Information Sciences 451–452 (2018) 34–49

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Privacy-preserving ridge regression on distributed data

Yi-Ruei Chen*, Amir Rezapour, Wen-Guey Tzeng

National Chiao Tung University, Hsinchu 30010, Taiwan

ARTICLE INFO

Article history:
Received 15 August 2016
Revised 28 March 2018
Accepted 30 March 2018
Available online 30 March 2018

Keywords:
Privacy-preserving regression
Ridge regression
Data privacy
Recommendation system

ABSTRACT

Ridge regression is a statistical method for modeling a linear relationship between a dependent variable and some explanatory values. It is a building-block that plays a major role in many learning algorithms such as recommendation systems. However, in many applications such as e-health, explanatory values contains private information owned by different patients that are not willing to share them, unless data privacy is guaranteed. In this paper, we propose a protocol for conducting privacy-preserving ridge regression (PPRR) over high-dimensional data. In our protocol, each user submits its data in an encrypted form to an evaluator and the evaluator computes a linear model of all users' data without learning their contents. The core encryption method is equipped with homomorphic properties to enable the evaluator to perform ridge regression over encrypted data. We implement our protocol and demonstrate that it is suitable for dealing with high-dimensional data distributed among millions of users. We also compare our protocol with the state-of-the-art solutions in terms of both computation and communication costs. The results show that our protocol outperforms most existing approaches based on secure multi-party computation, garbled circuit, fully homomorphic encryption, secret-sharing, and hybrid methods.

1. Introduction

Ridge regression is a statistical method for modeling a linear relationship between a dependent variable and some explanatory values. It is a building-block that plays a major role in many learning algorithms such as recommendation systems. A recommendation system learns user profiles through feedback so that the recommended item falls into the interest without requiring the user to make an explicit query. Regression technique analyzes a set of collected data and summarizes them in a compact form for determining how relevant an item is for a user.

Techniques for constructing a ridge regression, like other traditional machine learning algorithms, require the data to be in plaintext form for building a model. It means that a user who engages in an online service has to share its data with a service provider for regression. However, the user may refuse to do so if the shared data contain its personal information. For instance, a nationwide electronic medical record system (EMR) that encourages sharing of medical knowledge, has the potential to improve care coordination, healthcare quality, and many other areas of healthcare. The medical records may contain sensitive individual information so that the patients' are not willing to share them unless data privacy is guaranteed. The problem can be effectively addressed with the existence of a fully trusted party that carries out the computations on the behalf of the users. However, it is hard to find such a party in practice. In order to collect data adequately, it is important to take privacy into consideration while designing data analysis algorithms.

* Corresponding author.
E-mail addresses: yrchen.cs98g@nctu.edu.tw (Y.-R. Chen), rezapour@cs.nctu.edu.tw (A. Rezapour), wgtzeng@cs.nctu.edu.tw (W.-G. Tzeng).

https://doi.org/10.1016/j.ins.2018.03.061
0020-0255/© 2018 Elsevier Inc. All rights reserved.

The goal of this paper is to construct an efficient privacy-preserving ridge regression (PPRR) protocol over high-dimensional data. Each user u can submits its data $\mathbf{x}_u \in \mathbb{R}^d$ and $y_u \in \mathbb{R}$ in an encrypted form to a party called *evaluator* who is responsible to collect and analysis data. The evaluator aggregates the received data and performs regression without knowing data contents. We introduce a Crypto Service Provider (CSP) to initialize the cryptographic parameters of the system and help the evaluator to complete the regression task. CSP is not allowed to learn users' data and the regression model in our system.

Contributions. We propose a novel privacy-preserving ridge regression protocol for a large number of distributed data over high-dimensional data. We utilize secure summation method [4,6] and homomorphic encryption schemes [3,10,19,23] for data encryption. We carefully designed an efficient encryption method, which is equipped with homomorphic properties to enable the evaluator to perform regression over the encrypted data. Our PPRR protocol benefits from a number of advantages as follows. Firstly, the encryption cost of a user is asymptotically optimal. It is linear in the data dimension d. It also saves bandwidth cost and decreases network latency when numerous users join the system for data submission. Secondly, users can be offline after data submission and they are not required to participate in the subsequent computations. Thirdly, the computation tasks of both the evaluator and the CSP for aggregating users' data are highly parallelable. Their tasks can be computed roughly k times faster on k processors. Fourthly, the evaluator and the CSP need to only engage in one round of communication for exchanging a $d \times d$ data matrix. Lastly, we remove the dependency in the terms of both the number of users and the size of data from the regression computation. This enables our protocol to equip with the ability to deal with massive datasets as in [22].

We implement our protocol to evaluate the computation overhead with realistic settings. In addition, we compare it with the state-of-the-art solutions in terms of both computation and communication costs. The experimental results show the efficiency improvement of our protocol in almost all factors against the existing protocols. For instance, the regression computation time for the evaluator in Nikolaenko et al.'s method [22] is about 1.3 min with $d = 20$, whereas, a similar computation in our protocol takes only 8.8 s. The improvement becomes more substantial as d increases since the computational complexity of regression is $O(d^3)$. The importance of such improvement is immediately apparent in applications dealing with high-dimensional data.

We show that our protocol preserves data privacy assuming the evaluator and CSP are honest-but-curious and non-collusive. Also, we discuss the collusive cases for a subset of users, the evaluator and users, and the CSP and users. They can compromise data privacy of non-collusive users with a negligible probability. Moreover, our protocol can be extended to deal with a malicious evaluator or CSP, who attempts to misbehave for learning users' sensitive information.

Related works. The research of privacy-preserving regression has received considerable attention in recent years. Most of the proposed protocols focused on the data which are partitioned either horizontally or vertically across distributed servers [1,7–9,13,16–18,20,25,26,28]. A majority of these protocols employed secure multi-party computation (SMPC) framework to build a linear model over the joint dataset cooperatively. Du et al. [9] defined the S2-MLR (secure 2-party multivariate linear regression) and S2-MC (secure 2-party multivariate classification) problems. They developed a set of basic protocols for matrix computations (e.g., multiplication, inverse, etc.) as building blocks to solve the S2-MLR and S2-MC problems. Sanil et al. [25] further proposed a solution that enables multiple parties to compute the global regression coefficients from the local ones. They combined Powell's method and secure summation technique to enable each user to iteratively update the coefficients using its data.

Some of the approaches are based on secret sharing techniques [17,18]. Karr et al. [17] presented two secure linear regression methods based on secure data integration and secure multi-party computation, while considering different level of data confidentiality. The first method integrates datasets, while ensuring that no party can learn others data. The second one allows each party to share the local statistical information for computing the global regression coefficients using the secure summation protocol. In 2009, Karr et al. [18] used secure matrix product technique to allow multiple parties to estimate the coefficients and standard errors in linear regression. Parties can also verify the regression model without disclosing their private data. Recently, Cock et al. [7] proposed an information-theoretically secure linear regression protocol in the commodity-based model [2], which allows a trusted initializer pre-distributes some random strings. It later correlate the data to parties in the setup phase. Cock et al. [7] proposed a secure matrix multiplication and inversion protocol for the parties to compute the regression coefficients cooperatively. One shortcoming of the aforementioned SMPC based approaches is that they expect the data servers (or parties) to be on-line and participate in the computation throughout the entire process.

Another line of research made use of partially homomorphic encryption (PHE) schemes [16,20,24]. The learning algorithm is performed on ciphertext domain. Some solutions utilize fully homomorphic encryption (FHE) scheme [12] as a building block for encrypting user data and tackle the requirement of two non-colluding parties. However, data privacy is only assured when there is a trusted third party who can be responsible for key generation. Moreover, the approaches based on PHE schemes are usually complicated due to the limitations of the homomorphic property. On the other hand, the current FHE schemes are not efficient enough for large scale applications [21].

Recently, Nikolaenko et al. [22] proposed a hybrid approach for a large distributed dataset among million of users. Each user submits its encrypted data under an additive homomorphic encryption scheme. The evaluator aggregates them and executes a garbled regression circuit $\mathfrak{C}_{\text{CSP}}$ to compute the regression coefficient $\mathbf{w}$. The circuit $\mathfrak{C}_{\text{CSP}}$ is generated by CSP for implementing a linear system solver based on the Cholesky decomposition. Nevertheless, the garbled circuit method [27] is more efficient than the FHE schemes in regression phase. In particular, the size of $\mathfrak{C}_{\text{CSP}}$ does not depend on the number of

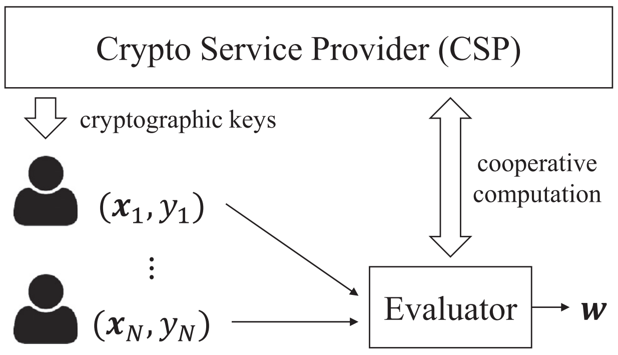


Fig. 1. Our system architecture.

users. Nikolaenko et al. [22] showed that their protocol performs substantially better than existing protocols based on secret sharing, SMPC, and garbled circuits. Data privacy is ensured if the evaluator and the CSP do not collude. However, a major shortcoming of the protocol is that the number of the gates in the circuit $\mathfrak{C}_{\text{CSP}}$ is large even if d is in the order of10′s. In addition. the computation and bandwidth costs of the protocol grow proportionally.

In 2016, Hu et al. [14] proposed a new efficient PPRR protocol without using Yao's garbled circuit. They designed a Packed Secure Multiplication Protocol (PSMC) by utilizing the Paillier encryption scheme. They transformed the ridge regession problem into the problem of solving linear equations so that they can use Gaussian elimination and Jacobi iterative methods to derive the learning model efficiently. The evaluation result showed that their protocol had a great reduction in computation time comparing with Nikolaenko et al.'s PPRR protocol.

In 2017, Gascón et al. [11] extended Nikolaenko et al.'s result for vertically partitioned data distributed among several users. They designed a secure multi/two-party inner product, which is supported by a trusted initializer for data aggregation. In regression phase, they used Yao's garble circuit to solving linear equations with non-collusive evaluator and CSP. Their circuit implemented Conjugate Gradient Descent (GCD) to provide a more efficient computation while maintaining accuracy and convergence rate. Another privacy-preserving linear regression is based on Gaussian elimination and LU decomposition was proposed by Bogdanov et al. [5].

In PPRR protocols proposed by Nikolaenko et al. [22], Hu et al. [14], Gascón et al. [11], and Bogdanov et al. [5] each user need to perform $O(d^2)$ operations for data encryption and submission. However, when d increases, their protocols become impractical due to expensive computation and communication costs since the costs are proportional to d^2.

2. Settings and threat model

In this section, we illustrate the system architecture and explain our system goals. We then give the threat model and system assumptions of our construction.

2.1. System architecture and system goal

As shown in Fig. 1, our system consists of three entities: N distributed users, evaluator, and Crypto Service Provider (CSP). The system is designed for the users to submit their data to the evaluator. The evaluator performs regression over the aggregated data and outputs a linear model. More precisely, each user $u = 1, \ldots, N$ owns its private data $\mathbf{x}_u = (x_{u,1}, x_{u,2}, \ldots, x_{u,d}) \in \mathbb{R}^d$ and $y_u \in \mathbb{R}$. CSP is responsible for initializing the whole system and generating system parameters. In our setting, CSP participates in some online steps to help the evaluator for generating $\mathbf{w}$.

Our goal is to ensure that during the protocol execution, neither the evaluator nor the CSP learns anything about users' private data ($\mathbf{x}_u$, y_u). In case that the evaluator colludes with some users, they should not be able to learn anything about private data contributed by the other users, except what is revealed by the result of the protocol. Similarly, in the case that CSP colludes with some users, they should be unable to learn anything about private data contributed by the other users.

2.2. Threat model and assumptions

In our system, we assume that the evaluator and CSP are honest-but-curious and do not collude. The former implies that they follow the protocol steps honestly but try to learn users' private data through observing protocol execution. The later implies that only one of them can be malicious at a time. Our protocol is designed to preserve the security under the following cases.

Malicious evaluator: The goal of the evaluator is to compute a correct parameter $\mathbf{w}$. Thus, the evaluator does not corrupt its computation for producing an incorrect result. However, the evaluator is motivated to misbehave for learning users' sensitive information. Our system requires that even such a malicious evaluator should be unable to learn anything beyond what is revealed by $\mathbf{w}$. Basically, the construction of our PPRR protocol in Section 4 assumes that the evaluator is honest-but-curious in protocol execution. We then explain in Section 5.2.1 how to extend our protocol to be cheating-resistance against a malicious evaluator.

Malicious CSP: CSP is also motivated to misbehave or disrupt evaluator's computation for learning users' data and $\mathbf{w}$. In Section 4, we describe that our PPRR protocol is secure against an honest-but-curious CSP. We then show in Section 5.2.1 how the evaluator can efficiently verify the correctness of CSP's computation with a high probability.

To simplify protocol description and analysis, we assume that the communication channels from a user to the evaluator and the channel between the evaluator and the CSP are secure. They can be implemented by incorporating a secure communication protocol such as SSL or TLS. We also assume that each user protects its private information properly and does not reveal them to any other entity. The private information includes users' data and secrets such as user individual random string. In Section 5.2, we discuss how to weaken this assumption.

Non-threats. Our protocol is *unable* to preserve the security under the following cases.

- Only evaluator or CSP can be malicious at a time, but not both of them. If the evaluator and the CSP collude, they can learn users' private data. The intuition behind this assumption is that both the evaluator and the CSP are well establish companies. Therefore, it is unlikely that they collude with each other. Otherwise they will lose their reputations and clients due to the unfavorable results.
- We assume that CSP correctly publishes system parameters. In addition, we assume that the evaluator and all users obtain the correct system parameters. This can be implemented using appropriate certificate authorities.

3. Background

In this section, we illustrate the used notations throughout this paper and introduce the ridge regression and LDLT decomposition methods.

3.1. Notations

We use bold uppercase letter $\mathbf{A}_{m\times n} = \left[a_{i,j}\right]_{m\times n}$ to represent an $m \times n$ matrix, where $a_{i,j}$ is the i-th row and j-th column entry. We use bold lowercase letter $\mathbf{x}_{m\times 1} = \left[x_i\right]_{m\times 1}$ to represent an $m \times 1$ matrix, where x_i is the i-th row entry. $\mathbf{x}_{m\times 1}$ can be either an m-dimension or m-tuple vector $(x_1, x_2, \ldots, x_m)$. We simply use $\mathbf{A}$ and $\mathbf{x}$ to represent $\mathbf{A}_{m\times n}$ and $\mathbf{x}_{m\times 1}$, respectively, when their dimensions are clearly understood. $\mathbf{A}^\top$ and $\mathbf{x}^\top$ are the transpose matrices of $\mathbf{A}$ and $\mathbf{x}$, respectively. $\mathbf{A}\circ\mathbf{B}$ is the element-wise (or Hadamard) production of two matrices $\mathbf{A} = \left[A_{i,j}\right]_{m\times n}$ and $\mathbf{B} = \left[B_{i,j}\right]_{m\times n}$, i.e., $\mathbf{A}\circ\mathbf{B} = \left[A_{i,j}B_{i,j}\right]_{m\times n}$.

3.2. Ridge regression

Given a set of input $\mathbf{x}_u = (x_{u,1}, x_{u,2}, \ldots, x_{u,d}) \in \mathbb{R}^d$ and its corresponding output $y_u \in \mathbb{R}$, for $1 \le u \le N$. The multiple linear regression problem is to learn $\mathbf{w} = (w_1, w_2, \ldots, w_d) \in \mathbb{R}^d$ such that $\mathbf{y} = \mathbf{X}\mathbf{w}$, where $\mathbf{X} = \left[x_{u,j}\right]_{N\times d}$ and $\mathbf{y} = \left[y_u\right]_{N\times 1}$.

The parameter $\mathbf{w}$ that satisfies $\mathbf{y} = \mathbf{X}\mathbf{w}$ may not exist. Hence, the *ridge regression* method estimates the closest approximation of $\mathbf{w}$ by minimizing the following objective function $E : \mathbb{R}^d \to \mathbb{R}$

$$E(\mathbf{w}) := \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 + \alpha\|\mathbf{w}\|^2. \tag{1}$$

For positive α, the regularization term $\alpha\|\mathbf{w}\|^2$ is used to prevent models from overfitting. By taking derivative w.r.t. $\mathbf{w}$, the minimization of (1) can be computed by solving the linear system

$$\mathbf{A}\mathbf{w} = \mathbf{b},$$

where $\mathbf{A} = \mathbf{X}^\top\mathbf{X} + \alpha\mathbf{I}$ and $\mathbf{b} = \mathbf{X}^\top\mathbf{y}$. Note that when $\alpha > 0$, the matrix $\mathbf{A}$ is symmetric and positive-definite.

3.3. LDLT Decomposition

The LDLT decomposition is used to decompose a square matrix $\mathbf{A}$ into the product of a lower triangular square matrix $\mathbf{L}$, a diagonal matrix $\mathbf{D}$, and $\mathbf{L}^\top$, i.e., $\mathbf{A} = \mathbf{L}\mathbf{D}\mathbf{L}^\top$. We solve $\mathbf{A}\mathbf{w} = \mathbf{b}$ by decomposing $\mathbf{A}$ as $\mathbf{L}\mathbf{D}\mathbf{L}^\top$ and solve $\mathbf{L}\mathbf{z} = \mathbf{b}$ by forward substitution, solving $\mathbf{D}\mathbf{z}' = \mathbf{z}$ by diagonal scaling, and solving $\mathbf{L}^\top\mathbf{w} = \mathbf{z}'$ by backward substitution. Table 1 demonstrates the LDLT decomposition algorithm. The overall complexity for the LDLT decomposition is $\Theta(d^3)$. The method for solving $\mathbf{L}\mathbf{D}\mathbf{L}^\top\mathbf{w} = \mathbf{b}$ takes $\Theta(d^2)$ operations.

The LDLT decomposition is a variant of the Cholesky decomposition. The Cholesky decomposition is numerically stable without the need of pivoting for symmetric and positive definite matrices. The LDLT decomposition is square-root free. It can be applied to a more general class of matrices. Some symmetric and indefinite matrices have no Cholesky decomposition, but they have an LDLT decomposition with negative entries in $\mathbf{D}$.

Table 1
LDLT decomposition algorithm.

```
Algorithm LDLT decomposition
Input: $\mathbf{A} = [\ a_{i,j}\ ]_{d\times d}$
Ouput: $(\mathbf{L}, \mathbf{D})$ s.t. $\mathbf{A} = \mathbf{LDL}^\top$
1 $\mathbf{L} = [\ \ell_{i,j}\ ]_{d\times d} \leftarrow \mathbf{I}$; $\mathbf{D} = [\ d_{i,j}\ ]_{d\times d} \leftarrow \mathbf{0}$;
2 for $k \leftarrow 1$ to $d$ do
3     $d_{k,k} \leftarrow a_{k,k}$;
4     for $i \leftarrow k+1$ to $d$ do
5         $a_{i,k} \leftarrow a_{ik}/a_{kk}$; $\ell_{i,k} \leftarrow a_{i,k}$;
6     for $j \leftarrow k+1$ to $d$ do
7         for $i \leftarrow j$ to $d$ do
8             $a_{i,j} \leftarrow a_{i,j} - a_{i,k}a_{k,k}a_{j,k}$;
9 return $(\mathbf{L}, \mathbf{D})$
```

4. Our PPRR protocol

In this section, we demonstrate the construction of our PPRR protocol, followed by its efficiency and security analyses. Our PPRR protocol consists of the following four phases:

- **Setup.** CSP generates the system parameters for all system entities.
- **Encryption.** Each user u encrypts its data and submits the encrypted data to the evaluator.
- **Aggregation.** The evaluator aggregates the received data with the assistance of CSP.
- **Regression.** The evaluator learns the model parameter $\mathbf{w}$ with the assistance of CSP.

4.1. An overview

Our data encryption is based on the *multiple* ElGamal encryption scheme over a finite cyclic group $\mathbb{G}$. Each user u encrypts its $\mathbf{x}_u$ and y_u with ElGamal public keys $T_i = g^{t_i}$, $0 \le i \le d$, as

$$g^r, \widetilde{\mathbf{x}}_u = [\ x_{u,i}T_i^r\ ]_{d\times 1}, \widetilde{y}_u = y_u T_0^r,$$

where r is a secret random string shared by users. Upon receiving users' data, the evaluator aggregates them as

$$\widetilde{\mathbf{A}} = \sum_u \widetilde{\mathbf{A}}_u = \sum_u \widetilde{\mathbf{x}}_u \widetilde{\mathbf{x}}_u^\top,$$
$$\widetilde{\mathbf{b}} = \sum_u \widetilde{\mathbf{b}}_u = \sum_u \widetilde{\mathbf{x}}_u \widetilde{y}_u$$

For regression, the evaluator uses the LDLT decomposition algorithm to decompose $\widetilde{\mathbf{A}} = \widetilde{\mathbf{L}}\widetilde{\mathbf{D}}\widetilde{\mathbf{L}}^\top$ and solves the linear system $\widetilde{\mathbf{A}}\widetilde{\mathbf{w}} = \widetilde{\mathbf{b}}$ to obtain $\widetilde{\mathbf{w}}$. The evaluator can solve the aforementioned linear systems, thanks to the homomorphic property of the encryption scheme. Eventually, the evaluator sends g^r and $\widetilde{\mathbf{w}}$ to CSP for decryption.

The multiple ElGamal encryption scheme remains secure regardless of using a common r [3,19]. However, the above protocol does not guarantee the privacy of $\mathbf{x}_u$'s with zero entry, i.e, the cipher of $x_{u,i} = 0$ is zero. In fact, plaintext domain of the classical ElGamal encryption scheme is $\mathcal{M} = \mathbb{Z}_p \setminus \{0\}$. Moerover, the encrypted data under the same T_i and randomness r become deterministic. That is, the evaluator can determine whether $x_{u,i}$ and $x_{v,i}$ of user u and v are the same from their corresponding ciphers $\widetilde{x}_{u,i}$ and $\widetilde{x}_{v,i}$ directly. In the extreme case, when the evaluator correctly guesses user u's entry $x_{u,i}$, it can decrypt every other user v's $x_{v,i}$ by computing $\widetilde{x}_{v,i}/(\widetilde{x}_{u,i}/x_{u,i})$.

To deal with the above issues, we apply the *secure summation* method [4,6]. Each user u randomly picks an individual and one-time used nonzero secret $\delta_u \in \mathbb{G}$ and incorporates δ_u to its data as follows:

$$x'_{u,i} = x_{u,i} + (i+1)\delta_u, \text{ for } 1 \le i \le d,$$
$$y'_u = y_u + (d+2)\delta_u.$$

These computations are all done over $\mathbb{G}$. The bit length of δ_u is equal to that of an element in $\mathbb{G}$. Therefore, δ_u perfectly hides $x_{u,i}$. Since δ_u is chosen randomly from $\mathbb{G}$, the evaluator can not learn $x_{v,i}$ from $\widetilde{x}'_{v,i} = (x_{v,i} + (i+1)\delta_v)T_i^r$, regardless of knowing the value of $x_{u,i}$.

Now, when the evaluator computes $\widetilde{\mathbf{A}}_u$ and $\widetilde{\mathbf{b}}_u$ for aggregation, there are extra parts.

$$(j+1)\delta_u x_{u,i} + (i+1)(\delta_u x_{u,j} + (j+1)\delta_u^2),$$
$$\delta_u x_{u,i} + (i+1)(\delta_u y_u + \delta_u^2).$$

In order to eliminate them, each user further submits an encryption of those extra parts under an additively homomorphic encryption scheme, e.g., the Paillier encryption scheme [23]. The evaluator aggregates the extra parts of all users and asks

CSP to re-encrypt them and add $\alpha\mathbf{I}$. The evaluator then eliminates the extra parts to obtain the encrypted $\widetilde{\mathbf{A}}$ and $\widetilde{\mathbf{b}}$ for computing ridge regression.

Remark. For data encryption, we use the secure summation method and multiple ElGamal encryption scheme simultaneously. These two techniques are complementary of each other. The secure summation fixes the deficiency of the multiple ElGamal encryption scheme with a shared r among distributed users. The multiple ElGamal encryption scheme protects users' data before and after aggregation.

4.2. The construction

This section gives the detailed construction of our PPRR protocol. The correctness of the protocol is also provided.

4.2.1. Setup phase

CSP sets a data dimension d, decides a domain $[0, 2^{\ell}]$ for each attribute, and prepares ElGamal and Paillier encryption schemes $\Pi = (\mathsf{ElG.K}, \mathsf{ElG.E}, \mathsf{ElG.D})$ and $\Psi = (\mathsf{Pai.K}, \mathsf{Pai.E}, \mathsf{Pai.D})$. CSP invokes key generation algorithm $\mathsf{ElG.K}$ to generate public-key and private-key of Π with $d+1$ pairs as follows.

$$\begin{aligned} \mathrm{PK}_{\mathrm{ElG}} &= \{\mathbb{G} = \mathbb{Z}_p, p, g, T_i = g^{t_i} : 0 \le i \le d\}, \\ \mathrm{SK}_{\mathrm{ElG}} &= \{t_i \in \mathbb{Z}_p^* : 0 \le i \le d\}. \end{aligned}$$

In addition, it envokes the key generation algorithm $\mathsf{Pai.K}$ to generate Ψ's key pair $(\mathrm{PK}_{\mathrm{Pai}}, \mathrm{SK}_{\mathrm{Pai}}) = (pk, sk)$.[1]CSP keeps $\mathrm{SK}_{\mathrm{ElG}}$ and $\mathrm{SK}_{\mathrm{Pai}}$ secret and publishes $\mathrm{PK}_{\mathrm{ElG}}$ and $\mathrm{PK}_{\mathrm{Pai}}$.

4.2.2. Encryption phase

In the beginning, users share a common random string $r \in \mathbb{Z}_p^*$. It can be accomplish with help of a trusted party [2] or using a distributed method. Every user u randomly picks a $\log_2 p$-bit string $\delta_u \in \mathbb{G}$, computes $V = g^r$, and encrypts data $\mathbf{x}_u$ and y_u as

$$\begin{aligned} \widetilde{\mathbf{x}}'_u &\leftarrow \left[\ x'_{u,i}T_i^r\ \right]_{d\times 1} \text{ with } x'_{u,i} = x_{u,i} + (i+1)\delta_u,\ 1 \le i \le d, \\ \hat{\mathbf{x}}'_u &\leftarrow \left[\ \mathsf{Pai.E}(pk, \delta_u x_{u,i})\ \right]_{d\times 1},\ 1 \le i \le d, \\ \hat{\Delta}_u &\leftarrow \mathsf{Pai.E}(pk, \delta_u^2), \\ \widetilde{y}'_u &\leftarrow y'_u T_0^r \text{ with } y'_u = y_u + \delta_u, \\ \hat{y}'_u &\leftarrow \mathsf{Pai.E}(pk, \delta_u y_u). \end{aligned}$$

Finally, u submits $(V, \widetilde{\mathbf{x}}'_u, \hat{\mathbf{x}}'_u, \hat{\Delta}_u, \widetilde{y}'_u, \hat{y}'_u)$ to the evaluator.

4.2.3. Aggregation phase

The evaluator sets initial matrices $\widetilde{\mathbf{A}}' \leftarrow \mathbf{0}_{d\times d}$, $\widetilde{\mathbf{b}}' \leftarrow \mathbf{0}_{d\times 1}$, $\hat{\mathbf{A}}' \leftarrow \mathbf{1}_{d\times d}$, and $\hat{\mathbf{b}}' \leftarrow \mathbf{1}_{d\times 1}$. Upon receiving a user's encrypted data $(V, \widetilde{\mathbf{x}}'_u, \hat{\mathbf{x}}'_u, \hat{\Delta}_u, \widetilde{y}'_u, \hat{y}'_u)$, the evaluator computes

$$\begin{aligned} \widetilde{\mathbf{A}}' &\leftarrow \widetilde{\mathbf{A}}' + \widetilde{\mathbf{x}}'_u(\widetilde{\mathbf{x}}'_u)^{\top}, \\ \hat{\mathbf{A}}' &\leftarrow \hat{\mathbf{A}}' \circ \left[\ (\hat{x}'_{u,i})^{j+1}(\hat{x}'_{u,j}\hat{\Delta}_u^{j+1})^{i+1}\ \right]_{d\times d}, \\ \widetilde{\mathbf{b}}' &\leftarrow \widetilde{\mathbf{b}}' + \widetilde{\mathbf{x}}'_u\widetilde{y}'_u, \\ \hat{\mathbf{b}}' &\leftarrow \hat{\mathbf{b}}' \circ \left[\ \hat{x}'_{u,i}(\hat{y}'_u\hat{\Delta}_u)^{i+1}\ \right]_{d\times 1}. \end{aligned}$$

After aggregating all (or enough amount of) users' data, the evaluator sends the result $(V, \hat{\mathbf{A}}', \hat{\mathbf{b}}')$ to CSP. CSP decrypts $\hat{\mathbf{A}}'$ and $\hat{\mathbf{b}}'$ under Ψ, subtracts $\alpha\mathbf{I}$, and encrypts them under Π:

$$\begin{aligned} \widetilde{\mathbf{A}}'' &\leftarrow \left[\ (\mathsf{Pai.D}(sk, \hat{a}'_{i,j}) - \alpha_{i,j})V^{t_i+t_j}\ \right]_{d\times d}, \\ \widetilde{\mathbf{b}}'' &\leftarrow \left[\ (\mathsf{Pai.D}(sk, \hat{b}'_i)V^{t_i+t_0}\ \right]_{d\times 1}, \end{aligned}$$

where $\alpha_{i,j} = \alpha$ when $i = j$, otherwise $\alpha_{i,j} = 0$. Finally, CSP returns $(\widetilde{\mathbf{A}}'', \widetilde{\mathbf{b}}'')$ and the evaluator computes $\widetilde{\mathbf{A}} = \widetilde{\mathbf{A}}' - \widetilde{\mathbf{A}}''$ and $\widetilde{\mathbf{b}} = \widetilde{\mathbf{b}}' - \widetilde{\mathbf{b}}''$.

[1] Paillier encryption scheme is additive homomorphic, that is, for two values x_1 and x_2,

$$\begin{aligned} &\mathsf{Pai.E}(pk, x_1)\mathsf{Pai.E}(pk, x_2) = \mathsf{Pai.E}(pk, x_1 + x_2), \\ &(\mathsf{Pai.E}(pk, x_1))^{x_2} = \mathsf{Pai.E}(pk, x_1 x_2). \end{aligned}$$

Correctness. For $\widetilde{\mathbf{A}}$: The (*i, j*)-th entry value of $\widetilde{\mathbf{A}}'$ is

$$\widetilde{a}'_{i,j} = \sum_u x'_{u,i} x'_{u,j} T^r_i T^r_j = \sum_u (x_{u,i}x_{u,j} + e_{u,i,j}) T^r_i T^r_j,$$

where $e_{u,i,j} = (j+1)\delta_u x_{u,i} + (i+1)(\delta_u x_{u,j} + (j+1)\delta_u^2)$. The (*i, j*)-th entry value of $\hat{\mathbf{A}}'$ is

$$\hat{a}'_{i,j} = \prod_u (\hat{x}'_{u,i})^{j+1} (\hat{x}'_{u,j} \hat{\Delta}_u^{j+1})^{i+1} = \prod_u \mathsf{Pai.E}(pk, e_{u,i,j}) = \mathsf{Pai.E}(pk, \sum_u e_{u,i,j}).$$

After CSP re-encrypts $\hat{\mathbf{A}}'$ as $\widetilde{\mathbf{A}}''$, the resulting matrix $\widetilde{\mathbf{A}} = \widetilde{\mathbf{A}}' - \widetilde{\mathbf{A}}''$ has no extra parts and contains $\alpha\mathbf{I}$.

For $\widetilde{\mathbf{b}}$: The *i*-th entry value of $\widetilde{\mathbf{b}}'$ is

$$\widetilde{\mathbf{b}}'_i = \sum_u x'_{u,i} y'_u T^r_i T^r_0 = \sum_u (x_{u,i} y_u + e_{u,i}) T^r_i T^r_0,$$

where $e_{u,i} = \delta_u x_{u,i} + (i+1)(\delta_u y_u + \delta_u^2)$. The *i*-th entry value of $\hat{\mathbf{b}}'$ is

$$\hat{b}'_i = \prod_u \hat{x}'_{u,i} (\hat{y}'_u \hat{\Delta}_u)^{i+1} = \prod_u \mathsf{Pai.E}(pk, e_{u,i}) = \mathsf{Pai.E}(pk, \sum_u e_{u,i}).$$

After CSP re-encrypts $\hat{\mathbf{b}}'$ as $\widetilde{\mathbf{b}}''$, the result $\widetilde{\mathbf{b}} = \widetilde{\mathbf{b}}' - \widetilde{\mathbf{b}}''$ has no extra parts.

4.2.4. Regression phase

The evaluator first solves the encrypted linear system $\widetilde{\mathbf{A}}\widetilde{\mathbf{w}} = \widetilde{\mathbf{b}}$ and obtains $\widetilde{\mathbf{w}}$. It then decrypts $\widetilde{\mathbf{w}}$ as follows. Notice that, all of the operations are done over $\mathbb{G}$.

(1) Run the LDLT algorithm to decompose $\widetilde{\mathbf{A}} = \widetilde{\mathbf{L}}\widetilde{\mathbf{D}}\widetilde{\mathbf{L}}^\top$.
(2) Solve $\widetilde{\mathbf{L}}\widetilde{\mathbf{z}} = \widetilde{\mathbf{b}}$ by forward substitution, $\widetilde{\mathbf{D}}\widetilde{\mathbf{z}}' = \widetilde{\mathbf{z}}$ by diagonal scaling, and $\widetilde{\mathbf{L}}^\top\widetilde{\mathbf{w}} = \widetilde{\mathbf{z}}'$ by backward substitution.
(3) The evaluator randomly chooses $\mathbf{s} = [s_i]_{d\times1}$, computes $\widetilde{\mathbf{w}}' = [\widetilde{w}'_i]_{d\times1} = \mathbf{s} \circ \widetilde{\mathbf{w}}$, and sends $(V, \widetilde{\mathbf{w}}')$ to CSP for decrypting $\widetilde{\mathbf{w}}'$:

$$\mathbf{w}' = [w'_i]_{d\times1} \leftarrow \left[\frac{\widetilde{w}'_i}{V^{t_0-t_i}}\right]_{d\times1}.$$

Finally, the evaluator computes the answer $\mathbf{w} \leftarrow [w'_i/s_i]_{d\times1}$.

Correctness. We show that the computation result is our desired answer as follows. For Step (1), we show that the LDLT algorithm with input $\widetilde{\mathbf{A}}$ outputs $(\widetilde{\mathbf{L}}, \widetilde{\mathbf{D}})$. It is similar to the case that the LDLT algorithm outputs (**L, D**) on input **A**, but it works over ciphertext domain. For Step (2), we show that after the forward substitution, diagonal scaling, and backward substitution, $\widetilde{\mathbf{w}}$ is an encrypted form of **w** with predictable encryption terms $V^{t_0-t_i}$ for $i = 1, 2, \ldots, d$. Finally, for Step (3), we show that the evaluator is able to query CSP for decrypting $\widetilde{\mathbf{w}}'$ without exposing the final result.

Step (1). Let $\widetilde{\mathbf{A}} = \begin{bmatrix} \widetilde{a}_{i,j} \end{bmatrix}_{d\times d}$ with $\widetilde{a}_{i,j} = a_{i,j} T^r_i T^r_j$ and $a_{i,j} = \sum_u x_{u,i} x_{u,j}$. We show that the LDLT algorithm with input $\widetilde{\mathbf{A}}$ outputs $(\widetilde{\mathbf{L}}, \widetilde{\mathbf{D}})$ as

$$\widetilde{\mathbf{L}} = \begin{bmatrix} 1 & 0 & \ldots & 0 \\ \widetilde{\ell}_{2,1} & 1 & \ldots & 0 \\ \widetilde{\ell}_{3,1} & \widetilde{\ell}_{3,2} & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \widetilde{\ell}_{d,1} & \widetilde{\ell}_{d,2} & \ldots & 1 \end{bmatrix}, \widetilde{\mathbf{D}} = \begin{bmatrix} \widetilde{d}_{1,1} & 0 & \ldots & 0 \\ 0 & \widetilde{d}_{2,2} & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \widetilde{d}_{d,d} \end{bmatrix},$$

where $\widetilde{\ell}_{i,j} = \ell_{i,j} \frac{T^r_i}{T^r_j}$ and $\widetilde{d}_{i,i} = d_{i,i} T^r_i T^r_i$, and the values of $\ell_{i,j}$'s and $d_{i,i}$'s are exactly the same values as in (**L, D**).

In the *k*-th iteration (line 3–8) of the LDLT algorithm with input $\widetilde{\mathbf{A}}$:

$$\widetilde{d}_{k,k} \leftarrow \widetilde{a}_{k,k} = a_{k,k} T^r_k T^r_k;$$

$$\widetilde{\ell}_{i,k} = \widetilde{a}_{i,k} \leftarrow \frac{\widetilde{a}_{i,k}}{\widetilde{a}_{k,k}} = \frac{a_{i,k} T^r_i T^r_k}{a_{k,k} T^r_k T^r_k} = \frac{a_{i,k}}{a_{k,k}} \frac{T^r_i}{T^r_k};$$

$$\widetilde{a}_{i,j} \leftarrow \widetilde{a}_{i,j} - \widetilde{a}_{i,k}\widetilde{a}_{k,k}\widetilde{a}_{j,k} = (a_{i,j} T^r_i T^r_j) - \left(a_{i,k} \frac{T^r_i}{T^r_k}\right)(a_{k,k} T^r_k T^r_k)\left(a_{j,k} \frac{T^r_j}{T^r_k}\right) = (a_{i,j} - a_{i,k} a_{k,k} a_{j,k}) T^r_i T^r_j.$$

As one can see, $\widetilde{a}_{i,j}$'s are computed the same as $a_{i,j}$'s except that the input to the LDLT algorithm is the encrypted $\widetilde{\mathbf{A}}$. It is easy to verify that the resulting $(\widetilde{\mathbf{L}}, \widetilde{\mathbf{D}})$ satisfies $\widetilde{\mathbf{A}} = \widetilde{\mathbf{L}}\widetilde{\mathbf{D}}\widetilde{\mathbf{L}}^\top$.

Step (2). From the result $(\widetilde{\mathbf{L}}, \widetilde{\mathbf{D}})$ in the last step, the following shows that our method in solving the encrypted linear equations outputs an encrypted $\widetilde{\mathbf{w}}$ with encryption terms $V^{t_0-t_i}$ for $i = 1, 2, \ldots, d$.

Table 2
The computation and communication costs in our PPRR protocol.

Computation cost #(operations)				Communication cost #(upload group elements)		
	CSP	User	Evaluator	CSP	User	Evaluator
Setup	$d+1$ ElG.K, 1 Pai.K	N/A	N/A	$d+2$	0	0
Encryption	N/A	1 R, $2d+1$ A, $2d+3$ M, $d+2$ Pai.E	N/A	N/A	6	N/A
Aggregation	$\frac{d^2+3d}{2}$ (Pai.D, A, M, E)	N/A	$N\frac{d^2+d}{2}+d$ A, $(N-1)(2d^2+5d+1)$ M, $(N-1)d$ E	$\frac{d^2+3d}{2}$	0	$\frac{d^2+3d+2}{2}$
Regression	d A, d E, d I, d M	N/A	$O(d^3)$ flops over $\mathbb{G}$	d	0	$d+1$

†R: Random string generation; A: Modular addition; M: Modular multiplicaiton; E: Modular exponentiation; I: Modular inversion.
†flops: floating-point operations (additions, subtractions, multiplications, and divisions)

- Solve $\widetilde{\mathbf{L}}\widetilde{\mathbf{z}} = \widetilde{\mathbf{b}}'$ by forward substitution: for $i \leftarrow 1$ to d,

$$\widetilde{z}_i \leftarrow \widetilde{b}'_i - \sum_{k=1}^{i-1} \widetilde{\ell}_{i,k}\widetilde{b}'_k = b'_i T_i^r T_0^r - \sum_{k=1}^{i-1}\left(\ell_{i,k}\frac{T_i^r}{T_k^r}\right)(b'_k T_k^r T_0^r) = \left(b'_i - \sum_{k=1}^{i-1}\ell_{i,k}b'_k\right)T_i^r T_0^r;$$

- Solve $\widetilde{\mathbf{D}}\widetilde{\mathbf{z}}' = \widetilde{\mathbf{z}}$ by diagonal scaling: for $i \leftarrow 1$ to d,

$$\widetilde{z}'_i \leftarrow \frac{\widetilde{z}_i}{\widetilde{d}_{i,i}} = \frac{z_i T_i^r T_0^r}{d_{i,i} T_i^r T_i^r} = \frac{z_i}{d_{i,i}}\frac{T_0^r}{T_i^r};$$

- Solve $\widetilde{\mathbf{L}}^\top\widetilde{\mathbf{w}} = \widetilde{\mathbf{z}}'$ by backward substitution: for $i \leftarrow d$ to 1,

$$\widetilde{w}_i \leftarrow \widetilde{z}'_i - \sum_{k=i+1}^{d}\widetilde{\ell}_{k,i}\widetilde{z}'_k = z'_i\frac{T_0^r}{T_i^r} - \sum_{k=i+1}^{d}\left(\ell_{k,i}\frac{T_k^r}{T_i^r}\right)\left(z'_k\frac{T_0^r}{T_k^r}\right) = \left(z'_i - \sum_{k=i+1}^{d}\ell_{k,i}z'_k\right)\frac{T_0^r}{T_i^r}.$$

Step (3). From the last step, we have $\widetilde{\mathbf{w}}' = [\widetilde{w}'_i]_{d\times 1} = [s_i w_i \frac{T_0^r}{T_i^r}]_{d\times 1}$. CSP decrypts $\widetilde{\mathbf{w}}'$ and obtains

$$\mathbf{w}' \leftarrow \left[\frac{\widetilde{w}'_i}{V^{t_0 - t_i}}\right]_{d\times 1} = \left[\frac{\widetilde{w}'_i}{T_0^r/T_i^r}\right]_{d\times 1} = \left[s_i w_i\right]_{d\times 1}.$$

Finally, the evaluator obtains the final answer $\mathbf{w} = [w_i]_{d\times 1} = [w'_i/s_i]_{d\times 1}$.

4.3. Efficiency analysis

We calculate the cost for the participants in each phase as follows. The overall costs are listed in Table 2.

Setup phase. CSP generates p, $\mathbb{G} = \langle g\rangle$, $d+1$ ElGamal key pairs (t_i, T_i)'s, and 1 Paillier key pair (pk, sk). CSP keeps $d+2$ private-keys t_i's and sk secret. It then publishes $\mathbb{G}$, $d+2$ public-keys T_i's, and pk.

Encryption phase. The computation cost of each user u for encrypting its data consists of the following operations: *random string generation* (R), *modular addition* (A), *modular multiplicaiton* (M), *modular exponentiation* (E), and *Paillier encryption* (Pai.E). Before encryption, u does 1 time of R to generate δ_u. Then u does d times of A for computing $(\delta_u, 2\delta_u, \ldots, (d+1)\delta_u)$ following by $d+1$ times of A for aggregating them with $(\mathbf{x}_u, y_u)$ to obtain $(\mathbf{x}'_u, y'_u)$. To encrypt $(\mathbf{x}'_u, y'_u)$ under Π, u does $d+1$ times of E for computing $(T_i)^r$'s, and then does $d+1$ times of M for encrypting $\mathbf{x}'_u$ and y'_u as $[(x'_{u,i})\cdot(T_i^r)]_{d\times 1}$ and $y'_u \cdot T_0^r$, respectively. To encrypt the extra parts under Ψ, u does $d+2$ times of M for computing $\delta_u\mathbf{x}_u$, δ_u^2, and $\delta_u y_u$. It then does $d+2$ times of Pai.E for encrypting them as $\hat{\mathbf{x}}'_u$, $\hat{\Delta}_u$, and $\hat{y}_u$, respectively. Finally, user u submits its encrypted data that contains 6 group elements to the evaluator.

Aggregation phase. The computation of both evaluator and CSP in aggregating users' data consists of A, M, and E operations. The matrices $\widetilde{\mathbf{A}}'$, $\hat{\mathbf{A}}'$, $\widetilde{\mathbf{A}}''$, and $\widetilde{\mathbf{A}}$ are symmetric. Hence, the evaluator and the CSP only need to compute and transmit the upper triangular parts ($\frac{d^2+d}{2}$ entries).

Evaluator's part. For each user u, the evaluator aggregates u's data by computing 4 matrices: $\widetilde{\mathbf{A}}'$, $\hat{\mathbf{A}}'$, $\widetilde{\mathbf{b}}'$, and $\hat{\mathbf{b}}'$. The evaluator does $\frac{d^2+d}{2}$ times of M and A operations for computing $\widetilde{\mathbf{A}}'$ and does d times of M and A operations for computing $\widetilde{\mathbf{b}}'$. To compute $\hat{\mathbf{A}}'$, it first needs $3\frac{(d^2+d)}{2}$ times of E and M operations. By re-arranging the operations carefully, the evaluator can compute $\hat{\mathbf{A}}'$ using d times of E and $\frac{3d^2+5d}{2}$ times of M operations. It makes the computations at least twice faster. Such an efficiency becomes more substantial as N and d increases.

- For each $i = 1, 2, \ldots, d$, compute $U_{i,i} \leftarrow (\hat{x}'_{u,i})^{i+1}$, $U_{i,i+1} \leftarrow U_{i,i}\cdot\hat{x}'_{u,i}, \ldots, U_{i,d} \leftarrow U_{i,d-1}\cdot\hat{x}'_{u,i}$.

- Compute $\hat{\Delta}_u^2 \leftarrow \hat{\Delta}_u \cdot \hat{\Delta}_u$, $\hat{\Delta}_u^3 \leftarrow \hat{\Delta}_u^2 \cdot \hat{\Delta}_u, \ldots, \hat{\Delta}_u^{d+1} \leftarrow \hat{\Delta}_u^d \cdot \hat{\Delta}_u$.
- For each $j = 1, 2, \ldots, d$, compute $V_j \leftarrow \hat{x}'_{u,j} \cdot \hat{\Delta}_u^{j+1}$.
- For each $j = 1, 2, \ldots, d$, compute $W_{1,j} \leftarrow V_j \cdot V_j$, $W_{2,j} \leftarrow W_{1,j} \cdot V_j, \ldots, W_{j,j} \leftarrow W_{j-1,j} \cdot V_j$.
- Compute $\mathbf{T}_u = [(\hat{x}'_{u,i})^{j+1}(\hat{x}'_{u,j}\hat{\Delta}_u^{j+1})^{i+1}]$ as $[U_{i,j}W_{i,j}]$ and then $\hat{\mathbf{A}}' \leftarrow \hat{\mathbf{A}}' \circ \mathbf{T}_u$.

Similarly, for each user u, the evaluator computes $\hat{\mathbf{b}}'$ using $2d+1$ times of M operation. To aggregate N users' data, the evaluator does $(N-1)\frac{d^2+d}{2}$ times of A, $(N-1)(2d^2+5d+1)$ times of M, and $(N-1)d$ times of E operations in total. In order to deliver the aggregated matrices $(\hat{\mathbf{A}}', \hat{\mathbf{b}}')$ to CSP, the evaluator transmits $\frac{d^2+d}{2} + d$ group elements. After CSP returns $(\widetilde{\mathbf{A}}'', \widetilde{\mathbf{b}}'')$, the evaluator does $\frac{d^2+d}{2} + d$ times of A operation to compute $(\widetilde{\mathbf{A}}, \widetilde{\mathbf{b}})$.

CSP's part. Upon receiving $(V, \hat{\mathbf{A}}', \hat{\mathbf{b}}')$, CSP helps the evaluator to re-encrypt them as $\widetilde{\mathbf{A}}''$ and $\widetilde{\mathbf{b}}''$, respectively. In doing so, it does $\frac{d^2+d}{2} + d$ times of $\mathsf{Pai.D}$ to decrypt $(\hat{\mathbf{A}}', \hat{\mathbf{b}}')$ as $(\mathbf{A}', \mathbf{b}')$ and d times of A to subtracts $\alpha\mathbf{I}$ from $\mathbf{A}'$. Then it does $\frac{d^2+d}{2} + d$ times of A and E operations to compute $t_i + t_j$, $t_i + t_0$, and $V^{t_i+t_j}$, $V^{t_i+t_0}$. Finally, it does $\frac{d^2+d}{2} + d$ times of M operation to compute $\widetilde{\mathbf{A}}''$ and $\widetilde{\mathbf{b}}''$ for the evaluator.

Regression phase. The evaluator solves $\widetilde{\mathbf{A}}\widetilde{\mathbf{w}} = \widetilde{\mathbf{b}}$ by using the standard LDLT algorithm, forward substitution, and backward substitution over $\mathbb{G}$. This method is commonly used for solving linear systems, which requires $O(d^3)$ floating-point operations (flops). The evaluator computes $\widetilde{\mathbf{w}}' = \mathbf{s} \circ \widetilde{\mathbf{w}}$ and sends $(V, \widetilde{\mathbf{w}})$ that contains $d+1$ group elements to CSP. For decryption, CSP computes $t_0 - t_i$, $V^{t_0-t_i}$, $(V^{t_0-t_i})^{-1}$, and $\widetilde{w}'_i \cdot (V^{t_0-t_i})^{-1}$ to obtain w'_i for each $i = 1, 2, \ldots, d$. CSP returns $\mathbf{w}'$ and the evaluator computes $w'_i s_i^{-1}$, for $i = 1, 2, \ldots, d$, to obtain the final answer $\mathbf{w}$.

4.4. Security analysis

In this section, we formally prove that our protocol preserves data privacy against the evaluator and the CSP. Recall that, we assume that the evaluator and the CSP are honest-but-curious in protocol execution. We also assume that users do not disclose their private data and secrets, i.e., we do not consider the collusion among users, evaluator, and CSP in this section. We will discuss some possible solutions to weaken the aforementioned assumptions in the next section.

4.4.1. Against CSP

In the aggregation phase, CSP can decrypt $\hat{\mathbf{A}}'$ and $\hat{\mathbf{b}}'$ to obtain the extra parts

$$\mathbf{A}' = \left[\ \sum_u (j+1)\delta_u x_{u,i} + (i+1)(\delta_u x_{u,j} + (j+1)\delta_u^2)\ \right]_{d\times d},$$
$$\mathbf{b}' = \left[\ \sum_u \delta_u x_{u,i} + (i+1)(\delta_u y_u + \delta_u^2)\ \right]_{d\times 1}.$$

They contain $\frac{d^2+d}{2} + d$ equations with $N(d+2)$ unknown variables δ_u and $(\mathbf{x}_u, y_u)$, for $1 \le u \le N$. Therefore, CSP cannot obtain $x_{u,i}$ or y_u when $N > \frac{d^2+3d}{2(d+2)} = \frac{d}{2}\frac{d+3}{d+2} > \frac{d}{2}$.

4.4.2. Against the evaluator

In the aggregation phase, the evaluator has both users encrypted data and the re-encrypted version of aggregated extra parts. We show that the evaluator cannot recover $x_{u,i}$'s and y_u's when the decisional Deffie–Hellman (DDH)[2] and decisional composite residuosity (DCR)[3] assumptions hold.

Our proof is based on a *data-recovery* (DR) game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. $\mathcal{C}$ initializes the system and generates all system parameters and publishes the public information PK. $\mathcal{C}$ then encrypts its data as a cipher $\mathbf{C}$ and sends (PK, $\mathbf{C}$) to $\mathcal{A}$. $\mathcal{A}$ outputs its guess $\breve{x}_{v,i^*}$ for the plaintext of an encrypted $\widetilde{x}_{v,i^*}$ in $\mathbf{C}$. The advantage of $\mathcal{A}$ for winning the DR game is

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{DR}} := \Pr[\breve{x}_{v,i^*} = x_{v,i^*}].$$

We call a protocol privacy-preserving if $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{DR}}$ is negligible for any poly-time adversary $\mathcal{A}$.

The following two lemmas show that, in our protocol, if there exists a poly-time adversary $\mathcal{A}$ with a non-negligible advantage, then, we can construct a poly-time algorithm $\mathcal{B}$ which calls $\mathcal{A}$ as a subroutine and breaks the underlying hard assumption (DDH or DCR assumption).

[2] [DDH assumption] Given $g, g^a, g^b, h \in \mathbb{G}$, it is intractable to determine whether $h = g^{ab}$. That is, the advantage

$$\mathsf{Adv}_{\mathcal{B}}^{\mathsf{DDH}} := |\Pr[\mathcal{B}(h) = 1 : h = g^{ab}] - \Pr[\mathcal{B}(h) = 1 : h \neq g^{ab}]|$$

is negligible for any poly-time algorithm $\mathcal{B}$.

[3] [DCR assumption] Given a composite $n = pq$ and an integer z, it is intractable to determine whether z is n-residue modulo n^2. That is, the advantage

$$\mathsf{Adv}_{\mathcal{B}}^{\mathsf{DCR}} := |\Pr[\mathcal{B}(z) = 1 : \exists y, z = y^n \bmod n^2] - \Pr[\mathcal{B}(z) = 1 : \forall y, z \neq y^n \bmod n^2]|$$

is negligible for any poly-time algorithm $\mathcal{B}$.

Table 3
Description of the $\mathcal{B}_{\rm DDH}$ algorithm for attacking the DDH assumption.

```
Algorithm $\mathcal{B}_{\rm DDH}$
Input: $\mathbb{G}$, $(g, g^a, g^b, h) \in \mathbb{G}^4$, $\mathcal{A}$, $\Psi$
Ouput: 1 for guessing $h = g^{ab}$ and 0 otherwise
1  /*generate public-key PK */
2  for $i \leftarrow 0$ to $d$ do $s_i \xleftarrow{\$} \mathbb{Z}_p^*$
3  $\text{PK}_{\text{ElG}} = \{\mathbb{G}, p, g, T_i : T_i \leftarrow (g^a)^{s_i}, 0 \le i \le d\}$;
4  $\text{PK}_{\text{Pai}} = \{pk\} \leftarrow \text{Pai.K}(\lambda)$;
5  for $u \leftarrow 1$ to $N$ do $(\mathbf{x}_u, y_u) \xleftarrow{\$} \mathbb{G}^d \times \mathbb{G}$;
6  /*generate encrypted data $\mathbf{C}_1$*/
7  $V \leftarrow g^b$; $\delta_u \xleftarrow{\$} \mathbb{G}$;
8  for $u \leftarrow 1$ to $N$ do
9      $\widetilde{\mathbf{x}}'_u \leftarrow [(x_{u,i} + (i+1)\delta_u)h^{s_i}]_{d\times 1}$;
10     $\hat{\mathbf{x}}'_u \leftarrow [\text{Pai.E}(pk, \delta_u x_{u,i})]_{d\times 1}$;
11     $\hat{\Delta}_u \leftarrow \text{Pai.E}(pk, \delta_u^2)$;
12     $\widetilde{y}'_u \leftarrow (y_u + \delta_u)h^{s_0}$;
13     $\hat{\mathbf{y}}'_u \leftarrow \text{Pai.E}(pk, \delta_u y_u)$;
14 /*generate re-encrypted data $\mathbf{C}_2$*/
15 $\widetilde{\mathbf{A}}'' \leftarrow \begin{bmatrix} \sum_u((j+1)\delta_u x_{u,i} \\ +(i+1)(\delta_u x_{u,j} + (j+1)\delta_u^2) - \alpha_{i,j})h^{s_i+s_j} \end{bmatrix}_{d\times d}$;
16 $\widetilde{\mathbf{b}}'' \leftarrow \left[\delta_u \sum_u x_{u,i} + (i+1)(y_u + \delta_u)h^{s_i+s_0}\right]_{d\times 1}$;
17 $\breve{x}_{v,i^*} \leftarrow \mathcal{A}(\text{PK}, \mathbf{C}_1, \mathbf{C}_2)$;
18 return 1 if $\breve{x}_{v,i^*} = x_{v,i^*}$ and 0 otherwise
```

Lemma 1. $\text{Adv}_{\mathcal{B}_{DDH}}^{DDH} \ge \text{Adv}_{\mathcal{A}}^{DR} - \frac{1}{2^\lambda}$.

Proof. We construct a $\mathcal{B}_{\rm DDH}$ algorithm that breaks the DDH assumption (as in Table 3) by utilizing $\mathcal{A}$ that attacks our protocol under the DR game.

Let $\mathbb{G} = \langle g \rangle$ be a finite cyclic group of order p, and g be the generator of $\mathbb{G}$. Let $(g, g^a, g^b, h) \in \mathbb{G}^4$ be an instance of DDH problem in $\mathbb{G}$. To generate public key PK_{EIG}, $\mathcal{B}_{\rm DDH}$ embeds the input challenge instance g^a into the public-key. Furthermore, $\mathcal{B}_{\rm DDH}$ sets the common random secret $V = g^b$. $\mathcal{B}_{\rm DDH}$ randomly generates users' data $(\mathbf{x}_u, y_u)$, $1 \le u \le N$ and encrypts them as $\mathbf{C}_1 = (V, \widetilde{\mathbf{x}}'_u, \hat{\mathbf{x}}'_u, \hat{\Delta}_u, \widetilde{y}'_u, \hat{\mathbf{y}}')$ and $C_2 = (\widetilde{\mathbf{A}}'', \widetilde{\mathbf{b}}'')$. $\mathcal{B}_{\rm DDH}$ simulates the protocol as follows:

(1) For PK, $\mathcal{B}_{\rm DDH}$ randomly chosen $s_i \in \mathbb{Z}_p^*$, $0 \le i \le d$ and sets $T_i = (g^a)^{s_i}$. $\mathcal{B}_{\rm DDH}$ generates $\text{PK}_{\text{Pai}} = \{pk\}$ using Ψ.
(2) For $\mathbf{C}_1$, $\mathcal{B}_{\rm DDH}$ sets $V = g^b, \widetilde{\mathbf{x}}'_u = \left[\ (x_{u,i} + (i+1)\delta_u)h^{s_i}\ \right]_{d\times 1}$, and $\widetilde{y}'_u = (y_u + \delta_u)h^{s_0}$. $\mathcal{B}_{\rm DDH}$ generates other encrypted data $\hat{\mathbf{x}}'_u$, $\hat{\Delta}_u$, and $\hat{\mathbf{y}}'$ under Ψ as discussed in our protocol construction.
(3) For $\mathbf{C}_2$, $\mathcal{B}_{\rm DDH}$ sets each entry of $\widetilde{\mathbf{A}}''$ as $\sum_u((j+1)\delta_u x_{u,i} + (i+1)(\delta_u x_{u,j} + (j+1)\delta_u^2) - \alpha_{i,j})h^{s_i+s_j}$ and each entry of $\widetilde{\mathbf{b}}''$ as $\sum_u(\delta_u x_{u,i} + (i+1)(\delta_u y_u + \delta_u^2))h^{s_i+s_0}$.

The advantage of $\mathcal{B}_{\rm DDH}$ for breaking the DDH assumption is:

$$\text{Adv}_{\mathcal{B}_{\rm DDH}}^{\rm DDH} = |\Pr[\breve{x}_{v,i^*} = x_{v,i^*} : h = g^{ab}] - \Pr[\breve{x}_{v,i^*} = x_{v,i^*} : h \ne g^{ab}]| \ge \text{Adv}_{\mathcal{A}}^{\rm DR} - \frac{1}{2^\lambda}.$$

In the DR game $\mathcal{B}_{\rm DDH}$ success if $\mathcal{A}$ correctly guesses $\breve{x}_{v,i^*}$ and $h = g^{ab}$. This is due to if $h = g^{ab}$, $h^{s_i} = (g^{ab})^{s_i} = (g^{as_i})^b = (T_i)^b$ and $h^{s_i+s_j} = (T_i)^b(T_j)^b$. The encrypted $\widetilde{\mathbf{x}}'_u$ and $\widetilde{y}'_u$ are thus valid since $\mathcal{B}_{\rm DDH}$ sets $T_i = g^{as_i}$ and $V = g^b$. Therefore, $\mathcal{B}_{\rm DDH}$ guesses correctly at least with the same probability as $\mathcal{A}$. Otherwise, if $h \ne g^{ab}$, $\mathcal{A}$ can only return $\breve{x}_{v,i^*}$ randomly. The probability that $\mathcal{A}$ randomly outputs $\breve{x}_{v,i^*} = x_{v,i^*}$ is $\frac{1}{2^\lambda}$. □

This implies that if the DDH assumption can not be computed in poly-time with a non-negligible probability, then a poly-time adversary $\mathcal{A}$ cannot exist with a non-negligible probability.

Similarly, we can construct a $\mathcal{B}_{\rm DCR}$ algorithm for breaking the DCR assumption and obtain the following lemma.

Lemma 2. $\text{Adv}_{\mathcal{B}_{DcH}}^{DCR} \ge \text{Adv}_{\mathcal{A}}^{DR} - \frac{1}{2^\lambda}$.

Proof. The proof of Lemma 2 is similar to the proof of Lemma 1, where $\mathcal{B}_{\rm DCR}$ algorithm embeds decisional composite residuosity (DCR) instance. □

Lemmas 1 and 2 states that our protocol is provably privacy preserving against the evaluator. That is, if the evaluator can breach user's data privacy in our protocol under the DR game, it can break DDH or DCR assumption. The following theorem states the result.

Theorem 1. *Our PPRR protocol ensures user's data privacy against the evaluator under the DR security game, assuming that DDH and DCR assumptions are hold.*

5. Extensions and discussions

In this section, we illustrate how to improve the computation time in data aggregation and extend the input space from $\mathbb{Z}_p$ to real numbers. We also discuss how to weaken some of the assumptions stated in our threat model.

5.1. Parallel data aggregation

Aggregation phase has the highest computation cost in our protocol. That is, the evaluator does $\mathcal{O}(Nd^2)$ operations for computing $(\widetilde{\mathbf{A}}', \widehat{\mathbf{A}}', \widetilde{\mathbf{b}}', \widehat{\mathbf{b}}')$ and CSP does $\mathcal{O}(d^2)$ operations for computing $(\widetilde{\mathbf{A}}'', \widetilde{\mathbf{b}}'')$. Thus, the computation times of both evaluator and CSP are highly affected as N and d increase. However, the computational tasks in this phase are highly parallelizable.

The evaluator and the CSP can both parallelize their tasks using k processors. The evaluator can partition users into k disjoint sets $\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_k$ and assign the sub-task of computing $\widetilde{\mathbf{A}}'[i]$ to processor i, where

$$\widetilde{\mathbf{A}}'[i] = \sum_{u \in \mathcal{U}_i} (\widetilde{\mathbf{x}}'_u)^\top \widetilde{\mathbf{x}}'_u$$

Then, the evaluator computes $\widetilde{\mathbf{A}}' = \sum_i \widetilde{\mathbf{A}}'[i]$ to obtain the aggregated users' data. Similarly, $\widehat{\mathbf{A}}'$, $\widetilde{\mathbf{b}}'$, and $\widehat{\mathbf{b}}'$ can be computed in parallel. For CSP, since each entry in the matrices $(\widetilde{\mathbf{A}}'', \widetilde{\mathbf{b}}'')$ can be computed independently, it can assign a processor to compute a portion of $(\widetilde{\mathbf{A}}'', \widetilde{\mathbf{b}}'')$ and combine the results without any further computations.

5.2. Security enhancements

In this section, we weaken the following assumptions of our threat model. (1) The evaluator and CSP are assumed to be honest-but-curious when executing the protocol. (2) Each user protects its private information properly and does not disclose them to any other entity.

To weaken the first assumption, we introduce a mechanism in Section 5.2.1 for the evaluator and the CSP to verify the correctness of computation results. To weaken the second assumption, Section 5.2.2 considers the situation, where some users disclose r and their δ's to the other entities.

5.2.1. Cheating-resistance

We consider the case where a malicious evaluator or CSP may intentionally corrupt the computation results or skip some computations for reducing the costs. We introduce two methods to deal with these issues. The foremost step is to enable every user u to verify the correctness of $\mathbf{w}$ by checking

$$\| \langle \mathbf{w}, \mathbf{x}_u \rangle - y_u \| \le \epsilon$$

for a small ϵ. If the verification does not pass for the majority of the users, we conclude that either the evaluator or CSP is dishonest in computation. Hence, the malicious evaluator or CSP cannot cause an incorrect $\mathbf{w}$ to be accepted by users.

The second method is to equip the evaluator and the CSP with the capability of validating each other's work. The evaluator is potentially motivated to misbehave and obtain some information about users' data. This violates our ultimate goal of preserving users' privacy. For instance, the evaluator may ignore the input of all users and only execute the rigid regression on the data provided by a single user. The resulting output could potentially leak some information about the targeted user. To deal with this issue, we ask the evaluator and the CSP to produce a compact zero-knowledge proof of their tasks to ensure the correctness of their computation.

5.2.2. Collusion-resistance

The collusion cases are rationally explained as follows.

Case 1 *Collusion attack from a subset of users.* A subset of collusive users are unable to learn anything about private data contributed by the other users. This is due to the secure communication channels between a user and the evaluator.

Case 2 *Collusion attack between the evaluator and a subset of users.* When the evaluator obtains r from user $u \in \mathcal{U}$, it can decrypt another user v's data $(\widetilde{\mathbf{x}}'_v, \widetilde{y}'_v)$, $v \notin \mathcal{U}$, by computing

$$\mathbf{x}'_v = \left[\frac{\widehat{x}'_{v,i}}{T_i^r}\right]_{d\times 1} = \left[x_{v,i} + (i+1)\delta_v\right]_{d\times 1},$$

$$y'_v = \frac{\widehat{y}'_v}{T_0^r} = y_v + \delta_v.$$

However, the evaluator is still unable to obtain $x_{v,\,i}$ or y_v since there are $d+1$ equations with $d+2$ unknown variables, δ_v and $(\mathbf{x}_v, y_v)$'s, in $(\mathbf{x}'_v, y'_v)$.

Case 3 *Collusion attack between CSP and a subset of users.* When CSP obtains r and $(\delta_u, (\mathbf{x}_u, y_u))$, $u \in \mathcal{U}$, the aggregated matrix $\mathbf{A}'$ contains $\frac{d^2+d}{2} + d$ equations with $(N - N')(d+2)$ unknown variables δ_v and $(\mathbf{x}_v, y_v)$ for $v \notin \mathcal{U}$. Hence, CSP cannot obtain user v's data $x_{v,\,i}$'s and y_v when $N - N' > \frac{d^2+3d}{2(d+2)} = \frac{d}{2}\frac{d+3}{d+2} \approx \frac{d}{2}$. Thus, users' data privacy is preserved against this collusion attack when the number of non-collusive users is greater than $\frac{d}{2}$.

5.3. Support for real numbers

In our protocol, the data message spaces for ElGamal (Π) and Paillier (Ψ) cryptosystems are $\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$ and $\mathbb{Z}_n = \{0, 1, \ldots, n-1\}$, respectively, where p and n are λ-bit numbers and $n \geq p$. However, in real world scenarios, users' data could contain real values. Therefore, we need to represent the real values as group elements in $\mathbb{Z}_p$ and $\mathbb{Z}_n$ in order to use our protocol correctly. This further implies that, we need to deal with negative numbers. In practice, there are many standard solutions for this issue in the literature [15]. We explain a simple and efficient solution to encrypt the real numbers under Π and Ψ as follows.

Decimal numbers. We use fix-point number representation to represent a real number as a decimal number with a fixed number of digits for the fractional part [15]. We introduce a scaling factor $\frac{1}{10^{\mu}}$ to scale up the input values as integers. After the protocol execution, we scale the computed result back to the original one by removing the factors according to the performed operations. More precisely, we represent an input number x as an integer $[x]$ with scaling factor of $\frac{1}{10^{\mu}}$, where

$$[x] = \lfloor x \cdot 10^{\mu} \rfloor$$

μ is a fixed positive integer.

All the operations in our protocol can be done in their integral version with the following scaling rules.

- Addition: $[a+b] = [a] + [b]$
- Multiplication: $[a \cdot b] = \frac{[a] \cdot [b]}{10^{\mu}}$
- Division: $[\frac{a}{b}] = \frac{[a]}{[b]} \cdot 10^{\mu}$

We use these rules to scale the computed results back to the original ones. Note that μ can be treated as a system parameter that is chosen for a desirable accuracy.

Negative numbers. For a negative number x, it is sufficient to shift x to the second half of $\mathbb{Z}_p$, i.e., $\{\frac{p-1}{2}+1, \frac{p-1}{2}+2, \ldots, p-1\}$, with $-1 = p-1 \bmod p$, $\ldots$, $-\frac{p+1}{2} = \frac{p-1}{2} + 1 \bmod p$. The non-negative numbers are then kept as the numbers in the first half of $\{0, 1, \ldots, \frac{p-1}{2}\}$. If the final computed result y is greater than $\frac{p}{2}$, we shift it back to $y - p$. This limits the value of input x within the interval $[-\frac{p+1}{2}, \frac{p-1}{2}]$. By choosing a sufficiently large p and an adequate bit length of the input data, we can prove that our protocol executes correctly.

6. Experiment and comparison

This section shows the evaluation results of our PPRR protocol over large users' data. We provide numerical results in Section 6.1 and a comparison of our protocol with the state-of-the-art ones in the subsequent section.

6.1. Numerical results

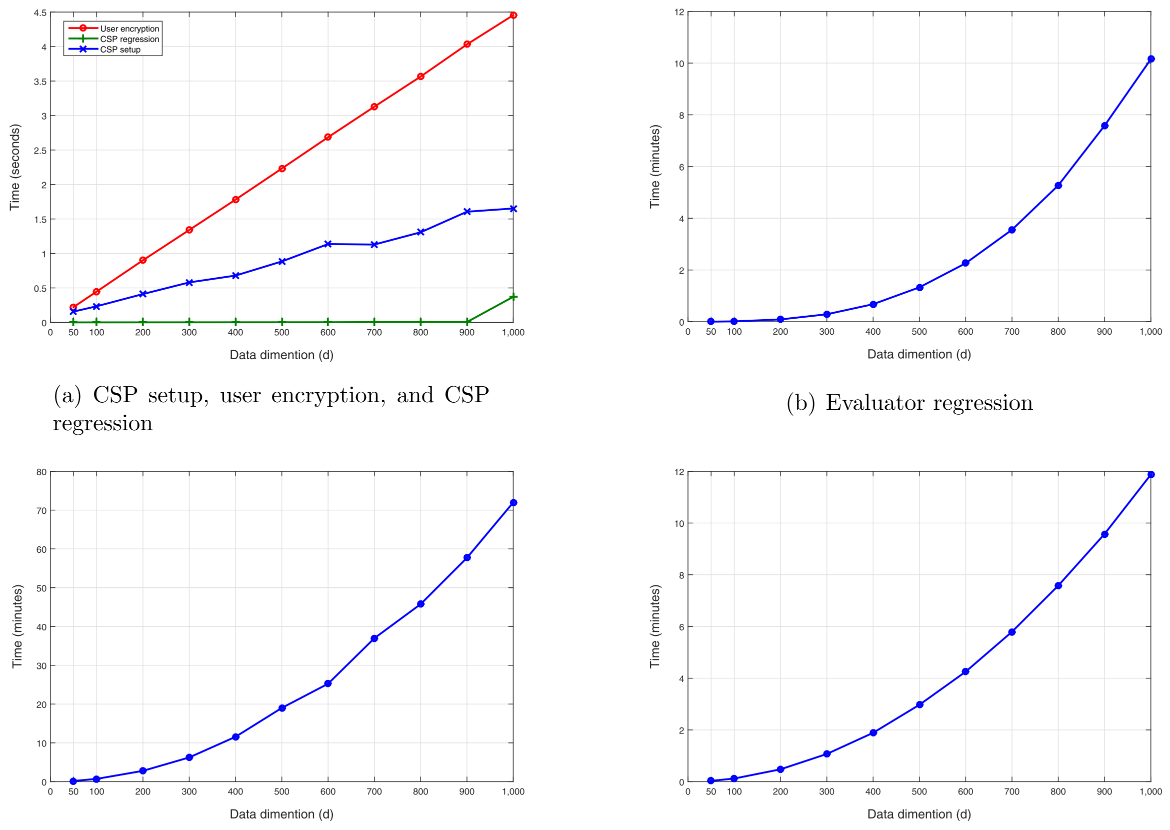
In this section, we show the computation time of executing each phase of our protocol. To show the impact of the input parameters over the performance, we synthetically generate user datasets for a different user number N and data dimension d. The experiments are performed on a single machine with 8-cores CPU at 1.5 GHz 64 GB RAM, running Ubuntu Linux 14.04. We implemented our protocol using C++ with GMP library 6.1.1. In our implementation, the evaluator and the CSP run their tasks in parallel with 4 processors and users employ a single processor. We set the security parameter $\lambda = 1,024$, that is, we use ElGamal and Paillier encryption schemes with a 1,024-bit modulus. We use 30 bits to represent a number, it can roughly represent a decimal number with 10 digits.

User data generation. We generate a random matrix $\mathbf{X} = \left[x_{i,j}\right]_{N \times d}$, where $x_{i,j}$ is a real number uniformly chosen from the interval [0,1]. Each row $\mathbf{x}_i$ of $\mathbf{X}$ represents a user's data, $1 \leq i \leq N$. To generate the corresponding output y_i of $\mathbf{x}_i$, for $1 \leq i \leq N$, we uniformly generate $\mathbf{w} = \left[w_i\right]_{N \times 1}$ with $w_i \in [0, 1]$ and compute $\mathbf{y} = \left[y_i\right]_{N \times 1}$ with $y_i = \sum_{j=1}^{d} x_{i,j} w_j$. We treat $\mathbf{X}$ and $\mathbf{y}$ matrices as the users' data in our protocol.

Execution time of each phase. Fig. 2 shows the execution time of each phase in our protocol. The x-axis is the data dimension (d), from 50 to 1000 dimensions. The y-axis is the execution time on the given dimension (d). Fig. 2(a) shows the computation time of CSP in both system setup and regression phase, and a user in data encryption. The results are linear in the data dimension d. For instance, when $d = 1,000$, it takes 4.4 s for a user to encrypt its data. It takes about 2.3 s for CSP to set up the system and decrypt the data in the regression phase.

Fig. 2(b) shows the computation time of the evaluator in regression phase. It is the time for solving a linear system by the LDLT decomposition with $O(d^3)$ operations, forward substitution with $O(d^2)$ operations, and backward substitution with $O(d^2)$ operations. When $d = 1000$, it takes about 10.1 min for the evaluator to execute the regression phase.

In Fig. 2(c) and 2(d), we investigate the computation time of the evaluator and the CSP in aggregation phase, respectively. The computation time of evaluator and CSP grows proportionally in Nd^2 and d^2, respectively. For instance, when $d = 1000$ and $N = 10^4$, it takes about 71.9 min for the evaluator to aggregate user's data. CSP needs about 12min for re-encrypting the aggregated data matrix.



(a) CSP setup, user encryption, and CSP regression

(b) Evaluator regression

(c) Evaluator aggregation (parallelized with 4 processors)

(d) CSP aggregation (parallelized with 4 processors)

Fig. 2. The execution time of each phase in our protocol.

6.2. Theoretical and numerical comparisons

In this section, we compare our protocol with Nikolaenko et al.'s [22] and Hu et al.'s [14] protocols in both theoretical and numerical perspectives. The compared protocols are shown to outperform the related protocols in implementation based on SMPC, garbled circuit, FHE, secret sharing, and hybrid methods. The theoretical and numerical comparison results are detailed in Table 4 and Table 5 respectively.

Theoretical comparison. In Table 4, we give an asymptotic comparison of each phase in our protocol with that in Nikolaenko et al.'s hybrid approach [22], and Hu et al. [14]'s Gaussian and Jacobi approaches. Nikolaenko et al. use the Paillier encryption scheme for data encryption and aggregation. They implement the secure regression computation via an enhanced Yao's garble circuit. Then the circuit is jointly executed by evaluator and CSP. Hu et al. also use the Paillier encryption scheme for data encryption and aggregation. In regression, they develop a packed secure multiplication protocol (PSMP) for the evaluator and the CSP to compute the data model with some interactions under the Gaussian's and Jacobi's methods.

In the setup phase, our protocol requires CSP to generate and publish $O(d)$ keys to the other entities. Nikolaenko et al. require CSP to generate a garble circuit $\mathfrak{C}_{\mathrm{CSP}}$ and send it to the evaluator. The size of $\mathfrak{C}_{\mathrm{CSP}}$ is large for high dimensional data since it is proportional to d^3. For instance, a circuit for 20-dimensional data with 24-bit numbers is about 270 MB [22]. The evaluator then runs an OT-protocol with CSP to acquire the garble input values of two random masking matrices. This step takes $O(d^2)$ operations for both evaluator and CSP to accomplish. The most efficient setup phase is the one in Hu et al. [14]'s approach. It only requires CSP to generate one pair of Paillier's keys.

In the encryption phase, our protocol provides the most efficient solution since each user only needs $O(d)$ operations to encrypt its data. The computation cost is asymptotically optimal since it is linear to the data dimension d. In contrast, the encryption cost in other protocols are proportional to d^2.

In the aggregation phase, the evaluator in all protocols requires $O(Nd^2)$ operations, which causes a heavy cost when N or d increases. In our protocol, the evaluator additionally performs $O(d^2)$ modular multiplication and addition operations for data aggregation. However, our protocol is able to provide a more efficient communication cost than the other protocols. Each user in our protocol only needs to submit $O(d)$ group elements to the evaluator, i.e., there are $O(Nd)$ communication

Table 4
A theoretical comparison of our PPRR protocol with the previous ones.

	NWIJBT [22]	Hu et al. [14]	Hu et al. [14]	Ours
Setup phase				
CSP computation	$O(\|\mathfrak{C}_{CSP}\| + d^2)$	$O(1)$	$O(1)$	$O(d)$
Evaluator computation	$O(d^2)$	idle	idle	idle
Communication	$O(\|\mathfrak{C}_{CSP}\| + d^2)$	$O(1)$	$O(1)$	$O(d)$
Encryption phase				
User computation	$O(d^2)$	$O(d^2)$	$O(d^2)$	$O(d)$
Aggregation phase				
CSP computation	$O(d^2)$	idle	idle	$O(d^2)$
Evaluator computation	$O(Nd^2)$	$O(Nd^2)$	$O(Nd^2)$	$O(Nd^2)$
Communication	$O(Nd^2)$	$O(Nd^2)$	$O(Nd^2)$	$O(Nd + d^2)$
Regression Phase				
CSP computation	idle	$O(d^3)$	$O(d^2)$	$O(d)$
Evaluator computation	$O(\|\mathfrak{C}_{CSP}\|)$	$O(d^3)$	$O(d^2)$	$O(d^3)$
Communication	idle	$O(d^3)$	$O(d^2)$	$O(d)$
Communication round, building block, and security				
Communication round	$O(1)$	$O(d)$	$O(\log_2 d)$	$O(1)$
Building Blocks	Yao's GC and Paillier enc.	Paillier enc.	Paillier enc.	ElGamal and Paillier enc.
Data privacy level	IND-CPA	IND-CPA	IND-CPA	data irrecoverable
Non-threat	collusion of the evaluator and CSP			

† $\mathfrak{C}_{CSP}$ is a garbled circuit for implementing a linear system solver based on the Cholesky decomposition with $O(d^3)$ operations. $|\mathfrak{C}_{CSP}|$ denotes the number of gates in $\mathfrak{C}_{CSP}$.
† N is the number of total users.
† d is the dimension of user data.

Table 5
A numerical comparison of our PPRR protocol with the previous ones.

	NWIJBT [22]	Hu et al. [14]	Hu et al. [14]	Ours (Java)	Ours (C++)
CSP's total computation time	N/A	15 s	50.7 s	8.3 s	0.86 s
Evaluator's total computation time	144 s[b]	4.2 s[a]	1.5 s[a]	209 s[b]	8.84 s
Total communication cost	770 MB	97.74 MB	97.58 MB	7.5 MB	7.5 MB
Choice of parameters (λ, d, N)	(2^{10}, 20, 10^4)	(2^{10}, 20, 1994)		(2^{10}, 20, 10^4)	
Environment	Java Runtime Engine 1.7				C++
Machine CPU	1.9 GHz	3.1 GHz		2.8 GHz	1.5 GHz
Machine RAM	64 GB	12 GB		4 GB	64 GB
Machine type	server	desktop		desktop	server
Machine OS	Ubuntu 12.04	Windows 10		Windows 7	Ubuntu 14.04

[a] The user number N in [14] is 1994, which is about 1/5 of $N = 10^4$ in other experiments. Since the computation cost in aggregation phase grows proportional to N, we think that the times 4.2 s and 1.5 s in [14] should be about 3 to 5 times slower for a fair comparison with other results.
[b] At the first glance, the evaluator's total computation time of our protocol (implemented in Java) and Nikolaenko et al.'s [22] protocol are much higher than Hu et al.'s [14] protocols. It is because that in Hu et al.'s [14] protocol, they use a "batch" Paillier encryption on users' data. As the discussion in Nikolaenko et al.'s [22] result, their protocol and our protocol can also apply batching technique for 30 elements in data encryption. It can significantly decrease the evaluator's computation time and total communications cost.

cost in data collection. In contrast, Nikolaenko et al.'s and Hu et al.'s approaches require $O(Nd^2)$ in total since each user needs to submit a $d \times d$ encrypted data matrix to the evaluator. Notice that in our and Nikolaenko et al.'s protocols, the evaluator and the CSP need an additional communication for decrypting the aggregated data matrix. In contrast, the evaluator in Hu et al. [14] is able to accomplish data aggregation by itself.

In the regression phase, our evaluator solves a linear system with $O(d^3)$ operations and asks CSP for decrypting the learning parameters with $O(d)$ communication and computation costs. The evaluator in Nikolaenko et al. protocol is able to compute the final result by using $\mathfrak{C}_{CSP}$ and garble inputs. In Hu et al. [14]'s protocol, the evaluator and the CSP need $O(d)$ and $O(\log_2 d)$ interactions for solving a linear system with Gaussian's and Jacobi's methods, respectively. It also requires the evaluator and CSP to jointly perform (d^3) and $O(d^2)$ operations.

According to the above discussions, although our protocol has many improvements in efficiency, however, our protocol can only guarantee data irrecoverable security. All other protocols achieve IND-CPA security since their data are protected under the Paillier encryption scheme. The construction of an IND-CPA (or stronger) secure PPRR with the same (or better) efficiency remains opened.

Numerical comparison. In Table 5, we provide a numerical comparison. We compare the total computation time of the evaluator and the CSP, in addition to the communication cost in protocol execution. The numerical values of other protocols are extracted from their evaluations. Our protocol is implemented using C++ language, however, we also provide an evaluation result implemented in Java language running on a desktop PC for a fair comparison with other results. In ours and Nikolaenko et al.'s experiments, we choose $\lambda = 1,024$, $d = 20$, and $N = 10^4$. Hu et al. [14] provide their result with the same values of λ and d, but smaller $N = 10^3$.

We first conclude that our protocol and Hu et al.'s protocol are more efficient than Nikolaenko et al.'s protocol in computation and communication costs. The main difference is from the use of garble circuit. As we know, there is a certain complexity in building the garble circuit for linear system solver and preparing the corresponding garble inputs. For example, a garble circuit with 30-bit for fix-point representation and $d = 12$ contains about 1.75×10^7 gates, and it takes about 50 s to be executed [22]. It significantly increases the execution time and communication costs for privacy-preserving ridge regression. For example, the total communication cost of Nikolaenko et al.'s in Table 5 is 770 MB, which is notably higher than 97 MB and 7.5 MB in Hu et al.'s and our protocols, respectively. We also observe that Hu et al.'s result achieved a great reduction in computation time. For example, our evaluator's total computation time, in Java implementation, is higher since the evaluator requires additional $O(Nd^2)$ operations over $\mathbb{G}$. Such improvement significantly decreased their data encryption time and communication cost. Nevertheless, our C++ parallel implementation shows that the evaluator's computation time is more suitable to be applied in practice.

Secondly, we observe that our protocol is more efficient than Hu et al.'s protocol in CSP's computation and communication costs as shown in Table 5. However, the evaluator's total computation time in our Java implementation is much higher than that of Hu et al.'s protocol. It is due to the fact that their N is about 1/5 of ours and they use "batch" encryption technique. As discussed in Nikolaenko et al.'s [22], their and our protocol can also employ batching technique for 30 elements in data encryption. Therefore, by either increasing N in Hu et al.'s protocol or adopting "batch" encryption technique in our protocol, our Java implementation outperforms Hu et al.'s protocol.

7. Conclusion

In this paper, we propose an efficient cryptographic protocol for large scale ridge regression. We designed our protocol based on secure summation methods, multiple ElGamal, and Paillier encryption schemes. We provide a security analysis to show that our protocol protects the private data. The experimental results on high-dimensional real world datasets show that our protocol offers a reasonable scalability. The theoretical and numerical comparisons show that our protocol outperforms state-of-art results based on secure multi-party computation, garbled circuit, fully homomorphic encryption, secret-sharing, and hybrid methods.

As a future work, we aim to develop a more efficient and secure PPRR protocol. We are also interested to extend our results to solve a more complex problem such as classification.

References

[1] R. Agrawal, R. Srikant, Privacy-preserving data mining, in: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD), 2000, pp. 439–450.

[2] D. Beaver, Commodity-based cryptography (extended abstract), in: Proceedings of the ACM Symposium on the Theory of Computing (STOC), 1997, pp. 446–455.

[3] M. Bellare, A. Boldyreva, J. Staddon, Randomness re-use in multi-recipient encryption schemeas, in: Proceedings of the Public Key Cryptography Conference (PKC), 2003, pp. 85–99.

[4] J.C. Benaloh, Secret sharing homomorphisms: keeping shares of a secret sharing, in: Proceedings of the Advances in Cryptology (CRYPTO), 1986, pp. 251–260.

[5] D. Bogdanov, L. Kamm, S. Laur, V. Sokk, Rmind: a tool for cryptographically secure statistical analysis, IEEE Trans. Dependable Secure. Comput. PP (99) (2017). 1–1

[6] C. Clifton, M. Kantarcioglu, J. Vaidya, X. Lin, M.Y. Zhu, Tools for privacy preserving data mining, SIGKDD Explor. 4 (2) (2002) 28–34.

[7] M.D. Cock, R. Dowsley, A.C.A. Nascimento, S.C. Newman, Fast, privacy preserving linear regression over distributed datasets based on pre-distributed data, in: Proceedings of the ACM Workshop on Artificial Intelligence and Security (AISec), 2015, pp. 3–14.

[8] W. Du, M.J. Atallah, Privacy-preserving cooperative scientific computations, in: Proceedings of the IEEE Computer Security Foundations Workshop (CSFW-14), 2001, pp. 273–294.

[9] W. Du, Y.S. Han, S. Chen, Privacy-preserving multivariate statistical analysis: Linear regression and classification, in: Proceedings of the International Conference on Data Mining (SDM), 2004, pp. 222–233.

[10] T.E. Gamal, A public key cryptosystem and a signature scheme based on discrete logarithms, IEEE Trans. Inf. Theory 31 (4) (1985) 469–472.

[11] A. Gascón, P. Schoppmann, B. Balle, M. Raykova, J. Doerner, S. Zahur, D. Evans, Privacy-preserving distributed linear regression on high-dimensional data, Proc. Priv. Enhanc. Technol. 4 (2017) 248–267.

[12] C. Gentry, Fully homomorphic encryption using ideal lattices, in: Proceedings of the Annual ACM Symposium on Theory of Computing (STOC), 2009, pp. 169–178.

[13] S. Han, W.K. Ng, L. Wan, V.C.S. Lee, Privacy-preserving gradient-descent methods, IEEE Trans. Knowl. Data Eng. 22 (6) (2010) 884–899.

[14] S. Hu, Q. Wang, J. Wang, S.S. Chow, Q. Zou, Securing fast learning! ridge regression over encrypted big data, in: Proceedings of the IEEE Trustcom/BigDataSE/ISPA, IEEE, 2016, pp. 19–26.

[15] K. Hwang, Computer arithmetic: principles, architecture and design, John Wiley & Sons, 1979.

[16] M. Kantarcioglu, C. Clifton, Privacy-preserving distributed mining of association rules on horizontally partitioned data, IEEE Trans. Knowl. Data Eng. 16 (9) (2004) 1026–1037.

[17] A.F. Karr, X. Lin, A.P. Sanil, J.P. Reiter, Secure regression on distributed databases, J. Comput. Gr. Stat. 14 (2) (2005) 1–18.

[18] A.F. Karr, X. Lin, A.P. Sanil, J.P. Reiter, Privacy-preserving analysis of vertically partitioned data using secure matrix products, J. Off. Stat. 25 (1) (2009) 125–138.

[19] K. Kurosawa, Multi-recipient public-key encryption with shortened ciphertext, in: Proceedings of the Public Key Cryptography Conference (PKC), 2002, pp. 48–63.

[20] Y. Lindell, B. Pinkas, Privacy preserving data mining, J. Cryptol. 15 (3) (2002) 177–206.

[21] M. Naehrig, K. Lauter, V. Vaikuntanathan, Can homomorphic encryption be practical? in: Proceedings of the ACM Cloud Computing Security Workshop (CCSW), 2011, pp. 113–124.
[22] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, N. Taft, Privacy-preserving ridge regression on hundreds of millions of records, in: Proceedings of IEEE Symposium on Security and Privacy (S&P), 2013, pp. 334–348.
[23] P. Paillier, Public-key cryptosystems based on composite degree residuosity classes, in: Proceedings of the EUROCRYPT, 1999, pp. 223–238.
[24] Y.N. Rob Hall Stephen E. Fienberg, Secure multiple linear regression based on homomorphic encryption, J. Off. Stat. 24 (4) (2011) 669–691.
[25] A.P. Sanil, A.F. Karr, X. Lin, J.P. Reiter, Privacy preserving regression modelling via distributed computation, in: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2004, pp. 677–682.
[26] J. Vaidya, C. Clifton, Privacy preserving association rule mining in vertically partitioned data, in: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2002, pp. 639–644.
[27] A.C.-C. Yao, How to generate and exchange secrets (extended abstract), in: Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS), 1986, pp. 162–167.
[28] J. Yi, J. Wang, R. Jin, Privacy and regression model preserved learning, in: Proceedings of the International Conference on Artificial Intelligence (AAAI), 2014, pp. 1341–1347.