IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS VOL.PP NO.99 YEAR 2012

Harnessing the Cloud for Securely Outsourcing Large-scale Systems of Linear Equations

Cong Wang, *Student Member, IEEE,* Kui Ren, *Member, IEEE,* Jia Wang, *Member, IEEE,* and Karthik Mahendra Raje Urs

Abstract—Cloud computing economically enables customers with limited computational resources to outsource large-scale computations to the cloud. However, how to protect customers' confidential data involved in the computations then becomes a major security concern. In this paper, we present a secure outsourcing mechanism for solving large-scale systems of linear equations (LE) in cloud. Because applying traditional approaches like Gaussian elimination or LU decomposition (aka. direct method) to such large-scale LE problems would be prohibitively expensive, we build the secure LE outsourcing mechanism via a completely different approach — iterative method, which is much easier to implement in practice and only demands relatively simpler matrix-vector operations. Specifically, our mechanism enables a customer to securely harness the cloud for iteratively finding successive approximations to the LE solution, while keeping both the sensitive input and output of the computation private. For robust cheating detection, we further explore the algebraic property of matrix-vector operations and propose an efficient result verification mechanism, which allows the customer to verify all answers received from previous iterative approximations in one batch with high probability. Thorough security analysis and prototype experiments on Amazon EC2 demonstrate the validity and practicality of our proposed design.

Index Terms—Confidential data, computation outsourcing, system of linear equations, cloud computing.

1 Introduction

Cloud Computing provides on-demand network access to a shared pool of configurable computing resources that can be rapidly deployed with great efficiency and minimal management overhead [2]. Under such unprecedented computing model, customers with computationally weak devices are no longer limited by the slow processing speed, memory, and other hardware constraints, but can enjoy the literally unlimited computing resources in the cloud through the convenient yet flexible pay-per-use manners [3].

Despite the tremendous benefits, the fact that customers and cloud are not necessarily in the same trusted domain brings many security concerns and challenges towards this promising computation outsourcing model [4]. Firstly, customer's data that are processed and generated during the computation in cloud are often sensitive in nature, such as business financial records, proprietary research data, and personally identifiable health information etc [5]. While applying ordinary encryption techniques to these sensitive information before outsourcing could be one way to combat the security concern, it also makes the task of computation over encrypted data in general a very difficult problem [6]. Secondly, since the operational details inside the cloud are not transparent enough to customers [5], no guarantee is provided on the quality of the computed results from the cloud. For example, for computations demanding a large amount of resources, there are huge financial incentives for the cloud server to be "lazy" if the customer cannot tell the correctness of the answer. Besides, possible software/hardware malfunctions and/or outsider attacks might also affect the quality of the computed results. Thus, we argue that the cloud is intrinsically *not secure* from the viewpoint of customers. Without providing a mechanism for secure computation outsourcing, i.e., to protect the sensitive input and output information of the outsourced computing needs and to validate the integrity of the computation result, it would be hard to expect cloud customers to turn over control of their computing needs from local machines to cloud solely based on its economic savings and resource flexibility.

Focusing on the engineering and scientific computing problems, this paper investigates secure outsourcing for widely applicable *large-scale* systems of linear equations (LE), which are among the most popular algorithmic and computational tools in various engineering disciplines that analyze and optimize real-world systems. By "large", we mean the storage requirements of the system coefficient matrix may easily exceed the available memory of the customer's computing device [7], like a modern portable laptop. In practice, there are many real world problems that would lead to very large-scale and even dense systems of linear equations with up to hundreds of thousands [8], [9] or a few million unknowns [10]. For example, a typical double-precision $50,000 \times 50,000$ system matrix resulted from electromagnetic application would easily occupy up to 20 GBytes storage space, seriously challenging the computational

• *Cong Wang, Kui Ren, Jia Wang, and Karthik Mahendra Raje Urs are all with the Department of Electrical and Computer Engineering, Illinois Institute of Technology, Chicago, IL 60616. E-mail: {cong, kren, jwang, karthik}@ece.iit.edu.*

A preliminary version [1] of this paper is to be presented at the 31th International Conference on Distributed Computing Systems (ICDCS 2011).

power of these low-end computing devices. Because the execution time of a computer program depends not only on the number of operations it must execute, but also on the location of the data in the memory hierarchy of the computer [7], solving such large-scale problems on customer's weak computing devices can be practically impossible, due to the inevitably involved huge IO cost. Therefore, resorting to cloud for such computation intensive tasks can be arguably the only choice for customers with weak computational power, especially when the solution is demanded in a timely fashion.

It is worth noting that in the literature, several cryptographic protocols for solving various core problems in linear algebra, including the systems of linear equations [11]–[16] have already been proposed from the secure multiparty computation (SMC) community. However, these approaches are in general ill-suited in the context of computation outsourcing model with large problem size. First and foremost, all these work developed under SMC model do not address the asymmetry among the computational power possessed by cloud and the customer, i.e., they all impose each involved party comparable computation burdens, which in this paper our design specifically intends to avoid (otherwise, there is no point for the customer seeking help from cloud). Secondly, the framework of SMC usually assumes each involved party knows a portion of the problem input information (e.g., each party knows only a share of the coefficient matrix). This assumption is not true any more in our model, where the information leakage on both the input and output of the LE problem to the cloud should be strictly forbidden. Last but not the least, almost all these solutions are focusing on the traditional direct method for jointly solving the LE, like the joint Gaussian elimination method in [12], [15], or the secure matrix inversion method in [13]. While working well for small size problems, these approaches in general do not derive practically acceptable solution time for large-scale LE, due to the expensive cubic-time computational burden for matrix-matrix operations and the huge IO cost on customer's weak devices (detailed discussions in Section 8).

The analysis from existing approaches and the computational practicality motivates us to design secure mechanism of outsourcing LE via a completely different approach — iterative method, where the solution is extracted via finding successive approximations to the solution until the required accuracy is obtained (to be introduced later in Section 2.3.1). Compared to direct method, iterative method only demands relatively simpler matrix-vector operations (with $\mathcal{O}(n^2)$ computational cost), which is much easier to implement in practice and widely adopted for large-scale LE [8], [10], [17]. To the best of our knowledge, no existing work has ever successfully tackled secure protocols for iterative methods on solving large-scale systems of LE in the computation outsourcing model, and we give the first study in this paper. Specifically, our mechanism utilizes the additive homomorphic property of public key encryption scheme, like the one proposed by Paillier [18], and allows customers with weak computational devices, starting from an initial guess, to securely harness the cloud for finding successive approximations to the solution in a privacy-preserving and cheating-resilient manner. For a linear system with $n \times n$ coefficient matrix, each iterative algorithm execution of the proposed mechanism only incurs $\mathcal{O}(n)$ local computational burden on customer's weak device and asymptotically eliminates the expensive IO cost, i.e., no unrealistic memory demands, which on the other hand may significantly downgrade the performance if the problem is solved by the customer alone. To detect any attempted result corruption by cloud server, we also propose a very efficient cheating detection mechanism to effectively verify in one batch of all the computation results by the cloud server from previous algorithm iterations with high probability. Our contributions can be summarized as follows:

1) For the first time, we formulate the problem in the computation outsourcing model for securely solving large-scale systems of LE via iterative methods, and provide the secure mechanism design which fulfills input/output privacy, cheating resilience, and efficiency.
2) Our mechanism brings computational savings as it only incurs $\mathcal{O}(n)$ local computation burden for the customer within each algorithm iteration and demands no unrealistic IO cost, while solving large-scale LE locally usually demands more than $\mathcal{O}(n^2)$ computation cost in terms of both time and memory requirements [10].
3) We explore the algebraic property of matrix-vector multiplication to design a batch result verification mechanism, which allows customers to verify all answers computed by cloud from previous iterations in one batch, and further ensures both the efficiency advantage and the robustness of the design.
4) The experiment on Amazon EC2 [19] shows our mechanism can help customers achieve up to $2.43\times$ savings when the sizes of the LE problems are relatively small ($n \leq 50,000$). Better efficiency gain can be easily anticipated when n goes to larger size. In particular, when n increases to $500,000$, the anticipated computational savings for customer can be up to 26.09 $\times$.

The rest of the paper is organized as follows. Section 2 introduces the system and threat model, and our design goals. Then we provide the detailed mechanism description in Section 4 and Section 5, both with security analysis. Section 6 gives the discussions on the practical implementation issues. Section 7 gives the performance evaluation, followed by Section 8 which overviews the related work. Finally, Section 9 gives the concluding remark of the whole paper.

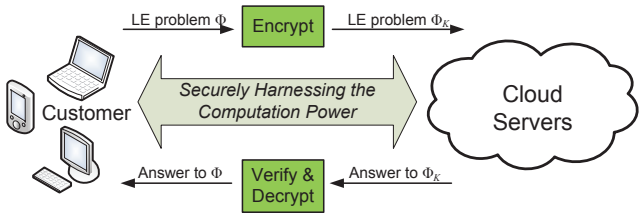

Fig. 1: Architecture of secure outsourcing problems of large-scale systems of linear equations in Cloud Computing

2 Problem Statement

2.1 System and Threat Model

We consider a computation outsourcing architecture involving two different entities, as illustrated in Fig. 1: the *cloud customer*, who wants to solve some computationally expensive LE problems with his low-end devices; the *cloud server* (CS), which has significant computation resources to provide utility computing services.

The customer has a large-scale system of LE problem $\mathbf{Ax} = \mathbf{b}$, denoted as $\Phi = (\mathbf{A}, \mathbf{b})$, to be solved. However, due to the lack of computing resources, like processing power, memory, and storage etc., he cannot carry out such expensive ($\mathcal{O}(n^\rho)(2 < \rho \leq 3)$) computation locally. Thus, the customer resorts to cloud server (CS) for solving the LE computation and leverages its computation power in a pay-per-use manner. Instead of directly sending original problem Φ, the customer first uses a secret key K to map Φ into some encrypted version Φ_K. Then, based on Φ_K, the customer starts the computation outsourcing protocol with CS, and harnesses the powerful resources of cloud in a privacy-preserving manner. The CS is expected to help the customer finding the answer of Φ_K, but supposed to learn as little as possible on the sensitive information contained in Φ. After receiving the solution of encrypted problem Φ_K, the customer should be able to first verify the answer. If it's correct, he then uses the secret K to map the output into the desired answer for the original problem Φ.

As later we shall see in the proposed model, the customer still needs to perform a one-time setup phase of encrypting the coefficient matrix with relatively costly $\mathcal{O}(n^2)$ computation.[1] But it is important to stress that this process can be performed under a trusted environment where the weak customer with no sufficient computational power outsources it to a trusted party. (Similar treatments have been utilized in [20]). The motivating example can be a military application where the customer has this one-time encryption process executed inside the military base by a trusted server, and then goes off into the field with access only to untrusted CS. Another example can be the customer has the system modeling coefficient matrix $\mathbf{A}$ encrypted on his company's workstation, and then uses his portable device outside while still hoping to make timely decisions (derive solutions $\mathbf{x_i}$) based on different observation $\mathbf{b_i}$ in the field, for $i = 1, 2, \ldots, s$. (Further discussion on amortization of such one-time cost is given in Section 6.2.) Thus, to make the rest of the paper easier to catch, we assume that CS is already in possession of the encrypted coefficient matrix, and the customer who knows the decryption key hopes to securely harness the cloud for on-demand computing outsourcing needs, i.e., solving LE problems $\{\mathbf{Ax} = \mathbf{b_i}\}$.

The security threats faced by the computation model primarily come from the malicious behavior of CS, which may behave beyond "honest-but-curious" model as assumed by some previous works on cloud data security (e.g., [21]–[23]). It is either because CS intends to do so or because it can be attacked/compromised. In addition to be persistently interested in analyzing the encrypted input sent by the customer and the encrypted output produced by the computation to learn the sensitive information, CS can also behave unfaithfully or intentionally sabotage the computation, e.g. to lie about the result to save the computing resources, while hoping not to be caught at the same time.

Finally we assume the communication channels between cloud server and the customer is authenticated and reliable, which can be achieved in practice with little overhead.

2.2 Design Goals

To enable secure and practical outsourcing of LE under the aforementioned model, our mechanism design should achieve the following security and performance guarantees.

- **Input/output Privacy**: No sensitive information from the customer's private data can be derived by the cloud server during performing the LE computation.
- **Robust Cheating Detection**: Output from faithful cloud server must be decrypted and verified successfully by the customer. No output from cheating cloud server can pass the verification by the customer with non-negligible probability.
- **Efficiency**: The local computation done by the customer should be substantially less than solving the original LE on his own. Here the computation burden is measured in terms of both time cost and memory requirements.

2.3 Preliminaries and Notations

2.3.1 Iterative Method

In many engineering computing and industrial applications, iterative method has been widely used in practice for solving large-scale systems of LE [8], and sometimes is the mandatory choice [17] over direct method due to

1. Since the encryption on each element of the matrix coefficient are independent, the whole one-time operation can be easily parallelized by using multithreading technique on the modern multi-core systems. For example, enabling double threading on a six core system could easily speedup the operation efficiency with a factor of 12.

its ease of implementation and relatively less computational power consumption, including the memory and storage IO requirement [10]. We now review some basics on the general form of stationary iterative methods for solving LE problems. A system of linear equations can be written as

$$\mathbf{Ax} = \mathbf{b}, \tag{1}$$

where $\mathbf{x}$ is the $n \times 1$ vector of unknowns, $\mathbf{A}$ is an $n \times n$ (non-singular) coefficient matrix and $\mathbf{b}$ is an $n \times 1$ right-hand side vector (so called constant terms). Most iterative methods involve passing from one iteration to the next by modifying a few components of some approximate vector solution at a time until the required accuracy is obtained. Without loss of generality, we focus on Jacobi iteration [17] here and throughout the paper presentation for its simplicity. Though extensions to other stationary iterative methods can be straightforward, we don't study them in the current work.

We begin with the decomposition: $\mathbf{A} = \mathbf{D} + \mathbf{R}$, where $\mathbf{D}$ is the diagonal component, and $\mathbf{R}$ is the remaining matrix. Then the Eq. (1) can be written as $\mathbf{Ax} = (\mathbf{D} + \mathbf{R})\mathbf{x} = \mathbf{b}$, and finally reorganized as: $\mathbf{x} = -\mathbf{D^{-1}} \cdot \mathbf{R} \cdot \mathbf{x} + \mathbf{D^{-1}} \cdot \mathbf{b}$. According to the Jacobi method, we can use an iterative technique to solve the left hand side of this expression for $\mathbf{x^{(k+1)}}$, using previous value for $\mathbf{x^{(k)}}$ on the right hand side. If we denote $\mathbf{T} = -\mathbf{D^{-1}} \cdot \mathbf{R}$ and $\mathbf{c} = \mathbf{D^{-1}} \cdot \mathbf{b}$, the above iterative equations can be simply represented as

$$\mathbf{x^{(k+1)}} = \mathbf{T} \cdot \mathbf{x^{(k)}} + \mathbf{c}. \tag{2}$$

The convergence is not always guaranteed for all matrices, but it is the case for a large body of LE problems derived from many real world applications [17].

2.3.2 Homomorphic Encryption

Our construction utilizes an efficient semantically-secure encryption scheme with additive homomorphic property. Given two integers x_1 and x_2, we have $\mathsf{Enc}(x_1 + x_2) = \mathsf{Enc}(x_1) * \mathsf{Enc}(x_2)$, and also $\mathsf{Enc}(x_1 * x_2) = \mathsf{Enc}(x_1)^{x_2}$. In our implementation we adopt the one presented by Paillier in [18]. The Paillier cryptosystem is a public key cryptosystem. Similar to RSA, the public key is N, which is the product of two large primes p and q. The plaintext space is $\mathbb{Z}_N$, while ciphertexts are represented as elements of group $\mathbb{Z}^*_{N^2}$. The encryption of message $x \in \mathbb{Z}_N$ is done by randomly choosing $r \in \mathbb{Z}^*_{N^2}$ and computing $\mathsf{Enc}(x) = g^x \cdot r^N \mod N^2$, where g is an element from $\mathbb{Z}^*_{N^2}$ with order multiple of N.

For a vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T \in (\mathbb{Z}_N)^n$, we use $\mathsf{Enc}(\mathbf{x})$ to denote the coordinate-wise encryption of $\mathbf{x}$: $\mathsf{Enc}(\mathbf{x}) = (\mathsf{Enc}(x_1), \mathsf{Enc}(x_2), \ldots, \mathsf{Enc}(x_n))^T$. Similarly, for some $n \times n$ matrix $\mathbf{T}$, where each of the component $\mathbf{T}[i, j]$ in $\mathbf{T}$ is from $\mathbb{Z}_N$, we denote the component-wise encryption of $\mathbf{T}$ as $\mathsf{Enc}(\mathbf{T})$, and we have $\mathsf{Enc}(\mathbf{T})[i, j] = \mathsf{Enc}(\mathbf{T}[i, j])$.

3 The Framework and Basic Solutions

In the Introduction we motivated the need for securely harnessing the cloud for solving large-scale LE problems. In this section, we start from the framework for our proposed mechanism design, and provide a basic outsourcing solution based on direct method. The basic solution fulfills the input/output privacy defined in Section 2.2, but does not meet the efficiency requirement. The analysis of this basic solution gives insights and motivations on our main mechanism design based on iterative methods.

3.1 The General Frameworks

To make the following presentation easier to follow, we first give the general high level description of the framework of the proposed mechanism, which consists of three phases: (ProbTransform, ProbSolve, ResultVerify).

- ProbTransform. *In this phase, cloud customer would initialize a randomized key generation algorithm and prepare the LE problem into some encrypted form Φ_K via key K for phase* ProbSolve*. Transformation and/or encryption operations will be needed when necessary.*
- ProbSolve. *In this phase, cloud customer would use the encrypted form Φ_K of LE to start the computation outsourcing process. In case of using iterative methods, the protocol ends when the solution within the required accuracy is found.*
- ResultVerify. *In this phase, the cloud customer would verify the encrypted result produced from cloud server, using the randomized secret key K. A correct output $\mathbf{x}$ to the problem is produced by decrypting the encrypted output. When the validation fails, the customer outputs $\perp$, indicating the cloud server was cheating.*

Note that the proposed framework suits for secure outsourcing mechanisms based on both direct method and iterative method. In the following, we first give a basic direct method based mechanism design.

3.2 Basic Direct Method based Mechanisms

We study in this subsection a straight-forward approach for encrypting the problem for direct solvers, and show that the local computation cost based on these techniques along may result in an unsatisfactory mechanism from the efficiency gain perspective.

Specifically, in the ProbTransform phase, the customer picks a random vector $\mathbf{r} \in \mathbb{R}^n$ as his secret keying material. Then he rewrites Eq. (1) as $\mathbf{A}(\mathbf{x} + \mathbf{r}) = \mathbf{b} + \mathbf{Ar}$. Let $\mathbf{y} = \mathbf{x} + \mathbf{r}$ and $\mathbf{b}' = \mathbf{b} + \mathbf{Ar}$, we have $\mathbf{Ay} = \mathbf{b}'$. To hide the coefficient matrix $\mathbf{A}$, the customer would select a random invertible matrix $\mathbf{Q}$ that has the same dimension as $\mathbf{A}$. Left-multiplying $\mathbf{Q}$ to both sides of $\mathbf{Ay} = \mathbf{b}'$ would give us

$$\mathbf{A}'\mathbf{y} = \mathbf{b}'', \tag{3}$$

where $\mathbf{A}' = \mathbf{QA}$ and $\mathbf{b}'' = \mathbf{Q}(\mathbf{b} + \mathbf{Ar})$. Clearly, as $\mathbf{Q}$ and $\mathbf{r}$ are chosen randomly and kept as secret, cloud has no way to know $(\mathbf{A}, \mathbf{b}, \mathbf{x})$, except the dimension of $\mathbf{x}$.

The customer can then start the ProbSolve phase by outsourcing $\Phi_K = (\mathbf{A}', \mathbf{b}'')$ to the cloud, who solves Φ_K and sends back answer $\mathbf{y}$. Once the correctness of $\mathbf{y}$ is verified, the customer can derive $\mathbf{x}$ via $\mathbf{x} = \mathbf{y} - \mathbf{r}$ for the original problem $\Phi = (\mathbf{A}, \mathbf{b})$.

Remark. While faithfully achieving the input/output privacy, the above approach is not attractive for the following two reasons: 1) The local problem transformation cost for matrix multiplication $\mathbf{QA}$ is $\mathcal{O}(n^3)$, which is comparable to the cost of solving $\mathbf{Ax} = \mathbf{b}$ [25]. Considering the extra cost of ResultVerify, the discussion of which we intentionally defer to a later Section 5, there is no guaranteed computational saving for the customer in this basic mechanism. 2) The local cubic time cost can become prohibitively expensive when n goes large to the orders of hundreds of thousand. Besides, it violates our assumption in Section 2 that the customer cannot carry out expensive $\mathcal{O}(n^\rho)(2 < \rho \leq 3)$ computation locally.

In the recent literature, Atallah et al. has proposed work for secure outsourcing matrix multiplication using only $\mathcal{O}(n^2)$ local complexity. However, in practice those work can hardly be applied to our case of calculating $\mathbf{Q} \cdot \mathbf{A}$ for Eq. (3), especially for large n. The reason is that in their work, either non-collusion servers are required [26] or scalar operations are expanded to polynomials and thus incur huge communication and computation overhead [27]. Both assumptions are difficult to be met in practice. (See detailed discussion on this at Section 8).

4 The Proposed Solution

The above observation and discussion shows that direct method based approach might not be a good option for resource-limited customers for secure outsourcing large-scale LE with computational savings in mind. This motivates us to design secure outsourcing mechanism using iterative method. To better facilitate the iterative method based mechanism design to be explored later, we first make some general but non-stringent assumptions about the system as follows: 1) we assume the system coefficient matrix $\mathbf{A}$ is a strictly diagonally dominant matrix[2]. It helps ensure the non-singularity of $\mathbf{A}$ and the convergence of the iterative method that is to be detailed later. Note that this is not a stringent requirement, as many real-world formulated LE problems satisfy this assumption, such as the statistical calculations [24], or the radar cross-section calculations [8] etc. 2) Although proper preconditioning techniques (e.g. see [7]–[10], [17] for details) on the coefficient matrix $\mathbf{A}$ can significantly improve the performance of any iterative method, we do not study the cost of these techniques in this paper. As we focus on the security design only, we assume the coefficient matrix $\mathbf{A}$ already ensures fast enough convergence behavior, i.e., the number of iterations $L \ll n$. 3) We assume the matrix $\mathbf{A}$ is first transformed to $\mathbf{T} = \mathbf{D}^{-1} \cdot \mathbf{R}$, where $\mathbf{A} = \mathbf{D} + \mathbf{R}$ as in Eq. (2), and then stored in cloud in its encrypted form $\mathsf{Enc}(\mathbf{T})$ via homomorphic encryption introduced in Section 2.3.2. As stated in our system model, this one-time setup phase is done before ProbTransform phase by some trusted workstation under different application scenarios. Hereinafter, we may interchangeably use the two forms of coefficient matrix $\mathbf{A}$ or $\mathbf{T}$ without further notice.

2. We focus on general dense matrices in this paper.

Algorithm 1: Problem Transformation Phase

Data: original problem $\Phi = (\mathbf{A}, \mathbf{b})$
Result: transformed problem as shown in Eq. (5)
begin
1 pick random $\mathbf{r} \in \mathbb{R}^n$;
2 compute $\mathbf{b}' = \mathbf{b} + \mathbf{Ar}$, and $\mathbf{c}' = \mathbf{D}^{-1} \cdot \mathbf{b}'$;
3 replace tuple $(\mathbf{x}, \mathbf{c})$ in Eq. (2) with $(\mathbf{y} = \mathbf{x} + \mathbf{r}, \mathbf{c}')$;
return transformed problem as Eq. (5);

For ease of presentation, we consider a semi-honest cloud server here, and defer the mechanisms of cheating detection to a later section.

4.1 Problem Transformation

The customer who has coefficient vector $\mathbf{b}$ and seeks solution $\mathbf{x}$ satisfying $\mathbf{Ax} = \mathbf{b}$ cannot directly starts the ProbSolve with cloud, since such interaction may expose the private information on final result $\mathbf{x}$. Thus, we still need a transformation technique to allow customer to properly hide such information first. Similar to the basic mechanism in Section 3.2, in the ProbTransform phase, the customer picks a random vector $\mathbf{r} \in \mathbb{R}^n$ as his secret keying material, and rewrites Eq. (1) as the new LE problem

$$\mathbf{Ay} = \mathbf{b}', \tag{4}$$

where $\mathbf{y} = \mathbf{x} + \mathbf{r}$ and $\mathbf{b}' = \mathbf{b} + \mathbf{Ar}$. Clearly, as long as the relationship $\mathbf{x} = \mathbf{y} - \mathbf{r}$ holds, then for any solution $\mathbf{x}$ satisfying Eq. (1), we can find a solution $\mathbf{y}$ satisfying the Eq. (4), and vice versa. Thus, the solution $\mathbf{x}$ to Eq. (1) can be found by solving a transformed LE problem in Eq. (4). At this point, both the output $\mathbf{x}$ and input tuple $\mathbf{b}$ have been perfectly hidden via random vector $\mathbf{r}$. Since our goal is to start the iterative process with the cloud for solving LE, we can reformulate Eq. (4) into the more convenient iterative form as Eq. (2), which we rewrite as follows:

$$\mathbf{y}^{(\mathbf{k+1})} = \mathbf{T} \cdot \mathbf{y}^{(\mathbf{k})} + \mathbf{c}', \tag{5}$$

where $\mathbf{T} = -\mathbf{D}^{-1} \cdot \mathbf{R}$, $\mathbf{c}' = \mathbf{D}^{-1} \cdot \mathbf{b}'$, and $\mathbf{A} = \mathbf{D} + \mathbf{R}$.

As a result, the problem input $\Phi = (\mathbf{A}, \mathbf{b})$ that needs to be protected is changed to tuple $\Phi_K = (\mathbf{T}, \mathbf{c}')$, where $\mathbf{T}$ has already been encrypted and stored as $\mathsf{Enc}(\mathbf{T})$ at cloud, and $\mathbf{c}'$ is just a randomly masked version of $\mathbf{b}$ via random $n \times 1$ vector $\mathbf{r}$. The problem output $\mathbf{x}$ is also masked to $\mathbf{y} = \mathbf{x} + \mathbf{r}$. Now we are ready for the next phase of ProbSolve. The whole procedure of ProbTransform is summarized in Algorithm 1.

Remark. This problem transformation on the local customer side only requires two matrix-vector multiplications: one for $\mathbf{b}' = \mathbf{b} + \mathbf{Ar}$ and one for $\mathbf{c}' = \mathbf{D^{-1}} \cdot \mathbf{b}'$. When n goes large, expensive IO cost at customer device could downgrade the performance of such operations. However, as soon we shall see, such one-time problem transformation cost can be easily amortized throughout the overall efficient iterative algorithm executions, especially when we deal with honest but curious adversaries. It is also worth noting that this transformation does not need to modify the matrix of $\mathbf{A}$ (or $\mathbf{T}$), which gives us advantage of reusing. Specifically, the customers with different constant terms $\mathbf{b_i}$ can utilize this transformation technique by choosing a different $\mathbf{r}$ and then harness the cloud for solving different LE problems $\{\mathbf{Ax} = \mathbf{b_i}\}$, as seen in Section 6.2.

4.2 The Iterative Problem Solving

After the problem transformation step, now we are ready to describe the phase of ProbSolve. The purpose of our protocol is to let the customer securely harness the cloud for the most expensive computation, i.e., the matrix-vector multiplication $\mathbf{T} \cdot \mathbf{y}^{(k)}$ in Eq. (5) for each algorithm iteration, $k = 1, 2, \ldots, L$. We show how it can be realized with the help of the additive homomorphic encryption introduced in Section 2.3.2. Since it is an iterative computing process, we only describe the very first round of the process as follows. We leave the convergence analysis and other input/output privacy guarantee in later subsections. For ease of presentation, in what follows we assume without loss of generality that our main protocol of solving LE works over integers. All arithmetic is modular with respect to the modulus N of the homomorphic encryption, and the modulus is large enough to contain the answer. The reason why this is a reasonable assumption and how our designs handle noninteger real numbers is given in more detail in Section 6 of practical considerations.

1) For the very first iteration, the customer starts the initial guess on the vector $\mathbf{y}^{(0)} = (y_1^{(0)}, y_2^{(0)}, \ldots, y_n^{(0)})^T$, and then sends it to the cloud.

2) The cloud server, in possession of the encrypted matrix $\mathsf{Enc}(\mathbf{T})$, computes the value $\mathsf{Enc}(\mathbf{T} \cdot \mathbf{y}^{(0)})$ by using the homomorphic property of the encryption:

$$\begin{aligned} \mathsf{Enc}(\mathbf{T} \cdot \mathbf{y}^{(0)})[i] &= \mathsf{Enc}(\sum_{j=1}^{n} \mathbf{T}[i,j] \cdot y_j^{(0)}) \\ &= \prod_{j=1}^{n} \mathsf{Enc}(\mathbf{T}[i,j])^{y_j^{(0)}}, \end{aligned} \tag{6}$$

for $i = 1, \ldots, n$, and sends $\mathsf{Enc}(\mathbf{T} \cdot \mathbf{y}^{(0)})$ to customer.

3) After receiving $\mathsf{Enc}(\mathbf{T} \cdot \mathbf{y}^{(0)})$, the customer decrypts and gets $\mathbf{T} \cdot \mathbf{y}^{(0)}$ using his private key. He then updates the next approximation $\mathbf{y}^{(1)} = \mathbf{T} \cdot \mathbf{y}^{(0)} + \mathbf{c}'$ via Eq. (5).

For the k-th iteration, it follows similarly that the customer provides the k-th approximation of $\mathbf{y}^{(k)}$ to the cloud, $k = 1, 2, \ldots, L$. The cloud computes $\mathsf{Enc}(\mathbf{T} \cdot \mathbf{y}^{(k)})$ for the customer for the next update of $\mathbf{y}^{(k+1)}$. The protocol execution continues until the result converges, as shown in Algorithm 2.

Algorithm 2: Iterative Problem Solving Phase

Data: transformed problem with input $\mathbf{c}'$ and $\mathsf{Enc}(\mathbf{T})$
Result: solution $\mathbf{x}$ to the original problem $\Phi = (\mathbf{A}, \mathbf{b})$
% L: maximum number of iterations to be performed;
% ϵ: measurement of convergence point;
begin
1 Customer picks $\mathbf{y}^{(0)} \in (\mathbb{Z}_N)^n$;
for $(k \leftarrow 0$ **to** $L)$ **do**
2 Customer sends $\mathbf{y}^{(k)}$ to cloud;
3 Cloud computes $\mathsf{Enc}(\mathbf{T}\mathbf{y}^{(k)})$ via Eq. (6);
4 Customer decrypts $\mathbf{T}\mathbf{y}^{(k)}$ via his private key;
if $||\mathbf{y}^{(k)} - \mathbf{y}^{(k+1)}|| \leq \epsilon$ **then**
5 **break** with convergence point $\mathbf{y}^{(k+1)}$;
6 **return** $\mathbf{x} = \mathbf{y}^{(k+1)} - \mathbf{r}$;

Remark. For the k-th iteration, the dominant customer's local computation overhead is only to decrypt the vector of $\mathsf{Enc}(\mathbf{T} \cdot \mathbf{y}^{(k)})$, which takes $\mathcal{O}(n)$ complexity, and in general does not require expensive IO cost, as demonstrated in our experiment in Section 7. This is less than the $\mathcal{O}(n^2)$ cost demanded by the matrix-vector multiplication $\mathbf{T} \cdot \mathbf{y}^{(k)}$ of Eq. (5) in terms of both time and memory requirements. Note that the decryption computation is generally quite expensive compared to the floating point arithmetic operation. Thus, the theoretical computation efficiency gain (in terms of both time and memory requirements) can only be exhibited when the problem size n goes very large. However, this is not a problem in our case, since we are specifically using iterative methods to securely solve large-scale LE. Later in Section 7, we will show some performance results and discuss the possible selections of deciding the proper size n for the proposed problem. Also note that the communication overhead between the customer and the cloud is only two vectors of size n for each iteration, which is reasonably efficient.

4.3 Convergence Analysis

When dealing with iterative methods, it is a must to determine whether and when the iteration will converge. Here we utilize the general convergence result from [17]: For LE problem $\mathbf{Ax} = \mathbf{b}$, if $\mathbf{A}$ is a strictly diagonally dominant or an irreducibly diagonally dominant matrix, then the associated Jacobi iterations converge for any initial guess of $\mathbf{x}_0$.

Since in our model the coefficient matrix $\mathbf{A}$ is readily diagonally dominant, to determine whether to terminate

the ProbSolve phase, the customer only needs to test if

$$||\mathbf{y}^{(k)} - \mathbf{y}^{(k+1)}|| \leq \epsilon, \tag{7}$$

for some small enough $\epsilon > 0$. And the termination point $\mathbf{y}^{(k+1)}$ will help get the final result $\mathbf{x}$ via $\mathbf{x} = \mathbf{y}^{(k+1)} - \mathbf{r}$. The local computation cost for each iteration is still $\mathcal{O}(n)$.

4.4 Input/output Privacy Analysis

4.4.1 Output Privacy Analysis

From above protocol instantiation, we can see that throughout the whole process, the cloud server only sees the plaintext of $\mathbf{y}^{(k)}$, the encrypted version of matrix $\mathsf{Enc}(\mathbf{T})$, and encrypted vectors $\mathsf{Enc}(\mathbf{T} \cdot \mathbf{y}^{(\mathbf{k})})$, $k = 1, 2, \ldots, L$. Since $\mathbf{y}$ is a blinded version of original solution $\mathbf{x}$, it is safe to send $\mathbf{y}$ to the cloud in plaintext. No information of $\mathbf{x}$ would be leaked as long as $\mathbf{r}$ is kept secret by the customer.

4.4.2 Input Privacy Analysis

While the output is protected perfectly, it is worth noting that some knowledge about the input tuple $\Phi_k = (\mathbf{T}, \mathbf{c}')$ could be implicitly leaked through the protocol execution itself. The reason is as follows: for each two consecutive iterations of the protocol, namely, the k-th and the $(k+1)$-th, the cloud server sees actually the plaintext of both $\mathbf{y}^{(k)}$ and $\mathbf{y}^{(k+1)}$. Thus, a "clever" cloud server could initiate a system of linear equations via Eq. (5) and attempts to learn the unknown components of $\mathbf{T}$ and $\mathbf{c}'$. More specifically, for the total L iterations, the cloud server could establish a series of $(L-1) \times n$ equations from $\mathbf{y}^{(k)}$, $k = 0, 1, \ldots, L-1$,[3] while hoping to solve $n^2 + n$ unknowns of $\mathbf{T}$ and $\mathbf{c}'$.

However, as we have assumed in Section 3.1 that various preconditioning techniques can ensure fast enough convergence behavior, we have the number of iterations $L \ll n$. (In fact, if L is close or even larger than n, there would be no advantage of using iterative method over direct method at all.) As a result, from the $(L-1) \times n$ equations, the $n^2 + n$ components of $\mathbf{T}$ and $\mathbf{c}$ is largely underdetermined and cannot be exactly determined by any means. Thus, as long as the cloud server has no previous knowledge of the coefficient matrix $\mathbf{A}$, we state that such bounded information leakage from $(L-1) \times n$ equations can be negligible, especially when the size of problem n goes very large.

In fact, we can further enhance the guarantee of input privacy by introducing a random scaling factor $a_k \in \mathbb{Z}_N$ for each iteration to break the linkability of two consecutive iterations of the protocol. Specifically, instead of sending $\mathbf{y}^{(k)}$ directly to the cloud server for the k-th iteration, the customer sends $a_k \cdot \mathbf{y}^{(k)}$ for each iteration of the ProbSolve. When the cloud server sends back the encrypted value $\mathsf{Enc}(a_k \cdot \mathbf{T}\mathbf{y}^{(k)})$, the customer just simply decrypts the vector of $a_k \cdot \mathbf{T}\mathbf{y}^{(k+1)}$, divides each component with a_k, and then updates the next approximation $\mathbf{y}^{(k+1)}$ via Eq. (5). Similarly, for the next iteration another random scaling factor a_{k+1} is multiplied to $\mathbf{y}^{(k+1)}$ before sent to the cloud server.

Remark. With the random scaling factor a_k, it is not possible to derive the original value $\mathbf{y}^{(k)}$ via $a_k\mathbf{y}^{(k)}$. Thus, the cloud server can no longer directly establish linear equations from received $a_k\mathbf{y}^{(k)}$ and $a_{k+1}\mathbf{y}^{(k+1)}$, but a series of non-linear equations with extra random unknowns $a_1, a_2, \ldots, a_L$. We should note that while this method further enhances the guarantee of input privacy by bringing extra randomness and nonlinearity of the system equations, it does not incurs any expensive operation, making the customer's local computation cost still $\mathcal{O}(n)$ within each iteration.

5 Cheating Detection

Till now, the proposed protocol works only under the assumption of honest but curious cloud server. However, such semi-trusted model is not strong enough to capture the adversary behaviors in the real world. In many cases, an unfaithful cloud server in reality could sabotage the protocol execution by either being lazy or intentionally corrupting the computation result, while hoping not to be detected. In the following, we propose to design result verification methods to handle these two malicious behaviors. Our goal is to verify the correctness of the solution by using as few as possible expensive matrix-vector multiplication operations.

For easy notation purposes, we denote $\mathbf{z}^{(k)} = \mathbf{T} \cdot \mathbf{y}^{(k)}$ as the expected correct responses from cloud server, and $\hat{\mathbf{z}}^{(k)} = \mathbf{T} \cdot \hat{\mathbf{y}}^{(k)}$ as the actual received value from cloud server, where $k = 1, 2, \ldots, \mathcal{L}$. Here we also assume $\mathcal{L} \leq L$, meaning the ResultVerify phase is initiated within at most $\mathcal{L}$ iterations.

5.1 Dealing with Lazy Adversary

We first consider detecting the laziness of cloud server. Since computing the addition and multiplication over encrypted domain could cost a lot of computational power, the cloud server might not be willing to commit service-level-agreed computing resources in order to save cost. More severely, for the k-th iteration, the adversary could simply reply the result $\mathbf{z}^{(k-1)}$ of the previous $(k-1)$-th iteration without computation.

As a result, the customer who uses $\mathbf{z}^{(k-1)}$ to update for the next $\mathbf{y}^{(k+1)}$ will get the result $\mathbf{y}^{(k+1)} = \mathbf{y}^{(k)}$. Consequently, he may be incorrectly led to believe the solution of equation $\mathbf{A}\mathbf{y} = \mathbf{b}'$ is found. Thus, for the malicious adversary, only checking the Eq. (7) is not sufficient to convince the customer that the solution has converged. According to Eq. (4) one further step has to be executed as

$$||\mathbf{A}\mathbf{y}^{(k+1)} - \mathbf{b}'|| \leq \epsilon. \tag{8}$$

Remark This checking equation incurs the local cost of $\mathcal{O}(n^2)$ for customer. While potentially expensive for

3. Note that $\mathbf{y}^{(L)}$ as the final answer is not transmitted to the cloud server.

large size of n, we should note that it does not have to be executed within every iteration. It only needs to be tested after the test on $\mathbf{y}^{(k)}$ and $\mathbf{y}^{(k+1)}$ via Eq. (7) is passed. If Eq. (7) is not passed, it means $\mathbf{y}^{(k+1)}$ is not the convergence point yet. On the other hand, if Eq. (7) is successfully passed, we can then initiate the test of Eq. (8). If Eq. (8) holds, we say the final solution is found, which is $\mathbf{x} = \mathbf{y}^{(k+1)} - \mathbf{r}$. If it doesn't, we can tell that the cloud server is cheating (being lazy). In either case, this matrix-vector multiplication of Eq. (8) only needs to be executed at most once throughout the protocol execution.

5.2 Dealing with Truly Malicious Adversary

While a lazy adversary only sends previous result as the current one, a truly malicious adversary can sabotage the whole protocol execution by returning arbitrary answers. For example, the malicious cloud server could compute Eq. (6) via arbitrary vectors $\hat{\mathbf{y}}^{(k)}$ other than customer's $\mathbf{y}^{(k)}$. In the worst case, it would make the protocol never converge, wasting the resources of the customer. Thus, we must design an efficient and effective method to detect such malicious behavior, so as to ensure the result quality. The straightforward way would be to redo the matrix-vector multiplication $\mathbf{T} \cdot \mathbf{y}^{(k)}$ and check if it equals to the received $\hat{\mathbf{z}}^{(k)}$ for each iteration k. This is not appealing since it consumes equivalent amount of resources in comparison to that of computing the results directly. Below we utilize the algebraic property of matrix-vector multiplication and design a method to test the correctness of all received answers $\hat{\mathbf{z}}^{(k)} = \mathbf{T} \cdot \hat{\mathbf{y}}^{(k)}$, $k = 1, 2, \ldots, \mathcal{L}$ in only one batch, i.e., using only one matrix-vector multiplication.

Suppose after $\mathcal{L}$ iterations, the solution still does not converge. The customer can initiate a ResultVerify phase by randomly selecting $\mathcal{L}$ numbers, $\alpha_1, \alpha_2, \ldots, \alpha_{\mathcal{L}}$ from $B \subset \mathbb{Z}_N$, where each α_k is of l-bit length and $l < \log N$. He then computes the linear combination $\boldsymbol{\theta}$ over the $\mathbf{y}^{(k)}$'s, which he has provided in the previous k iterations, $k = 1, 2, \ldots, \mathcal{L}$: $\boldsymbol{\theta} = \sum_{k=1}^{\mathcal{L}} \alpha_k \cdot \mathbf{y}^{(k)}$. Next, to test the correctness of all the intermediate results, $\{\hat{\mathbf{z}}^{(k)} = \mathbf{T} \cdot \hat{\mathbf{y}}^{(k)}\}$, $k = 1, 2, \ldots, \mathcal{L}$, received from cloud server, the customer simply checks if the following equation holds:

$$\mathbf{T} \cdot \boldsymbol{\theta} \stackrel{?}{=} \sum_{k=1}^{\mathcal{L}} \alpha_k \cdot \hat{\mathbf{z}}^{(k)}. \tag{9}$$

The above equation can be elaborated as follows:

$$\begin{aligned} \mathbf{T} \cdot \boldsymbol{\theta} &= \mathbf{T} \cdot \sum_{k=1}^{\mathcal{L}} \alpha_k \cdot \mathbf{y}^{(k)} \\ &= \sum_{k=1}^{\mathcal{L}} \alpha_k \cdot \mathbf{T} \cdot \mathbf{y}^{(k)} = \sum_{k=1}^{\mathcal{L}} \alpha_k \cdot \hat{\mathbf{z}}^{(k)}. \end{aligned}$$

Since each α_k is chosen randomly from $B = \{0,1\}^l \subset \mathbb{Z}_N$, we have the following theorem capturing the correctness and soundness of the cheating detection method:

Theorem *1:* The result verification Eq. (9) holds if and only if $\hat{\mathbf{z}}^{(k)} = \mathbf{T} \cdot \mathbf{y}^{(k)}$ for all $k = 1, 2, \ldots, \mathcal{L}$, with error probability at most 2^{-l}.

Proof: It is straightforward that the Eq. (9) always holds when the received values of each iteration are computed correctly. We now prove the soundness. Let $\hat{\mathbf{z}}^{(1)}, \hat{\mathbf{z}}^{(2)}, \ldots, \hat{\mathbf{z}}^{(\mathcal{L})}$ denote the actual received results from cloud server, among which there exist some unfaithfully computed results. And let $\mathbf{z}^{(1)}, \mathbf{z}^{(2)}, \ldots, \mathbf{z}^{(\mathcal{L})}$ denote the correct result of the matrix-vector multiplication of executed iteration.

Since the results are incorrect, there exist at least some k such that $\hat{\mathbf{z}}^{(k)} \neq \mathbf{z}^{(k)}$, for $k = 1, 2, \ldots, \mathcal{L}$. This inequality also indicates that at least one of the n components of vector $\hat{\mathbf{z}}^{(k)}$ and $\mathbf{z}^{(k)}$ is not equal to each other. Without loss of generality, we assume $\hat{\mathbf{z}}^{(k)}[1] \neq \mathbf{z}^{(k)}[1]$.

Now suppose the test Eq. (9) accepts the incorrect results on a particular choice of $\alpha_1, \alpha_2, \ldots, \alpha_{\mathcal{L}}$. From the correctness of Eq. (9) we must have

$$\sum_{k=1}^{\mathcal{L}} \alpha_k \cdot \hat{\mathbf{z}}^{(k)} = \sum_{k=1}^{\mathcal{L}} \alpha_k \cdot \mathbf{z}^{(k)} \tag{10}$$

Consider the 1st row of this vector equation, we have

$$\sum_{k=1}^{\mathcal{L}} \alpha_k \cdot \hat{\mathbf{z}}^{(k)}[1] = \sum_{k=1}^{\mathcal{L}} \alpha_k \cdot \mathbf{z}^{(k)}[1] \tag{11}$$

Since $\hat{\mathbf{z}}^{(k)}[1] \neq \mathbf{z}^{(k)}[1]$, we can rearrange the above equation as

$$\alpha_1 = -(\mathbf{z}^{(k)}[1] - \hat{\mathbf{z}}^{(k)}[1])^{-1} \cdot (\sum_{k=2}^{\mathcal{L}} \alpha_k \cdot (\mathbf{z}^{(k)}[1] - \hat{\mathbf{z}}^{(k)}[1])) \tag{12}$$

This means for any fixed choice of $\alpha_2, \alpha_3, \ldots, \alpha_{\mathcal{L}}$, there is exactly one choice of $\alpha_1 \in B = \{0,1\}^l \subset \mathbb{Z}_N$ (from Eq. (12)) such that Eq. (11) is true. Thus, if we fix $\alpha_2, \alpha_3, \ldots, \alpha_{\mathcal{L}}$ and draw α_1 randomly from B, the probability that Eq. (11) holds is 2^{-l}. The same is also true if we draw $\alpha_1, \alpha_2, \ldots, \alpha_{\mathcal{L}}$ independently at random. Note that we only consider the case where one of the n components of vector $\hat{\mathbf{z}}^{(k)}$ and $\mathbf{z}^{(k)}$ is not equal to each other. In case there are $\lambda > 1$ components are different, the probability that eq. (11) holds would be $2^{-\lambda \cdot l} < 2^{-l}$. Therefore, the test eq. (9) holds for unfaithful results is at most 2^{-l}.

Finally, we remark that some of our probability argument from Eq. (12) is inspired by Bellare's analysis of their fast batch verification for modular exponentiation and digital signature schemes [28].

□

Remark. It is easy to tell that the computation overhead of Eq. (9) is only bounded by one matrix-vector multiplication of the left-hand-side of the equation (recall $\mathcal{L} \leq L \ll n$). The size of l is a tradeoff between efficiency and security. While a conservative choice of high security should probably require l around 80 bit, for a fast check

a reasonable choice of 20 bits of l is also acceptable [29]. Note that in practice this result verification does not need to happen very frequently, because the property of batch verification ensures the quality of all previous received values $\{\hat{\mathbf{z}}^{(k)} = \mathbf{T} \cdot \hat{\mathbf{y}}^{(k)}\}$, $k = 1, 2, \ldots, \mathcal{L}$. Thus the customer can preset the threshold $\mathcal{L}$ as sufficiently large such that either he detects the unfaithful behavior of cloud server or the program will converge soon after $\mathcal{L}$ iterations. In the best case, Eq. (9) only needs to be instantiated once. Combined with Eq. (8), we can see that our method for cheating detection indeed achieves as few as possible expensive matrix-vector multiplication operation. As a result, the overall design asymptotically eliminates the expensive IO cost on customer throughout the successive approximation process for seeking the solution as well as result verification.

6 Practical Considerations

In the previous section, we assumed that the proposed protocol works well over integer values. Now we discuss these practical issues on how to adapt our proposed solution to work over non-integer and even negative values, and justify that our solution is a reasonable one. In addition, we will also describe some specific application scenarios of our proposed mechanism where we can amortize the one-time $\mathcal{O}(n^2)$ setup cost by reusing the coefficient matrix $\mathbf{A}$. Finally, we show the utilization of cost associativity of cloud computing can actually help us save much of protocol running time during the implementation on cloud. These discussions also give us insights of designing our experiments on the real cloud platform in the next section.

6.1 Dealing with Non-Integer Numbers

Remember in the Paillier cryptosystem, the message space is $\mathbb{Z}_N$: $\{0, 1, \ldots, N-1\}$. Thus, we have to deal with issues of both negative numbers and real numbers in order to use the primitive correctly. We begin with the handling of negative numbers. Since the homomorphic property of Paillier cryptosystem is over modulo arithmetic, we can shift the negative numbers to the interval of $\{(N-1)/2+1, \ldots, N-1\}$, with $-1 = N-1 \mod N$, while keeping non-negative numbers in the interval of $\{0, 1, \ldots, (N-1)/2\}$. When the decrypted value $y_i > N/2$, we shift y_i back to the result $y_i - N$. By doing so we limit the value of input to the interval of $(-N/2, (N-1)/2]$. But since N is of 1024 bit length, it ensures the interval is large enough for accommodating all the computation to be performed in our scheme.

As for the real value $\mathbf{y} \in \mathbb{R}^n$, or $y_i \in \mathbb{R}$, we can introduce some scaling factor (e.g., the multipliers of 10) to first scale up the value into integer value. Then after the protocol execution, we can scale the result back to the original value by removing the scaling factor.[4] More formally speaking, let $1/q$ be the selected multiplier, where q is some small enough quantization factor. Then we can approximate the value of $y_i \in \mathbb{R}$ as $\bar{y}_i = \lfloor y_i/q \rfloor \in \mathbb{Z}_N$. This may results in some accuracy problem. But if we select sufficiently thin quantization factor like $q = 10^{-3}$, it may lose little accuracy. Obviously, the additive homomorphic property of Paillier encryption scheme still holds, since

$$\mathsf{Dec}(\mathsf{Enc}(\bar{y}_1) \cdot \mathsf{Enc}(\bar{y}_2)) = \bar{y}_1 + \bar{y}_2 = \frac{y_1 + y_2}{q}. \tag{13}$$

This allows cloud to perform an arbitrarily number of sums among ciphertext without worrying the rounding errors. Also, the property of plaintext multiplication also holds, since

$$\mathsf{Dec}(\mathsf{Enc}(\bar{y}_1)^{\bar{y}_2}) = \bar{y}_1 \cdot \bar{y}_2 = \frac{y_1 \cdot y_2}{q^2}. \tag{14}$$

Here the presence of factor q^2 from Eq. (14) indicates that the size of the encrypted value would grow exponentially with the number of multiplications over plaintext. However, this is not an issue in our scheme, because the computation of Eq. (6) in the ProbSolve phase only needs one-time multiplication between each scaled value pair $\overline{\mathbf{T}[i,j]}$ and $\bar{y}_j^{(k)}$ for $j = 1, 2, \ldots, n$. Consequently, after the customer decrypts the result $\mathbf{T} \cdot \mathbf{y}^{(k)}$ of Eq. (6), the compensation factor for the final real value of each component would be q^2 (but not q).

6.2 Application over a Series of Equations $\{\mathbf{Ax} = \mathbf{b}_i\}$

Lots of real world engineering computing problems can be formulated directly or indirectly as systems of linear equations. Moreover, many of these problems would not only demand the solution of a single system of linear equations $\mathbf{Ax} = \mathbf{b}$, but also require to solve a matrix equation multiple times for blocks of right-hand-side constant terms $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_s$. (See Chapter 1.4 from [10] and references therein for such example applications of large dense system of LE on electromagnetic problems and others.) Our proposed mechanism can be applied to these problem scenarios and explore the flexibility in reusing encrypted matrix $\mathsf{Enc}(\mathbf{T})$ in Eq. (5), which not only amortizes the one-time cost on trusted workstation for encrypting the coefficient matrix by a factor of s, but also increases the usability of the proposed mechanism.

However, the encrypted matrix cannot be reused in unlimited times, since it would give cloud server accumulated information to establish matrix equations to solve for $\mathbf{T}$. (see Section 4.4). Thus, for security purposes, the number s of reuse should never exceeds $\lfloor n/\tilde{L} \rfloor$, where $\tilde{L}$ denotes the expected maximum number of iterations for solving one matrix equation $\mathbf{Ax} = \mathbf{b}_i$, $i = 1, 2, \ldots, s$. In this way, we can ensure the group of equations on matrix $\mathbf{T}$ is always underdetermined (and thus have unlimited solutions) and guarantees the acceptable input privacy.

4. In the literature, using scaling factors to handle non-integer values is not new. We acknowledge that similar treatment can be found in many previous work, e.g., [30]–[32], to list a few.

6.3 Utilizing the Cost Associativity of Cloud Computing

To better utilize the benefits of cloud, we propose to explore the famous "cost associativity" of cloud computing, which was first suggested by [33], to speedup the cloud sides computation, i.e., the Eq. (6) in our scheme. Cost associativity states that using 1000 EC2 instances [19] in the cloud for 1 hour costs the same as using 1 instances for 1000 hours. This gives us advantage to parallelize the computation of Eq. (6), which can be decomposed into subtasks running simultaneously on multiple available EC2 instances.

Specifically, when we implement our scheme, we can first separate the encrypted matrix $\mathsf{Enc}(\mathbf{T})$ into t submatrices, each formed by n/t rows out of the n rows of $\mathsf{Enc}(\mathbf{T})$. (we assume n can be divided exactly by t here). Then instead of sending $\mathsf{Enc}(\mathbf{T})$ to a single EC2 instance, we send each submatrix of $\mathsf{Enc}(\mathbf{T})$ to t different EC2 instances in the cloud. When doing the computation of Eq. (6) for the k-th iteration, the customer can send $\mathbf{y}^{(k)}$ to this cluster of t EC2 instances and instruct each of them to compute only partial result based on its possessed submatrix, which would be a subvector with n/t components. After collecting all the subvectors, the customer simply decrypts and merges them to get the result of $\mathbf{T} \cdot \mathbf{y}^{(k)}$. In general, initiating t EC2 instances in the cloud allows us to reduce the overall cloud side computation time to $1/t$, with no extra cost to customers. This will be validated in our experiment in the next section.

7 Performance Analysis

We implement our mechanisms using C language. Algorithms utilize the GNU Scientific Library, the GNU Multiple Precision Arithmetic Library, and the Paillier Library with modulus N of size 1024 bit. The customer side process is conducted on a laptop with Intel Core 2 Duo processor running at 2.16 GHz, 1 GB RAM, and a 5400 RPM Western Digital 250 GB Serial ATA drive with an 8MB buffer. The cloud side process is conducted on Amazon Elastic Computing Cloud (EC2) with High-Memory instance type [19]. The scaling factor for real numbers is set to be 10^3. Our randomly generated diagonally-dominant test benchmark focuses on the large-scale problems only, where n ranges from 5,000 to 50,000. The solutions are all converged within 50 iterations. Since the computation dominates the running time as evidenced by our experiment, we ignore the communication latency between the customer and the cloud for this application. All results represent the mean of 10 trials. In order to handle large-scale matrix-vector operations, proper matrix splitting approaches are to be used, which also demonstrates how the IO cost could significantly downgrade the performance if the whole computation is solely performed on the customer's local machine.

TABLE 1: Transformation cost for different size of problems.

Benchmark			ProbTransform cost
#	dimension	storage size	transformation time
1	n = 5,000	200 MB	7.35 sec
2	n = 8,000	512 MB	28.17 sec
3	n = 1,0000	800 MB	47.61 sec
4	n = 20,000	3.2 GB	less than 4 mins
5	n = 30,000	7.2 GB	less than 7 mins
6	n = 40,000	12.8 GB	less than 13 mins
7	n = 50,000	20 GB	less than 23 mins

7.1 Problem Transformation Cost

We first summarize the cost for customer performing ProbTransform. As shown in Section 4.1, the transformation cost is dominated by the two matrix-vector multiplication in Eq. (5). Note that when n goes large, the resulted matrix would be too large to be hold in customer's local machine memory. Thus, the matrix-vector multiplication cannot be performed in one step. Instead, the matrix has to be split into multiple submatrices, and each time only a submatrix can be loaded in the memory for computing a portion of the final result. In our experiment with 1 GB RAM laptop, we split the matrix into submatrices with 200 MB each for easy in-memory operation. The time results for different problem sizes are shown in Table 1. For the largest benchmark size $n = 50,000$, the problem transformation only costs around 22 minutes on our laptop. Compared to the baseline experiment where the customer solves the equation by himself (shown in the next section), such computational burden should be considered practically acceptable and it can be easily amortized throughout the overall iterative algorithm executions.

7.2 Local Computation Comparison for Problem Solving

In our protocol by harnessing the computation power of cloud, the dominant operation in each iteration for customer is only to perform n decryptions. If the customer solves the problem by himself, which is the baseline of our comparison, the dominant computation burden within each iteration would be the matrix-vector multiplication with the input size n^2. We compare the two computation cost in table 2, where both timing results and estimated memory consumption for each single algorithm iteration are reported. To better present the trend of the efficiency gain between the two experiments, the timing comparison results are also plotted in Fig. 2.

To have a fair comparison, again we have to consider the memory requirements incurred by the two operation. In particular, when n goes large, the IO time has to be taken into consideration. Similar to the transformation cost test, in our baseline experiment, each matrix is split into submatrices with 200 MB each for easy in-memory arithmetic operation. In this way, when performing the

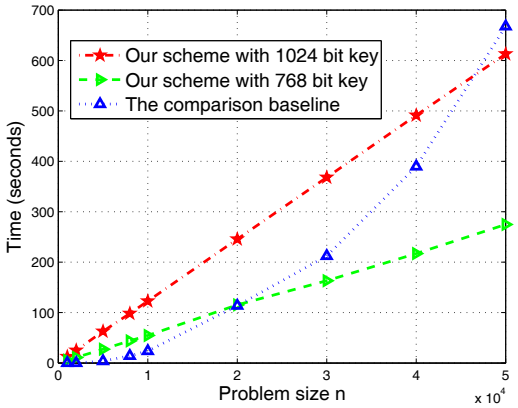

Fig. 2: Comparison of customer local computation cost among baseline and our scheme with different choices of key length.

matrix-vector multiplication for a $50,000 \times 50,000$ matrix with 20 GB space, at least 100 times expensive IO operations for a 200 MB submatrix have to be performed, which significantly increases the total time cost in our baseline experiment, as shown in Fig. 2. On the other hand, our proposed scheme only demands local n decryption operations, which does not have such high memory demands. For 1024 bit key, holding the $50,000 \times 1$ encrypted vector only needs $50,000 \times 256$Bytes < 13.0 MB memory, which can be easily satisfied by modern portable computing devices. Thus, the total local computation cost simply goes linearly with the problem size n. For completeness, we also conduct the experiment with reduced key length of the Paillier cryptosystem. This is motivated by applications where only short term guarantees of secrecy (the coefficient matrix of $\mathbf{A}$) may be required. (See short key experiments in [32], [34] for example.) Thus, if the customer decides a smaller key length is acceptable, the total timing will be reduced.

We can see that the crossover point occurs around $n = 46,000$ for a 1024 bit key and $n = 20,000$ for a 768 bit key in Fig. 2, where the trend of the efficiency gain among $\mathcal{O}(n)$ and $\mathcal{O}(n^2)$ is also clearly shown. In case of 768 bit key, when $n = 50,000$, the customer's local computation cost in the baseline experiment would be $2.43\times$ more than the proposed scheme. Note that $n = 50,000$ is not an unreasonably large matrix. Many real world application, e.g. problems from electromagnetic community, could easily lead to a dense system of linear equations with more than 200,000 unknowns [10]. Though in this work we didn't try problem size larger than 50,000, the better efficiency gain for larger-scale problems can be easily anticipated (see Table 3) from the clear trend among $\mathcal{O}(n)$ and $\mathcal{O}(n^2)$ shown in Fig. 2. For example, when $n = 500,000$, the anticipated computational saving for customer can be up to 26. $09\times$.

7.3 Cloud Computation Cost

The cloud side computation cost for each iterated algorithm execution is given in Table 4. The third and forth columns lists the cloud computation time when there is only one instance running. However, as stated previously in Section 6.3, we can utilize the cost associativity of cloud computing to speedup the cloud server computation via task parallelization without introducing additional cost to customers. Thus, the fifth and the sixth columns lists the estimated cloud computation time when multiple t Amazon EC2 instances are running simultaneously. By configuring a proper choice of $t = 100$, even for the largest size of the problem $n = 50,000$, the cloud side computation can be finished within around 20 minutes for each round. Given the security property our mechanism has provided, such time cost should be deemed reasonable in practice.

8 Related Work

8.1 Secure Computation Outsourcing

Recently, a general result of secure computation outsourcing has been shown viable in theory [20], which is based on Yao's garbled circuits [35] and Gentry's fully homomorphic encryption (FHE) scheme [36]. However, applying this general mechanism to our daily computations would be far from practical, due to the extremely high complexity of FHE operation and the pessimistic circuit sizes that can hardly be handled in practice. Instead of outsourcing general functions, in the security community, Atallah et al. explore a list of customized solutions [26], [27], [37] for securely outsourcing specific computations. In [37], they give the first investigation of secure outsourcing of numerical and scientific computation, including LE. Though a set of problem dependent disguising techniques are proposed, they explicitly allow private information leakage. Besides, the important case of result verification is not considered. In [26], Atallah et al. give a protocol design for secure matrix multiplication outsourcing. The design is built upon the assumption of two non-colluding servers and thus vulnerable to colluding attacks. Later on in [27], Atallah et al. give an improved protocol for secure outsourcing matrix multiplications based on secret sharing, which outperforms their previous work [26] in terms of single server assumption and computation efficiency. But the drawback is that due to secret sharing technique, all scalar operations in original matrix multiplication are expanded to polynomials, introducing significant communication overhead. Considering the case of the result verification, the communication overhead must be further doubled, due to the introducing of additional pre-computed "random noise" matrices. In short, these solutions, although elegant, are still not efficient enough for immediate practical uses on large-scale problems, which we aim to address for the secure LE outsourcing in this paper. Very recently, Wang et al. [38] give the first study of secure outsourcing of linear programming

TABLE 2: Customer computation cost comparison among baseline experiment and the proposed mechanism for single algorithm iteration. The asymmetric speedup is the ratio of the time cost in the baseline over the time cost in the proposed scheme, which captures the customer efficiency gain. The entry with "×" indicates the positive efficiency gain is achieved.

Benchmark			Baseline cost		The proposed ProbSolving cost			Asymmetric Speedup	
#	dimension	storage size	time	memory	time (768-bit)	time (1024-bit)	memory	768-bit	1024-bit
1	n = 5,000	200 MB	3.68 sec	200 MB	27.46 sec	62.81 sec	1.3 MB	0.13	0.06
2	n = 8,000	512 MB	14.09 sec	200 MB	43.94 sec	98.24 sec	2.0 MB	0.32	0.14
3	n = 10,000	800 MB	23.78 sec	200 MB	54.74 sec	122.72 sec	2.6 MB	0.43	0.19
4	n = 20,000	3.2 GB	113.65 sec	200 MB	115.32 sec	245.81 sec	5.1 MB	0.99	0.46
5	n = 30,000	7.2 GB	212.33 sec	200 MB	163.29 sec	368.12 sec	7.7 MB	1.3 ×	0.58
6	n = 40,000	12.8 GB	389.76 sec	200 MB	217.20 sec	491.32 sec	10.2 MB	1.79 ×	0.79
7	n = 50,000	20 GB	667.63 sec	200 MB	274.70 sec	612.87 sec	13.0 MB	2.43 ×	1.09 ×

TABLE 3: The anticipated cost comparison among baseline experiment and the proposed mechanism for larger-scale problems where $n > 50000$. Under current experiment setting, only the estimated timing results for single algorithm iteration are reported.

Benchmark		Baseline	The proposed ProbSolving cost		Asymmetric Speedup	
#	size	time cost	time (768-bit)	time (1024-bit)	768-bit key	1024-bit key
1	n = 200,000	3.1 hours	18.2 mins	40.8 mins	10.28 ×	4.60 ×
2	n = 500,000	19.8 hours	45.6 mins	1.7 hours	26.09 ×	11.65 ×
3	n = 1,000,000	79.7 hours	1.5 hour	3.4 hours	52.44 ×	23.41 ×

TABLE 4: Cloud side computation cost for different choices of keys and number of simultaneously running EC2 instances t.

Benchmark		Time cost with one instance		Estimated time with instances $t = 10$	
#	size	768-bit key	1024-bit key	768-bit key	1024-bit key
1	n = 5,000	12 mins	20 mins	1.2 mins	2 mins
2	n = 8,000	30 mins	53 mins	3 mins	5.3 mins
3	n = 10,000	48 mins	1.3 hours	4.8 mins	7.8 mins
Benchmark		Time cost with one instance		Estimated time with instances $t = 100$	
#	size	768-bit key	1024-bit key	768-bit key	1024-bit key
4	n = 20,000	3.2 hours	5.4 hours	1.9 mins	3.2 mins
5	n = 30,000	7.2 hours	12.3 hours	4.3 mins	7.4 mins
6	n = 40,000	12.9 hours	20.7 hours	7.7 mins	12.4 mins
7	n = 50,000	20.2 hours	34.4 hours	12.1 mins	20.6 mins

in cloud computing. Their solution is based on problem transformation, and has the advantage of bringing customer savings without introducing substantial overhead on cloud. However, those techniques involve cubic-time computational burden matrix-matrix operations, which the weak customer in our case is not necessarily able to handle for large-scale problems.

8.2 Secure Two-party Computation

Securely solving LE has also been studied under the framework of SMC [35]. As we discussed in Introduction, work under SMC may not be well-suited for the computation outsourcing model, primarily because they generally do not address the computation asymmetry among different parties. Besides, in SMC no single involved party knows all the problem input information, making result verification usually a difficult task. But in our model, we can explicitly exploit the fact that the customer knows all input information and thus design efficient batch result verification mechanism. Under the SMC setting, Cramer and Damgård et al. initiate the study of secure protocols for solving various linear algebra problems [11]. Their work is done in the information theoretic multi-party setting, and focus on achieving constant round complexity. Later on, some other theoretical work on this topic have been proposed [12]–[14], each improving the previous results in terms of different round complexity and communication complexity, respectively. Readers are referred to [14] for the detailed comparison. Note that these work are all developed under the theoretical SMC framework and may not be directly applied to our settings of computation outsourcing. Besides, even the most communication efficient work such as [12], [13] focus only on the direct method based solution, and thus are not necessarily able to tackle the large-scale problem that we are handling in this paper. Du et al. also [16] propose a scheme under SMC setting for solving LE based on secret sharing.

But they utilize the heavy oblivious transfer protocol which could incur large communication and computation complexity of the two involved parties. Recently, Troncoso-Pastoriza et al. [15] give the first study on iterative methods for collaboratively solving systems of linear equations under the SMC framework, which is the closest related work that we are aware of in the literature. However, their work can only support very limited number iterative algorithm execution and cannot support reasonably large-scale systems of linear equations due to the continuing growth of the ciphertext in each iteration, which is not the case in our design. In addition to the computation asymmetry issue mentioned previously, they only consider honest-but-curious model and thus do not guarantee that the final solution is correct under the possible presence of truly malicious adversary.

8.3 Computation Delegating and Cheating Detection

Detecting the unfaithful behaviors for computation outsourcing is not an easy task, even without consideration of input/output privacy. Verifiable computation delegation, where a computationally weak customer can verify the correctness of the delegated computation results from a powerful but untrusted server without investing too much resources, has found great interests in theoretical computer science community. Some recent general result can be found in Goldwasser et al. [39]. In distributed computing and targeting the specific computation delegation of one-way function inversion, Golle et al. [40] propose to insert some pre-computed results (images of "ringers") along with the computation workload to defeat untrusted (or lazy) workers. Szada et al. [41] extend the ringer scheme and propose methods to deal with cheating detection of other classes of computation outsourcing, including optimization tasks and Monte Carlo simulations. In [42], Du. et al. propose a method of cheating detection for general computation outsourcing in grid computing. The server is required to provide a commitment via a Merkle tree based on the results it computed. The customer can then use the commitment combined with a sampling approach to carry out the result verification, without re-doing much of the outsourced work. However, all above schemes allow server actually see the data and result it is computing with, which is strictly prohibited in secure computation outsourcing model for data privacy. Thus, the problem of result verification essentially becomes more difficult, when both input/output privacy is demanded. Our work leverages the algebraic property of matrix-vector multiplication and effectively integrate the batch result verification within the mechanism design, introducing very small amount of extra overhead on the customer.

Difference from Conference Version Portions of the work presented in this paper have previously appeared as an extended abstract in [1]. We have revised the article a lot and improved many technical details as compared to [1]. The primary improvements are as follows: Firstly, we provide a new mechanism design on secure outsourcing LE via direct method in Section 3.2. The discussion on the pros and cons of this mechanism justifies our observation and motivation for choosing iterative method as our base for the secure outsourcing mechanism design. Secondly, we provide a new Section 6, where we thoroughly discuss the series of practical techniques and mechanism parameter considerations that should be taken into account when implementing the mechanism for specific applications. Thirdly, we provide a new Section 8.3 in Related Work which include discussions and comparisons over another important branch of research works on data computation delegation and result verification, which are closely related to our cheating detection mechanism design. Finally, we provide a complete yet rigorous security proof for the theorem 1 in Section 5.2, which is lacking in [1].

9 Concluding Remarks

In this paper, we investigated the problem of securely outsourcing large-scale LE in cloud computing. Different from previous study, the computation outsourcing framework is based on iterative methods, which has the benefits of easy-to-implement and less memory requirement in practice. This is especially suitable for the application scenario, where computational constrained customers want to securely harness the cloud for solving large-scale problems. We also investigated the algebraic property of the matrix-vector multiplication and developed an efficient and effective cheating detection scheme for robust result verification. Thorough security analysis and extensive experiments on the real cloud platform demonstrate the validity and practicality of the proposed mechanism.

Acknowledgement

This work was supported in part by the US National Science Foundation under grant CNS-0831963.

References

[1] C. Wang, K. Ren, J. Wang, and K. M. R. Urs, "Harnessing the cloud for securely solving large-scale systems of linear equations," in *Proc. of the 31st International Conf. on Distributed Computing Systems (ICDCS)*.

[2] P. Mell and T. Grance, "Draft nist working definition of cloud computing," Referenced on Jan. 23rd, 2010 Online at http://csrc.nist.gov/groups/SNS/cloud-computing/index.html, 2010.

[3] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, April 2010.

[4] Cloud Security Alliance, "Security guidance for critical areas of focus in cloud computing," 2009, online at http://www.cloudsecurityalliance.org.

[5] Sun Microsystems, Inc., "Building customer trust in cloud computing with transparent security," 2009, online at https://www.sun.com/offers/details/sun_transparency.xml.

[6] C. Gentry, "Computing arbitrary functions of encrypted data," *Commun. ACM*, vol. 53, no. 3, pp. 97–105, 2010.

[7] K. Forsman, W. Gropp, L. Kettunen, D. Levine, and J. Salonen, "Solution of dense systems of linear equations arising from integral-equation formulations," *IEEE Antennas and Propagation Magazine*, vol. 37, no. 6, pp. 96–100, 1995.

[8] A. Edelman, "Large dense numerical linear algebra in 1993: The parallel computing influence," *International Journal of High Performance Computing Applications*, vol. 7, no. 2, pp. 113–128, 1993.

[9] V. Prakash, S. Kwon, and R. Mittra, "An efficient solution of a dense system of linear equations arising in the method-of-moments formulation," *Microwave and Optical Technology Letters*, vol. 33, no. 3, pp. 196–200, 2002.

[10] B. Carpentieri, "Sparse preconditioners for dense linear systems from electromagnetic applications," Ph.D. dissertation, CERFACS, Toulouse, France, 2002.

[11] R. Cramer and I. Damgård, "Secure distributed linear algebra in a constant number of rounds," in *Proc. of CRYPTO*, 2001, pp. 119–136.

[12] K. Nissim and E. Weinreb, "Communication efficient secure linear algebra," in *Proc. of TCC*, 2006, pp. 522–541.

[13] E. Kiltz, P. Mohassel, E. Weinreb, and M. K. Franklin, "Secure linear algebra using linearly recurrent sequences," in *Proc. of TCC*, 2007.

[14] P. Mohassel and E. Weinreb, "Efficient secure linear algebra in the presence of covert or computationally unbounded adversaries," in *Proc. of CRYPTO*, 2008, pp. 481–496.

[15] J. R. Troncoso-Pastoriza, P. Comesaña, and F. Pérez-González, "Secure direct and iterative protocols for solving systems of linear equations," in *Proc. of 1st International Workshop on Signal Processing in the EncryptEd Domain (SPEED)*, 2009, pp. 122–141.

[16] W. Du and M. J. Atallah, "Privacy-preserving cooperative scientific computations," in *Proc. of 14th IEEE Computer Security Foundations Workshop (CSFW)*, 2001, pp. 273–294.

[17] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed. Society for Industrial and Applied Mathematics, 2003.

[18] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proc. of EUROCRYPT*, 1999, pp. 223–238.

[19] Amazon.com, "Amazon elastic compute cloud," Online at http://aws.amazon.com/ec2/, 2009.

[20] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers," in *Proc. of CRYPTO*, Aug. 2010.

[21] C. Wang, N. Cao, J. Li, K. Ren, and W. Lou, "Secure ranked keyword search over encrypted cloud data," in *Proc. of ICDCS*, 2010.

[22] J. Li, Q. Wang, C. Wang, N. Cao, K. Ren, and W. Lou, "Fuzzy keyword search over encrypted data in cloud computing," in *Proc. of IEEE INFOCOM'10 Mini-Conference*, San Diego, CA, USA, March 2010.

[23] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained access control in cloud computing," in *Proc. of INFOCOM*, 2010.

[24] G. Dahlquist and A. Bjorck, *Numerical Methods*. Courier Dover, 2003.

[25] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. MIT press, 2008.

[26] D. Benjamin and M. J. Atallah, "Private and cheating-free outsourcing of algebraic computations," in *Proc. of 6th Conf. on Privacy, Security, and Trust (PST)*, 2008, pp. 240–245.

[27] M. Atallah and K. Frikken, "Securely outsourcing linear algebra computations," in *Proc. of ASIACCS*, 2010, pp. 48–59.

[28] M. Bellare, J. Garay, and T. Rabin, "Fast batch verification for modular exponentiation and digital signatures," in *Proceedings of Eurocrypt 1998, volume 1403 of LNCS*. Springer-Verlag, 1998, pp. 236–250.

[29] J. Camenisch, S. Hohenberger, and M. Pedersen, "Batch verification of short signatures," in *Proceedings of Eurocrypt EUROCRYPT, volume 4515 of LNCS*. Springer-Verlag, 2007, pp. 243–263.

[30] C. Orlandi, A. Piva, and M. Barni, "Oblivious Neural Network Computing via Homomorphic Encryption," *EURASIP Journal on Information Security*, vol. 2007, pp. 1–10, 2007.

[31] S. Han and W. K. Ng, "Privacy-preserving linear fisher discriminant analysis," in *Proc. of the 12th Pacific-Asia conf. on Advances in knowledge discovery and data mining*.

[32] S. Han, W. K. Ng, L. Wan, and V. C. Lee, "Privacy-preserving gradient-descent methods," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 6, pp. 884–899, 2010.

[33] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," University of California, Berkeley, Tech. Rep. UCB-EECS-2009-28, Feb 2009.

[34] J. Bethencourt, D. X. Song, and B. Waters, "New techniques for private stream searching," *ACM Trans. Inf. Syst. Secur.*, vol. 12, no. 3, 2009.

[35] A. C.-C. Yao, "Protocols for secure computations (extended abstract)," in *Proc. of FOCS*, 1982, pp. 160–164.

[36] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc of STOC*, 2009, pp. 169–178.

[37] M. J. Atallah, K. N. Pantazopoulos, J. R. Rice, and E. H. Spafford, "Secure outsourcing of scientific computations," *Advances in Computers*, vol. 54, pp. 216–272, 2001.

[38] C. Wang, K. Ren, and J. Wang, "Secure and practical outsourcing of linear programming in cloud computing," in *Proc. of IEEE INFOCOM*, 2011.

[39] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum, "Delegating computation: interactive proofs for muggles," in *Proc. of STOC*, 2008, pp. 113–122.

[40] P. Golle and I. Mironov, "Uncheatable distributed computations," in *Proc. of CT-RSA*, 2001, pp. 425–440.

[41] D. Szajda, B. G. Lawson, and J. Owen, "Hardening functions for large scale distributed computations," in *Proc. of IEEE Symposium on Security and Privacy*, 2003, pp. 216–224.

[42] W. Du, J. Jia, M. Mangal, and M. Murugesan, "Uncheatable grid computing," in *Proc. of ICDCS*, 2004, pp. 4–11.