1-1-2015

Secure Cloud Computing for Solving Large-Scale Linear Systems of Equations

Xuhui Chen

Secure cloud computing for solving large-scale linear systems of equations

By

Xuhui Chen

A Thesis
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Electrical and Computer Engineering
in the Department of Electrical and Computer Engineering

Mississippi State, Mississippi

December 2015

Secure cloud computing for solving large-scale linear systems of equations

By

Xuhui Chen

Approved:

Qian (Jenny) Du
(Major Professor)

James E. Fowler
(Committee Member/Graduate Coordinator)

Pan Li
(Committee Member)

Jason M. Keith
Dean
Bagley College of Engineering

Name: Xuhui Chen

Date of Degree: December 11, 2015

Institution: Mississippi State University

Major Field: Electrical and Computer Engineering

Major Professor: Prof. Qian(Jenny) Du

Title of Study: Secure cloud computing for solving large-scale linear systems of equations

Pages of Study: 37

Candidate for Degree of Master of Science

Solving large-scale linear systems of equations (LSEs) is one of the most common and fundamental problems in big data. But such problems are often too expensive to solve for resource-limited users. Cloud computing has been proposed as an efficient and cost-effective way of solving such tasks. Nevertheless, one critical concern in cloud computing is data privacy. Many previous works on secure outsourcing of LSEs have high computational complexity and share a common serious problem, i.e., a huge number of external memory I/O operations, which may render those outsourcing schemes impractical. We develop a practical secure outsourcing algorithm for solving large-scale LSEs, which has both low computational complexity and low memory I/O complexity and can protect clients privacy well. We implement our algorithm on a real-world cloud server and a laptop. We find that the proposed algorithm offers significant time savings for the client (**up to** 65%) compared to previous algorithms.

DEDICATION

To my family.

ACKNOWLEDGEMENTS

It is a genuine pleasure to express my deep sense of thanks and gratitude to my advisor and my committee members for their professional supervisory and generous help during my Master's study. They have guided me through research and even my life. I really appreciate all the support they have given to me.

Besides, I would like to extend my sincere thank all my colleagues, including Ming Li, Sergio Salinas, Arun Thapa, Weixian Liao, Kaijin Zhang, Changqing Luo, Sheng Cai, and Jinlong Ji. Discussing problems with them is such an exciting experience. Particularly, I would like to thank Dr. Sergio Salinas. He has so many innovative thoughts. To work with him is always full of surprise. This work could not be completed without their suggestions and supports. It's my great pleasure being a member of our cohesive NEST lab.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Background

In recent years, many large-scale systems and applications have emerged which deal with huge volumes of data. For example, social networks need to monitor and record interactions among millions or billions of users [3] [8]; scientists need to sequence the genome of complex organisms [10]; and power system operators collect enormous amounts of data from the electric grid for real-time monitoring and offline analysis [7]. Due to advances in computer memory, large-scale data sets can be stored in a cost-effective manner. However, analyzing them requires extensive computing resources which are usually very expensive capital investments. Therefore, both individuals and organizations face a formidable challenge in trying to analyze large-scale data sets in a timely and cost-effective way. This challenge has attracted significant attention from industry, academia and governments, who have identified it as a new technology field, called Big Data [13, 18].

Solving large-scale linear systems of equations (LSEs) of the form $\mathbf{A}\mathbf{x} = \mathbf{b}$ is one of the most common and fundamental problems in big data. But such problems are often too expensive to solve for resource-limited users. Some researchers and commercial entities suggest cloud computing as a timely, efficient, and cost-effective way of solving such computing tasks [4, 9, 12, 21, 22]. In cloud computing, clients outsource their computing tasks to the cloud, which contains a large amount of computing resources and offers them on a pay per-use basis [20]. In this computing paradigm, clients share the cloud resources with each other, and avoid purchasing, installing, and maintaining sophisticated and expensive computing hardware and software.

Nevertheless, one critical concern in cloud computing is data privacy. To be more prominent, in many cases, clients's LSEs contain private data that should remain hidden from the cloud for ethical, legal, or security reasons. For example, a person's genome could be disclosed by a computing task from a health-care provider [18]; a company's data set may reveal proprietary processes, which are an attractive target for corporate espionage; or data from power system operators contain information that can be exploited to attack the electric grid [16]. Thus, in order for people to really adopt cloud computing, we have to design tools and technologies that allow clients to outsource the computing of their LSEs to the cloud while preserving the privacy of their data. The fact that clients lack computing and storage resources also limits the complexity of operations that they can perform to hide their data from the cloud, which makes secure outsourcing an even more challenging problem.

1.2 Related Works

Some previous works on securely outsourcing computing tasks to the cloud could be used to solve LSEs. However, they suffer from high computing requirements. In [5], Gennaro et al. utilize fully homomorphic encryption (FHE) to securely outsource computations to the cloud. Although the scheme is very attractive and offers theoretical privacy guarantees, FHE itself is a computationally intensive operation, and large-scale computations based on FHE ciphertexts are very expensive, even for the cloud. Wang et al. [23] [25] propose methods to privately outsource a linear programming problem. A client may employ these methods to find the solution to an LSE by requesting the cloud to solve a special linear program. Unfortunately, to protect data privacy, the client needs to perform a matrix-matrix multiplication that is prohibitively expensive because this operation has a computational complexity of $\mathcal{O}(n^\rho)$ for $2 < \rho \leq 3$ (for $n \times n$ matrices).

Recently a few secure outsourcing algorithms have been developed specifically for solving LSEs. Lei et al. [14] and Atallah et al. [1] design secure matrix inversion algorithms that use matrix permutations to preserve data privacy. To find the solution to an LSE, a client performs operations with $\mathcal{O}(n^2)$ complexity. Wang et al. [24] develop an iterative algorithm specifically to solve LSEs, where a client transforms and encrypts the coefficient matrix using homomorphic encryption, and the cloud carries out computations on ciphertexts. Specifically, the client needs to perform two matrix-vector multiplications, which require $\mathcal{O}(n^2)$ floating-point (flops) operations, and $\mathcal{O}(n^2)$ homomorphic encryptions. Note that performing homomorphic encryptions has high computational complexity ($\mathcal{O}(\log_2 e)$ flops per encrypted value, were e is the key size). Although it is proposed that the client could outsource this computation to a trusted third-party, it may not always exist. The use of homomorphic encryption also forces the cloud to operate on ciphertexts, which then has to use specialized linear algebra software and performs operations with higher computational complexity. Besides, the proposed algorithm only works for LSEs whose coefficient matrices are diagonally dominant, and the privacy will be compromised if the number of iterations approaches or exceeds n. Later on, Chen et al. [2] also propose similar solutions to outsourcing linear programs and LSEs while preserving users' privacy. We notice that most such works' computational complexities are still high.

More importantly, previous works [1, 2, 14, 24] share a common serious problem, i.e., a huge number of memory I/O operations. This problem has been largely neglected in the previous secure outsourcing algorithm design. But we stress that the number of times an algorithm accesses a matrix is of particular importance for outsourcing a large-scale LSE and can eventually render the algorithm impractical. The reason is as follows. Most often a client lacks enough RAM memory to store a large-scale matrix completely at once. So, instead of working on RAM memory directly, as is the case with smaller matrices, the client can only load a small section of the large-scale matrix at a time and write the results to external memory when it is done. However, reading and writing operations from and into external memory have a very high latency compared to the same operations in RAM. For example, our experiments show that reading a matrix once with dimension $30,000 \times 30,000$ and size 10GB on a laptop that has 4GB RAM and a hard disk at 5400RPM would take about 15 minutes. Therefore, any practical algorithm for large-scale LSEs should only incur as small the number of memory I/O operations for the client as possible. To better capture the special memory I/O requirement of large-scale LSEs, we propose a new definition of memory I/O complexity, which is the number of values that are read/written from/into external memory. Previous works have very high storage complexity. For example, a client needs to access the elements of a matrix at least four times in [1, 14], and five times in [24], respectively, which may take an unacceptably long time due to the latency of the huge number of I/O operations in practice.

Aiming to reduce both computational and memory I/O complexities, in this paper, we develop an efficient and practical secure outsourcing algorithm for solving large-scale LSEs. Specifically, to protect its data privacy, a client generates a random matrix to transform the original coefficient matrix $\mathbf{A}$ into matrix $\hat{\mathbf{A}}$ via a matrix addition. We prove that matrix $\hat{\mathbf{A}}$ is computationally indistinguishable from a random matrix. Then, based on the conjugate gradient method, the client finds the solution vector $\mathbf{x}$ iteratively with the help of the cloud. Since the client delegates expensive matrix-vector operations to the cloud, it has computational complexity of $\mathcal{O}(n^2)$. Besides, it has very low memory I/O complexity of $4n^2 + 2n$. The algorithm preserves the privacy of the client, i.e, hides $\mathbf{A}$, $\mathbf{x}$, and $\mathbf{b}$, by letting the cloud operate on the transformed matrix $\hat{\mathbf{A}}$ and some intermediate values, rather than on $\mathbf{A}$, $\mathbf{x}$, or $\mathbf{b}$. Moreover, since matrix $\hat{\mathbf{A}}$ is the result of a linear algebra operation, the cloud can use traditional linear algebra software, and avoid the costly exponentiations required for ciphertext-based operations as in [24].

We summarize our main contributions as follows.

- We develop an efficient and practical algorithm to securely outsource the computation of large-scale LSEs
- The proposed algorithm requires operations with low computational and storage complexities from the client. In particular, the computational complexity is $\mathcal{O}(n^2)$ and the memory I/O complexity is $4n^2 + 2n$ read/write operations. We compare both complexities of our algorithm with those of previous algorithms and find that our algorithm is much more efficient.
- We show that the cloud is unable to obtain any information about the client's LSE. Different from [24] where privacy can only be protected if the algorithm converges in fewer than n iterations, the privacy in our algorithm can be protected no matter how many iterations are needed.

- We implement our algorithm on a real-world cloud server and a laptop. We find that the proposed algorithm offers significant time savings for the client (up to 65%) compared to previous algorithms.

The rest of the paper is organized as follows. In Section 2 we introduce the considered system architecture and threat model. Section 3 describes the proposed privacy-preserving matrix transformation. Section 4 presents in detail the proposed algorithm for secure outsourcing of LSEs, while Section 5 analyzes the computational complexity, storage complexity, and privacy of the proposed algorithm. We present our experimental results in Section 6, and conclude the paper in Section 7.

CHAPTER 2

PROBLEM FORMULATION

In this section, we present the system architecture considered in this paper, and introduce the threat model.

2.1 System Architecture

We consider an asymmetric two-party computing architecture as shown in Fig. 2.1, where a cloud client (CC) is resource-limited while a remote cloud server (CS) has abundant computing resources. The CC intends to solve a large-scale computing task, but cannot complete it on its own. So the CC offloads the most expensive computations to the CS and collaborates with it to find the solution to the large-scale LSE. In this work, we concentrate on the computing task of finding the solution to a large-scale LSE:

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{2.1}$$

where $\mathbf{A} \in \mathbf{R}^{m \times n}$ $(m \geq n)$ is the coefficient matrix, $\mathbf{x} \in \mathbb{R}^{n \times 1}$ is the solution vector, and $\mathbf{b} \in \mathbb{R}^{m \times 1}$ is the constant vector. We assume that $\mathbf{A}$, $\mathbf{x}$, and $\mathbf{b}$ contain, or could reveal private information of the CC.

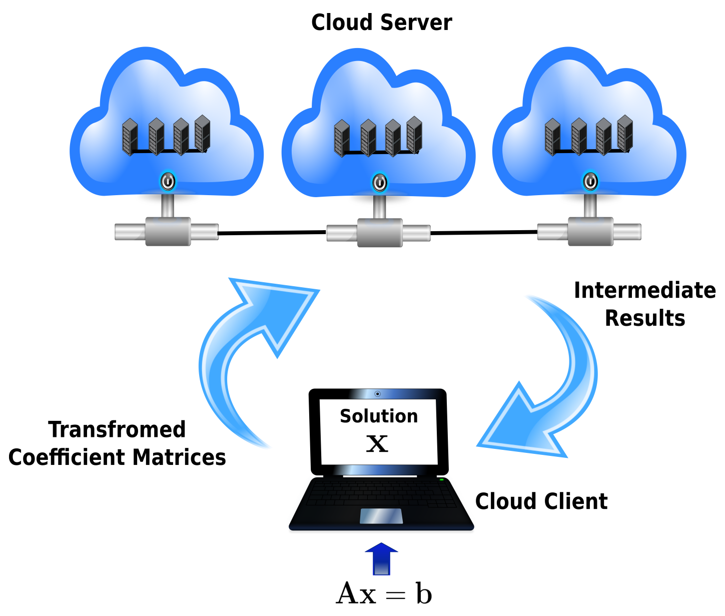

Figure 2.1

A Secure Architecture for Outsourcing LSEs

2.2 Threat Model

We assume a malicious threat model for the CS. That is, the CS tries to extract information from the CC's data and from the results of its own computations, and may attempt to deviate from the proposed protocols and return erroneous results.

To enable the CC to securely delegate computing tasks to the CS, the data that the CC shares with the CS should appear random. This notion of privacy is known as computational indistinguishability [11], and is defined as follows.

2.2.1 Definition 1 Computational Indistinguishability

Two probability ensembles $X = \{X_s\}_{n\in\mathbb{N}}$ and $Y = \{Y_s\}_{n\in\mathbb{N}}$, are computationally indistinguishable if for every probabilistic polynomial time distinguisher D there exists a negligible function $\mu(n)$ such that

$$\big|Pr[D(X_s) = 1] - Pr[D(Y_s) = 1]\big| < \mu(s) \tag{2.2}$$

where the notation $D(X_s)$ means that x is chosen according to distribution X_s and then $D(x)$ is run. Distinguisher $D(x)$ returns 1 if it determines that x is chosen according to distribution X_s.

Moreover, this definition can be extended to the case where a distinguisher D has access to multiple samples of the vectors X and Y, i.e., when comparing two matrices.

2.2.2 Definition 2

Let $\mathbf{R} \in \mathbb{R}^{m\times n}$ be a random matrix with entries in its jth column sampled from a uniform distribution with interval $[-L_j, L_j]$ $\forall j \in [1, n]$. Matrices $\mathbf{R}$ and $\hat{\mathbf{A}} \in \mathbb{R}^{m\times n}$ are computationally indistinguishable if for any probabilistic polynomial time distinguisher $D(\cdot)$ there exists a negligible function $\mu(\cdot)$ such that

$$|P[D(r_{i,j}) = 1] - Pr[D(\hat{a}_{i,j}) = 1]| \leq \mu(n) \tag{2.3}$$

where $i \in [1, m],\ j \in [1, n]$, $r_{i,j}$ is the element in the ith row and jth column of $\mathbf{R}$, and $\hat{a}_{i,j}$ is the element in the ith row and jth column of $\hat{\mathbf{A}}$. Distinguisher $D(r_{i,j})$ outputs 1 when it identifies the input as a uniform distribution in the range $[-L_j, L_j]$, and zero otherwise.

CHAPTER 3

A PRIVACY-PRESERVING MATRIX TRANSFORMATION

Before delving into details about our proposed algorithm for outsourcing large-scale LSEs, we first present a privacy-preserving matrix transformation scheme.

3.1 Conceal Information

To delegate a computing task to the CS, the CC first needs to perform some computations on its data. These computations should hide the data from the CS, require light computational effort from the CC, and at the same time allow the CS to return a meaningful result. To this end, we design a light-weight privacy-preserving matrix transformation based on matrix addition that offers computational indistinguishability, that is, a probabilistic polynomial-time algorithm is unable to differentiate between the transformed matrix and a random matrix with non-negligible probability.

In particular, the CC hides its private information in the coefficient matrix $\mathbf{A}$ by applying a matrix addition as follows:

$$\hat{\mathbf{A}} = \mathbf{A} + \mathbf{Z} \tag{3.1}$$

where $\mathbf{Z} \in \mathbb{R}^{m\times n}$ is a random matrix, and $\hat{a}_{i,j} = a_{i,j} + z_{i,j}\ \forall\ i \in [1, m],\ j \in [1, n]$ where $a_{i,j}$ is the element in the ith row and jth column of $\mathbf{A}$. We assume that the values of matrix $\mathbf{A}$ are within the range $[-K, K]$, where $K = 2^l$ $(l > 0)$ is a positive constant.

To reduce the CC's computational complexity, the random matrix $\mathbf{Z}$ is formed by a vector outer-product, i.e.,

$$\mathbf{Z} = \mathbf{u}\mathbf{v}^\top \tag{3.2}$$

where $\mathbf{u} \in \mathbb{R}^{m\times 1}$ is a vector of uniformly distributed random variables with probability density functions as follows:

$$f_{\mathcal{U}}(u_i) = \begin{cases} \frac{1}{2c} & -c < u_i < c \\ 0 & \text{otherwise} \end{cases}, \tag{3.3}$$

where $c = 2^p$ $(p > 0)$ is a positive constant, and $i \in [1, m]$. Vector $\mathbf{v} \in \mathbb{R}^{n\times 1}$ is a vector of arbitrary positive constants ranging from 2^l and 2^{l+q} $(q > 0)$. Note that element $z_{i,j} = u_i v_j$ $(\forall i \in [1, m],\ j \in [1, n])$ of matrix $\mathbf{Z}$ is the product of a random variable and a positive constant. Thus, $z_{i,j}$ is also a random variable with its probability density function defined as [15]:

$$f_{\mathcal{Z}}(g_{i,j}) = \begin{cases} \frac{1}{2L_j} & -L_j < g_{i,j} < L_j \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

where $L_j = cv_j$ $(\forall j \in [1, n])$ and hence is between 2^{p+l} and 2^{p+l+q}. We can now arrive at a theorem about the computational indistinsguishability between $\hat{\mathbf{A}}$ and a matrix with columns filled with values taken from uniform distributions.

3.2 A Theorem

Let $\mathbf{R}$ be a random matrix with entries in its jth column sampled from a uniform distribution with interval $[-L_j, L_j]$ ($\forall j \in [1, n]$). Matrices $\mathbf{R}$ and $\hat{\mathbf{A}}$ are computationally indistinguishable.

3.2.1 Proof

According to Definition 2.2.2, we need to show that $r_{i,j}$ and $\hat{a}_{i,j}$ ($\forall i \in [1, m], j \in [1, n]$) are computationally indistinguishable for matrices $\mathbf{R}$ and $\hat{\mathbf{A}}$ to be computationally indistinguishable. In particular, we show that any probabilistic polynomial time distinguisher D cannot distinguish $\hat{a}_{i,j}$ from $r_{i,j}$ for any $\forall i \in [1, m]$, $j \in [1, n]$ except with non-negligible success probability.

Recall that values from $\mathbf{R}$ and $\mathbf{A}$ are in the intervals $[-L_j, L_j]$ and $[-K, K]$, respectively. Thus, we have $\hat{a}_{i,j} \in [-K - L_j, K + L_j]$, and hence $r_{i,j}, \hat{a}_{i,j} \in [-2^\kappa, 2^\kappa]$ where $\kappa = p + l + q + 1$. The best strategy for distinguisher D when presented with a sample $x = \hat{a}_{i,j}$ is to return $b \leftarrow \{0, 1\}$ with equal probability if $-L_j \leq x \leq L_j$, and 1 if $x < -L_j$ or $x > L_j$. Therefore, when $x = \hat{a}_{i,j}$, we have that the success probability of the distinguisher is given by

$$\begin{aligned} Pr[D(\hat{a}_{i,j}) &= 1] \\ &= \frac{1}{2} Pr[-L_j \leq \hat{a}_{i,j} \leq L_j] \\ &\quad + Pr[\hat{a}_{i,j} < -L_j] + Pr[\hat{a}_{i,j} > L_j] \\ &= \frac{1}{2}(1 - Pr[\hat{a}_{i,j} < -L_j] - Pr[\hat{a}_{i,j} > L_j]) \\ &\quad + Pr[\hat{a}_{i,j} < -L_j] + Pr[\hat{a}_{i,j} > L_j] \end{aligned} \tag{3.5}$$

where

$$\begin{aligned} Pr[\hat{a}_{i,j} > L_j] &= Pr[a_{i,j} + z_{i,j} > L_j] \\ &= Pr[z_{i,j} > L_j - a_{i,j}] \\ &\leq Pr[z_{i,j} > L_j - K] \\ &= \frac{K}{2L_j} \end{aligned} \tag{3.6}$$

Similarly, we find that $Pr[\hat{a} < -L_j] \leq \frac{K}{2L_j}$. Consequently, we have that the probability of success for distinguisher D, when $x = \hat{a}_{i,j}$, is bounded as follows:

$$Pr[D(\hat{a}_{i,j} = 1)] \leq \frac{1}{2} + \frac{K}{2L_j} \tag{3.7}$$

On the other hand, if $x = r_{i,j}$, we can obtain that $Pr[D(r_{i,j}) = 1] = \frac{1}{2}$. According to equation (2.3), for any $\forall i \in [1, m]$, $j \in [1, n]$ we get that

$$|Pr[D(\hat{a}_{i,j}) = 1] - Pr[D(r_{i,j}) = 1]| \leq \frac{K}{2L_j} \tag{3.8}$$

Note that $K = 2^l$ and $L_j \in [2^{p+l}, 2^{p+l+q}]$. Thus, we have

$$\mu(\kappa) = \frac{K}{2L_j} \leq \frac{2^l}{2^{p+l}} = \frac{1}{2^p} = \frac{1}{2^{\kappa-l-q-1}} \tag{3.9}$$

which is a negligible function. This concludes the proof.

CHAPTER 4

SECURE OUTSOURCING OF LARGE-SCALE LSES

In this section, we develop a practical and light-weight algorithm to securely outsource a large-scale LSE to the CS based on the conjugate gradient method (CGM).

4.1 The Conjugate Gradient Method

We notice that solving the LSE in (2.1) is equivalent to solving the following unconstrained quadratic program

$$\min f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{A}'\mathbf{x} - \mathbf{b}'\mathbf{x} \tag{4.1}$$

where $\mathbf{A}'$ is symmetric and positive definite [19]. Therefore, we use the CGM algorithm that solves the above optimization problem to solve (2.1).

Specifically, as any gradient directions (GD) method, the CGM employs a set of vectors $\mathcal{P} = \{\mathbf{p}_0, \mathbf{p}_1, \dots \mathbf{p}_n\}$ that are conjugate with respect to $\mathbf{A}'$, that is, at iteration k the following condition is met:

$$\mathbf{p}_k^\top \mathbf{A}'\mathbf{p}_i = 0 \text{ for } i = 0, \dots, k-1. \tag{4.2}$$

Using the conjugacy property of vectors in $\mathcal{P}$, we can find the solution in at most n steps by computing a sequence of solution approximations as follows:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k \tag{4.3}$$

where α_k is the one-dimensional minimizer of (4.1) along $\mathbf{x}_k + \alpha_k \mathbf{p}_k$. The minimizer α_k can be found by setting (4.1) to zero and taking its gradient when $\mathbf{x} = \mathbf{x}_{k+1}$

$$\nabla f(\mathbf{x}_{k+1}) = \mathbf{A}'\mathbf{x}_{k+1} - \mathbf{b}' \;\; = \;\; 0. \tag{4.4}$$

By replacing $\mathbf{x}_{k+1}$ with (4.3) and multiplying by $\mathbf{p}_k^\top$ from the left, we get

$$\alpha_k = \frac{-\mathbf{r}_k^\top \mathbf{p}_k}{\mathbf{p}_k^\top \mathbf{A}' \mathbf{p}_k} \tag{4.5}$$

where $\mathbf{r}_k = \mathbf{A}'\mathbf{x}_k - \mathbf{b}'$ is called the residual.

Moreover, we can find the residual iteratively based on (4.3) as follows:

$$\begin{aligned} \mathbf{r}_{k+1} &= \mathbf{A}'\mathbf{x}_{k+1} - \mathbf{b}' \\ &= \mathbf{A}'(\mathbf{x}_k + \alpha_k \mathbf{p}_k) - \mathbf{b}' = \mathbf{r}_k + \alpha_k \mathbf{A}'\mathbf{p}_k. \end{aligned} \tag{4.6}$$

Efficiently finding the set of conjugate vectors $\mathcal{P}$ is a major challenge in GD methods. The CGM algorithm offers an efficient way of finding $\mathcal{P}$ that has low storage and computational complexity. In particular, the CGM finds a new conjugate vector $\mathbf{p}_{k+1}$ at iteration k by a linear combination of the negative residual, i.e., the steepest descent direction of $f(\mathbf{x})$, and the current conjugate vector $\mathbf{p}_k$, that is,

$$\mathbf{p}_{k+1} = -\mathbf{r}_{k+1} + \beta_{k+1}\mathbf{p}_k \tag{4.7}$$

where β_{k+1} is chosen in such a way that $\mathbf{p}_{k+1}^\top$ and $\mathbf{p}_k$ meet condition (4.2). By multiplying $\mathbf{p}_k^\top \mathbf{A}'$ from the left in (4.7), we get

$$\mathbf{p}_k^\top \mathbf{A}' \mathbf{p}_{k+1} = -\mathbf{p}_k^\top \mathbf{A}' \mathbf{r}_{k+1} + \mathbf{p}_k^\top \mathbf{A}' \beta_{k+1} \mathbf{p}_k, \tag{4.8}$$

which leads to

$$\beta_{k+1} = \frac{\mathbf{p}_k^\top \mathbf{A}'(\mathbf{p}_{k+1} + \mathbf{r}_{k+1})}{\mathbf{p}_k^\top \mathbf{A}'\mathbf{p}_k}. \tag{4.9}$$

Since as mentioned above $\mathbf{p}_k^\top$ and $\mathbf{p}_{k+1}^\top$ are conjugate with respect to $\mathbf{A}'$, we have $\mathbf{p}_k^\top \mathbf{A}'\mathbf{p}_{k+1} = 0$. Note that $\mathbf{p}_k^\top \mathbf{A}'\mathbf{r}_{k+1}$ is a scalar and $\mathbf{A}$ is symmetric. Thus, we have

$$\beta_{k+1} = \frac{\mathbf{r}_{k+1}^\top \mathbf{A}'\mathbf{p}_k}{\mathbf{p}_k^\top \mathbf{A}'\mathbf{p}_k}. \tag{4.10}$$

Moreover, since $\mathbf{x}_k$ minimizes $f(\mathbf{x})$ along $\mathbf{p}_k$, it can be shown that $\mathbf{r}_k^\top \mathbf{p}_i = 0$ for $i = 0, 1, \ldots, k-1$ [19]. Using this fact and equation (4.7), a more efficient computation for (4.5) can be found, namely,

$$\begin{aligned}\alpha_k &= \frac{-\mathbf{r}_k^\top(-\mathbf{r}_k + \beta_k \mathbf{p}_{k-1})}{\mathbf{p}_k^\top \mathbf{A}'\mathbf{p}_k} \\ &= \frac{\mathbf{r}_k^\top \mathbf{r}_k}{\mathbf{p}_k^\top \mathbf{A}'\mathbf{p}_k}. \end{aligned} \tag{4.11}$$

Similarly, using (4.6), we can find a more efficient formulation for β_{k+1}. First, we replace $\mathbf{A}'\mathbf{p}_k$ with $\frac{1}{\alpha}(\mathbf{r}_{k+1} - \mathbf{r}_k)$ in (4.10) to get

$$\beta_{k+1} = \frac{\mathbf{r}_{k+1}^\top(\mathbf{r}_{k+1} - \mathbf{r}_k)}{\mathbf{p}_k^\top(\mathbf{r}_{k+1} - \mathbf{r}_k)} \tag{4.12}$$

Then, using the fact that $\mathbf{p}_k^\top \mathbf{r}_{k+1} = 0$ and $\mathbf{r}_{k+1}^\top \mathbf{r}_k = 0$ [19], we find that

$$\beta_{k+1} = -\frac{\mathbf{r}_{k+1}^\top \mathbf{r}_{k+1}}{\mathbf{p}_k^\top \mathbf{r}_k}. \tag{4.13}$$

By replacing $\mathbf{p}_k$ with $-\mathbf{r}_k + \beta_k \mathbf{p}_{k-1}$ above, and applying $\mathbf{p}_{k-1}^\top \mathbf{r}_k = 0$, we get

$$\beta_{k+1} = \frac{\mathbf{r}_{k+1}^\top \mathbf{r}_{k+1}}{\mathbf{r}_k^\top \mathbf{r}_k}. \tag{4.14}$$

To summarize the above, the CGM algorithm is as follows. At iteration $k = 0$, we have

$$\mathbf{r}_0 = \mathbf{A}'\mathbf{x}_0 - \mathbf{b}' \tag{4.15}$$

$$\mathbf{p}_0 = -\mathbf{r}_0 \tag{4.16}$$

$$k = 0 \tag{4.17}$$

and at iteration $k > 0$ we have the following iterative equations:

$$\alpha_k = \frac{\mathbf{r}_k^\top \mathbf{r}_k}{\mathbf{p}_k^\top \mathbf{A}'\mathbf{p}_k} \tag{4.18}$$

$$\mathbf{r}_{k+1} = \mathbf{r}_k + \alpha_k \mathbf{A}'\mathbf{p}_k \tag{4.19}$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k \tag{4.20}$$

$$\beta_{k+1} = \frac{\mathbf{r}_{k+1}^\top \mathbf{r}_{k+1}}{\mathbf{r}_k^\top \mathbf{r}_k} \tag{4.21}$$

$$\mathbf{p}_{k+1} = -\mathbf{r}_{k+1} + \beta_{k+1}\mathbf{p}_k \tag{4.22}$$

Compared to other methods, e.g., Gaussian eliminations, QR decomposition, CGM offers a feasible algorithm for extremely large-scale systems.

4.2 The Privacy-preserving CGM Algorithm

In what follows, we describe our proposed privacy-preserving CGM algorithm (PPCG-M) that exploits the structure of the CGM method to securely shift the relatively more expensive operations, i.e., matrix-vector multiplications, in each iteration to the CS.

4.2.1 LSE Transformation

As shown in Section 6.2, the CGM algorithm only works with symmetric and positive definite matrices. Therefore, the CC can transform the original LSE (2.1) to the following equivalent LSE:

$$\mathbf{A}'\mathbf{x} = \mathbf{b}' \tag{4.23}$$

where $\mathbf{A}' = \mathbf{A}^\top\mathbf{A}$ is symmetric and positive definite, and $\mathbf{b}' = \mathbf{A}^\top\mathbf{b}$.

Since computing $\mathbf{A}'$ requires a matrix-matrix multiplication, which has complexity of $\mathcal{O}(n^\rho)$, the CC can outsource the computation to the CS. To be more prominent, the CC generates a random matrix $\mathbf{Z}_0 = \mathbf{u}_0\mathbf{v}_0^\top$ as described in Section 3 and then sends a masked matrix $\hat{\mathbf{A}}_0$ to the CS:

$$\hat{\mathbf{A}}_0 = \mathbf{A} + \mathbf{Z}_0. \tag{4.24}$$

As proved before, $\hat{\mathbf{A}}_0$ is computationally indistinguishable from a random matrix and hence does not reveal any information about $\mathbf{A}$. The CS carries out the following secure computation:

$$\begin{aligned} \mathbf{G} &= \hat{\mathbf{A}}_0^\top\hat{\mathbf{A}}_0 \\ &= \mathbf{A}^\top\mathbf{A} + \mathrm{M} \end{aligned} \tag{4.25}$$

where $\mathrm{M} = \mathbf{Z}_0^\top\mathbf{A} + \mathbf{A}^\top\mathbf{Z}_0 + \mathbf{Z}_0^\top\mathbf{Z}_0$. Thus, upon receiving $\mathbf{G}$, the CC can obtain the symmetric positive definite matrix $\mathbf{A}'$ by

$$\mathbf{A}' = \mathbf{G} - \mathrm{M}. \tag{4.26}$$

To avoid matrix-matrix multiplications in the calculation of $\mathbf{M}$, the CC can replace $\mathbf{Z}_0$ with $\mathbf{u}_0\mathbf{v}_0^\top$, i.e.,

$$\mathbf{M} = \mathbf{v}_0(\mathbf{u}_0^\top \mathbf{A}) + (\mathbf{A}^\top \mathbf{u}_0)\mathbf{v}_0^\top + \mathbf{v}_0(\mathbf{u}_0^\top \mathbf{u}_0)\mathbf{v}_0^\top. \tag{4.27}$$

We summarize this LSE transformation scheme in Algorithm 4.1. Next, the CC and the CS collaboratively carry out the CGM algorithm to solve $\mathbf{A}'\mathbf{x} = \mathbf{b}'$. Note that $\mathbf{A}'$ can be calculated just once for many LSEs that share the same $\mathbf{A}'$ but have different $\mathbf{b}'$'s. For example, power system operators solve many state estimation problems for system monitoring and control. These problems have different measurements, i.e., $\mathbf{b}'$'s, but the same $\mathbf{A}'$ which depends on network topology and does not change frequently. Thus, finding $\mathbf{A}'$ once is enough to solve a large number of LSEs.

Table 4.1

Algorithm 1 LSE Transformation

```
REQUIRE $CC \leftarrow \mathbf{A}$,$\mathbf{b}$
1. Generate matrix $\mathbf{Z_0}$
2. Construct $\hat{\mathbf{A}}_0$ using (4.24) and send it to CS
3. Receive $\mathbf{G}$
4. Calculate $\mathbf{A}'$ using (4.26)
5. Calculate $\mathbf{b}' = \mathbf{A}^\top \mathbf{b}$ using
ENSURE $\mathbf{A}', \mathbf{b}'$
```

4.2.2 Initialization

In the initialization step, the CC sets the initial solution vector $\mathbf{x_0}$ to a random vector of $\mathbb{R}^{n\times 1}$, and tries to compute $\mathbf{r}_0$ and $\mathbf{p}_0$ according to equations (4.15) and (4.16). Since computing $\mathbf{A}'\mathbf{x}_0$ requires a matrix-vector multiplication, the CC can outsource this computation to the CS.

Particularly, the CC generates a masked coefficient matrix $\hat{\mathbf{A}}' = \mathbf{A}' + \mathbf{Z}$, where $\mathbf{Z} = \mathbf{u}\mathbf{v}^\top$ as described in Section 3, and sends it together with $\mathbf{x_0}$ to the CS. The CS helps the CC compute the term $\mathbf{A}'\mathbf{x}_0$ in a privacy-preserving manner by computing the following intermediate value

$$\mathbf{h}_0 = \hat{\mathbf{A}}'\mathbf{x}_0. \tag{4.28}$$

Upon receiving $\mathbf{h}_0$, the CC computes the residual vector as follows

$$\begin{aligned} \mathbf{r}_0 &= \mathbf{A}'\mathbf{x}_0 - \mathbf{b}' \\ &= \mathbf{h}_0 - \mathbf{u}(\mathbf{v}^\top\mathbf{x}_0) - \mathbf{b}'. \end{aligned} \tag{4.29}$$

By computing $\mathbf{v}^\top\mathbf{x}_0$ first in equation (4.29), the CC computes vector-vector computations only, which have linear complexity. This is possible due to the fact that $\mathbf{Z}$ is a rank-one matrix and can be decomposed into an outer-vector product. If we had formed $\mathbf{Z}$ arbitrarily, the client would not experience any computational or storage complexity gains.

At the end of the initialization step, the client sets the conjugate vector $\mathbf{p}_0 = -\mathbf{r}_0$, and transmits it along with $\mathbf{r}_0$ to the CS.

4.2.3 Main Iteration

Exploring the CGM iteration, i.e., equations (4.18)-(4.22), we notice that equations (4.18) and (4.19) need matrix-vector multiplications involving the coefficient matrix $\mathbf{A}'$, while the rest of the equations only require vector-vector multiplications. We exploit these equations to design an efficient collaborative computation between the CC and the CS, where the CS helps compute (4.18) and (4.19), and the CC carries out the rest of the equations by itself. To protect the CC's privacy, the CS carries out computations with the transformed matrix $\hat{\mathbf{A}}'$, instead of $\mathbf{A}'$. In what follows, we describe a set of operations that allow the CC to efficiently find $\mathbf{x}_n$, while protecting its data privacy.

To compute α_k, the CC and the CS carry out equation (4.18) in two steps. First, the CS computes an intermediate vector

$$\mathbf{t}_k \;=\; \mathbf{p}_k^\top \hat{\mathbf{A}}' \mathbf{p}_k \tag{4.30}$$

Second, the CC finds α_k using $\mathbf{t}_k$ as follows

$$\alpha_k \;=\; \frac{\mathbf{r}_k^\top \mathbf{r}_k}{\mathbf{t}_k - (\mathbf{p}_k^\top \mathbf{u})(\mathbf{v}^\top \mathbf{p}_k)}. \tag{4.31}$$

Similarly, the CC exploits the CS's resources to find $\mathbf{r}_{k+1}$. The CS first calculates the intermediate vector

$$\mathbf{f}_k \;=\; \hat{\mathbf{A}}' \mathbf{p_k} \tag{4.32}$$

which allows the CC to compute $\mathbf{r_{k+1}}$ as follows

$$\mathbf{r}_{k+1} \;=\; \mathbf{r}_k + \alpha_k(\mathbf{f}_k - \mathbf{u}\mathbf{v}^\top \mathbf{p}_k). \tag{4.33}$$

Note that when calculating α_k and $\mathbf{r}_{k+1}$ we have also used the fact that $\mathbf{Z}$ is rank-one to provide computational gains to the CC. That is, the CC carries out the computations of α_k and $\mathbf{r}_{k+1}$ in linear time via vector-vector multiplications.

Equations (4.20)-(4.22) only require vector-vector operations, hence they all can be computed entirely by the CC itself. At the end of the kth iteration, the CC transmits $\mathbf{p}_{k+1}$

to the CS for the next iteration $k+1$. Iterations terminate according to the stopping criteria suggested by Golub and Van Loan [6], i.e., $\sqrt{\mathbf{r}_k^\top \mathbf{r}_k} \leq \nu ||b||_2$, where ν is a tolerance value.

We summarize the PPCGM algorithm for the CC in Algorithm 4.2. Moreover, we note that since the CS has an economic incentive to allocate less computational resources to the CC and return erroneous solutions, the CC should be able to verify the results from the CS. In particular, at the end of the algorithm the CC can multiply $\mathbf{A}'$ by the obtained solution vector $\mathbf{x}$, and compare the product to the constant vector $\mathbf{b}'$. As in [24], the solution vector $\mathbf{x}$ can be deemed correct if $||\mathbf{A}'\mathbf{x} - b_p'|| \leq \epsilon$, where ϵ is a small value. Since the result verification is not the main focus of this paper, we refer the readers to other works for more detailed discussions.

Table 4.2

Algorithm 2 A Privacy-Preserving Conjugate Gradient Method (PPCGM)

```
**REQUIRE** $CC \leftarrow \mathbf{A}'$,$\mathbf{b}'$
1. Generate $\mathbf{u}$, $\mathbf{v}$, and $\mathbf{Z}$ using (3.2)
2. Calculate $\hat{\mathbf{A}}' = \mathbf{A}' + \mathbf{Z}$ and transmit it and $\mathbf{x}_0$ to the CS
3. Receive $\mathbf{h}_0$
4. Calculate $\mathbf{r}_0$ using (4.29) and $\mathbf{p}_0 = -\mathbf{r}_0$, and transmit to cloud
5. WHILE $\sqrt{\mathbf{r}_k^\top \mathbf{r}_k} > \nu||\mathbf{b}||_2$ do
6. Receive $\mathbf{t}_k$ and $\mathbf{f}_k$
7. Compute $\alpha_k$ using (4.31)
8. Compute $\mathbf{r}_{k+1}$ using (4.33)
9. Compute $\mathbf{x}_{k+1}$, $\beta_{k+1}$, $\mathbf{p}_{k+1}$ using (4.20), (4.21) and (4.22), respectively.
10. Transmit $\mathbf{p}_{k+1}$ to cloud
11. ENDWHILE
ENSURE $\mathbf{x}_n$
```

CHAPTER 5

PERFORMANCE AND PRIVACY ANALYSIS

In this section we analyze the computational and memory I/O complexity of the proposed PPCGM algorithm, and compare them with those of the previous works We also present a thorough privacy analysis. Note that previous works can only work with square coefficient matrices. To perform fair comparisons, we assume that they employ our proposed Algorithm 4.1 to securely transform an arbitrary coefficient matrix into a square matrix.

5.1 Computational Complexity

We define the computational complexity of a party as the number of floating-point (flops) operations (additions, subtractions, multiplications, and divisions), bitwise operations, and encryptions that the party needs to perform. We note that an encryption takes many flops, and a flop takes some bitwise operations. To determine the overall computational complexity for the client in PPCGM, we look into Algorithm 4.1 and Algorithm 4.2 in detail.

If the original coefficient matrix is not symmetric and definite positive, the CC runs Algorithm 4.1 to construct such a matrix in equation (4.23). To this end, the CC computes 4 matrix-matrix additions and subtractions ($\mathbf{1 \times mn + 3 \times n^2}$), 2 matrix-vector products

$(\mathbf{2} \times \mathbf{n}(\mathbf{2m} - \mathbf{1}))$, 3 outer vector products $(3 \times n^2)$, 1 inner vector product $(2n - 1)$, and 1 scalar-vector product (n), which takes $\mathbf{5mn} + \mathbf{6n^2} + \mathbf{n} - \mathbf{1}$ flops.

In line 4.2 of Algorithm 4.2, the client generates the random vector $\mathbf{u}$ and the random matrix $\mathbf{Z}$. To get $\mathbf{u}$, the client uses a pseudo-random number generator like the Mersenne Twister [17], which takes a constant number of bitwise operations per random number. To get $\mathbf{Z}$, the client multiplies $\mathbf{u}$ times the vector of constants $\mathbf{v}^\top$ via n^2 flops. We assume the constants in $\mathbf{v}$ are pre-chosen by the client. In line 4.2, the client constructs the transformed coefficient matrix $\hat{\mathbf{A}}'$ through a matrix addition, which takes n^2 flops. Thus, the total number of operations required to get $\hat{\mathbf{A}}'$ is $2n^2$ flops and Cn bitwise operations, where C is the number of bitwise operations needed to generate a random number. In line 4.2, the client computes $\mathbf{r}_0$ through one vector-vector multiplications $(2n - 1)$, one vector-scalar multiplication (n), and two vector-vector subtractions $(2n)$, which takes $5n - 1$ flops in total. Note that $\mathbf{p}_0$ can be computed by the CS. The total computational complexity of the initialization phase for the CC is $2n^2 + 5n - 1$ flops plus Cn bitwise operations.

To find α_k in line 4.2, according to equation (4.31), the client performs 4 vector-vector multiplications $(4 \times (2n - 1))$, a scalar subtraction, and a scalar division, which has a total of $8n - 2$ flops. In line 4.2, the client performs 1 vector-vector multiplication $(2n - 1)$, 2 vector-scalar multiplications $(2n)$, and 2 vector-vector additions $(2n)$ to find $\mathbf{r}_{k+1}$. This computation has a cost of $6n - 1$ flops. Similarly, we observe $\mathbf{x}_{k+1}$, β_{k+1}, $\mathbf{p}_{k+1}$ need $2n$, $4n - 1$, and $2n$ flops, respectively. Totally, the client performs $\mathbf{22Kn} - \mathbf{4K}$ flops after $\mathbf{K}$ iterations.

Therefore, the total computational complexity of the PPCGM algorithm is $\mathcal{O}(\mathbf{mn})$ flops plus $\mathcal{O}(n)$ bitwise operations.

In the scheme proposed by [14], as mentioned earlier, an arbitrary coefficient matrix is transformed into a square matrix first, which takes $\mathbf{5mn + 6n^2 + n - 1}$ flops. Then, a client encrypts its coefficient matrix by multiplying it with two permutation matrices, which takes $2n^2$ flops. Similarly, the client performs $2n^2$ multiplications to decrypt the received inverse matrix. To solve an LSE the client performs an additional matrix-vector multiplication. Thus, the total computational complexity is $\mathbf{5mn + 12n^2 - 1}$, i.e., $\mathcal{O}(\mathbf{mn})$, flops.

The secure outsourcing proposed in [24] requires a client to perform a problem transformation that takes one diagonal matrix inversion, a matrix-vector multiplication, the multiplication of diagonal matrix and a matrix with a zero diagonal, the multiplication of a diagonal matrix and a vector, and an additive homomorphic encryption of the elements of the coefficient matrix. These operations take a total of $3n^2 - n$ flops plus n^2 homomorphic encryptions. Then in each iteration the client decrypts a vector and performs a vector addition, which takes n flops and n decryptions. Considering the transformation of an arbitrary coefficient matrix into a square matrix, the total computational complexity for this work is $\mathcal{O}(\mathbf{mn})$ flops $+\mathcal{O}(n^2)$ encryptions.

A brief summary of computational complexity comparison between our algorithm and previous works is shown in Table 5.1. To facilitate the comparison between our proposed algorithm and previous works, we assume that only Algorithm 2 is executed, i.e., the input coefficient matrix $\mathbf{A}$ is symmetric and positive definite and the CS skips Algorithm 1.

5.2 Memory I/O Complexity

As mentioned before, to better capture the special memory I/O requirement of large-scale LSEs, we propose a new definition of memory I/O complexity, which is the number of values that areas follows. If the original LSE system is not symmetric and positive definite, the CC runs Algorithm 4.1, which needs to read the original coefficient matrix and write the new coefficient matrix. These operations take $\mathbf{2mn}$ I/O operations. In line 4.2 of Algorithm 4.2, to construct $\hat{\mathbf{A}}'$, the CC reads $\mathbf{A}'$ and writes of $\hat{\mathbf{A}}'$ to external memory, which takes $2n^2$ I/O operations. Computing $\mathbf{r}_0$ requires one read of $\mathbf{b}'$ which takes n I/O operations. In the main iteration phase, the CC is able to make all of its operations within the RAM memory. At the final iteration, the CC stores the solution $\mathbf{x}^*$ into the external memory, which takes n I/O operations. Therefore, the total memory I/O complexity of our scheme is no more than $4n^2 + 2n$.

In [14], the CC hides its coefficient matrix using permutation matrices that need one read of $\mathbf{A}$ and one write of the resulting matrix, which takes $2n^2$ I/O operations. The client decrypts the received inverse matrix similarly, which takes another $2n^2$ I/O operations. To find the solution vector, the CC performs a read of the inverse matrix and vector $\mathbf{b}$ and a write of the final solution, which takes $n^2 + 2n$ I/O operations. Note that transforming an arbitrary matrix into a square matrix for matrix inversion incurs $2n^2$ memory I/O operations. The memory I/O complexity in [14] for general matrices is thus $7n^2 + 2n$.

In [24], the CC protects its data by transforming the problem through a matrix transformation, which takes $3n^2 + n$ I/O operations, and by encrypting the coefficient matrix, which takes additional $2n^2$ I/O operations. At the final iteration, the CC also stores the result in external memory which takes n I/O operations. Similarly, an arbitrary matrix needs to be transformed into a square matrix first, which takes $2n^2$ I/O operations. Thus, the total memory I/O complexity for the CC is $7n^2 + 2n$ I/O operations.

A summary of memory I/O complexity comparison between our algorithm and previous works is also shown in Table 5.1.

Table 5.1

Computational and Memory I/O Complexity Comparison

Algorithm	**Computational Complexity**	**Memory I/O Complexity**	**Matrix Type**
Gennaro et. al [5]	$\mathcal{O}(n^2)$ FHE crypt ops	$6n^2 + 2n$ I/O ops	General
Wang et. al [23] [25]	$\mathcal{O}(n^\rho)$ flops	$6n^2 + 2n$ I/O ops	General
Lei et. al [14]	$\mathcal{O}(n^2)$ flops	$7n^2 + 2n$ I/O ops	General
Attallah et. al [1]	$\mathcal{O}(n^2)$ flops	$8n^2 + 2n$ I/O ops	General
Wang et. al [24]	$\mathcal{O}(n^2)$ flops $+\mathcal{O}(n^2)$ crypt ops	$7n^2 + 2n$ I/O ops	Diagonally Dominant
Our scheme	$\mathcal{O}(n^2)$ flops + $\mathcal{O}(n)$ bit ops	$4n^2 + 2n$ I/O ops	General

5.3 Privacy Analysis

Exploring the PPCGM algorithm proposed in Section 4, we observe that the CS only has access to the transformed coefficient matrix $\hat{\mathbf{A}}'$ and the conjugate vector $\mathbf{p}_k$. According to Theorem 3.2, the transformed matrix $\hat{\mathbf{A}}'$ is computationally indistinguishable from a random matrix. Thus, the CS cannot derive any information about the coefficient matrix $\mathbf{A}'$ from the transformed matrix $\hat{\mathbf{A}}$.

We also observe that the CS is unable to derive information about the solution vector $\mathbf{x}_n$. Specifically, to calculate $\mathbf{x}_n$ the CS needs the knowledge of α_k, which is calculated with $\mathbf{r}_k$. However, the CC keeps α_k and $\mathbf{r}_k$ private. We also note from (4.22) that even if the CS stores $\mathbf{p}_k$ for all k, it cannot calculate $\mathbf{r}_k$ because β_k is kept private by the CC. Moreover, from (4.19), $\mathbf{r}_k$ also remains unknown from the CS since it would need the coefficient matrix $\mathbf{A}'$ to find it.

In addition, by keeping α_k and $\mathbf{r}_k$ private, the CC also prevents the CS to learn about the vector $\mathbf{b}'$ and hence $\mathbf{b}$.

We also note that different from [24] where privacy can only be protected if the algorithm converges within n iterations, the privacy in our algorithm can be protected no matter how many iterations are needed.

CHAPTER 6

EXPERIMENTAL RESULTS

In this section, we evaluate the computational and memory I/O complexity of the proposed scheme for secure outsourcing of large-scale LSEs. We implement both the CC and the CS parts of the algorithm in Matlab 2013b. We run the CC on a laptop with a dual-core 2.4GHz CPU, 4GB RAM memory, and a 320GB hard disk at 5,400RPM. The CS is implemented on Amazon Elastic Compute Cloud (EC2). As explained in Section 4.2.3, transforming $\mathbf{A}$ into $\mathbf{A}'$ can be done just once for many LSEs. Therefore, we focus on the performance of solving $\mathbf{A}'\mathbf{x} = \mathbf{b}'$ with coefficient matrices of dimension $n \times n$, with n ranging from $5,000$ to $30,000$.

6.1 Computing Complexity

We first explore the computing time of our algorithm, in which the CC shifts most of the computational burden to the cloud by outsourcing matrix-vector computations. We compare our results to the iterative algorithm in [24] with 768-bit encryptions, which is their best performing case. We notice that [24] employs the Jacobi method to solve diagonally dominant matrices, which may not converge when applied to a general matrix. Thus, we compare the time that the CC takes to complete the computations in each iteration of both algorithms. The results are summarized in Table 6.1. We observe that the CC is able to complete each iteration much faster in our scheme than in [24]. For example, in the case when $n = 5,000$ the CC in our algorithm completes an iteration in only 0.7ms, while it takes 27s in [24], a difference of five orders of magnitude. We also notice that as the dimension of the matrix increases, our performance gain is even more obvious. This is due to the use of regular arithmetic in our scheme but the use of the computationally expensive homomorphic decryptions in [24].

Table 6.1

Comparison of Average Computing Time Per Iteration for the CC

Matrix Size	**Our Algorithm**	**[24]**
$n = 5,000$	0.70 ms	27.82 s
$n = 8,000$	0.72 ms	46.06 s
$n = 10,000$	0.76 ms	56.32 s
$n = 30,000$	1.50 ms	121.81 s

6.2 Memory I/O complexity

We then evaluate the total memory I/O complexity of our algorithm. Again, since [24] may not converge when applied to a general matrix, we compare such cost of our scheme with that of [14]. We can see from Table 6.2 that our algorithm has much lower memory I/O cost compared to [14], which is consistent with our I/O complexity analysis. For example, when $n = 5,000$, the CC's total memory I/O access time is 3.4 minutes in our scheme while 6.0 minutes in [14]. When n goes to $30,000$, the CC's total memory I/O access

Table 6.2

Comparison of Total Memory I/O Access Time for the CC

Matrix Size	**Our Algorithm**	**[14]**
$n = 5,000$	3.4 min	6.0 min
$n = 8,000$	13.5 min	23.6 min
$n = 10,000$	14.2 min	24.8 min
$n = 15,000$	23.4 min	40.8 min
$n = 30,000$	64.7 min	171.3 min

time is 64.7 minutes in our scheme while 171.3 minutes in [14]. This shows a significant difference, i.e., up to 62% time saving in our algorithm.

We also explore the total running time of the proposed algorithm in Table 6.3. We disregard the communication time with the cloud so that we can focus on the total computing and memory I/O access time. We observe that the total running time savings offered by our algorithm are very attractive. For example, in the case of $n = 5,000$ our scheme solves the large-scale LSE in 3.6 minutes, while the scheme in [14] takes 6.2 minutes, indicating 42% time saving in our algorithm. Moreover, in the case of $n = 30,000$, the total running time of our algorithm is 66.9 minutes, compared to a total of 192.7 minutes in [14]. Thus, our algorithm achieves as high as 65% time saving, which is very impressive. In addition, from Table 6.2 and Table 6.3, we can tell that memory I/O operations lead to a very significant part of the total running time, which shows the impact that memory I/O complexity has on the overall algorithm performance. Therefore, low I/O complexity is indispensable for a practical outsourcing algorithm.

We finally plot Fig. 6.1 to more clearly compare the total running time of our algorithm with that of [14]. In accordance to our theoretical results, we observe that the total running time in each algorithm grows quadratically as the dimension of the coefficient matrix n increases. We also notice that the time saving of our algorithm becomes more and more significant compared to that of [14] as n increases.

Table 6.3

Comparison of Total Running Time

Matrix Size	**Our Algorithm**	**[14]**
$n = 5,000$	3.6 min	6.2 min
$n = 8,000$	13.7 min	24.6 min
$n = 10,000$	14.5 min	26.7 min
$n = 15,000$	23.9 min	43.6 min
$n = 30,000$	66.9 min	192.7 min

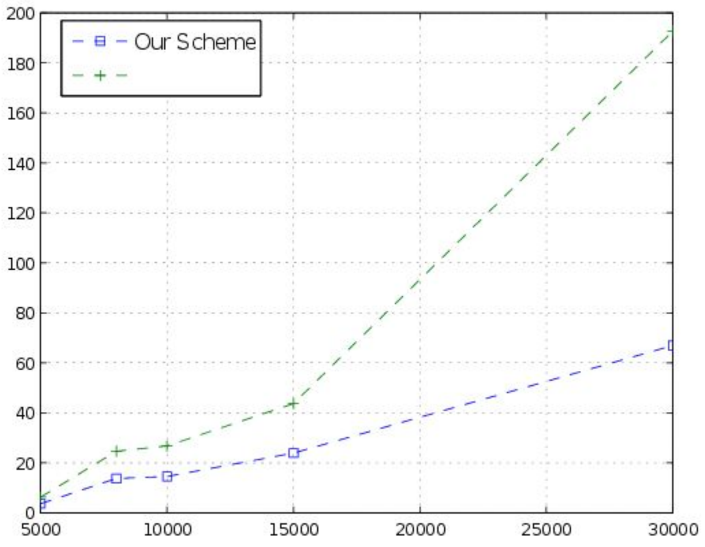

Figure 6.1

The total running time of our algorithm compared with that of [14].

CHAPTER 7

CONCLUSIONS AND FUTURE WORKS

In this thesis, we have investigated the problem of securely outsourcing large-scale sparse LSEs. In particular, to protect the cloud client's privacy, we develop a privacy-preserving matrix transformation based on linear algebra and show that the resulting matrix is computationally indistinguishable from a random one. Then, we propose a algorithm based on the conjugate gradient method that can solve large-scale LSEs efficiently while preserving the client's privacy. Formal analysis shows that our proposed algorithm has much lower computational and memory I/O complexities than previous works, and protects the client's privacy well. We finally conduct extensive experiments on Amazon Elastic Compute Cloud (EC2) and find that our algorithm offers significantly less total running time compared to previous works.

REFERENCES

[1] M. J. Atallah, K. Pantazopoulos, J. R. Rice, and E. E. Spafford, "Secure outsourcing of scientific computations," *Trends in Software Engineering*, M. V. Zelkowitz, ed., vol. 54 of *Advances in Computers*, Elsevier, 2002, pp. 215 – 272.

[2] F. Chen, T. Xiang, and Y. Yang, "Privacy-preserving and verifiable protocols for scientific computation outsourcing to the cloud," *Journal of Parallel and Distributed Computing*, vol. 74, no. 3, 2014, pp. 2141 – 2151.

[3] H.-C. Chu, D.-J. Deng, and J.-H. Park, "Live Data Mining Concerning Social Networking Forensics Based on a Facebook Session Through Aggregation of Social Data," *Selected Areas in Communications, IEEE Journal on*, vol. 29, no. 7, August 2011, pp. 1368–1376.

[4] H. Demirkan and D. Delen, "Leveraging the capabilities of service-oriented decision support systems: Putting analytics and big data in cloud," *Decision Support Systems*, vol. 55, no. 1, 2013, pp. 412–421.

[5] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers," *Advances in Cryptology CRYPTO 2010*, Santa Barbara, CA, USA, 2010.

[6] G. H. Golub and C. F. V. Loan, *Matrix Computations*, The John Hopkins University Press, 2013.

[7] A. Ipakchi and F. Albuyeh, "Grid of the future," *Power and Energy Magazine, IEEE*, vol. 7, no. 2, March 2009, pp. 52–62.

[8] C. Jiang, Y. Chen, and K. Liu, "Graphical Evolutionary Game for Information Diffusion Over Social Networks," *Selected Topics in Signal Processing, IEEE Journal of*, vol. 8, no. 4, Aug 2014, pp. 524–536.

[9] U. Kang, D. Chau, and C. Faloutsos, "Pegasus: Mining billion-scale graphs in the cloud," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Kyoto, Japan, March 2012.

[10] R. R. Kao, D. T. Haydon, S. J. Lycett, and P. R. Murcia, "Supersize me: how whole-genome sequencing and big data are transforming epidemiology," *Trends in Microbiology*, vol. 22, no. 5, 2014, pp. 282–291, Special Issue: Omics: Fulfilling the Promise.

[11] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*, Chapman and Hall/ CRC, 2008.

[12] E. Kohlwey, A. Sussman, J. Trost, and A. Maurer, "Leveraging the Cloud for Big Data Biometrics: Meeting the Performance Requirements of the Next Generation Biometric Systems," *IEEE World Congress on Services (SERVICES)*, Washington DC, USA, July 2011.

[13] T. Kraska, "Finding the Needle in the Big Data Systems Haystack," *Internet Computing, IEEE*, vol. 17, no. 1, Jan 2013, pp. 84–86.

[14] X. Lei, X. Liao, T. Huang, H. Li, and C. Hu, "Outsourcing the Large Matrix Inversion Computation to a Public Cloud," *IEEE Transaction on Cloud Computing*, vol. 1, no. 1, January-June 2013, pp. 78–87.

[15] A. Leon-Garcia, *Probability, Statistics, and Random Processes for Electrical Engineers*, Prentice Hall, 2008.

[16] Y. Liu, P. Ning, and M. K. Reiter, "False Data Injection Attacks Against State Estimation in Electric Power Grids," *ACM Conference on Computer and Communications Security*, Chicago, Illinois, USA, 2009.

[17] M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator," *ACM Transactions on Modeling and Computer Simulation*, vol. 8, no. 1, January 1998, pp. 3–30.

[18] President's Council of Advisors on Science and Technology, "BIG DATA AND PRIVACY: A TECHNOLOGICAL PERSPECTIVE,", May 2014.

[19] J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer, 2006.

[20] S. Sakr, A. Liu, D. Batista, and M. Alomari, "A Survey of Large Scale Data Management Approaches in Cloud Environments," *IEEE Communications Surveys Tutorials*, vol. 13, no. 3, March 2011, pp. 311–336.

[21] E. E. Schadt, M. D. Linderman, J. Sorenson, L. Lee, and G. P. Nolan, "Computational Solutions to Large-Scale Data Mnagement and Analysis," *Nature Reviews Genetics*, vol. 11, no. 9, September 2010, pp. 647–657.

[22] Y. Simmhan, S. Aman, A. Kumbhare, R. Liu, S. Stevens, Q. Zhou, and V. Prasanna, "Cloud-Based Software Platform for Big Data Analytics in Smart Grids," *Computing in Science Engineering*, vol. 15, no. 4, July 2013, pp. 38–47.

[23] C. Wang, K. Ren, and J. Wang, “Secure and Practical Outsourcing of Linear Programming in Cloud Computing,” *International Conference on Computer Communications*, Shangai, China, 2011.

[24] C. Wang, K. Ren, J. Wang, and Q. Wang, “Harnessing the Cloud for Securely Outsourcing Large-Scale Systems of Linear Equations,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, June 2013, pp. 1172–1181.

[25] C. Wang, B. Zhang, and K. R. J. A. Roveda, “Privacy-Assured Outsourcing of Image Reconstruction Service in Cloud,” *IEEE Transactions on Emerging Topics in Computing*, vol. 1, no. 1, June 2013, pp. 166–177.