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

K.Satyanarayana Murthy MSc(Maths),B.Ed,PGDCA
Asst.Professor
Depart of Humanities and Basic Sciences
PYDAH College of Engineering, Patavala, Kakinada
(Affiliated to JNTU-Kakinada, Andhrapradesh – 533001)

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 LEs 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.

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

1 Introduction

In cloud computing, customers with computationally weak devices are now 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 [2]. Despite the tremendous benefits, the fact that customers and cloud are not necessarily in the same trusted domain brings many security concerns and challenges toward this promising computation outsourcing model [3]. First, 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. 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 [4]. Second, since the operational details inside the cloud are not transparent enough to customers [3], 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 (CS) 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 data and to validate the computation result integrity, it would be hard to expect customers to turn over control of their computing needs from local machines to cloud solely based on its economic savings.

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 [9], [10], [11], [12], [13], [14] have already been proposed from the

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. For example, by applying Newton's method, to solve a system modeled by nonlinear equations converts to solve a sequence of systems of linear equations. Also, by interior point methods, system optimization problems can be converted to a system of nonlinear equations, which is then solved as a sequence of systems of linear equations as mentioned above. By "large," we mean the storage requirements of the system coefficient matrix may easily exceed the available memory of the customer's computing device [5], like a modern portable laptop. In practice, there are many real-world problems that would lead to very large-scale and even systems of linear equations with up to hundreds of thousands [6], [7] or a few million unknowns [8].

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 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 on the location of the data in the memory hierarchy [5], solving such large-scale problems on custo- mer's weak computing devices can be practically impos- sible, due to the inevitably involved huge IO cost. Thus, resorting to cloud for such computation intensive tasks can be arguably the only choice for customers with weak computing .

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, 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 to seek help from cloud). Second, the framework of SMC usually does not directly consider the computation result verification as an indispensable security requirement, due to the assumption that each involved party is semihonest. This assumption is not true any more in our model, where any unfaithful behavior by the cloud during the computation 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 [10], or the secure matrix inversion method in [11]. 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 (see discussions in Appendix D, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2012.206).

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. Compared to direct method, iterative method only demands relatively simpler matrix-vector operations with $O(n^2)$ computational cost, which is much easier to implement in practice and widely adopted for large-scale LE [6], [8], [15]. 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 encryption scheme, e.g., the Paillier cryptosystem [16], and allows customers with weak computing 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 n coefficient matrix, the proposed mechanism is based on a one-time amortizable setup with n^2 cost. Then, in each iterative algorithm execution, the proposed mechanism only incurs n local computational burden to the customer and asymptotically eliminates the expensive IO cost, i.e., no unrealistic memory demands. To ensure computation result integrity, 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. Both designs ensure computational savings for the customer. Our contributions are summarized below: For the first time, we formulate the problem of securely outsourcing large-scale systems of LE via iterative methods, and provide mechanism designs fulfilling input/output privacy, cheating resilience, and efficiency.

1. Our mechanism brings computational savings. Within each iteration, it incurs $O(n)$ computation burden for the customer and demands no unrealistic IO cost, while solving large-scale LE locally incurs $O(n^2)$ per-iteration cost in terms of both time and memory requirements [8].
2. We explore the algebraic property of matrix-vector operations to design a batch verification mechanism, which allows customers to verify all results of previous iterations from cloud in one batch. It ensures both the efficiency advantage and robustness of the design.
3. The experiment on Amazon EC2 [17] shows our mechanism helps customers achieve up to 2:22 × savings when the sizes of the LE are relatively small ($n \leq 50{,}000$). Better efficiency gain can be easily anticipated when n goes to larger size. In particular, when $n = 500{,}000$, the anticipated savings can be up to 26:09×.

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 and security analysis in Sections 3, 4, and 5. Section 6 gives the performance evaluation, followed by Section 7 which overviews the related work. Finally, Section 8 gives the concluding remark.

2 Problem Statement

System and Threat Model

We consider a computation outsourcing architecture involving *cloud customer* and *cloud server* illustrated in Fig. 1. The customer has a large-scale LE problem $A = b$, denoted as (A, b), to be solved. However, due to the lack of computing resources, he cannot carry out such expensive (n^p $2 < p \leq 3$) computation locally. Thus, the customer resorts to cloud server for solving the LE problem. For data protection, the customer first uses a secret key K to map $$ into some encrypted version $_K$. Then, based on $_K$, the customer starts the computation outsourcing protocol with CS, and harnesses the cloud resources in a privacy-preserving manner. The CS is expected to help the customer finding the answer of $_K$,

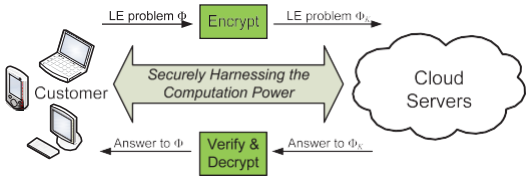

Fig. 1. Architecture of secure outsourcing large-scale systems of linear equations in cloud computing.

but supposed to learn as little as possible on the sensitive information in Ø. After receiving the solution of encrypted problem $Ø_K$, 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 Ø. 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 $Oðn^2Þ$ 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 [18]). 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 A encrypted on his company's workstation, and then uses his portable device outside while still hoping to make timely decisions (derive solutions x_i) based on different observation b_i in the field, for $i ¼ 1; 2; ... ; s$. 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 $fAx ¼ big$.

The security threats primarily come from the malicious behaviors of CS, which may behave beyond "honest-but-curious" model as assumed by other works on cloud security (e.g., [19], [20], [21]). In addition to being interested in learning the sensitive input/output information of Ø, CS can 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.

Design Goals

To enable secure and practical outsourcing of LE under the aforementioned model, we have the following design goals: 1) Input/output privacy: No sensitive information from the customer's private data can be derived by the cloud server during faithfully performing the LE computation; 2) Robust cheating detection: Output from faithful cloud server must be verified successfully by the customer. No output from cheating cloud server can pass the verification with nonnegligible probability. 3) Efficiency: The local computation burden, in terms of both time and memory requirements, for the customer should be much less than solving the original LE on his own.

1. The encryption on each element of the matrix coefficient is independent, and thus can be easily parallelized. Doing each of these one-time element encryptions does not have to load the whole coefficient matrix in memory in the first place. As an example of parallelization, enabling double threading on a six core system could easily speedup the operation efficiency with a factor of 12.

Preliminaries and Notations

Iterative Method

In many engineering computing and industrial applications, iterative method has been widely used in practice for solving large-scale LE [6], and sometimes is the mandatory choice [15] over direct method due to its ease of implementation and relatively less computational power consumption, including the memory and storage IO requirement [8]. We now review some basics on the general form of stationary iterative methods for solving LE problems. A system of linear equations is written as

$$Ax ¼ b; \qquad ð1Þ$$

where x is the $n×1$ vector of unknowns, A is an $n×n$ (nonsingular) coefficient matrix, and b is an $n×1$ 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 [15] here and throughout the paper presentation for its simplicity. Though extensions to other stationary iterative methods can be possible, we don't study them in the current work. We begin with the decomposition: $A ¼ D þ R$, where D is the diagonal component, and R is the remaining matrix. Then, the (1) can be written as $Dx þ Rx ¼ b$, and finally reorganized as: $x ¼ D^{-1} b - D^{-1} Rx$. According to the Jacobi method, we can use an iterative technique to solve the left hand side of this expression for $x^{ðkþ1Þ}$, using previous value for $x^{ðkÞ}$ on the right hand side. If we denote iteration matrix $T ¼ D^{-1} R$ and $c ¼ D^{-1} b$, the above iterative equations can be represented as

$$x^{ðkþ1Þ} ¼ T · x^{ðkÞ} þ c: \qquad ð2Þ$$

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 [15].

Homomorphic Encryption

Our construction utilizes a semantically secure encryption scheme with additive homomorphic property. Given two integers x_1 and x_2, we have $Encðx_1 þ x_2Þ ¼ Encðx_1Þ · Encðx_2Þ$, and also $Encðx_1 · x_2Þ ¼ Encðx_1Þ^{x_2}$.[2] In our implementation we adopt the Paillier cryptosystem [16]. For a vector $x ¼ ðx_1; x_2; ... ; x_nÞ^T ∈ ðℤ_NÞ^n$, we use $EncðxÞ$ to denote the coordinate-wise encryption of x: $EncðxÞ ¼ ðEncðx_1Þ; Encðx_2Þ; ... ; Encðx_nÞÞ$. For some $n × n$ matrix T, where each of the component $T[i; j]$ in T is from $ℤ_N$, we denote the component-wise encryption of T as $EncðTÞ$,

and we have $\mathrm{Enc}(T_{[i,j]}) = \mathrm{Enc}(T_{[i,j]})$.

3 The Basic Solution

In this section, The basic solution is described under the general framework consisting of three phases: (ProbTransform, ProbSolve, ResultVerify). The analysis of this basic solution gives insights and motivations on our main mechanism design based on iterative methods.

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

$$A'y = b''; \quad (3)$$

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

The customer can then start the ProbSolve phase by outsourcing $\mathcal{O}_K = (A'; b'')$ to the cloud, who solves $\mathcal{O}_K$ and sends back answer y. After verifying the correctness of y, the customer can derive the original x via $x = y - r$.

Remark. While achieving the input/output protection, this approach is not attractive for the following reasons:

1) The local problem transformation cost for matrix multiplication QA is $O(n^3)$, which is comparable to the cost of solving $Ax = b$ [22]. 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. 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 $O(n^p)(2 < p \leq 3)$ computation locally.

In the recent literature [23], [24], Atallah et al. have proposed works for secure outsourcing matrix multiplication using only $O(n^2)$ local complexity. However, in practice those works can hardly be applied to our case of calculating $Q \cdot A$ for (3). The reason is that in their works, either noncollusion servers are required [23], or scalar operations are expanded to polynomials and thus incur huge communication and computation overhead [24]. Both assumptions are difficult to be met in practice. (See detailed discussion at Section 7).

4 The Proposed Solution

such as the statistical calculations [25], the radar cross-section calculations [6], etc.

2. Although proper preconditioning techniques (e.g., [5], [6], [7], [8], [15]) on the coefficient matrix A can significantly improve the performance of iterative method, we do not study the cost of these techniques in this paper. As we focus on the security design, we assume the coefficient matrix A already ensures fast enough convergence behavior, i.e., the number of iterations $L \ll n$.
3. We assume the matrix A is first transformed to $T = D^{-1} \cdot R$, where $A = D + R$ as in (2), and then stored in cloud in its encrypted form $\mathrm{Enc}(T)$ via an additive homomorphic encryption. 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 A or T without further notice.
4. For ease of presentation, we defer the cheating detection to Section 5.

Problem Transformation

For protection of result x, the customer who has coefficient vector b and seeks solution x satisfying $Ax = b$ cannot directly start the ProbSolve with cloud. Thus, we need a transformation technique to allow the customer to properly hide such information first. Similar to the basic mechanism, in the ProbTransform phase, the customer picks a random vector $r \in \mathbb{R}^n$ as his secret keying material, and rewrites (1) as the new LE problem

$$Ay = b'; \quad (4)$$

where $y = x + r$ and $b' = b + Ar$. Clearly, the solution x to (1) can be found by solving a transformed LE problem in (4), and vice versa. At this point, both the output x and input tuple b have been well hidden by random vector r. Next, we reformulate (4) into the iterative form similar as (2):

$$y^{(k+1)} = T \cdot y^{(k)} + c'; \quad (5)$$

where $T = -D^{-1} \cdot R$, $c' = D^{-1} \cdot b'$, and $A = D + R$. Now the problem input $\mathcal{O} = (A; b)$ is changed to tuple

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 mechanism design to be explored next, we first make some general but nonstringent assumptions about the system as follows:

1. We assume the coefficient matrix A is a general nonsingular matrix that ensures a solution to the system after convergence of iterative approximations.[2] For simplicity, one such example for A is to be a strictly diagonally dominant matrix. Note that this is not a stringent requirement, as many real-world formulated LE problems satisfy this assumption,

2. According to the general result in [15], the necessary and sufficient condition for convergence is that the maximum modulus of eigenvalues of iteration matrix $T = -D^{-1} \cdot R$ is less than 1.

$\emptyset_K(T; c')$, where T has already been encrypted and stored as $Enc_{pk}(T)$ at cloud, and c' is just a randomly masked version of b via random $n \times 1$ vector r. The output x is also masked by $y = x + r$. This whole procedure is summarized as Algorithm 1 in Appendix B, available in the online supplemental material.

Remark. This problem transformation only requires locally two matrix-vector multiplications: $b' = b + Ar$ and $c' = D^{-1} \cdot b'$ with $n^2 + n$ scalar multiplications. When n goes large, expensive IO cost at customer device might downgrade the performance of such operations. By comparing (2) and (5), it is easy to see that this transformation does not affect the matrix of A (or T), which gives us advantage of reusing. Specifically, the customers with different constant terms b_i can run this transformation multiple times by choosing a different r each time and then harness the cloud for solving different LE problems $\{Ax = b_i\}$, as seen in Appendix A.2, available in the online supplemental material.The Iterative Problem Solving

After the problem transformation step, now we are ready for the ProbSolve phase. Our goal is to let the customer securely harness the cloud for the most expensive computa- tion, i.e., the matrix-vector multiplication $T \cdot y^{(k)}$ in (5) for each algorithm iteration, $k = 1; 2; \ldots ; L$. Since it is an iterative computing process, we only describe the very first round of the process as follows. We leave the analysis of convergence and input/output protection in later sections. In what follows, we assume 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. Details on how to handle noninteger numbers is given in Appendix A, available in the online supplemental material.

1. For the very first iteration, the customer starts the initial guess on the vector $y^{(0)} = (y^{(0)}; y^{(0)}; \ldots; y^{(0)})^T$,

Since we assume the matrix A ensures convergence, to determine the termination, the customer tests if

$$\|y^{(k)} - y^{(k+1)}\| \le s; \qquad (7)$$

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

Input/Output Privacy Analysis

Output Privacy Analysis

and then sends it to the cloud.

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

$$\mathrm{Enc}(T \cdot y^{(0)})[i] = \mathrm{Enc}\left(\sum_{j=1}^{n} T[i,j] \cdot y_j^{(0)}\right) = \prod_{j=1}^{n} \mathrm{Enc}(T[i,j])^{y_j^{(0)}} \qquad (6)$$

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

3. After receiving $\mathrm{Enc}(T \cdot y^{(0)})$, the customer decrypts and gets $T \cdot y^{(0)}$ using his private key. He then updates the next approximation $y^{(1)} = T \cdot y^{(0)} + c^{(0)}$ via (5).

For the kth iteration, it follows that the customer sends the kth approximation $y^{(k)}$ to cloud. The cloud sends $\mathrm{Enc}(T \cdot y^{(k)})$ to the customer for the next update of $y^{(k+1)}$. The protocol continues until the result converges, as shown in Algorithm 2 in Appendix B, available in the online supplemental material.

Remark. In each iteration, the dominant customer's compu-

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

this analysis can be easily extended to the case of outsourcing a series of equations $\{Ax = b_i\}$, since for each individual $Ax = b_i$, an independently picked random r_i can be used to protect the output privacy accordingly.

Input Privacy Analysis

While the output is protected well, it is worth noting that some knowledge about the input tuple $\Phi_k = (T, c^{(0)})$ could be implicitly leaked through the protocol execution itself. The

reason is as follows: for each two consecutive iterations of the protocol, namely, the kth and the $(k+1)$th, the cloud server sees actually the plaintext of both $y^{(k)}$ and $y^{(k+1)}$. Thus, a "clever" cloud server could initiate a system of linear equations via (5) and attempts to learn the unknown components of T and $c^{(0)}$. More specifically, for the total L iterations, the cloud server could establish a series of $(L-1) \times n$ equations from $y^{(k)}$, $k = 0, 1, \ldots, L-1$,[3] while hoping to solve $n^2 + n$ unknowns of T and $c^{(0)}$.

However, as we have assumed in Section 4 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 $n^2 + n$

tation overhead is to decrypt the vector of $\mathrm{Enc}(T \cdot y)$, which takes $O(n)$ complexity, and in general does not require expensive IO cost. This is theoretically less than the $O(n^2)$ cost demanded by the matrix-vector multiplication $T \cdot y^{(k)}$ of (5) in terms of time and memory requirements. Note that the theoretical computation efficiency gain can only be exhibited when the problem size n goes large, since the decryption computation is generally more expensive than the plaintext arithmetic operation. But large-scale LE is exactly the case we are focusing on by using iterative methods. Later in Section 6, we show performance results and discuss the possible selections of the size n for the problems. 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 small.

4.3 Convergence Analysis

When dealing with iterative methods, it is a must to determine whether and when the iteration will converge. components of T and 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 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. Here, the choice of a_k should not introduce overflows with respect to the large arithmetic modulo N. Specifically, instead of sending $y^{(k)}$ to the cloud server, the customer sends $a_k \cdot y^{(k)}$ for the kth iteration of the ProbSolve. When the cloud server sends back the encrypted value $\mathrm{Enc}(a_k T y^{(k)})$, the customer just simply decrypts the vector of $a_k T y^{(k+1)}$, divides each component with a_k, and then updates the next approximation $y^{(k+1)}$ via (5). Similarly,

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

for the next iteration another random scaling factor a_{k+1} is multiplied to $y^{(k+1)}$ before sent to the cloud server.

Remark. With the random scaling factor a_k, the original value of $y^{(k)}$ is well protected via $a_k y^{(k)}$. Thus, the cloud server can no longer directly establish linear equations from received $a_k y^{(k)}$ and $a_{k+1} y^{(k+1)}$, but a series of nonlinear equations with extra random unknowns $a_1; a_2; \ldots; a_L$. 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, Finally, we should note that the above analysis on the input privacy does not affect the output protection of x. This is because the random secret r protects x from y and the original constant term b from transformed c'.

5 Cheating Detection

Till now, the proposed protocol works only under the assumption of honest but curious cloud server. However, in many cases, an unfaithful cloud server could sabotage the protocol execution by either being lazy or intentionally corrupting the computation result. Next, 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. In the following, we denote $z^{(k)} = T \cdot y^{(k)}$ as the expected correct responses, and $\hat{z}^{(k)} = T \cdot \hat{y}^{(k)}$ as the actual received value from cloud server, where matrix-vector multiplication of (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 (6) via arbitrary vectors $\hat{y}^{(k)}$ other than customer's $y^{(k)}$. 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 $T \cdot y^{(k)}$ and check if it equals to the received $\hat{z}^{(k)}$ 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 $\hat{z}^{(k)} = T \cdot \hat{y}^{(k)}$, $k = 1; 2; \ldots; L$ in only one batch, i.e., using only one matrix- vector multiplication. Note that batch verification is not a new idea and has been studied in cryptographic contexts, e.g., fast digital signature verifications [26].

Suppose after L iterations, the solution still does not converge. The customer can initiate a ResultVerify phase by randomly selecting L numbers, $a_1; a_2; \ldots; a_L$ from $B \in \mathbb{Z}_N$, where each a_k is of l-bit length and $l < \log N$. He then computes the linear combination β over the $y^{(k)}$'s, which he has provided in the previous k iterations, $k = 1; 2; \ldots; L$:

$$\beta = \sum_{k=1}^{L} a_k y^{(k)}.$$

Next, to test the correctness of all the intermediate results $\hat{y}^{(k)} = T \cdot \hat{y}^{(k)}$, $k = 1; 2; \ldots; L$, received from cloud server, the customer simply checks if the following equation holds:

$$\sum_{k=1}^{L} a_k \hat{z}^{(k)} \stackrel{?}{=} T \cdot \beta + \sum_{k=1}^{L} a_k b. \quad (9)$$

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 kth iteration, the adversary could simply reply the result $z^{(k-1)}$ of the previous $(k-1)$th iteration without computation.

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

$$\|Ay^{(k+1)} - b\| \le s. \qquad (8)$$

Remark. This checking equation incurs the local cost of $O(n^2)$ for customer. While potentially expensive for 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 $y^{(k)}$ and $y^{(k+1)}$ via (7) is passed. If (7) is not passed, it means $y^{(k+1)}$ is not the convergence point yet. On the other hand, if (7) is successfully passed, we can then initiate the test of (8). If (8) holds, we say the final solution is found, which is $x = y^{(k+1)} - r$. If it doesn't, we can tell that the cloud server is cheating (being lazy). In either case, this Since each a_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 (9) holds if and only if* $\hat{z}^{(k)} = T \cdot y^{(k)}$ *for all* $k = 1, 2, \ldots, L$, *with error probability at most* 2^{-l}.

Proof. See Appendix C, available in the online supplemental material. □

Remark. It is easy to tell that the computation overhead of (9) is only bounded by one matrix-vector multiplication of the left-hand-side of the equation (recall $L \le n$). The size of l is a tradeoff between efficiency and security.

$$T \cdot \& \stackrel{?}{=} \sum_{k=1} a_k \cdot \hat{z}^{(k)}: \qquad (9)$$

The above equation can be elaborated as follows:

$$T \cdot \& = T \cdot \sum_{k=1} a_k \cdot y^{(k)} = \sum_{k=1}^{1} a_k \cdot T \cdot y^{(k)} = \sum_{k=1}^{1} a_k \cdot \hat{z}^{(k)}:$$

TABLE 1
Transformation Cost for Different 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 = 10,000	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

reasonable choice of 20 bits of l is also acceptable [27]. 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 $f^{\hat{}}z^{\eth k\flat} \frac{1}{4} T \cdot y^{\wedge \eth k\flat}g$, $k \frac{1}{4} 1; 2; \ldots; L$. Thus, the custo- mer can preset the threshold L as sufficiently large such that either he detects the unfaithful behavior of cloud server or the program will converge soon after L iterations. In the best case, (9) only needs to be instantiated once. Combined with (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 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 1,024 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 5,400 RPM Western Digital 250 GB Serial ATA drive with an 8 MB buffer. The cloud side process is conducted on Amazon Elastic Computing Cloud (EC2) with High-Memory instance type [17]. Our randomly generated diagonally dominant test benchmark focuses on the large-scale problems only, where n ranges from 5,000 to 50,000, and serve for the purpose of validating the performance of the design. To ensure good condition number and the convergence of solutions, we first generate a random matrix with coefficient ranges from —1 to 1 and then add it with a diagonal matrix with large diagonals, e.g., 100 in case of $n_{\frac{1}{4}}20;000$. The scaling factor for real numbers is set to be 10^3 (See Appendix A, available in the online supplemental material). The solutions are all converged within 50 iterations when termination threshold $s \frac{1}{4} 0{:}001$. Since the computation dominates the running time as evidenced by our experiment, we ignore the communication cost. All results represent the mean of 10 trials. In order to handle large-scale matrix-vector operations, proper matrix splitting approaches are used, which demonstrates how the IO cost could significantly downgrade the performance if the whole computation is solely performed on the customer's local machine.

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 (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. This has taken into account the memory occupation from OS load-up and easy in-memory operations and suits the assumption of weak customer device. The time results for different problem sizes are shown in Table 1. For the largest benchmark size $n \frac{1}{4} 50;000$, 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 for getting one problem solution, when we compare the customer's local average computation cost per iteration.

Local Computation Comparison

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 siz

Paillier cryptosystem. This is motivated by applications

TABLE 2
The Average per-Iteration Cost for Customer Computation Comparison

		Benchmark		Baseline		The proposed design			Asymmetric Speedup	
		storage size	memory	time (768-bit)	time (1024-bit)	memory	768-bit	1024-bit
#	dimension	200 MB	time	200 MB	27.82 sec	163.18 sec	1.3 MB		
1	n = 5,000		3.68 sec					0.13	0.06
2	n = 8,000	512 MB	14.09 sec	200 MB	45.06 sec	99.38 sec	2.0 MB	0.31	0.14
3	n = 10,000	800 MB	23.78 sec	200 MB	56.32 sec	124.31 sec	2.6 MB	0.42	0.19
4	n = 20,000	3.2 GB	113.65 sec	200 MB	121.81 sec	252.30 sec	5.1 MB	0.93	0.45
5	n = 30,000	7.2 GB	212.33 sec	200 MB	171.78 sec	376.67 sec	7.7 MB	1.24 ×	0.56
6	n = 40,000	12.8 GB	389.76 sec	200 MB	232.79 sec	506.91 sec	10.2 MB	1.67 ×	0.77
7	n = 50,000	20 GB	667.63 sec	200 MB	301.41 sec	639.58 sec	13.0 MB	2.22 ×	1.04 ×

The entry with "×" indicates the positive efficiency gain is achieved.

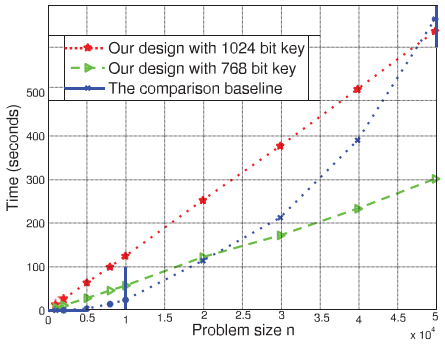

Fig. 2. Comparison of customer's average Per-Iteration computation cost among baseline and our scheme with different choices of key length.

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. Note that the reported measurements are the average per-iteration cost, which have taken into account the problem transformation in amortized fashion already. 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 operations. 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 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 demands. For 1,024 bit key, each ciphertext is of size 2,048 bit, i.e., 256 byte (see Section 2.3.2). Therefore, 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. To show possible tradeoffs between security and performance, we also conduct the experiment with reduced key length of the

where only short term guarantees of secrecy (the coefficient matrix of A) may be required. (See short key experiments in [28], [29], for example.) Thus, if the customer accepts a smaller key length, the total timing will be reduced.

We can see that the crossover point occurs around n 48;000 for a 1,024 bit key and n 22;000 for a 768 bit key in Fig. 2, where the trend of the efficiency gain among n and 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 2:22 more than the proposed scheme. Note that n 50;000 is not an unreason- ably 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 [8]. 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 from the clear trend among n and n^2 shown in Fig. 2. For example, when n 500;000, the anticipated computational saving for customer can be up to 26. 09 . Note that the choice of in-memory storage does not affect the theoretical analysis on the computation gain. Also, from the experi- mental observation, as long as n goes large, i.e., for large- scale LE problems, the computational savings can always be expected by the customer.

Cloud Computation Cost

The cloud side computation cost for each iterated algorithm execution is given in Table 3. The third and forth columns lists the cloud computation time when there is only one instance running. However, as stated in Appendix A.5, available in the online supplemental material, 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 can be deemed reasonable.

7 Related Work

Recently, a general result of secure computation outsourcing has been shown viable in theory [18], which is based on

TABLE 3
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

Yao's garbled circuits [30] and Gentry's fully homomorphic encryption (FHE) scheme [31]. 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 [23], [24], [32] for securely outsourcing specific computations. In [32], 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 [23], Atallah and Bejanmin give a protocol design for secure matrix multiplication outsourcing. The design is built upon the assumption of two noncolluding servers and thus vulnerable to colluding attacks. Later on in [24], Atallah and Frikken give an improved protocol for secure outsourcing matrix multiplications based on secret sharing, which outperforms their previous work [23] 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 precomputed "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. Wang et al. [33] give the first study of secure outsourcing of linear programming 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 may not be handled by the weak customer in our assumption. Very recently, Blanton et al. [34] explored secure outsourcing all-pair distance calculations of large-scale biometric data. Their focus is on result verification, which leverages certain structures of the distance computations and the framework of adding fake items and random sampling.

Difference from conference version [1]. First, we provide a new mechanism design on secure outsourcing LE via direct method in Section 3. Second, we thoroughly discuss the series of practical techniques and mechanism parameter considerations when implementing the mechanism for specific applications in Appendix A, available in the online supplemental material. Third, we provide extended literature survey on data computation delegation and result verification in Appendix D, available in the online supplemental material. Fourth, we provide a complete yet rigorous security proof for the Theorem 1 in Section 5.2 and Appendix C, available in the online supplemental material. Finally, we greatly improved the performance evaluation with more clarified experimental settings and evaluation comparison.

8 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. In particular, the design requires a one-time amortizable setup phase with $O(n^2)$ cost, and then each following iterative algorithm execution only incurs $O(n)$ local computational cost with the benefits of easy-to-implement and less memory requirement in practice. 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.

Acknowledgments

This work was supported in part by the US National Science Foundation (NSF) under grant CNS-1262277, CNS-1116939, and by Amazon web service research grant. The preliminary result was published at ICDCS 2011 [1].

References

[1] C. Wang, K. Ren, J. Wang, and K. Mahendra Raje Urs, "Harnessing the Cloud for Securely Solving Large-Scale Systems of Linear Equations," *Proc. 31st Int'l Conf. Distributed Computing Systems (ICDCS),* pp. 549-558, 2011.

[2] M. Armbrust et al., "A View of Cloud Computing," *Comm. ACM,* vol. 53, no. 4, pp. 50-58, Apr. 2010.

[3] Cloud Security Alliance, "Security Guidance for Critical Areas of Focus in Cloud Computing," http://www.cloudsecurityalliance.org, 2009.

[4] C. Gentry, "Computing Arbitrary Functions of Encrypted Data," *Comm. ACM,* vol. 53, no. 3, pp. 97-105, 2010.

[5] 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, Dec. 1995.

[6] A. Edelman, "Large Dense Numerical Linear Algebra in 1993: The Parallel Computing Influence," *Int'l J. High Performance Computing Applications,* vol. 7, no. 2, pp. 113-128, 1993.

[7] 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.

[8] B. Carpentieri, "Sparse Preconditioners for Dense Linear Systems from Electromagnetic Applications," PhD dissertation, CERFACS, Toulouse, France, 2002.

[9] R. Cramer and I. Damgård, "Secure Distributed Linear Algebra in a Constant Number of Rounds," *CRYPTO: Proc. Ann. Int'l Cryptology Conf. Advances in Cryptology,* 2001.

[10] K. Nissim and E. Weinreb, "Communication Efficient Secure Linear Algebra," *Proc. Third Conf. Theory of Cryptography (TCC),* pp. 522-541, 2006.

[11] E. Kiltz, P. Mohassel, E. Weinreb, and M.K. Franklin, "Secure Linear Algebra Using Linearly Recurrent Sequences," *Proc. Fourth Conf. Theory of Cryptography (TCC),* pp. 291-310, 2007.

[12] P. Mohassel and E. Weinreb, "Efficient Secure Linear Algebra in the Presence of Covert or Computationally Unbounded Adversaries," *CRYPTO: Proc. 28th Ann. Int'l Cryptology Conf.,* pp. 481-496, 2008.

[13] J.R. Troncoso-Pastoriza, P. Comesaña, and F. Pérez-González, "Secure Direct and Iterative Protocols for Solving Systems of Linear Equations," *Proc. First Int'l Workshop Signal Processing in the EncryptEd Domain (SPEED),* pp. 122-141, 2009.

[14] W. Du and M.J. Atallah, "Privacy-Preserving Cooperative Scientific Computations," *Proc. IEEE 14th Computer Security Foundations Workshop (CSFW),* pp. 273-294, 2001.

[15] Y. Saad, *Iterative Methods for Sparse Linear Systems,* second ed. Soc. for Industrial and Applied Math., 2003.

[16] P. Paillier, "Public-Key Cryptosystems Based on Composite Degree

Residuosity Classes," *EUROCRYPT: Proc. 17th Int'l Conf. Theory and Application of Cryptographic Techniques,* pp. 223-238, 1999.

[17] Amazon.com, "Amazon Elastic Compute Cloud," http://aws.amazon.com/ec2/, 2009.

[18] R. Gennaro, C. Gentry, and B. Parno, "Non-Interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers," *CRYPTO: Proc. 30th Ann. Conf. Advances in Cryptology ,* pp. 465-482, 2010.

[19] C. Wang, N. Cao, J. Li, K. Ren, and W. Lou, "Secure Ranked Keyword Search over Encrypted Cloud Data," *Proc. IEEE 30th Int'l Conf. Distributed Computing Systems (ICDCS),* pp. 253-262, 2010.

[20] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving Secure, Scalable, and Fine-Grained Access Control in Cloud Computing," *Proc. IEEE INFOCOM,* pp. 534-542, 2010.

[21] C. Wang, K. Ren, S. Yu, and K. Mahendra Raje Urs, "Achieving Usable and Privacy-Assured Similarity Search Over Outsourced Cloud Data," *Proc. IEEE INFOCOM,* pp. 451-459, 2012.

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

[23] D. Benjamin and M.J. Atallah, "Private and Cheating-Free Outsourcing of Algebraic Computations," *Proc. Sixth Conf. Privacy, Security, and Trust (PST),* pp. 240-245, 2008.

[24] M. Atallah and K. Frikken, "Securely Outsourcing Linear Algebra Computations," *Proc. Fifth ACM Symp. Information, Computer and Comm. Security (ASIACCS),* pp. 48-59, 2010.

[25] G. Dahlquist and A. Bjorck, *Numerical Methods.* Dover Publications, 2003.

[26] M. Bellare, J. Garay, and T. Rabin, "Fast Batch Verification for Modular Exponentiation and Digital Signatures," *Eurocrypt: Proc. Int'l Conf. the Theory and Application of Cryptographic Techniques,* pp. 236-250, 1998.

[27] J. Camenisch, S. Hohenberger, and M. Pedersen, "Batch Verification of Short Signatures," *EUROCRYPT: Proc. 26th Ann. Int'l Conf. Advances in Cryptology,* pp. 243-263, 2007.

[28] J. Bethencourt, D.X. Song, and B. Waters, "New Techniques for Private Stream Searching," *ACM Trans. Information Systems Security,* vol. 12, no. 3, article 16, 2009.

[29] S. Han, W.K. Ng, L. Wan, and V.C. Lee, "Privacy-Preserving Gradient-Descent Methods," *IEEE Trans. Knowledge and Data Eng.,* vol. 22, no. 6, pp. 884-899, June 2010.