Efficient Secure Outsourcing of Large-scale Quadratic Programs

Sergio Salinas
Department of Electrical Engineering and Computer Science
Wichita State University
Wichita, KS, USA
salinas@cs.wichita.edu

Changqing Luo
Department of Electrical Engineering and Computer Science
Case Western Reserve University
Cleveland, OH, USA
cxl881@case.edu

Weixian Liao
Department of Electrical Engineering and Computer Science
Case Western Reserve University
Cleveland, OH, USA
wxl393@case.edu

Pan Li
Department of Electrical Engineering and Computer Science
Case Western Reserve University
Cleveland, OH, USA
lipan@case.edu

ABSTRACT

The massive amount of data that is being collected by today's society has the potential to advance scientific knowledge and boost innovations. However, people often lack sufficient computing resources to analyze their large-scale data in a cost-effective and timely way. Cloud computing offers access to vast computing resources on an on-demand and pay-per-use basis, which is a practical way for people to analyze their huge data sets. However, since their data contain sensitive information that needs to be kept secret for ethical, security, or legal reasons, many people are reluctant to adopt cloud computing. For the first time in the literature, we propose a secure outsourcing algorithm for large-scale quadratic programs (QPs), which is one of the most fundamental problems in data analysis. Specifically, based on simple linear algebra operations, we design a low-complexity QP transformation that protects the private data in a QP. We show that the transformed QP is computationally indistinguishable under a chosen plaintext attack (CPA), i.e., CPA-secure. We then develop a parallel algorithm to solve the transformed QP at the cloud, and efficiently find the solution to the original QP at the user. We implement the proposed algorithm on the Amazon Elastic Compute Cloud (EC2) and a laptop. We find that our proposed algorithm offers significant time savings for the user and is scalable to the size of the QP.

CCS Concepts

•Mathematics of computing → Quadratic programming; •Security and privacy → Distributed systems security; Mathematical foundations of cryptography; **•Theory of computation →** *Parallel algorithms;*

Keywords

Secure outsourcing; Quadratic programs; Parallel computing; Big data

ASIA CCS '16, May 30-June 03, 2016, Xi'an, China

ISBN 978-1-4503-4233-9/16/05…$15.00

DOI: http://dx.doi.org/10.1145/2897845.2897862

1. INTRODUCTION

From universities to multinational companies, modern organizations collect huge amounts of data that have great potential to advance scientific and engineering knowledge, and accelerate innovations [6, 11, 23]. For example, biomedicine researchers can develop novel treatments by finding patterns in large-scale genomic databases [5]; e-commerce companies can offer better product recommendations by analyzing billions of customer transactions [1]; power engineers can perform real-time analysis like state estimation and power flow optimization based on the enormous amount of data collected from the electric grid [13]; and financial firms can improve their investment strategies by analyzing the daily deluge of stock market data. Obviously, we have massive data in all these fields, and such data needs to be stored, managed, and more importantly, analyzed. However, organizations face a formidable challenge in trying to analyze such massive amounts of data in a timely and cost-effective way.

In particular, it is infeasible for users to analyze large-scale data sets on traditional computer hardware due to its limited computing capacity and RAM (random access memory). To overcome this limitation, many governments have built supercomputers that can complete very heavy computing tasks, but have large installation and operating costs (in the range of tens of millions of dollars or even higher) and usually have restricted access. Besides, even an in-house computing cluster can be very expensive and may still lack enough memory and computing power to analyze large-scale data sets for big data applications [18, 24].

This challenge has attracted significant attention from industry, academia and governments. Recently, cloud computing has been proposed as an efficient and economical way for

resource-limited users to analyze massive data sets. In this computing paradigm, cloud clients outsource their computing tasks to a cloud server [9, 15, 17, 28, 29], which contains a large amount of computing resources and offers them on an on-demand and pay-per-use basis [26]. Therefore, cloud clients can share the cloud resources with each other, and avoid purchasing, installing, and maintaining sophisticated and expensive computing hardware and software.

Although most organizations recognize the advantages of cloud computing, many of them are reluctant to adopt it due to privacy concerns [22]. To be more prominent, in many cases, users' data are very sensitive and should be kept secret from the cloud for ethical, security, or legal reasons. For example, outsourcing genomic databases to the cloud could reveal the owners of the DNA sequences; employing the cloud to implement recommendation systems for e-commerce can give unauthorized access to users' shopping habits; a power company's data may disclose the topology of the system, thus enabling attacks on the electric grid [21]; and solving portfolio optimization problems at the cloud may compromise financial firms' private stock allocation strategies. In fact, users' private data is vulnerable to malicious cloud service providers, who can directly snoop into its users' data, and third-party adversaries, who can launch a number of attacks against the cloud [25], [30]. Therefore, to enable scientists and engineers to revolutionize their fields through the analysis of large-scale data, we have to design outsourcing tools that preserve their data privacy. Moreover, the fact that people usually lack computing and memory storage resources limits the complexity of the operations that they can perform to hide their data, which makes secure outsourcing an even more challenging problem.

We observe that many problems that involve sensitive large-scale data sets employ quadratic programs (QPs), i.e., optimization problems with a quadratic objective and affine constraints, as a fundamental computing block. For example, support vector machines, i.e., machine learning algorithms that can be used in genome pattern search, recommendation systems, etc., can be formulated as QPs [7]. Sequential quadratic programming, which can be used to solve complex nonlinear optimization problems in science and engineering, solves a QP at every iteration. Besides, QPs naturally arise in many practical problems like portfolio optimizations. Therefore, in this work, we concentrate on securely and efficiently outsourcing QPs, a fundamental computing task for large-scale data analysis. The problem of securely outsourcing large-scale computations is particularly challenging and different from before because the few local computing and memory resources at users greatly limit the amount of computations they can perform to protect and process their data. To the best of our knowledge, we are the first to investigate the secure outsourcing of QPs.

Some works on secure outsourcing of large-scale computations to the cloud have proposed traditional cryptographic techniques, such as homomorphic encryption, to protect the user's data and analyze them at the cloud. In particular, Gennaro et al. [10] propose fully homomorphic encryption (FHE), which allows secure outsourcing of a function to the cloud. Wang et al. [32] develop an iterative algorithm where a user and the cloud collaboratively solve a linear system of equations. To protect its privacy, the user must first apply partial homomorphic encryption on its data, which has a high computational complexity ($\mathcal{O}(\log_2 e)$ flops per encrypted value, were e is the key size). Similarly, Liu et al. [20] employ homomorphic encryption to solve gradient descent problems at the cloud. Although this scheme offers theoretical privacy guarantee, it requires the user to perform computationally expensive operations. Moreover, by using homomorphic encryption, the user forces the cloud to carry out operations on ciphertexts, which needs to be handled with specialized linear algebra software, and hence adds significant overhead to already expensive large-scale computations at the cloud.

There are a few works which securely outsource a few problems to the cloud by employing some mapping functions. Wang et al. [31] [33] privately outsource a linear programming problem by applying an affine mapping function to the objective function and constraint matrices. However, the client needs to perform a matrix-matrix multiplication, which is prohibitively expensive. In our previous work [27], we randomize the coefficient matrix of a large-scale linear systems of equations and outsource the most expensive computations of an iterative solution method to the cloud. Although its computational complexity is low, the user needs to exchange vectors with the cloud at every iteration, which may introduce communication delays. Besides, Lei et al. [19] and Atallah et al. [2] design secure matrix inversion algorithms that use linear algebra operations to offload the inversion of a matrix to the cloud. However, matrix inversion is a very computationally expensive task, even for the cloud. Moreover, when used to solve other problems, matrix inversion is usually an intermediate step (e.g., in the solution of linear systems of equations, the coefficient matrix needs to be multiplied by the constant vector after inversion), and hence it may incur heavy communication cost, and sometimes even infeasible, to communicate the matrix back to the user before the algorithm can continue. We can see that such works impose large computational complexity on the user, and/or need to transmit a large amount of data between the user and the cloud, which may not be practical for large-scale data sets.

In this paper, we develop an efficient and practical secure outsourcing algorithm for solving large-scale QPs. Specifically, the user protects its QP's private data by multiplying the objective and constraint matrices by random sparse matrices. We show that the transformed QP is computationally indistinguishable both in value and in structure under a chosen plaintext attack (CPA), i.e., CPA-secure. Then, based on the dual problem theory and the Gauss-Seidel algorithm, the cloud finds the solution to the transformed QP, and sends the results to the user, who can then efficiently find the solution to its original QP. The algorithm preserves the user's privacy by letting the cloud operate on the transformed QP, rather than on any of the original problem matrices. Since the user only performs operations with random sparse matrices, its computational complexity is $\mathcal{O}(\max\{n^2, mn\})$, where n is the optimization variable's dimension and m is the number of constraints. Moreover, we have parallelized the algorithm at the cloud, which speeds up the overall computing time. The algorithm only employs traditional linear algebra software, avoiding costly exponentiations required for cyphertext operations as in encryption-based works. Further, since the user only communicates with the cloud to find the transformed QP and receive the results, the proposed algorithm has minimal communication complexity.

We summarize our main contributions as follows.

- For the first time in the literature, we develop an efficient and practical algorithm to securely outsource large-scale QPs to the cloud.
- The proposed algorithm requires the user to perform computations with only $\mathcal{O}(\max\{n^2, mn\})$ complexity. To accelerate the computations, the cloud runs the algorithm in parallel and only employs traditional linear algebra software, which has very low computational burden as well.
- In the proposed algorithm, the user only communicates with the cloud to transform its QP and to receive the solution, resulting in a constant number of data exchanges between the user and the cloud.
- We prove that the transformed QP problem can protect the user's data privacy. In particular, the transformed QP has the property of computational indistinguishability under a CPA.
- We implement our proposed algorithm on the Amazon EC2 platform and a laptop. We find that our algorithm achieves significant time savings for the user compared to solving the problem by itself, and that its parallel speedup is scalable.

The rest of the paper is organized as follows. In Section 2, we introduce the considered system architecture and threat model. Section 3 describes our privacy-preserving matrix transformation. Section 4 describes the dual problem and its Gauss-Seidel solution method. Section 5 presents in detail the proposed algorithm for secure outsourcing of QPs. We demonstrate our experimental results in Section 7, and conclude the paper in Section 8.

2. PROBLEM FORMULATION

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

2.1 System Architecture

We consider an asymmetric two-party architecture formed by a resource-constrained user and a resource-abundant cloud. The user aims to solve a large-scale optimization problem that has a quadratic objective function and affine constraints, i.e., a quadratic programming problem of the form [3]

$$\min_{\mathbf{x}} \quad \phi(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{Q}\mathbf{x} - \mathbf{b}^\top \mathbf{x} \tag{1a}$$

$$\text{subject to} \quad \mathbf{A}\mathbf{x} \leq \mathbf{c} \tag{1b}$$

where $\mathbf{x} \in \mathbb{R}^{n\times 1}$ is the optimization variable, and $\mathbf{Q} \in \mathbb{R}^{n\times n}$ and $\mathbf{b} \in \mathbb{R}^{n\times 1}$ are the quadratic and linear coefficients, respectively. Matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ and vector $\mathbf{c} \in \mathbb{R}^{m\times 1}$ define the set of affine constraints. The user aims to find the optimal solution $\mathbf{x}^*$ such that the objective in (1a) is minimized and the inequalities in (1b) are satisfied. We assume that the coefficient matrix $\mathbf{Q}$ is positive definite and symmetric, and hence, the QP in (1) has a unique solution. We also assume that $\mathbf{A}$ is full-rank.

Due to the large size of the QP, the user is unable to solve it by itself in a feasible amount of time. Thus, to find the optimal solution, the user outsources the most computationally expensive tasks to the cloud. We show this architecture in Fig. 1.

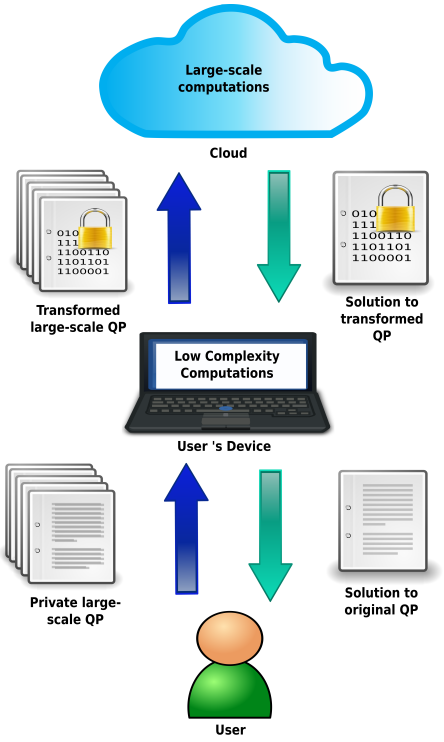


Figure 1: An Architecture for Securely Outsourcing QPs.

Besides, we denote the set of index pairs that point to non-zero elements in a matrix $\mathbf{K} \in \mathbb{R}^{m\times n}$ as follows

$$\mathcal{S}_{\mathbf{K}} = \{(i,j)|k_{i,j} \neq 0 \ \forall i \in [1,m], \forall j \in [1,n]\} \tag{2}$$

where i and j denote the ith and jth column of $\mathbf{K}$, respectively.

2.2 Threat Model

We assume a malicious threat model for the cloud. In particular, the cloud attempts to learn the user's private data from the user's outsourced data and the results of its own computations. Additionally, the cloud may deviate from the proposed protocols or return erroneous results.

Therefore, to securely outsource the computation of the QP problem, we adopt the concept of computational indistinguishability under a CPA [16]. In a matrix, we notice that the non-zero elements' values and positions both carry private information. In the following, we formally define computational indistinguishability under a CPA, i.e., CPA security, for these two types of private information, respectively.

We first introduce the definition of a pseudorandom function as follows, which will be utilized to construct matrix transformations with CPA security.

Definition 1. *Let F be a function and f a truly random function. We say F is a pseudorandom function if for all probabilistic polynomial-time distinguishers D, there exists a negligible function μ such that*

$$|Pr[D^F(1^n) = 1] - Pr[D^f(1^n) = 1]| \leq \mu \tag{3}$$

Distinguishers D^F and D^f have oracle access to functions F and f, respectively.

Definition 2. *Computational Indistinguishability in Value: We say that a matrix transformation scheme has indistinguishable transformations in value under a chosen-plaintext attack (or is CPA-secure in value) if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there exists a negligible function μ, such that the probability of distinguishing two matrix transformations in value in a CPA indistinguishability experiment is less than $1/2 + \mu$.*

Definition 2 establishes the inability of an attacker to tell apart the non-zero values in a matrix $\mathbf{K}$ from those in another matrix. Moreover, the positions of the non-zero elements in $\mathbf{K}$ (i.e., $\mathbf{K}$'s structure), contain private information that should also be hidden from the cloud.

To protect a matrix's structure, we propose to permute its rows and columns in such a way that the non-zero elements occupy positions that are indistinguishable from those of the non-zero elements in another matrix. We give the definition of secure permutation below.

Definition 3. *Computational Indistinguishability in Structure: We say that a permutation scheme has indistinguishable permutations under a chosen-plaintext attack (or is CPA-secure in structure) if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there exists a negligible function μ, such that the probability of distinguishing two permutations in a CPA indistinguishability experiment is less than $1/2 + \mu$.*

3. A PRIVACY-PRESERVING MATRIX TRANSFORMATION

To securely upload a QP problem to the cloud, the user must first conceal its private data by performing certain computations. In this section, we describe a privacy-preserving transformation that conceals a matrix's non-zero elements and structure and can be used to conceal a QP.

3.1 Privacy-preserving Matrix Multiplications

We propose that the user efficiently conceals the non-zero values of a private matrix by employing sparse random matrix multiplications. In particular, consider a private matrix $\mathbf{H} \in \mathbb{R}^{m \times n}$, with non-zero elements $h_{i,j} \leftarrow \{0,1\}^K$ for $(i,j) \in \mathcal{S}_\mathbf{H}$, where $\mathcal{S}_\mathbf{H}$ is the structure of $\mathbf{H}$ as defined in (2). The user can hide $\mathbf{H}$'s non-zero values by performing the following multiplications

$$\tilde{\mathbf{H}} = \mathbf{DHF} \tag{4}$$

In (4), $\mathbf{D} \in \mathbb{R}^{m \times m}$ is a diagonal matrix defined as

$$d_{i,j} = \begin{cases} v_i & i = j \quad \text{for } i,j \in [1,m] \\ 0 & \text{otherwise} \end{cases}.$$

The value v_i (for $i \in [1,m]$) is the ith element of a vector $\mathbf{v_H} \in \mathbb{R}^{m \times 1}$ and determined by using a pseudorandom function $F_c : \{0,1\}^w \times \{0,1\}^w \rightarrow \{0,1\}^w$, i.e.,

$$v_i = F_c(r_i, g) \ \forall i \in [1,m] \tag{5}$$

where r_i is a random string and g is a constant one. Matrix $\mathbf{F} \in \mathbb{R}^{n \times n}$ is also a diagonal matrix, i.e.,

$$f_{i,j} = \begin{cases} t_i & i = j \text{ for } i,j \in [1,n] \\ 0 & \text{otherwise} \end{cases},$$

where t_i is the ith element of vector $\mathbf{t} \in \mathbb{R}^{n \times 1}$ and set to an arbitrary positive constant.

Consequently, the non-zero elements of $\tilde{\mathbf{H}}$ in (4) are given by

$$\tilde{h}_{i,j} = d_{i,i} f_{j,j} h_{i,j} = v_i t_j h_{i,j} \tag{6}$$

for $(i,j) \in \mathcal{S}_\mathbf{H}$. We denote these computations as

$$Mask^{F_c}(r_i, t_j, h_{i,j}) = \tilde{h}_{i,j}. \tag{7}$$

We can now arrive at a theorem about the CPA-security in value of the matrix multiplications in (4).

Theorem 1. *If $F_c(\cdot,\cdot)$ is a pseudorandom function, the matrix multiplications in (4) are computationally indistinguishable in value under a CPA.*

Proof. According to Definition 2, we need to show that given two arbitrary matrices $\mathbf{H}^1$ and $\mathbf{H}^2$ with the same structure as $\mathbf{H}$, i.e., $\mathcal{S}_{\mathbf{H}^1} = \mathcal{S}_{\mathbf{H}^2} = \mathcal{S}_\mathbf{H}$, 1) $\tilde{\mathbf{H}}^1$ and $\tilde{\mathbf{H}}^2$ have the same structure; and 2) that $\tilde{h}^1_{i,j}$ and $\tilde{h}^2_{i,j}$ $(\forall (i,j) \in \mathcal{S}_{\tilde{\mathbf{H}}})$ are indistinguishable under a CPA.

To prove 1), we show that both $\tilde{\mathbf{H}}^1$ and $\tilde{\mathbf{H}}^2$ have the same structure as $\mathbf{H}$. This is because multiplications by diagonal matrices preserve the positions and the quantity of the non-zero elements. Thus, we can see that both $\tilde{\mathbf{H}}_1$ and $\tilde{\mathbf{H}}_2$ have the same structure as $\mathbf{H}$'s, i.e., $\mathcal{S}_\mathbf{H}$.

To prove 2), we need to show that a probabilistic polynomial time distinguisher D cannot distinguish $\tilde{h}^1_{i,j}$ from $\tilde{h}^2_{i,j}$ for any $(i,j) \in \mathcal{S}_\mathbf{H}$ with a probability significantly higher than $1/2$ in a CPA experiment.

Specifically, suppose a probabilistic polynomial-time adversary $\mathcal{A}$ carries out a CPA indistinguishability experiment $\mathcal{A}^{F_c}$ as shown in Algorithm 1. In particular, an adversary $\mathcal{A}$ outputs two arbitrary numbers, $h^0_{i,j}, h^1_{i,j} \leftarrow \{0,1\}^K$. A bit $b \leftarrow \{0,1\}$ is randomly chosen, and $Mask^F(r'_i, t_j, h^b_{i,j}) = \tilde{h}^b_{i,j}$ is computed and given to $\mathcal{A}$, where r'_i is a random number. $\mathcal{A}$ has oracle access to $Mask^F$ and eventually outputs b'. If $b' = b$, we say that $\mathcal{A}$ succeeds and set $\mathcal{A}^{F_c} = 1$.

Algorithm 1 A CPA Indistinguishability Experiment: $\mathcal{A}^{F_c}$

1: An adversary $\mathcal{A}$ outputs two arbitrary numbers $h^0_{i,j}, h^1_{i,j} \leftarrow \{0,1\}^K$.
2: A random bit $b \leftarrow \{0,1\}$ is chosen. $Mask^F(r'_i, t_j, h^b_{i,j}) = \tilde{h}^b_{i,j}$ is computed and given to $\mathcal{A}$, where r'_i is randomly chosen.
3: $\mathcal{A}$ continues to have oracle access to $Mask^F$ and outputs a bit b'.

Output: 1 if $b' = b$ and 0 otherwise.

Now consider an experiment $\mathcal{A}^{f_c}$ that is exactly the same as $\mathcal{A}^{F_c}$ except that a truly random function $f_c : \{0,1\}^w \rightarrow \{0,1\}^w$ is used in place of F_c. $\mathcal{A}$'s probability of success, i.e., $\mathcal{A}^{f_c} = 1$, depends on two cases:

1. *The oracle chooses the same random number r'_i used to compute $\tilde{h}^b_{i,j}$ to answer at least one of $\mathcal{A}$'s queries.* In this case, $\mathcal{A}$ can easily tell which of its values was masked and hence correctly get $b' = b$, i.e., $A^{f_c} = 1$. We denote this case as C_0.

2. *The oracle never chooses the same random number r'_i used to compute $\tilde{h}^b_{i,j}$ to answer $\mathcal{A}$' queries.* Since

$Mask^{f_c}$ outputs truly random numbers, $\mathcal{A}$ learns nothing about which of its values was masked. Thus, the probability that $\mathcal{A}$ outputs $b' = b$ is $\frac{1}{2}$. We denote this case as C_1.

Therefore, the probability of $\mathcal{A}^{f_c} = 1$, i.e., $\mathcal{A}$ succeeding, can be calculated as:

$$\begin{aligned} Pr[\mathcal{A}^{f_c} = 1] &= Pr[\mathcal{A}^{f_c} = 1|C_0]Pr[C_0] \\ &\quad + Pr[\mathcal{A}^{f_c} = 1|C_1]Pr[C_1] \\ &\leq Pr[C_0] + Pr[\mathcal{A}^{f_c} = 1|C_1] \end{aligned}$$

Since $\mathcal{A}$ is a polynomial time adversary, it can at most make $\alpha(w)$ queries to the oracle, where $\alpha(\cdot)$ is a polynomial function. Hence, in $\mathcal{A}^{f_c}$, $\mathcal{A}$ can query the oracle at most $\alpha(w)$ times. Considering that the values returned by the oracle to $\mathcal{A}$ are truly random numbers, the probability that $\mathcal{A}$ succeeds, i.e., $\mathcal{A}^{f_c} = 1$, is

$$Pr[\mathcal{A}^{f_c} = 1] \leq \frac{\alpha(w)}{2^w} + \frac{1}{2} \tag{8}$$

Next, we define the function μ as follows:

$$\mu(w) = Pr[\mathcal{A}^{F_c} = 1] - \frac{1}{2},$$

and hence we have

$$Pr[\mathcal{A}^{F_c} = 1] = \frac{1}{2} + \mu(w). \tag{9}$$

Intuitively, if $\mu(w)$ is not negligible, then the difference between (8) and (9) is also not negligible. Thus, an adversary $\mathcal{A}$ would be able to distinguish a truly random function and a pseudorandom function.

To formally prove this, we use $\mathcal{A}$ to construct a distinguisher D. To this end, D emulates the CPA indistinguishability experiment for $\mathcal{A}$ as described in Algorithm 2 and observes whether $\mathcal{A}$ succeeds or not. If $\mathcal{A}$ succeeds, D guesses that its input is a pseudorandom function, while if $\mathcal{A}$ fails, D guesses that this oracle is a truly random function.

Algorithm 2 Distinguisher D

1: D is given access to an oracle $\mathbb{O}$.
2: When $\mathcal{A}$ queries with two arbitrary numbers $h^0_{i,j}, h^1_{i,j}$, choose a random bit $b \leftarrow \{0,1\}$, compute $\tilde{h}^b_{i,j} = v_i t_j h^b_{i,j}$ where v_i is the output of the oracle $\mathbb{O}$ and t_j is as defined before, and return it to $\mathcal{A}$.
3: Continue answering any oracle queries of $\mathcal{A}$. Eventually, $\mathcal{A}$ outputs b'.

Output: 1 if $b' = b$ and 0 otherwise.

We observe that if D's oracle uses a truly random function, the view of $\mathcal{A}$ when called by D as a sub-routine is identical to its view when called by $\mathcal{A}^{f_c}$. Therefore, we have that

$$Pr[D^{f_c} = 1] = Pr[\mathcal{A}^{f_c} = 1]. \tag{10}$$

On the other hand, if D's oracle is a pseudorandom function, then the view of $\mathcal{A}$ when called by D is identically distributed to its view when called by $\mathcal{A}^{F_c}$. Thus, we get

$$Pr[D^{F_c} = 1] = Pr[\mathcal{A}^{F_c} = 1]. \tag{11}$$

Taking the difference of equations (11) and (10), we get

$$Pr[D^{F_c} = 1] - Pr[D^{f_c} = 1] \geq \mu(w) - \frac{\alpha(w)}{2^w}$$

Since we have assumed that F_c is a pseudorandom function, the term $\mu(w) - \frac{\alpha(w)}{2^w}$ must be negligible by Definition 2. Moreover, since $\alpha(w)$ is polynomial, this implies that $\mu(w)$ must also be negligible, making our value masking transformation secure under CPA.

By union bound, this concludes the proof. □

3.2 Privacy-preserving Matrix Permutations

Although the matrix transformation in equation (4) hides the values of the non-zero elements in $\mathbf{H}$, it reveals their original positions, i.e., $\mathbf{H}$'s structure, which is also private. Next, we design secure permutations that can hide $\mathbf{H}$'s structure by randomly reordering the rows and columns of $\tilde{\mathbf{H}}$.

To randomly permute $\tilde{\mathbf{H}}$'s row index vector $\mathbf{e} \in \mathbb{R}^{m\times 1}$, the user computes the following

$$\mathbf{e}' = \mathcal{M}(\mathbf{e}), \quad \hat{\mathbf{e}}' = F(\mathbf{r}, \mathbf{e}'), \quad \hat{\mathbf{e}} = \mathcal{M}^{-1}(\hat{\mathbf{e}}') \tag{12}$$

where $\mathcal{M} : \mathbb{R}^m \to \{0,1\}^k$ ($k = \lceil \log_2 m! \rceil$) is a function that maps index vectors to bit strings; $F : \{0,1\}^k \to \{0,1\}^k$ is a pseudorandom permutation; $\mathbf{r} \in \{0,1\}^k$ is a random bit string; and $\mathcal{M}^{-1} : \{0,1\}^k \to \mathbb{R}^m$ is the inverse of $\mathcal{M}$. We denote these computations as

$$Perm^F(\mathbf{r}, \mathbf{e}) = \hat{\mathbf{e}} \tag{13}$$

Similarly, we can denote by $Perm^F(\mathbf{r}', \mathbf{u})$ the random permutation of column index vector $\mathbf{u} \in \mathbb{R}^n$, where $\mathbf{r}' \in \{0,1\}^{k'}$ ($k' = \lceil \log_2 n! \rceil$) is a random bit string.

The user applies the random permutations $Perm(\mathbf{r}, \mathbf{e})$ and $Perm(\mathbf{r}', \mathbf{u})$ to $\tilde{\mathbf{H}}$ through the following multiplications

$$\hat{\mathbf{H}} = \mathbf{E}\tilde{\mathbf{H}}\mathbf{U} \tag{14}$$

where $\mathbf{E} \in \mathbb{R}^{m\times m}$ and $\mathbf{U} \in \mathbf{R}^{n\times n}$ are random permutation matrices, and their elements are defined by

$$\begin{aligned} e_{i,j} &= \delta_{\pi(i),j} \quad \forall i \in [1,m], j \in [1,m] \\ u_{i,j} &= \delta_{\pi(i),j} \quad \forall i \in [1,n], j \in [1,n] \end{aligned}$$

where i and j are the row and column indexes, respectively, and the function $\pi(\cdot)$ maps an original index i to its permuted index, i.e., $\pi(i) = \hat{e}_i$ (for $i \in [1,m]$) and $\pi(i) = \hat{u}_i$ (for $i \in [1,n]$). Besides, the Kronecker delta function is given by

$$\delta_{i,j} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}.$$

The CC is able to recover the original matrix by applying the inverse permutations, i.e.,

$$\tilde{\mathbf{H}} = \mathbf{E}^\top \hat{\mathbf{H}} \mathbf{U}^\top \tag{15}$$

where $\top$ denotes the matrix transpose operation. To reach this result, we have used the orthogonal property of permutation matrices, i.e, $\mathbf{E}^\top\mathbf{E} = \mathbf{I}$ and $\mathbf{U}\mathbf{U}^\top = \mathbf{I}$, where $\mathbf{I}$ is the identity matrix.

We now state a theorem about the CPA-security in structure of the above matrix permutations in (14).

Theorem 2. *If $F(\cdot,\cdot)$ is a pseudorandom function, then the row and column permutations described in (14) are computationally indistinguishable in structure under a CPA.*

Proof. The proof follows a similar approach to that for Theorem 1 and hence is omitted here due to space limit. □

4. THE LAGRANGE DUAL PROBLEM

Since directly solving the original QP in (1) needs computationally expensive methods, such as the active-set or interior point method, in this section, we first find an equivalent optimization problem that only has non-negativity constraints, called the Lagrange dual problem.

We first form the Lagrangian $\mathcal{L} : \mathbb{R}^{n\times 1} \times \mathbb{R}^{m\times 1} \to \mathbb{R}$ as follows [3]:

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = \quad \phi(\mathbf{x}) + \boldsymbol{\lambda}^\top(\mathbf{A}\mathbf{x} - \mathbf{c}) \tag{16}$$

where $\boldsymbol{\lambda} \in \mathbb{R}^{m\times 1}$ is the vector of dual variables, and $\mathbf{A}$ and $\mathbf{c}$ are defined in (1).

We define the Lagrange dual function $g : \mathbb{R}^{m\times 1} \to \mathbb{R}$ as the minimum value of the Lagrangian over $\mathbf{x}$ (for $\boldsymbol{\lambda} \in \mathbb{R}^{m\times 1}$), i.e.,

$$g(\boldsymbol{\lambda}) = \inf_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}). \tag{17}$$

To solve the Lagrange dual function, we take the derivative of equation (16) with respect to $\mathbf{x}$ and set it to zero as follows:

$$\mathbf{x}^\top\mathbf{Q} - \mathbf{b}^\top + \boldsymbol{\lambda}^\top\mathbf{A} = \mathbf{0} \tag{18}$$

where we have used the definition of $\phi(\cdot)$ in (1a). Taking the transpose in (18) and solving for $\mathbf{x}$, we get

$$\mathbf{x} = \mathbf{Q}^{-1}(\mathbf{b} - \mathbf{A}^\top\boldsymbol{\lambda}). \tag{19}$$

Then, by plugging (19) into (16) and considering that $\mathbf{Q}$ is symmetric, we can rewrite the Lagrange dual function as

$$\begin{aligned} g(\boldsymbol{\lambda}) = &-\frac{1}{2}\boldsymbol{\lambda}^\top\mathbf{A}\mathbf{Q}^{-1}\mathbf{A}^\top\boldsymbol{\lambda} \\ &- \boldsymbol{\lambda}^\top(\mathbf{c} - \mathbf{A}\mathbf{Q}^{-1}\mathbf{b}) \\ &- \frac{1}{2}\mathbf{b}^\top\mathbf{Q}^{-1}\mathbf{b}. \end{aligned} \tag{20}$$

By using equation (20) as the objective function and requiring $\boldsymbol{\lambda}$ to be non-negative, we arrive at the dual problem of the QP in (1), i.e.,

$$\min_{\boldsymbol{\lambda}} \quad g(\boldsymbol{\lambda}) = \frac{1}{2}\boldsymbol{\lambda}^\top\mathbf{P}\boldsymbol{\lambda} + \boldsymbol{\lambda}^\top\mathbf{r} \tag{21a}$$

$$\text{subject to} \quad \boldsymbol{\lambda} \geq \mathbf{0} \tag{21b}$$

where $\mathbf{P} = \mathbf{A}\mathbf{Q}^{-1}\mathbf{A}^\top$ and it is positive definite and symmetric, and $\mathbf{r} = \mathbf{c} - \mathbf{A}\mathbf{Q}^{-1}\mathbf{b}$. We denote the solution to (21) as $\boldsymbol{\lambda}^*$.

Since the problem (1) is convex and the affine constraints are feasible, then the strong duality holds [3] and we have that

$$\mathbf{x}^* = \mathbf{Q}^{-1}(\mathbf{b} - \mathbf{A}^\top\boldsymbol{\lambda}^*). \tag{22}$$

That is, we can use the result of the dual problem in (21) to solve the QP in (1)

5. SECURE OUTSOURCING OF LARGE-SCALE QPS

In this section, we design a secure and efficient outsourcing algorithm for large-scale QP problems based on the dual problem found in Section 4.

5.1 An Iterative Solution to QPs

Before we delve into details about our secure outsourcing algorithm, we first present the Gauss-Seidel algorithm (GSA), an iterative solution method for optimization problems that we employ to solve the dual problem in (21).

The main idea of the GSA is to iteratively update an element of the solution vector $\boldsymbol{\lambda}$ in such a way that it minimizes the objective function when all the other elements are kept constant. In particular, the GSA can solve (21) as follows:

$$\begin{aligned} \lambda_j(t+1) = \arg\min_{\lambda_j \geq 0} \; & g(\lambda_1(t+1), \ldots, \lambda_{j-1}(t+1), \\ & \lambda_j, \lambda_{j+1}(t), \ldots, \lambda_m(t)) \end{aligned} \tag{23}$$

where $j \in [1, m]$. Intuitively, the algorithm updates λ_j (for all $j \in [1, m]$) one at a time, and uses the most recent updates as they become available.

Let $(\lambda_1(t+1), \ldots, \lambda_{j-1}(t+1), \lambda_j, \lambda_{j+1}(t), \ldots, \lambda_m(t))$ be denoted as $\boldsymbol{\lambda}_j(t)$. We can solve equation (23) analytically by taking the partial derivative of $g(\boldsymbol{\lambda}_j(t))$ in (21a) with respect to λ_j and setting it to zero:

$$\frac{\partial g(\boldsymbol{\lambda}_j(t))}{\partial \lambda_j} = r_j + \mathbf{p}_j\boldsymbol{\lambda}_j(t) = 0$$

where r_j is the jth element of $\mathbf{r}$, and $\mathbf{p}_j$ is the jth row of $\mathbf{P}$.

Then, the unconstrained minimum of $g(\cdot)$ along the jth coordinate starting from $\lambda_j(t)$ is attained at

$$\lambda'_j(t+1) = \lambda_j(t) - \frac{1}{p_{j,j}}(r_j + \mathbf{p}_j\boldsymbol{\lambda}_j(t)) \qquad \forall j \in [1, m]$$

where t is the iteration index.

Thus, taking into account the non-negativity constraint and holding all the other variables constant, we get that the Gauss-Seidel iteration, when λ_j is updated, is

$$\begin{aligned} \lambda_j(t+1) &= \max\{0, \lambda'_j(t+1)\} \\ &= \max\{0, \lambda_j(t) - \frac{1}{p_{j,j}}(r_j + \mathbf{p}_j\boldsymbol{\lambda}_j(t))\}, \end{aligned} \tag{24}$$

$$\lambda_i(t+1) = \lambda_i(t) \qquad \forall i \neq j. \tag{25}$$

After all the $\lambda_i(t+1)$'s $(1 \leq i \leq m)$ are updated, the GSA iteration proceeds to the next.

5.2 A Parallel Solution to QPs

We exploit the structure of matrix $\mathbf{P}$ to derive a parallel version of the above GSA algorithm, which we call PGSA. Specifically, since $\mathbf{P}$ is a symmetric matrix, according to (24), if the kth element of $\mathbf{p}_j$ is equal to zero, i.e., $p_{j,k} = 0$ and hence $p_{k,j} = 0$, then we can update λ_j independently of λ_k.

To find the groups of λ_j's that can be updated in parallel, we first model the relationships between them as an undirected graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, were $\mathcal{V}$ is a set of vertexes and $\mathcal{E}$ is the set of edges. We build the graph $\mathcal{G}$ by placing an edge between two variables whose updates depend on each other. Thus, the vertex and edge sets are defined as

$$\begin{aligned} \mathcal{V} &= \{\lambda_1, \ldots, \lambda_m\} \\ \mathcal{E} &= \{(\lambda_i, \lambda_j) | p_{i,j} \neq 0\} \end{aligned}$$

where the notation (λ_i, λ_j) indicates there is an edge between λ_i and λ_j.

Thus, in the graph $\mathcal{G}$, a group of variables that are all connected need to be updated sequentially, and different groups can be updated in parallel. Specifically, we group variables

that belong to the same connected component of $\mathcal{G}$ into a set $\mathcal{C}_l$ ($l \in [1, L]$), where L is the total number of connected components. We denote the vector of variables in the set $\mathcal{C}_l$ as

$$\boldsymbol{\lambda}^l = (\lambda_1^l, \ldots, \lambda_{p^l}^l)$$

where $p^l = |\mathcal{C}_l|$, and the vector containing all variables as

$$\boldsymbol{\lambda} = (\boldsymbol{\lambda}^1, \ldots, \boldsymbol{\lambda}^L).$$

Note that the user can efficiently find the connected components of $\mathcal{G}$ by using the depth-first algorithm [4]. Then, the GSA sequentially updates each vector $\boldsymbol{\lambda}^l$'s variables as described in Section 5.1 and updates different vectors $\boldsymbol{\lambda}^l$ for all $l \in [1, L]$ in parallel.

In particular, we rewrite equation (23) regarding $\boldsymbol{\lambda}^l$'s variables as follows:

$$\lambda_k^l(t+1) = \arg\min_{\lambda_k^l \geq 0} g(\boldsymbol{\lambda}^1, \ldots, \lambda_1^l(t+1), \ldots, \lambda_{k-1}^l(t+1), \lambda_k^l, \lambda_{k+1}^l(t), \ldots, \lambda_{p^l}^l(t), \ldots, \boldsymbol{\lambda}^L) \quad (26)$$

for any $k \in [1, p^l]$, where vectors $\boldsymbol{\lambda}^q$ for all $q \neq l$ are assumed to be constants. Denote the above variables in $g(\cdot)$ by $\boldsymbol{\lambda}_k^l(t)$. The solution to (26) for the kth variables in $\boldsymbol{\lambda}^l$ is thus given by

$$\begin{aligned}\lambda_k^l(t+1) &= \max\{0, \lambda'^l_k(t+1)\} \\ &= \max\{0, \lambda_k^l(t) - \frac{1}{p_{k,k}}(r_k + \mathbf{p}_k \boldsymbol{\lambda}_k^l(t))\} \quad (27a)\end{aligned}$$

$$\lambda_q^l(t+1) = \lambda_q^l(t) \qquad \forall q \neq k. \quad (27b)$$

Note that in (27a) the terms $p_{k,q}\lambda_q^m = 0$ for all $m \neq l$, and thus variables in $\boldsymbol{\lambda}^l$ can be updated without knowledge of the variables in other vectors.

The iteration is carried out until the sequence $\{\boldsymbol{\lambda}(t)\}$ converges, i.e., until $||\boldsymbol{\lambda}(t) - \boldsymbol{\lambda}(t+1)||_2 \leq \nu$, where $\nu > 0$ is the stopping parameter.

Algorithm 3 The Parallel Gauss-Seidel Algorithm

Input: $\mathbf{P}$, $\mathbf{r}$
1: Build $\mathcal{G}$
2: Form the sets $\mathcal{C}_l$ $\forall l \in [1, L]$ by finding the connected components in $\mathcal{G}$
3: $\boldsymbol{\lambda}(0) \leftarrow \boldsymbol{\lambda}_0$
4: $t \leftarrow= 1$, $\nu(1) \leftarrow \nu_0$
5: **while** $\nu(t) > \nu$ **do**
6: **for** $l = 1$ **to** L **do**
7: Compute $\lambda_j(t+1)$ $\forall j \in \mathcal{C}_l$ in parallel using equation (27)
8: **end for**
9: $\nu(t+1) = ||\boldsymbol{\lambda}(t) - \boldsymbol{\lambda}(t+1)||_2$
10: $t = t + 1$
11: **end while**
Output: $\boldsymbol{\lambda}(t)$

Next, we state a theorem about the convergence and correctness of the PGSA algorithm.

Theorem 3. *Let $g(\cdot)$ be a convex and continuous function as defined in (21). Let $\{\boldsymbol{\lambda}^l(t)\}$ be the sequence generated by the PGSA algorithm when updating $\boldsymbol{\lambda}$. Then every limit point of $\{\boldsymbol{\lambda}(t)\}$ minimizes $g(\cdot)$ over all $\lambda_j \in \mathcal{C}_l$.*

Proof. We first show that the sequence $\{\boldsymbol{\lambda}(t)\}$ converges. Suppose that there is only one group of connected components $\mathcal{C}_1$, i.e.,

$$\mathcal{C}_1 = \{\lambda_j | j \in [1, m]\},$$

and thus $\boldsymbol{\lambda} = \boldsymbol{\lambda}^1$. Let

$$\mathbf{z}_j(t) = (\lambda_1(t+1), \ldots, \lambda_j(t+1), \lambda_{j+1}(t), \ldots, \lambda_m(t))$$

Since by definition $\lambda_j(t+1)$ is a solution to equation (23), we have

$$g(\boldsymbol{\lambda}(t)) \geq g(\mathbf{z}_1(t)) \geq g(\mathbf{z}_2(t)) \geq \cdots \geq g(\boldsymbol{\lambda}(t+1)). \quad (28)$$

Notice that the GSA update reduces the value of the objective function $g(\cdot)$ after each iteration.

Suppose that $\boldsymbol{\lambda}^* = (\lambda_1^*, \ldots, \lambda_m^*)$ is a limit point of the sequence $\{\boldsymbol{\lambda}(t)\}$. Then, we get $\boldsymbol{\lambda}^* \geq \mathbf{0}$. Let $\{\boldsymbol{\lambda}(t_k)\}$ be a subsequence of $\{\boldsymbol{\lambda}(t)\}$ that converges to $\boldsymbol{\lambda}^*$. From (28), we know that the sequence $\{g(\boldsymbol{\lambda}(t_k))\}$ converges to either a finite number or $-\infty$. Using the convergence of $\boldsymbol{\lambda}(t_k)$ to $\boldsymbol{\lambda}^*$ and the continuity of $g(\cdot)$, we see that $\{g(\boldsymbol{\lambda}(t_k))\}$ converges to $g(\boldsymbol{\lambda}^*)$, which implies that the entire sequence $\{g(\boldsymbol{\lambda}(t))\}$ converges to $g(\boldsymbol{\lambda}^*)$. To show that $\{g(\boldsymbol{\lambda}(t)\}$ converges when $L > 1$, we can follow a similar procedure for $\boldsymbol{\lambda}^l$ (for all $l \in [1, L]$) assuming vectors $\boldsymbol{\lambda}_q$ (for all $q \neq l$) are kept constant.

Next, we show that every limit point $\boldsymbol{\lambda}^*$ in fact minimizes $g(\cdot)$. Particularly, we first consider the case when $L = 1$ and show that $\lambda_1(t_k + 1) - \lambda_1(t_k)$ converges to zero. Assume the contrary, or equivalently, that $\mathbf{z}_1(t_k) - \boldsymbol{\lambda}(t_k)$ does not converge to zero. Let $\gamma(t_k) = ||\mathbf{z}_1(t_k) - \boldsymbol{\lambda}(t_k)||_2$. By possibly restricting to a subsequence of t_k, we may assume that there exists a $\gamma_0 > 0$ such that $\gamma(t_k) > \gamma_0$ for all k. Let $\mathbf{s}_1(t_k) = (\mathbf{z}_1(t_k) - \boldsymbol{\lambda}(t_k))/\gamma(t_k)$. Hence, we have $\mathbf{z}_1(t_k) = \boldsymbol{\lambda}(t_k) + \gamma(t_k)\mathbf{s}_1(t_k)$ and $||\mathbf{s}_1(t_k)||_2 = 1$. Note that $\mathbf{s}_1(t_k)$ belongs to a compact set and hence has a limit point $\mathbf{s}_1^*$. By restricting to a further subsequence of $\{t_k\}$, we assume that $\mathbf{s}_1(t_k)$ converges to $\mathbf{s}_1^*$.

Fixing an $\epsilon \in [0, 1]$, we see that $0 \leq \epsilon\gamma_0 \leq \gamma(t_k)$, and that $\boldsymbol{\lambda}(t_k) + \epsilon\gamma_0\mathbf{s}_1(t_k)$ is on the segment joining $\boldsymbol{\lambda}(t_k)$ and $\mathbf{z}_1(t_k) = \boldsymbol{\lambda}(t_k) + \gamma(t_k)\mathbf{s}_1(t_k)$. Due to the convexity of $g(\cdot)$, and the fact that $\mathbf{z}_1(t_k)$ minimizes $g(\cdot)$ over all $\boldsymbol{\lambda}$ that differ from $\boldsymbol{\lambda}(t_k)$ along the first component, we get

$$\begin{aligned}g(\mathbf{z}_1(t_k)) &= g(\boldsymbol{\lambda}(t_k) + \gamma(t_k)\mathbf{s}_1(t_k)) \\ &\leq g(\boldsymbol{\lambda}(t_k) + \epsilon\gamma_0\mathbf{s}_1(t_k)) \\ &\leq g(\boldsymbol{\lambda}(t_k)).\end{aligned}$$

Taking the limit as $k \to \infty$, we obtain

$$g(\boldsymbol{\lambda}^*) \leq g(\boldsymbol{\lambda}^* + \epsilon\gamma_0\mathbf{s}_1^*) \leq g(\boldsymbol{\lambda}^*)$$

where we have used the fact that $g(\mathbf{z}_1(t_k))$ converges to $g(\boldsymbol{\lambda}^*)$ due to equation (28). Thus, we have that

$$g(\boldsymbol{\lambda}^* + \epsilon\gamma_0\mathbf{s}_1^*) = g(\boldsymbol{\lambda}^*)$$

for every $\epsilon \in [0, 1]$. Therefore, this contradicts the convexity of $g(\cdot)$ because we have that $\gamma_0\mathbf{s}_1^* \neq \mathbf{0}$. This contradiction establishes that $\mathbf{z}_1(t_k) - \boldsymbol{\lambda}(t_k)$ converges to zero, and specifically, that $\mathbf{z}_1(t_k)$ converges to $\boldsymbol{\lambda}^*$.

Besides, according to equation (28), we get that

$$g(\mathbf{z}_1(t_k)) \leq g(\lambda_1, \lambda_2(t_k), \ldots, \lambda_m(t_k)) \qquad \forall \lambda_1 \geq 0.$$

Taking the limit as $k \to \infty$ and using the fact that $\mathbf{z}_1(t_k)$ converges to $\boldsymbol{\lambda}^*$, we obtain

$$g(\boldsymbol{\lambda}^*) \leq (g(\lambda_1, \lambda_2^*, \ldots, \lambda_m^*)) \qquad \forall \lambda_1 \geq 0.$$

Therefore, taking into account that $g(\cdot)$ is convex, we conclude that λ_1^* is the minimizer along the first variable, i.e.,

$$\nabla_1 g(\boldsymbol{\lambda}^*)^\top(\lambda_1 - \lambda_1^*) \geq 0, \qquad \forall \lambda_1 \geq 0.$$

By following a similar argument to the above for $j > 1$ and putting the inequalities together, we get

$$\nabla g(\boldsymbol{\lambda}^*)^\top(\boldsymbol{\lambda} - \boldsymbol{\lambda}^*) \geq 0, \qquad \forall \lambda_j \geq 0. \tag{29}$$

Since $g(\cdot)$ is convex, (29) is a necessary and sufficient condition for $\boldsymbol{\lambda}^*$ to be the global minimizer.

In the case of $L > 1$, we can follow a similar process for $\boldsymbol{\lambda}^l$ for all $l \in [1, L]$ when $\boldsymbol{\lambda}^p$ (for all $p \neq l$) is kept constant. This concludes the proof. □

5.3 A Secure Outsourcing Method for QPs

In what follows, we describe our secure QP solver (SQPS), which solves the QP in Section 2. Specifically, in the SQPS, the user and the cloud collaboratively find a secure version of the dual problem, which is then solved by the cloud. Based on the solution to the secure dual problem, the user finds the solution to the original QP.

5.3.1 Secure Formulation of the Dual Problem

As shown in Section 4, the user can find the dual problem by solving for $\mathbf{P}$ and $\mathbf{r}$ in (21). However, due to their massive size, the user is unable to compute these matrices on its own. Besides, to solve the dual problem, the user needs to securely outsource it to the cloud. In the following, we develop a scheme that allows the user to securely find and outsource the dual problem of a QP by employing the matrix transformation proposed in Section 3.

In particular, to securely find the dual problem's coefficient matrix $\mathbf{P}$ in the objective function, the user and the cloud collaborate to compute two matrix multiplications, i.e., $\mathbf{W} = \mathbf{A}\mathbf{Q}^{-1}$ and $\mathbf{P} = \mathbf{W}\mathbf{A}^\top$. To this end, the user first applies the matrix transformations (4) and (14) to $\mathbf{Q}$ and $\mathbf{A}$ in (1) as follows:

$$\bar{\mathbf{Q}} = \mathbf{V_Q}\mathbf{Q}\mathbf{T_Q},$$

where $\mathbf{V_Q}$ is formed by a random permutation matrix and a random diagonal matrix, i.e., $\mathbf{V_Q} = \mathbf{E_Q}\mathbf{D_Q}$, and $\mathbf{T_Q}$ by a diagonal matrix of arbitrary positive constants and a random permutation matrix, i.e., $\mathbf{T_Q} = \mathbf{F_Q}\mathbf{U_Q}$. Similarly, the user conceals $\mathbf{A}$ as follows:

$$\bar{\mathbf{A}} = \mathbf{V_A}\mathbf{A}\mathbf{T_Q}$$

where $\mathbf{V_A} = \mathbf{E_A}\mathbf{D_A}$.

After receiving $\bar{\mathbf{Q}}$ and $\bar{\mathbf{A}}$, the cloud inverts $\bar{\mathbf{Q}}$, i.e.,

$$\bar{\mathbf{Q}}^{-1} = \mathbf{T_Q}^{-1}\mathbf{Q}^{-1}\mathbf{V_Q}^{-1},$$

computes the matrix multiplication

$$\bar{\mathbf{W}} = \bar{\mathbf{A}}\bar{\mathbf{Q}}^{-1} = \mathbf{V_A}\mathbf{W}\mathbf{V_Q}^{-1},$$

where $\mathbf{W} = \mathbf{A}\mathbf{Q}^{-1}$, and sends $\bar{\mathbf{Q}}^{-1}$ and $\bar{\mathbf{W}}$ to the user.

To compute $\mathbf{P} = \mathbf{W}\mathbf{A}^\top$, the user finds and conceals $\mathbf{W}$, i.e.,

$$\bar{\mathbf{W}}' = \mathbf{V_W}(\mathbf{V_A}^{-1}\bar{\mathbf{W}}\mathbf{V_Q})\mathbf{T_W} = \mathbf{V_W}\mathbf{W}\mathbf{T_W}$$

where $\mathbf{V_W} = \mathbf{E_W}\mathbf{D_W}$ and $\mathbf{T_W} = \mathbf{F_W}\mathbf{U_W}$, conceals $\mathbf{A}^\top$, i.e.,

$$\bar{\mathbf{A}}' = \mathbf{T_W}^{-1}\mathbf{A}^\top\mathbf{V_{A^\top}}$$

where $\mathbf{V_{A^\top}} = \mathbf{E_{A^\top}}\mathbf{D_{A^\top}}$, and uploads $\bar{\mathbf{W}}'$ and $\bar{\mathbf{A}}'$ to the cloud. Note that $\mathbf{T_W}^{-1}$ can be easily computed by the user since it is the multiplication of a diagonal matrix and an orthogonal permutation matrix.

Then, the cloud computes

$$\bar{\mathbf{P}} = \bar{\mathbf{W}}'\bar{\mathbf{A}}' = \mathbf{V_W}\mathbf{P}\mathbf{V_{A^\top}}$$

and transmits it to the user.

In the last step, the user finds $\mathbf{P}$ as follows:

$$\mathbf{P} = \mathbf{V_W}^{-1}\bar{\mathbf{P}}\mathbf{V_{A^\top}}^{-1}.$$

Based on the cloud results, the user can also compute the linear objective coefficient vector $\mathbf{r}$:

$$\mathbf{r} = \mathbf{c} - \mathbf{W}\mathbf{b}. \tag{30}$$

5.3.2 Initialization

To securely carry out the main iteration of the PGSA at the cloud, the user conceals and uploads the dual problem's coefficient matrix $\mathbf{P}$, the linear coefficient vector $\mathbf{r}$, and the initial solution vector $\boldsymbol{\lambda}_0$. Specifically, the user computes

$$\hat{\mathbf{P}} = \mathbf{V_P}\mathbf{R_P}\mathbf{P}\mathbf{V_P}^{-1}, \tag{31}$$

where $\mathbf{R_P} > \mathbf{0}$ is a diagonal matrix, whose elements are generated by a pseudorandom function like in (5), aiming to protect values of the diagonal elements in $\mathbf{P}$, and $\mathbf{V_P} = \mathbf{E_P}\mathbf{D_P}$ is formed by random permutation and diagonal matrices as described in Section 3. Besides, the user calculates

$$\hat{\mathbf{r}} = \mathbf{V_P}\mathbf{R_P}\mathbf{r},$$

and

$$\hat{\boldsymbol{\lambda}}_0 = \mathbf{V_P}\boldsymbol{\lambda}_0, \tag{32}$$

where $\boldsymbol{\lambda}_0 \in \mathbb{R}^{m\times 1}$ and its elements are positive and uniformly chosen at random.

After receiving the above variables, to form the vectors $\hat{\boldsymbol{\lambda}}^l$ ($l \in [1, L]$) that can be updated in parallel, the cloud finds the connected components of the matrix $\hat{\mathbf{P}}$ as described in Section 5.2, and update them based on equation (27).

Note that in the original dual problem (21), we have the constraint $\boldsymbol{\lambda} \geq 0$. To enable the cloud to accurately determine the sign of $\hat{\lambda'}^l_k(t+1)$ (for all $k \in [1, p^l]$, $l \in [1, L]$) in equation (27), the user shares with the cloud the signs of the non-zero elements in $\mathbf{V_P}$ by uploading the following vector

$$\mathbf{s_{V_P}} = \mathbf{v_P} \oslash |\mathbf{v_P}|.$$

Here, since $\mathbf{V_P}$ is an $n \times n$ matrix where in each row there is only one non-zero element, we denote by $\mathbf{v_P}$ the vector of the n non-zero elements in $\mathbf{V_P}$. Moreover, $\oslash$ and $|\cdot|$ denote element-wise division and element-wise absolute value, respectively.

5.3.3 Main Iteration

The cloud carries out the PGSA update in (27) as follows:

$$\hat{\lambda'}^l_k(t+1) = \hat{\lambda}^l_k(t) - \frac{1}{\hat{p}_{k,k}}(\hat{r}_k + \hat{\mathbf{p}}_k\hat{\boldsymbol{\lambda}}^l_k(t))$$

for all $k \in [1, p^l]$, $l \in [1, L]$, where $\hat{\mathbf{p}}_k$ is the kth row of $\hat{\mathbf{P}}$.

Furthermore, suppose that $\hat{\lambda'}^l_k$ is the jth element in the vector $\hat{\boldsymbol{\lambda}}$. To check the sign of the corresponding $\hat{\lambda}^l_k(t+1)$,

the cloud employs vector $\mathbf{s_{V_P}}$ as follows:

$$\hat{\lambda}_k^l(t+1) = \begin{cases} \hat{\lambda'}_k^l(t+1), & \text{if } \hat{\lambda'}_k^l(t+1) \geq 0 \wedge s_j > 0 \\ \hat{\lambda'}_k^l(t+1), & \text{if } \hat{\lambda'}_k^l(t+1) \leq 0 \wedge s_j < 0 \\ 0, & \text{if } \hat{\lambda'}_k^l(t+1) < 0 \wedge s_j > 0 \\ 0, & \text{if } \hat{\lambda'}_k^l(t+1) > 0 \wedge s_j < 0 \end{cases}$$

where s_j is the jth element of $\mathbf{s_{V_P}}$.

The cloud continues the iteration until the vector of updates converges, that is, until $||\hat{\boldsymbol{\lambda}}^l(t) - \hat{\boldsymbol{\lambda}}^l(t+1)||_2 \leq \nu$ for all $l \in [1, L]$, where $0 < \nu < 1$ is the stopping parameter. Once the cloud determines that the algorithm has converged, it transmits $\boldsymbol{\lambda}^*$ and $\bar{\mathbf{Q}}^\top$ to the user, who computes the solution to the dual problem as follows:

$$\boldsymbol{\lambda}^* = \mathbf{V_P}^{-1}\hat{\boldsymbol{\lambda}}^*,$$

and the solution to the original QP problem by

$$\mathbf{x}^* = \mathbf{T_Q}\bar{\mathbf{Q}}^{-1}\mathbf{V_Q}(\mathbf{b} - \mathbf{A}^\top\boldsymbol{\lambda}^*).$$

Since the user only performs computations with transformation matrices and vectors, its computational complexity is quadratic regarding the QP's dimension. Specifically, to hide its private matrices in Section 5.3.1, the user transforms matrices $\mathbf{Q}$, $\mathbf{A}$, $\mathbf{W}$, $\mathbf{A}^\top$ and finds matrix $\mathbf{P}$, which take $\mathcal{O}(\max\{n^2, mn\})$ complexity. In Section 5.3.2, similarly, the user performs matrix transformations with $\mathcal{O}(n^2)$ complexity. In Section 5.3.3, to recover the solution to the original problem, the user performs matrix-vector operations and matrix transformations, which have $\mathcal{O}(\max\{n^2, mn\})$ complexity. Therefore, the overall computational complexity of the user in the algorithm is $\mathcal{O}(\max\{n^2, mn\})$, which is practical.

5.4 Result Verification

Since the cloud may return erroneous results, we propose a result verification procedure for the user based on the Karush-Kuhn-Tucker (KKT) conditions. In particular, based on the KKT conditions, the user verifies that the solution $\mathbf{x}^*$ it computed minimizes the QP's objective function $\phi(\cdot)$ and satisfies the constraints of the QP problem in (1) by evaluating the following

$$\mathbf{A}\mathbf{x}^* - \mathbf{c} \leq \mathbf{0}, \tag{33}$$

$$\boldsymbol{\lambda}^* \geq \mathbf{0}, \tag{34}$$

$$\boldsymbol{\lambda}^{*\top}(\mathbf{A}\mathbf{x}^* - \mathbf{c}) = \mathbf{0}, \tag{35}$$

$$(\mathbf{Q}\mathbf{x}^* - \mathbf{b}) + \mathbf{A}\boldsymbol{\lambda}^* = \mathbf{0}. \tag{36}$$

Equations (33) and (34) verify the feasibility of $\mathbf{x}^*$ and $\boldsymbol{\lambda}^*$, respectively. The complimentary slackness in equation (35) validates the equality of the dual and primal solutions. The fourth KKT condition in (36) shows that the solution $\mathbf{x}^*$ minimizes $\phi(\cdot)$. If the KKT conditions hold, the user determines that the solution $\mathbf{x}^*$ that it computed based on the cloud results is correct. Otherwise it determines that the cloud is acting unfaithfully. Since the evaluation of the KKT conditions only requires matrix-vector multiplications, the client is able to efficiently evaluate them.

6. PRIVACY ANALYSIS

Inspecting the proposed secure outsourcing algorithms, we observe that the cloud only has access to the securely transformed QP, and hence it is unable to learn private information from the user.

Specifically, in the process of secure formulation of the dual problem as in Section 5.3.1, the user shares with the cloud the transformed matrices $\hat{\mathbf{Q}}$, $\bar{\mathbf{A}}$, $\bar{\mathbf{A}}'$, and $\bar{\mathbf{W}}'$. According to Theorem 1 and Theorem 2, the transformed matrices are computationally indistinguishable both in value and in structure under a CPA. Thus, the cloud cannot derive any information about the elements of the original QP's matrices $\mathbf{Q}$, $\mathbf{A}$, $\mathbf{b}$, or $\mathbf{c}$ from the transformed matrices that the user uploads.

In the process of solving the dual problem as in Section 5.3.2, the user uploads $\hat{\mathbf{P}}$, $\hat{\mathbf{r}}$, and $\hat{\boldsymbol{\lambda}}_0$ to the cloud. Similarly, based on Theorem 1 and Theorem 2, the cloud is unable to determine the matrices $\mathbf{P}$, $\mathbf{r}$, or $\boldsymbol{\lambda}_0$ from $\hat{\mathbf{P}}$, $\hat{\mathbf{r}}$, or $\hat{\boldsymbol{\lambda}}_0$.

Furthermore, the cloud is unable to learn anything about the dual solution vector $\boldsymbol{\lambda}^*$ or the solution vector $\mathbf{x}^*$. In particular, for the cloud to calculate $\boldsymbol{\lambda}^*$ from $\hat{\boldsymbol{\lambda}}^*$, it needs knowledge about the elements in matrix $\mathbf{V_P}$, which the user keeps secret. We also observe that $\mathbf{x}^*$ remains unknown to the cloud because the cloud would need access to $\mathbf{T_Q}$, $\mathbf{V_Q}$, $\mathbf{b}$, and $\mathbf{A}$, but the user never uploads them.

7. EXPERIMENT RESULTS

In this section, we describe the implementation of our proposed algorithm and throughly analyze its performance.

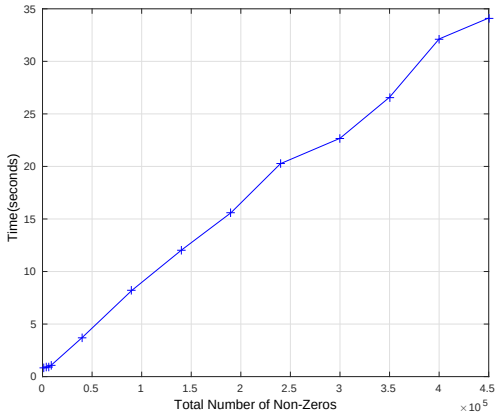
7.1 Experiment Setup

To closely replicate a real-world scenario, we run the user part of the algorithm on a laptop with a dual-core 2.4GHz CPU, 8GB RAM memory, and a 150GB solid state drive, and the cloud part on an Amazon Elastic Compute Cloud (EC2) cluster with 4 computing nodes. We implement the user and cloud parts of the algorithm on Matlab 2014b and Python's linear algebra software SciPy [14], respectively. Moreover, we employ Apache Spark [35] to manage the storage and computations at the EC2 computing cluster.

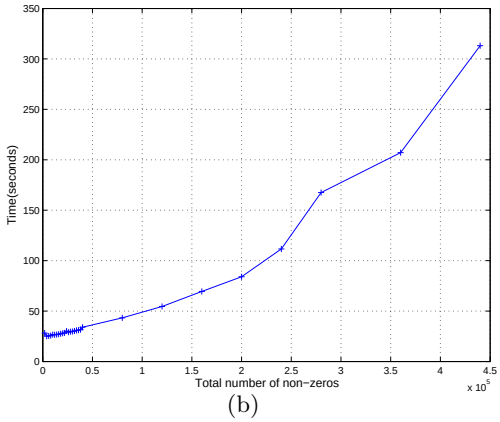
We test the performance of our algorithm with both random and real-world QPs in diverse fields, e.g., aerospace engineering, circuit design, power system analysis, and operations research, taken from well known repositories [8,12]. In addition, since efficient matrix computations at the cloud have been studied before [34], we disregard the cloud's time to perform matrix multiplication and inversion (i.e., $\bar{\mathbf{Q}}^{-1}$, $\bar{\mathbf{W}}$ and $\bar{\mathbf{P}}$). We instead focus on the time complexity of computing the PGSA algorithm and finding the connected components in $\hat{\mathbf{P}}$ at the cloud, and the time complexity of performing the QP transformation and solution computation at the user.

7.2 Computing Time

We first analyze the computing time of our secure outsourcing algorithm PGSA at both the user and at the cloud, and show the results in Fig 2. In particular, we first measure the time that the user takes to complete its part of the PPGSA algorithm, that is, the time it takes to find the transformed QP plus the time needed to find the solution $\mathbf{x}^*$. Fig. 2(a) shows the user's computing time for QPs with increasing amounts of non-zero elements. We observe that even when the QP contains a large quantity of non-zero elements the user can still finish its computing very quickly. For example, the computing time of the user when the quadratic coefficient matrix in the dual problem has 4.5×10^5 non-zero



(a)



(b)

Figure 2: Computing time of the PGSA algorithm at the user and cloud for different QP sizes. (a) Computing time at the user. (b) Computing time at the cloud.

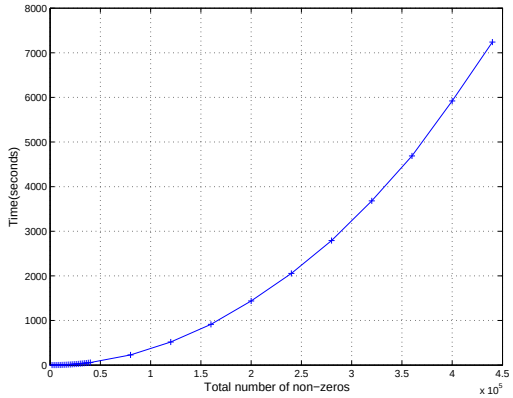


Figure 3: Computing time of the GSA algorithm at the user.

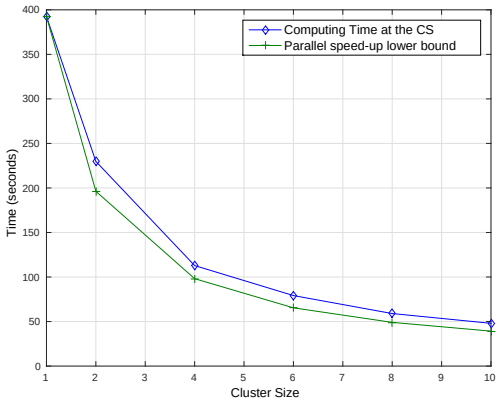


Figure 4: Comparison of computing time at the cloud and its lower bound.

elements is only 34s. Furthermore, we see that the computing time of the user is approximately linear with regard to the number of non-zero elements, which is practical for large-scale problems.

Besides, in Fig. 2(b), we show the time that the cloud takes to solving the securely transformed dual problem with different amounts of non-zero values. In particular, we measure the computing time of the iterations in PGSA as well as the depth-first algorithm to find the connected components. We find that the computing time of the cloud is low even when the size of the QP is very large. For instance, the cloud takes 324s to solve a QP with 4.5×10^5 non-zero elements in the quadratic matrix of the dual problem, which is practical in real-world scenarios.

Next, we investigate the computational savings offered by our secure outsourcing algorithm PGSA. Specifically, we compare the time when the user solves the original QP by itself using GSA with that when the user and the cloud collaborate to solve it using PGSA. We first show the time that the user takes to solve QPs with increasing amounts of non-zero values by itself in Fig. 3, and we can see that it has a quadratic complexity. Then, we compare the overall computing times of the local GSA and PGSA algorithms in Table 1. We first observe that our secure outsourcing algorithm PGSA has significantly shorter computing time compared to solving the QP by the user itself with the GSA algorithm. For example, the user by itself takes over 7000s to solve a QP with 4.5×10^5 non-zeros elements in the quadratic coefficient matrix of the dual problem, while the user and the cloud can collaboratively solve the same problem in less than 350s, which represents a 95% reduction in computing time. This is due to the ability of our algorithm to exploit the cloud's extensive resources and run in parallel. Notice that we are only using a cluster with 4 nodes, and we already see significant computing time savings for all QP sizes.

7.3 Scalability

To explore the scalability of our algorithm, we compare the computing time of the cloud with different cluster sizes with the least computing time that we can expect. To find this lower bound on the computing time of the cloud, we take the cloud's computing time with a one-node cluster and divide it by the number of nodes in the cluster. Besides, to fully explore the time savings due to parallelization offered by our algorithm, we solve QPs that have at least ten strongly connected components in the quadratic coefficient matrix and hence can be potentially allocated among the computing nodes. Fig. 4 shows both the cloud's computing

Table 1: Comparison of computing time of local GSA and that of the proposed PGSA

Non-zeros in matrix P	Local GSA	Total PGSA	PGSA at the user	PGSA at the cloud	User's speedup
8×10^4	228.8s	45.2s	1.9 s	43.3s	120.4
1.2×10^5	516.9s	57.5	2.9s	54.6s	178.2
1.6×10^5	914.3s	73.15s	3.7 s	69.4s	247.1
2.0×10^5	1439.0s	88.7s	4.6 s	84.1s	312.8
2.4×10^5	2055.2s	122.6s	5.0 s	111.6s	411.04
2.8×10^5	2791.5s	173.9s	6.4 s	167.5s	436.17
3.6×10^5	4689.7s	215.3s	8.2 s	207.1s	571.9
4.4×10^5	7240.4s	322.0s	8.9 s	313.1s	813.5

time and the lower bound on the computing time for a QP with 10000 variables under different cluster sizes. We observe that our algorithm is very close to the lower bound on the computing time. Therefore, our secure outsourcing algorithm is scalable, which is a crucial requirement for solution methods for large scale problems.

8. CONCLUSIONS

In this paper, for the first time in the literature, we have investigated the problem of securely outsourcing large-scale quadratic programs (QPs) to the cloud. To protect the users' private information in their QPs, we develop a low complexity matrix transformation scheme. Specifically, the user conceals the QP's non-zero elements and structure by multiplying the non-zero elements with random numbers, and randomly permutating the rows and columns of its matrices, respectively. We show that the resulting QP is computationally indistinguishable both in value and in structure under a CPA, and thus user can confidently share it with the cloud. Then, based on the dual problem theory and the Gauss-Seidel algorithm, we design an iterative and parallel algorithm PGSA to solve the transformed dual problem in the cloud. Based on the solution to the transformed dual problem, the user can find the solution to its original QP with minimal computing resources. We have implemented the proposed algorithm on the Amazon Elastic Compute Cloud (EC2) and a laptop, and extensively evaluate its performance. Our results show that the proposed algorithm can solve large-scale QPs very efficiently, offers significant time savings to the user, and is scalable.

9. ACKNOWLEDGMENTS

This work was partially supported by the U.S. National Science Foundation under grants CNS-1343220, and CNS-1149786.

10. REFERENCES

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6):734–749, June 2005.

[2] M. J. Atallah, K. Pantazopoulos, J. R. Rice, and E. E. Spafford. Secure outsourcing of scientific computations. In M. V. Zelkowitz, editor, *Trends in Software Engineering*, volume 54 of *Advances in Computers*, pages 215 – 272. Elsevier, 2002.

[3] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[4] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, 2001.

[5] F. F. Costa. Big data in biomedicine. *Drug Discovery Today*, 19(4):433–440, 2014.

[6] N. R. Council. *Frontiers in Massive Data Analysis*. he National Academies Press, Washington, DC, 2013.

[7] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000.

[8] T. A. Davis and Y. Hu. The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software*, 38(1):1–25, 2011.

[9] H. Demirkan and D. Delen. Leveraging the capabilities of service-oriented decision support systems: Putting analytics and big data in cloud. *Decision Support Systems*, 55(1):412–421, 2013.

[10] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Advances in Cryptology CRYPTO 2010*, Santa Barbara, CA, USA, 2010.

[11] Q. Hardy. IBM, G.E. and others create big data alliance, 2015.

[12] C. M. I. Maros. A repository of convex quadratic programming problems. *Optimization Methods and Software*, pages 671–681, 1999.

[13] A. Ipakchi and F. Albuyeh. Grid of the future. *Power and Energy Magazine, IEEE*, 7(2):52–62, March 2009.

[14] E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python, 2001.

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

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

[17] 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. In *IEEE World Congress on Services (SERVICES)*, Washington DC, USA, July 2011.

[18] T. Kraska. Finding the needle in the big data systems haystack. *Internet Computing, IEEE*, 17(1):84–86, Jan 2013.

[19] 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*, 1(1):78–87, January-June 2013.

[20] F. Liu, W. K. Ng, and W. Zhang. Encrypted gradient descent protocol for outsourced data mining. In *International Conference on Advanced Information Networking and Applications (AINA)*, Gwangju, Korea, 2015.

[21] Y. Liu, P. Ning, and M. K. Reiter. False data injection attacks against state estimation in electric power grids. In *ACM Conference on Computer and Communications Security*, Chicago, Illinois, USA, 2009.

[22] Cloud Security Allliance. Security guidancefor critical areas of focus in cloud computing v3.0. https://cloudsecurityalliance.org/guidance/csaguide.v3.0.pdf, 2011.

[23] President's Council of Advisors on Science and Technology. Big data: Seizing opportunities, preserving values, 2014.

[24] President's Council of Advisors on Science and Technology. Big data and privacy: A technological perspective, May 2014.

[25] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, you, get off my cloud exploring information leakage in third party compute clouds. In *ACM Conference on Computer and Communications Security*, Chicago, IL, USA, November 2009.

[26] S. Sakr, A. Liu, D. Batista, and M. Alomari. A survey of large scale data management approaches in cloud environments. *IEEE Communications Surveys Tutorials*, 13(3):311–336, March 2011.

[27] S. Salinas, X. Chen, C. Luo, and P. Li. Efficient secure outsourcing of large-scale linear systems of equations. In *IEEE Conference of Computer Communications (INFOCOM)*, Hong Kong, China, April 2015.

[28] 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*, 11(9):647–657, September 2010.

[29] 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*, 15(4):38–47, July 2013.

[30] S. Subashini and V. Kavitha. A survey on security issues in service delivery models of cloud computing. *Journal of Network and Computer Applications*, 34(1):1–11, 2011.

[31] C. Wang, K. Ren, and J. Wang. Secure and practical outsourcing of linear programming in cloud computing. In *International Conference on Computer Communications*, Shangai, China, 2011.

[32] 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*, 24(6):1172–1181, June 2013.

[33] 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*, 1(1):166–177, June 2013.

[34] J. Xiang, H. Meng, and A. Aboulnaga. Scalable matrix inversion using mapreduce. In *International Symposium on High-performance Parallel and Distributed Computing*, Vancouver, BC, Canada, 2014.

[35] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. In *USENIX Conference on Hot Topics in Cloud Computing*, Boston, MA, USA, 2010.