathsf{DO}_{\mathsf{Alice}}$'s input is $X^{\mathsf{Alice},0}, \vec{y}^{\mathsf{Alice},0}$ we have

$$A^{\mathsf{Alice},0} := \left(X^{\mathsf{Alice},0}\right)^T \cdot X^{\mathsf{Alice},0} = I$$

and

$$\vec{b}^{\mathsf{Alice},0} := \left(X^{\mathsf{Alice},0}\right)^T \cdot \vec{y}^{\mathsf{Alice},0} = y^{\mathsf{Alice},0}$$

whereas in the second execution, when $\mathsf{DO}_{\mathsf{Alice}}$'s input is $X^{\mathsf{Alice},1}, \vec{y}^{\mathsf{Alice},1}$ we have

$$A^{\mathsf{Alice},1} := \left(X^{\mathsf{Alice},1}\right)^T \cdot X^{\mathsf{Alice},1} = I$$

and

$$\vec{b}^{\mathsf{Alice},0} := \left(X^{\mathsf{Alice},1}\right)^T \cdot \vec{y}^{\mathsf{Alice},1} = y^{\mathsf{Alice},1}$$

These are then merged into a single matrix and response vector. The merged matrices are $A^0 := A^{\mathsf{Alice},0} + A^{\mathsf{Bob}} = 2I$ in the first case, and $A^1 := A^{\mathsf{Alice},1} + A^{\mathsf{Bob}} = 2I$ in the second case. As for the response vectors, these are

$$\vec{b}^0 := \vec{b}^{\mathsf{Alice},0} + \vec{b}^{\mathsf{Bob}} = \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix}$$

in the first case, and

$$\vec{b}^1 := \vec{b}^{\mathsf{Alice},1} + \vec{b}^{\mathsf{Bob}} = \begin{pmatrix} 1 \\ 1 \\ 2 \end{pmatrix}$$

in the second case. The solutions $\vec{w}^0, \vec{w}^1$ to the linear systems $A^0\vec{w}^0 = \vec{b}^0$ and $A^1\vec{w}^1 = \vec{b}^1$, respectively, are

$$\vec{w}^0 = \begin{pmatrix} 1 \\ 1/2 \\ 1/2 \end{pmatrix}, \quad \vec{w}^1 = \begin{pmatrix} 1/2 \\ 1/2 \\ 1 \end{pmatrix}$$

Therefore, the output sparse models would be

$$\vec{w}^{0\prime} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad \vec{w}^{1\prime} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

respectively, which reveals which feature in the input of $\mathsf{DO}_{\mathsf{Alice}}$ is most correlated with the response vector.

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 $1 \leq j \leq k$, $\mathrm{pjct}_{F_j}(A)$ 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 [4, 29]). 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 4, 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. [29], 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 [29] (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 $d \leq n \in \mathbb{N}$. 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 Iterative Ridge of Figure 1, except with negligible probability $negl(\kappa)$.*

Proof. We will prove a stronger claim: we will prove that if the FHE scheme has *perfect* correctness, then the outputs of SIR and Iterative 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 Iterative Ridge algorithm (Figure 1) in the following points:[16]

(1) **Correctness of computing over a finite ring:** Iterative 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 Iterative 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. Iterative 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 Iterative 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 Iterative 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 Iterative Ridge it is computed directly on the ordering.
(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,

[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 Iterative 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).

whereas in Iterative 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 Iterative 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 overflows will occur (see Remark 5.3), 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 [29] (and holds here because N is sufficiently large, see Remark 5.3).

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 8), 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 $\det(A) \cdot \operatorname{adj}(A)$ 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 Iterative Ridge). Referring to Figure 3, the (scaled, encrypted) model which $\mathcal{S}_2$ sends to $\mathcal{S}_1$ is $\vec{\zeta}_F = \operatorname{expd}_F\left(\operatorname{adj}(\Gamma) \cdot \vec{\beta}\right)$ (see Remark 5.5 for a description of $\operatorname{expd}_F(\cdot)$). $\mathcal{S}_1$ then removes the masking by computing

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

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

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

and

$$\begin{aligned}&\forall F \subseteq [d], \forall B \in \mathbb{Z}_N^{d\times d}, \forall \vec{v} \in \mathbb{Z}_N^{|F|}\ :\\&\operatorname{pjct}_F\left(\mathcal{N}_F B \mathcal{N}_F \cdot \operatorname{expd}_F(\vec{v})\right) = \operatorname{pjct}_F(B) \cdot \vec{v}\end{aligned} \tag{8}$$

Using Equation 7, we have:

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

where the equality denoted by (1) follows from the definition of A and from the fact that $R = \operatorname{expd}_F(R')$ (so $R = \mathcal{N}_F R \mathcal{N}_F$), the equality denoted by (2) used Equation 7, and the rightmost equality follows from the definitions of A'_F and R' (see Figure 3).

Therefore,

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

Next, we observe that:

$$\begin{aligned}&\forall F \subseteq [d], \forall \vec{v}, \vec{u} \in \mathbb{Z}_N^{d}\ :\\&\operatorname{pjct}_F(\vec{v} + \vec{u}) = \operatorname{pjct}_F(\vec{v}) + \operatorname{pjct}_F(\vec{u})\end{aligned} \tag{9}$$

which implies that

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

where (1) is by the definition of $\vec{\beta}$, (2) is by the definition of $\vec{\beta}_F$, (3) follows from Equation 9, (4) follows from Equation 8 because $A_F = \mathcal{N}_F A \mathcal{N}_F$ and $\vec{r} = \operatorname{expd}_F(\vec{r}')$, and (5) follows from the definition of A'_F.

Putting the two together, we have

$$\begin{aligned}\vec{\zeta} &= \operatorname{adj}(\Gamma) \cdot \vec{\beta}\\&= \det(\Gamma) \cdot \left(R'\right)^{-1} \cdot \left(A'_F\right)^{-1} \cdot \left[\operatorname{pjct}_F\left(\vec{b}_F\right) + A'_F \cdot \vec{r}'\right]\\&= \det(\Gamma) \cdot \left(R'\right)^{-1} \cdot \left[\vec{w}'_F + \vec{r}'\right].\end{aligned} \tag{10}$$

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

Next, we notice that

$$\begin{aligned}&\forall F \subseteq [d], \forall \vec{v}, \vec{u} \in \mathbb{Z}_N^{|F|}\ :\\&\operatorname{expd}_F(\vec{v} + \vec{u}) = \operatorname{expd}_F(\vec{v}) + \operatorname{expd}_F(\vec{u})\end{aligned} \tag{11}$$

Therefore,

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

where (1) follows from the definition of ζ, (2) follows from Equation 10, and (3) follows from Equation 11.

Next, notice that

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

which implies that

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

where (1) was established above, (2) follows from Equation 12, and (3) follows from the definition of $\vec{w}'_F$ and $\vec{r}$.

Moreover, notice that

$$\begin{aligned}&\forall F\subseteq[d],\forall B,C\in\mathbb{Z}_N^{|F|\times|F|}\ :\\ &\mathsf{expd}_F(B\cdot C)=\mathsf{expd}_F(B)\cdot\mathsf{expd}_F(C)\end{aligned}\tag{13}$$

in particular, this implies that

$$\begin{aligned}R\cdot\mathsf{expd}_F\left((R')^{-1}\right) &=^{(1)} \mathsf{expd}_F(R')\cdot\mathsf{expd}_F\left((R')^{-1}\right)\\ &=^{(2)} \mathsf{expd}_F\left(R'\cdot(R')^{-1}\right)\\ &= \mathsf{expd}_F\left(I_{|F|}\right)\end{aligned}\tag{14}$$

where (1) follows from the definition of R, (2) follows from Equation 13, and $I_{|F|}$ denotes the $|F|\times|F|$ identity matrix. Consequently, we have that

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

where (1) follows from substituting $\vec{\zeta}_F$ with the value computed above, (2) follows from Equation 14, and (3) follows from the fact that $\vec{w}_F,\vec{r}$ have 0 in every index $i\notin F$.

Finally, notice that

$$\det(\Gamma)=\det\left(A'_F\cdot R'\right)=\det\left(R'\right)\cdot\det\left(A'_F\right)\tag{15}$$

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

Putting it together, we have that

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

where (1) follows by substituting $R\cdot\vec{\zeta}_F$ with the value computed above, and (2) follows from Equation 15. 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 9). 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 Iterative 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 $0<x\le(N-1)/2$ as "positive", and values $(N-1)/2<x<N$ as "negative". We will show that $\mathsf{Ord}_i=\left|\left\{z_j\ :\ |z_j|<|z_i|\right\}\right|$. 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 $\mathsf{Ord}_i=\left|\left\{j\ :\ z_i^2-z_j^2>0\right\}\right|$. 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 $\alpha_{i,j}\neq0$), 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 9 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 9). 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 Iterative Ridge). Indeed, as described in the analysis of Item 3 above, the scalde ridge phase outputs a scaled model $\vec{z}_F = \det\left(A'_F\right) \cdot \vec{w}_F$ (where $\vec{w}_F$ is a permuted version of the actual output model). $\mathcal{S}_1$ additionally obtains an encryption of

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

(where (1) follows from Equation 15) from which $\mathcal{S}_1$ computes $\det\left(A'_F\right)^{-1}$ (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 [23, 65]).

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

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

where (1) follows from Equation 12. Moreover, by the definition of A and $\vec{b}$ we have:

$$A = P^T\left(X^TX+\lambda I\right)P = P^TX^TXP + P^T\cdot\lambda I\cdot P$$
$$=^{(1)} (XP)^T\cdot XP + \lambda I$$

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

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

Therefore, performing the learning over $A, \vec{b}$ (which are permuted versions of $X^TX+\lambda I$ 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., κ, λ, N, D, $n_1,\ldots,n_m$, d, s, ℓ, rej, 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 $\left(\mathsf{pk}_N, \mathsf{sk}_N\right), \left(\mathsf{pk}_D, \mathsf{sk}_D\right)$, and random encryptions $\mathbf{P}_2 \leftarrow \mathsf{Enc}\left(\mathsf{pk}_N, 0_d\right)$ and $\mathbf{P}_2^T \leftarrow \mathsf{Enc}\left(\mathsf{pk}_N, 0_d\right)$, where 0_d denotes the $d\times d$ all-zeros matrix. Additionally, for every honest DO_j, it generates a random encryption $\mathbf{A}^j \leftarrow \mathsf{Enc}\left(\mathsf{pk}_N, 0_d\right)$, 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 9) 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 $\cup_{t=1}^{l-1} S^t_{\mathsf{del}}$. 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 5.6. 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 $\cup_{t=1}^{l-1} S^t_{\mathsf{del}}$, 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 [29], that the masking perfectly hides the data matrix and response vectors (see Lemma 8.7 below). We note that Giacomelli et al. [29] 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 [29, Lemma 1].

Lemma 8.7 (Lemma 1 from [29]). *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 12, 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 11 implies that $\vec{v} = \mathsf{expd}_F\left(\vec{b} + A' \cdot \vec{r}'\right)$. Second, notice that by Equation 13, 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 9): for every $i, j \in F_{l-1}, i \neq j$, generates a random $\mathsf{df}^l_{i,j} \leftarrow \mathbb{Z}_N$, and encrypts $\mathbf{df}^l_{i,j} \leftarrow \mathsf{Enc}\left(\mathsf{pk}_N, \mathsf{df}^l_{i,j}\right)$. 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^l_{del} 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 $S^l_{\mathsf{del}} \subseteq F_{l-1}$ of size $\left|S^l_{\mathsf{del}}\right| = v_l$. Then, it sets $F_l = F_{l-1} \setminus S^l_{\mathsf{del}}$, and $f_l = f_{l-1} - v_l$.
(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 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.

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

$\mathcal{H}_0$**:** 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 9) 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 9 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 $\vec{b}^l := \mathsf{expd}_F(\vec{v})$ and $\Gamma^l = \mathsf{expd}_F(M)$.
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 5.6). 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.

□

Remark 8.9. In SIR, the output model is revealed to all parties. As discussed in Section 2.5, this involves some privacy risks, which may be alleviated by revealing the output model only to some of the parties. We now explain why SIR remains secure in this setting. Consider first the variant in which $\mathcal{S}_2$ does not reveal the output model to any party. This variant is still secure against an adversary corrupting $\mathcal{S}_1$ and a strict subset of data owners. Indeed, the simulator in the proof of Lemma 8.5 does not use the output model in the simulation, so the same proof works also when $\mathcal{S}_1$ does not learn the output model. Next, consider the variant in which only an external output party O learns the output model. The difference from the previous variant is that now $\mathcal{S}_2$ also doesn't learn the model. We claim that SIR is still secure against an adversary corrupting $\mathcal{S}_2$ and a strict subset of data owners. Indeed, in the proof of Lemma 8.6, the simulator uses the output model only to generate the final message sent from $\mathcal{S}_1$ to $\mathcal{S}_2$ in Step 2 of Figure 8. The proof can be adjusted to the variant in which $\mathcal{S}_2$ does not learn the output model as follows. First, the simulator does not receive the output model as output, and is not required to simulate the final message from $\mathcal{S}_1$ to $\mathcal{S}_2$ (this message does not exist in this variant). Second, hybrid $\mathcal{H}_6$ is not needed in the proof, since $\mathcal{H}_5$ is already exactly the simulated view. Finally, the view of the external party O can be simulated exactly as the integral model $\vec{w}^{\mathsf{int}}$ is generated from the rationally-reconstrcted output model in the proof of Lemma 8.6.

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 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*
$$O\left(d \cdot (d+\mathsf{thr}^2)\cdot\lceil\frac{d}{\mathsf{sl}}\rceil\cdot\left(\ell + \log(n^2+\lambda) + \log d\right)\right)$$
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 [4], 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 [2, 42]) 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 [4].)

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 [4, 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 9), 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 9) 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

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

using the formula for an infinite geometric sum, we get

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

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 that $\log N = O\left(d(\ell+\log(n^2+\lambda)+\log d)\right)$. 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. □

10 SYSTEM AND EMPIRICAL EVALUATION

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

10.1 Data

The Cancer Genome Atlas (TCGA), a landmark cancer genomics program, molecularly characterized over 20,000 primary cancer and matched normal human tissue samples spanning 33 cancer types. The program integrates contributions from many researchers coming from diverse disciplines and from multiple institutions. The data spans genomic, epigenomic, transcriptomic, and proteomic data measured on the aforementioned samples. We used a small portion of this data for our experiments. Concretely we used randomly selected portions of one of the breast cancer transcriptomics data matrices. We start with a matrix with features normalized to lie in $[-1,1]$ with 3-digit precision. The matrix has 781 rows (samples/instances) and $> 10K$ columns (features, representing genes profiled). Each (i,j) entry of the matrix represents the expression level of gene j in sample i. We use $\mathcal{D}$ to denote this full TCGA breast cancer expression profiling matrix. The TCGA data also includes 781 TIL levels for this cohort, as part of additional data to support biological and clinical interpretation. TIL levels quantify tumor infiltrating immune cells in the (tumor) samples.

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

10.2 Comparing IR to Filter, Ridge and Lasso, on Our Example Dataset

To quantify the advantage of iterative ridge (IR) over the filter, ridge, Lasso, and truncated ridge (truncate) approaches (the baseline algorithms) we ran experiments in the clear. Data was generated as detailed in Section 10.1. We generated 1000 data sets, each consisting of $d = 100$ features selected uniformly at random. The 781 samples are 80/20 split into training and test. For each of the 1000 datasets we ran IR, filter, ridge, Lasso and truncated ridge on the training data. IR selects 10 features as specified in Figure 1. Filter selects the top 10 features according to their Pearson correlation to the target vector (on the training data); ridge regression was executed with the regularization parameter set to its default value $\lambda = 1$ while in Lasso, we set it to get the same number of non-zero feature as in IR; truncate executes ridge regression and then removes features with low weight to maintain the 10 features with the highest weights. In each one of these cases we then obtain the actual model (coefficients) and apply it to the test data. In Figure 10 we compare the resulting MSEs for IR to the other approaches. The histogram represents the observed results for the percentage difference:

$$\Delta = \frac{\mathsf{MSE}(\text{baseline}) - \mathsf{MSE}(\mathsf{IR})}{\mathsf{MSE}(\mathsf{IR})}\cdot 100 \tag{16}$$

We clearly observe a significant performance advantage for IR in our example gene expression dataset. Concretely, IR outperforms (non-negative Δ) filter, ridge, Lasso and truncate in 100.0%, 77.4%, 93.9% and 100.0% of the experiments, respectively. The mean percentage difference is 27.23%, 16.43%, 27.06% and 17.08%, respectively. We repeated the experiment, selecting 8 out of $d = 40$ features, showing similar results: IR outperforms filter, ridge, Lasso and truncate in 100.0%, 79.5%, 93.5% and 100.0% of the experiments, respectively.

10.3 System Implementation Details

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

Permuting the input. In a preliminary step $\mathcal{S}_1$ and $\mathcal{S}_2$ participate in a protocol to permute the input: $A = (P_1 \cdot P_2)^T \cdot T^1 \cdot (P_1 \cdot P_2)$ (for T^1 an intermediate value, see Figure 6), where P_1 and P_2 are random permutation matrices chosen by $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively. There are several ways to compute this step. One option is for $\mathcal{S}_1$ to perform all the computations using FHE (with multiplicative depth 2). Another option – which is the one used in our implementation – is to utilize a protocol between $\mathcal{S}_1$ and $\mathcal{S}_2$ that uses masking and requires only additive homomorphism. This step is executed only once at the onset of the protocol, and its running time is inconsequential compared to the total protocol runtime (e.g., with 40 features, this step took 1,152 seconds out of 84,690 seconds. See Table 2).

Ranking weights. To decide which entries of $\vec{w}$ to remove in each iteration, we compute a full ranking over the entries (see Figure 9). This involves comparing pairs of entries. Our experiments show that the majority of the running time is spent in this step. For example, the total real time of SIR when $d = 40$ was 84,690 seconds, while computing the ranks took 76,184 seconds (see Table 2). To Compare a single pair w_i, w_j of entries, $\mathcal{S}_1$ compares $w_i^2 - w_j^2 + r_{ij}$ to ranges received from $\mathcal{S}_2$, to determine whether $w_i^2 - w_j^2 \in [0, N/2]$ (and hence $w_i \geq w_j$). In our implementation $\mathcal{S}_1$ performed all $\binom{d}{2}$ comparisons. An alternative is to use an approach similar to Bitonic sort (or Batcher sort, see [5]). However, while a sorting approach performs only $\frac{d}{2}\binom{\lceil \log_2 d \rceil}{2}$ comparisons, it is less amenable to parallelization. Specifically, the sorting approach is faster when $\lceil \frac{1}{\text{CPUs}}\binom{d}{2} \rceil > \lceil \frac{d/2}{\text{CPUs}} \rceil \cdot \binom{\lceil \log_2 d \rceil}{2}$, so for the number of features and CPUs in our experiments the naive all-pairs approach was faster. We stress that to utilize the Bitonic sort alternative (which would be faster for larger d's), one only needs to implement the comparison step using Bitonic sort, making our protocol flexible, efficiently supporting both regimes of d.

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

- *Ranking steps.* Computing the ranks of features in $\vec{w}$, when the entries are given in binary. These steps involve a 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 *additive* homomorphism, but necessitate multiple (co-prime) plaintext moduli to implement the CRT. Preferably, these plaintext moduli should be as large as possible, because larger moduli mean less elements in the CRT encoding.

We used two different FHE libraries (with different schemes) to implement these two types of steps. For ranking we use BGV in HElib [38] 2.1.0, because it supports bootstrapping (unlike SEAL). For CRT steps we use B/FV in SEAL [55] 4.0, in which it is easier to configure keys for multiple plaintext moduli. Switching between the two libraries and schemes is done by $\mathcal{S}_2$, who generated the keys (pk_N, sk_N) and (pk_D, sk_D) in HElib and SEAL respectively. In detail, in HElib [38] 2.1.0 we initialize the keys while setting the plaintext modulo to 17^2, and the cyclotomic polynomial degree to 78,881. This resulted in ciphertexts with 7,000 slots, supporting a multiplicative depth of 28, as well as bootstrapping. In SEAL we initialize the keys while setting the cyclotomic polynomial degree to 8,192, a chain length of 7, chain bits of 25 and prime bits of 30. This resulted in a ciphertext with 4,096 slots that supports additive homomorphism.

10.4 Evaluation Setup

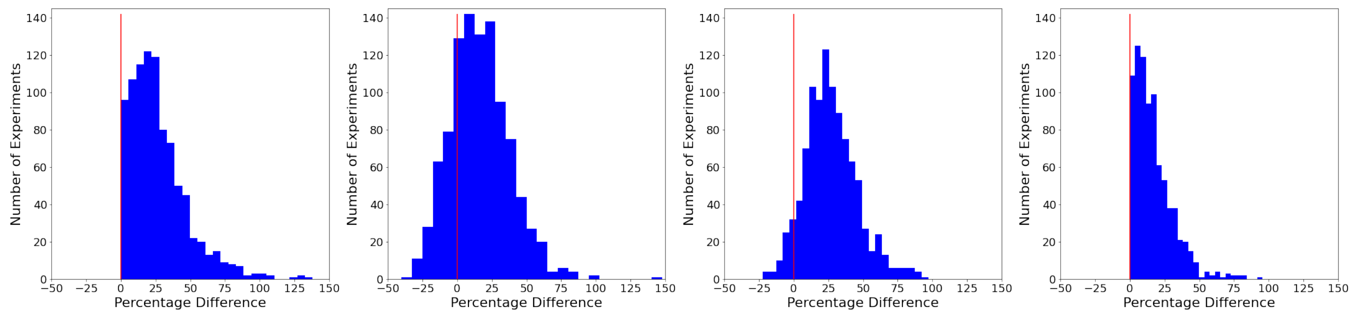
We executed SIR (Protocol 2) on the data generated from TCGA described in Section 10.1, and measured performance of the data owners and servers. We executed two types of experiments: end-to-end and single-iteration experiments. In all experiments rej was set to 10% and the ring size N was determined by Equation 4.

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

Single-iteration experiments measure performance when executing a single iteration (Protocol 2, Step 3, i.e., computing scaled ridge, ranking the features, and removing the features with the smallest weight) on different values of d. Concretely, we take $5 \leq d \leq 40$ features, in increments of 5.

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

What we measured. We report the total runtime, as well as the runtime of each sub-task in SIR: *encrypt* (Figure 5, Step 2); *permute* (Figure 6, Step 3); *mask* (Figure 3, Step 2); *unmask* (Figure 3, Step 5);



(a) IR vs Filter **(b) IR vs Ridge** **(c) IR vs Lasso** **(d) IR vs Truncate**

Figure 10: IR outperforms filter, ridge, Lasso and truncate in terms of the resulting MSE on the test data, for our example datasets (1000 random test sets generated from the breast cancer gene expression data matrix from TCGA [60]). The histograms depict the number of experiments (y-axis) for which we observe a given value for the percentage difference Δ of Eq 16 (x-axis).

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

Table 2: Runtime (seconds), memory consumption (GB), and parallelization ratio (in parenthesis) in end-to-end SIR executions.

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

Table 3: Runtime (seconds) in a single SIR iteration, and parallelization ratio (in parenthesis).

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

Table 4: Runtime (seconds) of SIR on a growing number of records (n).

decrypt (Figure 3, Step 3);[18] *solve* (Figure 3, Step 4);[19] *pair diff* (Figure 9, Step 1); *ranges* (Figure 9, Step 2); *ranks* (Figure 9, Step 3). Furthermore, we report the *parallelization ratio*, which is the ratio between runtime when executing the computation on a single CPU (as measured by the operating system) vs. the runtime on our 100-cores system. Intuitively, this indicates the average number of CPUs that were busy performing the task, where a higher ratio means the task is more amenable to parallelization as it utilize more CPUs (the maximum being 100).

10.5 SIR Performance

Tables 2-3 summarize the performance exhibited in the end-to-end experiments and the single-iteration experiments, respectively. Table 4 presents SIR runtime on various database sizes.

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

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

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

needed for our encodings (for small d's), so the total number of ciphertexts grew only mildly in d.

Parallelization. As d grows, most tasks become more parallelizeable. In particular, the ranking task –which dominates the bulk of the runtime– is "embarrassingly" parallelizable since we make all $\binom{d}{2}$ comparisons, which can be executed in parallel. We note that if we had an *unlimited number of CPUs*, the ranking runtime would essentially be the time of a single comparison. For example, when $(d, N) = (40, 2^{1260})$ (cf. Table 3 bottom row), our ranking utilized an average of 91 CPUs (the system had 100 CPUs), whereas having access to $\binom{d}{2}$ CPUs is expected to improved the ranking runtime by a factor of $\binom{40}{2}/91 \approx 8.6$, thus improving the total runtime by 6×.

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

11 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. Our protocol enjoys rigorous security, and scales favorably with the number of records and data owners. Moreover, our protocol naturally gives a privacy-preserving ridge truncation protocol, which is less accurate (as we have shown), but simpler and faster, and therefore may be preferred in some cases.

The design of our protocol is based, amongst other consideration, on certain potential attacks that can be developed when partial or intermediate information is leaked. In particular, we show in Section 7 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. As explained in Sections 2.5 and 7, SIR is susceptible to inversion attacks. Determining the exact extent to which such attacks are harmful, and devising measures to protect against them, are left for future work.

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

ACKNOWLEDGMENTS

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

REFERENCES

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

[2] Ehud Aharoni, Allon Adir, Moran Baruch, Nir Drucker, Gilad Ezov, Ariel Farkash, Lev Greenberg, Ramy Masalha, Guy Moshkowich, Dov Murik, Hayim Shaul, and Omri Soceanu. 2023. HeLayers: A Tile Tensors Framework for Large Neural Networks on Encrypted Data. *Proc. Priv. Enhancing Technol. To appear.* 2023 (2023). https://arxiv.org/abs/2011.01805

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

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

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

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

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

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

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

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

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

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

[13] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2012. (Leveled) fully homomorphic encryption without bootstrapping. In *ITCS'12*. ACM, 309–325.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[42] Xiaoqian Jiang, Miran Kim, Kristin Lauter, and Yongsoo Song. 2018. Secure Outsourced Matrix Computation and Application to Neural Networks. In *CCS'18.* 1209–1222.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[68] Yuncheng Wu, Shaofeng Cai, Xiaokui Xiao, Gang Chen, and Beng Chin Ooi. 2020. Privacy preserving vertical federated learning for tree-based models. *arXiv preprint arXiv:2008.06170* (2020).

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

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

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