LinSOS: Secure Outsourcing of Linear Computations Based on Affine Mapping

Kai Zhou and Jian Ren
Department of ECE, Michigan State University, East Lansing, MI 48824-1226
Email: {zhoukai, renjian}@msu.edu

***Abstract*—Linear computational problems emerge from various fields such as engineering and finance. Due to the large scale of these problems, they are often hard to be processed by resource-constrained devices. Thus, outsourcing becomes a natural solution. In this paper, we propose a Secure OutSourcing scheme (LinSOS) for Linear computations. The proposed scheme (LinSOS) is based on affine mapping and imposes only linear operations at local environment. As a result, the end-users can enjoy impressive computational gains from outsourcing. We also provide a verification scheme such that end-users can always receive valid results. Our extensive security and complexity analysis and performance comparison with existing schemes show that LinSOS is both secure and efficient.**

***Index Terms*—Cloud computing, computation outsourcing, security, efficiency**

I. Introduction

Cloud computing is becoming prevalent in providing end-users various services such as remote storage and computation outsourcing. As as result, the resource-constrained end-users can outsource expensive computational tasks to the cloud in a pay-per-use manner. Despite the tremendous benefits from outsourcing, security and correctness of the results have become two major concerns. This is because the cloud server may try to get as much information as possible from the outsourced data. The cloud server may also return false results in order to save computational resources.

To address these issues, various secure outsourcing mechanisms have been proposed. Among these schemes, the outsourcing of linear computational problems receives much attention due to their broad applications. For example, in [1]–[3], different disguising techniques have been developed to securely outsource some basic scientific operations such as matrix multiplication, matrix inversion and convolution. In [4], the authors focus on the outsourcing of linear programming. In [5], a secure outsourcing scheme is proposed for large-scale systems of linear equations. However, all these schemes are specially designed for a particular kind of linear computation. Thus there lacks a systematic way to address the secure outsourcing of linear computations.

In this paper, we aim at developing a secure outsourcing scheme that is suitable for general linear computational problems. The basic idea is based on affine mapping. Under an affine mapping, the independent variables of the original computational problem are mapped to a new group of variables. As a result, the original problem is transformed to a new form that can be outsourced to the cloud with the original variables concealed. Then the cloud can solve the outsourced problem and return the corresponding result to the end-user. On receiving the returned result, the end-user can recover the valid result to the original problem under an inverse affine mapping efficiently.

The main contributions of this paper can be summarized as follows: (i) We propose a secure outsourcing scheme (LinSOS) that is widely applicable to general linear computational problems. (ii) LinSOS is highly efficient since it imposes processing of $\mathcal{O}(n^2)$ complexity at local environment. (iii) We also propose a verification scheme to guarantee the correctness of the returned results. (iv) We investigate the trade-offs between security and complexity such that the end-users can select different outsourcing strategies according to their own computational constraints and security demands in a cost-aware manner.

The rest of this paper is organized as follows. In Section II, we introduce our system model and threat model. In Section III, we present our design for system of linear equations based on affine mappings and give the security and complexity analysis. In Section IV, we show that the design and analysis can be extended to other linear computations such as linear programming. Then the result verification scheme is introduced in Section V. We evaluate the performance of our scheme and compare it with several existing works in Section VI. We conclude our work in Section VII.

II. System and Threat Model

In this paper, we consider a computation outsourcing system consisting of two entities: the end-user and the cloud. An end-user expects to outsource to the cloud a linear computational problem denoted by $F(\mathbf{x})$, where $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ is the input sequence. The whole outsourcing scheme can be divided into four phases:

1) **Problem transformation**: the end-user transforms the original problem $F(\mathbf{x})$ to a new form $G(\mathbf{y})$, where $\mathbf{y}$ is the new input sequence. Then the transformed problem $G(\mathbf{y})$ is outsourced to the cloud.
2) **Cloud processing**: the cloud solves $G(\mathbf{y})$ and returns the result $\mathbf{y}^*$as well as a proof Γ to the end-user.
3) **Result verification**: based on $\mathbf{y}^*$ and Γ, the end-user verifies the correctness of the returned result.
4) **Result recovery**: if the returned result passes the verification, the end-user recovers the original result $\mathbf{x}^*$ from $\mathbf{y}^*$ through an inverse affine mapping. Otherwise, an error is reported.

978-1-4799-6664-6/16/$31.00 ©2016 IEEE

We assume that the cloud is *malicious* for two reasons. First, the cloud is curious. It may try to collect any sensitive information revealed by the outsourced problem. Second, the cloud may cheat during the computation process in order to save computational resources. As a result, a secure outsourcing scheme should not only preserve the private information but also guarantee that valid results can be successfully recovered at local side.

III. Design and Analysis of LinSOS

In this section, we illustrate each phase of LinSOS. Especially, we explain in detail the process of affine mapping based problem transformation that constitutes the essence of the outsourcing scheme. Then, we give the complexity analysis of LinSOS based on which we introduce our cost-aware design. At last, our security analysis shows that LinSOS is secure in protecting end-user's private information. We deploy system of linear equations as a case study and we then show that LinSOS can be well extended to other linear computations.

A. Problem Transformation Based on Affine Mapping

We assume that the end-user wants to outsource a system of linear equations $F(\mathbf{x}) : \mathbf{Ax} = \mathbf{b}$, where $\mathbf{A} \in \mathbb{R}^{n\times n}$, $\mathbf{b} \in \mathbb{R}^n$ and $\mathbf{x} \in \mathbb{R}^n$ are the independent variables. Under an affine mapping, the independent variables of the original problem are mapped to a new group of variables. That is, $\mathbf{x} = \mathbf{Ky} + \mathbf{r}$, where $\mathbf{K} \in \mathbb{R}^{n\times n}$ and $\mathbf{r} \in \mathbb{R}^n$. Correspondingly, the original problem $\mathbf{Ax} = \mathbf{b}$ is transformed to $\mathbf{AKy} = \mathbf{b} - \mathbf{Ar}$. Denote $\mathbf{A}' = \mathbf{AK}$ and $\mathbf{b}' = \mathbf{b} - \mathbf{Ar}$, then we can get the transformed problem $G(\mathbf{y}) : \mathbf{A}'\mathbf{y} = \mathbf{b}'$ that can be outsourced to the cloud. Then the cloud can solve $G(\mathbf{y})$ to get the result $\mathbf{y}^*$ which is returned to the end-user.

It is clear that as long as $\mathbf{K}$ is nonsingular, the above mentioned affine mapping is a one-to-one mapping. Thus $\mathbf{x}$ is uniquely determined by $\mathbf{y}$. Since the solutions to the two problems satisfy $\mathbf{x}^* = \mathbf{Ky}^* + \mathbf{r}$, given $\mathbf{y}^*$ returned by the cloud, the end-user is able to recover $\mathbf{x}^*$ uniquely at local side. From this, we can conclude that the affine mapping based outsourcing scheme is correct. Provided that the result returned by the cloud is correct, end-user is guaranteed to recover a valid result to the original problem.

B. Complexity Analysis

From the above outsourcing scheme, we can see that the computational overhead mainly lies in matrix multiplication $\mathbf{AK}$. Thus in our discussion, we focus on the complexity of computing $\mathbf{AK}$ and utilize the number of multiplications among the matrix entries, denoted as M, as a measurement. To multiply two arbitrary $n \times n$ matrices, the typical complexity is $\mathcal{O}(n^3)$, which is generally believed to be too high and unacceptable for mobile client computation. Thus, our goal is to limit the computational complexity to $\mathcal{O}(n^2)$ by selecting special forms of $\mathbf{K}$.

In our design, we provide four options of $\mathbf{K}$ described as follows:

1) **Option-1**: $\mathbf{K}$ is a diagonal matrix that has the format $\mathbf{K} = \{k_{ij} | k_{ij} = 0, \forall i \neq j\}$.
2) **Option-2**: $\mathbf{K}$ is a permutation matrix that has exactly one non-zero entry in each row and each column in the matrix.
3) **Option-3**: $\mathbf{K}$ is a band matrix that has an upper half-bandwidth p and a lower half-bandwidth q such that $k_{ij} = 0$ for $i > j+p$ and $j > i+q$. The total bandwidth of $\mathbf{K}$ is denoted by $W = p + q + 1$. For simplicity, we assume that $\mathbf{K}$ has an equal upper and lower half-bandwidth $p = q = \omega$, then $W = 2\omega + 1$.
4) **Option-4**: $\mathbf{K}$ is a sparse matrix with density d which is defined as the ratio of non-zero elements in the matrix. We assume that the number of non-zero elements in each row and each column of $\mathbf{K}$ is up-bounded by a constant θ.

The number of multiplications M in each scheme can be calculated and the results are presented in Table I. In summary, we demonstrate that by selecting special forms of $\mathbf{K}$, the complexity of multiplying $\mathbf{K}$ with an arbitrary matrix $\mathbf{A}$ is $\mathcal{O}(n^2)$. Since matrix multiplication is the most expensive part of end-user's processing, we can derive that the overall computational complexity for the end-user is $\mathcal{O}(n^2)$, which is within the end-user's computational constraints.

C. Security Analysis

In this section, we first show that LinSOS can meet the basic security requirement, that is to protect the input $\{\mathbf{A}, \mathbf{b}\}$ and the output $\{\mathbf{x}^*\}$ of the system of linear equations. Then we analyze how the four options can provide different security levels.

Under the affine mapping, the system of equations is transformed to $\mathbf{A}'\mathbf{y} = \mathbf{b}'$, where $\mathbf{A}' = \mathbf{AK}$ and $\mathbf{b}' = \mathbf{b} - \mathbf{Ar}$. Therefore, both $\mathbf{A}$ and $\mathbf{b}$ are concealed by $\mathbf{K}$ and $\mathbf{r}$, which are kept secret at local side. Additionally, since the original solution is recovered by $\mathbf{x}^* = \mathbf{Ky}^* + \mathbf{r}$, without knowing $\mathbf{K}$ and $\mathbf{r}$, the cloud cannot recover $\mathbf{x}^*$. Thus, all the four options of LinSOS are secure in concealing the input $\{\mathbf{A}, \mathbf{b}\}$ and the output $\{\mathbf{x}\}$ of the system of linear equations.

Then, besides the input and output, we want to quantitatively analyze how much (side) information the cloud can learn from the outsourced problem. We introduce diffusion and confusion as general criteria to characterize the different security levels that LinSOS can provide. These two concepts were first introduced by Shannon [6] and have been successfully applied to security analysis in cryptography such as block ciphers [7]. Due to the similarity between block ciphers and our proposed affine mapping scheme, we apply diffusion and confusion parameter which are defined below as two merits to measure the security of LinSOS.

Definition 1. Under affine mapping $\mathbf{x} = \mathbf{Ky} + \mathbf{r}$, the matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ is transformed to $\mathbf{A}' = \mathbf{AK}$. Then diffusion parameter α and confusion parameter β are defined as the number of different entries in $\mathbf{A}$ and the number of different entries in $\mathbf{K}$ that appear in each entry of $\mathbf{A}'$.

When $\mathbf{K}$ is a diagonal matrix, the entry a'_{ij} in $\mathbf{A}'$ can be calculated as $a'_{ij} = k_{ii}a_{ij}$. For this affine mapping, as defined above, we have the diffusion parameter $\alpha_1 = 1$ and confusion

Table I
COMPLEXITY AND SECURITY OF EACH SCHEME

Scheme	Complexity (M)	Diffusion (α)	Confusion (β)
Option-1	n^2	1	1
Option-2	n^2	1	1
Option-3	Wn^2	W	W
Option-4	θn^2	θ	θ

parameter $\beta_1 = 1$. By investigating $\mathbf{A}'$, it is obvious that each column in $\mathbf{A}'$ is related in a simple way to that in $\mathbf{A}$ such that the i^{th} column in $\mathbf{A}'$ is the multiplication of the i^{th} column in $\mathbf{A}$ with k_{ii}. In this way, only based on $\mathbf{A}'$, the cloud can easily know the ratio between any two entries within the same column in $\mathbf{A}$.

For $\mathbf{K}$ to be a permutation matrix, the diffusion and confusion parameters can be calculated as $\alpha_2 = 1$ and $\beta_2 = 1$ which are the same as in option-1. However, the difference is that $\mathbf{A}'$ in option-2 can be regarded as the result of permuting the columns of $\mathbf{A}'$ obtained from option-1. Thus, although the cloud can get a knowledge of the ratio between two entries in the same column of $\mathbf{A}$, it is not sure which particular column those two entries belong to. Therefore, option-2 is more secure than option-1.

For $\mathbf{K}$ to be a band matrix with upper half-bandwidth and lower half-bandwidth both equal to ω, we have the diffusion and confusion parameter as $\alpha_3 = \beta_3 = 2(\omega + 1) = W$. Since each entry in $\mathbf{A}'$ is a linear combination of α_3 entries in $\mathbf{A}$ and β_3 entries in $\mathbf{K}$, the ratio information of entries in $\mathbf{A}$ is concealed. However, the disadvantage is that the cloud can still learn how a particular entry in $\mathbf{A}'$ is composed. For example, suppose $\omega = 1$, the cloud can know for sure that $a'_{ij} = a_{i(j-1)} + a_{ij} + a_{i(j+1)}$.

At last, for $\mathbf{K}$ to be a sparse matrix, we assume that there are exactly θ non-zero entries in each row and column of $\mathbf{K}$. Then we have diffusion and confusion parameters $\alpha_4 = \beta_4 = \theta$. Similar to option-3, the ratio information of entries in $\mathbf{A}$ can be concealed. Moreover, since the non-zero entries are randomly positioned in the sparse matrix $\mathbf{K}$, the cloud is unable to know how each entry in $\mathbf{A}'$ is composed.

From the above analysis, we can conclude that larger confusion and diffusion parameters will guarantee better protection of side information. We summarize the computational complexity and security of LinSOS in Table I.

From the above complexity and security analysis, we can see the trade-off between the computational complexity and security. In summary, from option-1 to option-4, the security levels that they can provide increase at a cost of computational power. Thus LinSOS actually provides end-users with the flexibility to choose the outsourcing schemes that are most suitable for them. These four options thus give a cost-aware outsourcing scheme for end-users to address the various security demands and computational constraints.

IV. EXTENSION TO LINEAR PROGRAMMING

In this section, we will demonstrate that our design and analysis for system of linear equations can be well applied to linear programming. We consider a linear programming problem denoted by

$$\begin{aligned} \text{minimize} \quad & \mathbf{c}^T\mathbf{x} \\ \text{subject to} \quad & \mathbf{Ax} = \mathbf{b}, \mathbf{Dx} \leq \mathbf{0}, \end{aligned} \tag{1}$$

where $\mathbf{b}, \mathbf{c} \in \mathbb{R}^n$, $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\mathbf{D} \in \mathbb{R}^{s\times n}$ $(m, s \leq n)$. Under the affine mapping $\mathbf{x} = \mathbf{Ky} + \mathbf{r}$, the problem is transformed to

$$\begin{aligned} \text{minimize} \quad & \mathbf{c}^T\mathbf{Ky} + \mathbf{c}^T\mathbf{r} \\ \text{subject to} \quad & \mathbf{AKy=b} - \mathbf{Ar}, \mathbf{DKy} \leq -\mathbf{Dr}, \end{aligned}$$

from which we can see that the input $\{\mathbf{c}^T, \mathbf{A}, \mathbf{b}, \mathbf{D}\}$ can be concealed by the secret key $\mathbf{K}$ and $\mathbf{r}$. It is obvious that the computational bottleneck lies in the multiplication of $\mathbf{K}$ with $\mathbf{A}$ and $\mathbf{D}$. Thus the same complexity and security analysis for system of linear equations apply for linear programming. That is the complexity of the previous four options is all bounded by $\mathcal{O}(n^2)$. In terms of security, the four options are all secure in protecting the input and output while providing different levels of protection of other side information.

V. RESULTS VERIFICATION

In this section, we propose a result verification scheme to ensure the correctness of the result. For system of linear equations, it is sufficient to verify directly whether $\|\mathbf{Ax}^*\| < \varepsilon$, where ε is a pre-defined error tolerance. The complexity of this verification process is $\mathcal{O}(n^2)$.

For optimization problems, the general idea of our proposed verification scheme is to outsource the problem twice under two different affine mappings. To be specific, under the affine mappings $\mathbf{x} = \mathbf{K}_1\mathbf{y} + \mathbf{r}_1$ and $\mathbf{x} = \mathbf{K}_2\mathbf{z} + \mathbf{r}_2$, the original problem $F(\mathbf{x})$ is transformed to $G(\mathbf{y})$ and $H(\mathbf{z})$, respectively. Then the cloud solves the two outsourced problems and returns the corresponding results $\mathbf{y}^*$ and $\mathbf{z}^*$. Thus the end-user can utilize the condition $\mathbf{K}_1\mathbf{y}^* + \mathbf{r}_1 = \mathbf{K}_2\mathbf{z}^* + \mathbf{r}_2$ as a criterion to verify whether the returned results are valid. The output of a linear programming problem can be divided into three cases: normal, infeasible and unbounded. In the following, we will present verification schemes for the three different cases separately.

A. Normal Case

The above proposed verification scheme works well for the normal case. That is if the equality $\mathbf{K}_1\mathbf{y}^* + \mathbf{r}_1 = \mathbf{K}_2\mathbf{z}^* + \mathbf{r}_2$ holds, the end-user can make sure that a valid result can be recovered. This is because whatever the correct result is (normal, infeasible or unbounded), the cloud is not able to manipulate two results that satisfy the equality condition without actually conduct the computation process. And this verification process for normal case forms the basis for the verification for other cases.

B. Infeasible Case

The above verification scheme would fail if the cloud simply returns an infeasible result for any outsourced problem. To deal with this issue, we utilize phase I method as described

in [8, Chapter 11] to check the feasibility of the problem. For a linear programming problem described in Equation (1), a corresponding phase I problem $F_I(\mathbf{x})$ can be constructed as:

$$\begin{aligned} \text{minimize} \quad & \rho \\ \text{subject to} \quad & \mathbf{Ax}=\mathbf{b}, \mathbf{Dx} < \rho, \end{aligned}$$

where ρ is a single variable. It is obvious that when ρ is large enough, $F_I(\mathbf{x})$ is always feasible.

Suppose $\mathbf{x}^*$ minimizes the objective function and ρ^* is the corresponding minimum value. The phase I problem is designed in such a way that when $\rho^* \leq 0$, the original problem $F(\mathbf{x})$ is feasible and $F(\mathbf{x})$ is infeasible otherwise. Thus when the cloud indicates that the solutions to the two outsourced problem $G(\mathbf{y})$ and $H(\mathbf{z})$ are infeasible, it then generates the corresponding two phase I problems $G_I(\mathbf{y})$ and $H_I(\mathbf{z})$ and computes the optimal points $\mathbf{y}^*$ and $\mathbf{z}^*$ and the minimum values ρ_G^* and ρ_H^*, respectively. Then at the local side, the verification is the same as that in the normal case. That is only when $\rho_G^* > 0$ and $\rho_H^* > 0$ and the equality $\mathbf{K}_1\mathbf{y}^* + \mathbf{r}_1 = \mathbf{K}_2\mathbf{z}^* + \mathbf{r}_2$ holds can the end-user be guaranteed to receive valid solutions.

C. Unbounded Case

In the unbounded case, the cloud indicates that the objective function $\mathbf{c}^T\mathbf{x} \to -\infty$ in its domain. To verify the result, we can construct the corresponding *Lagrangian* L as

$$L(\mathbf{x}, \mathbf{u}, \mathbf{v}) = \mathbf{c}^T\mathbf{x} + \mathbf{u}(\mathbf{Ax} - \mathbf{b}) + \mathbf{vDx},$$

where $\mathbf{u} \in \mathbb{R}^m$ and $\mathbf{v} \in \mathbb{R}^s$ are the associated *Lagrange multiplier vectors*. Then based on this Lagrangian $L(\mathbf{x}, \mathbf{u}, \mathbf{v})$, a *Lagrange dual function* can be constructed as

$$\Phi(\mathbf{u}, \mathbf{v}) = \inf_{\mathbf{x}\in\mathcal{D}} L(\mathbf{x}, \mathbf{u}, \mathbf{v}) = \inf_{\mathbf{x}\in\mathcal{D}} \left(\mathbf{c}^T\mathbf{x} + \mathbf{u}(\mathbf{Ax} - \mathbf{b}) + \mathbf{vDx}\right),$$

where $\mathcal{D}$ is the domain of the optimization problem. From this definition, it is easy to prove that $\forall \mathbf{v} \succeq 0$, we have the following inequality:

$$\Phi(\mathbf{u}, \mathbf{v}) \leq L(\mathbf{x}^*, \mathbf{u}, \mathbf{v}) \leq \mathbf{c}^T\mathbf{x}^*,$$

where $\mathbf{c}^T\mathbf{x}^*$ denotes the optimal value of the objective function. The above inequality gives a lower bounded of the objective function that depends on the selection of $\mathbf{u}$ and $\mathbf{v}$. Thus, among all the selections of $\mathbf{u}$ and $\mathbf{v}$, find the optimal lower bound is equivalent to solving the following optimization problem:

$$\text{maximize} \quad \Phi(\mathbf{u}, \mathbf{v}) \qquad \text{subject to} \quad \mathbf{v} \succeq 0.$$

If the original problem is unbounded below, the transformed problem described above should be infeasible since it gives a lower bound of the optimal value in the original problem. Thus the remaining task is to verify the feasibility of the above problem, which has been illustrated in the infeasible case. Let the cloud solve the phase I problems of the two Lagrange dual problems and return the optimal solutions denoted by $(\rho_G^*, \mathbf{y}^*, \mathbf{u}_G^*, \mathbf{v}_G^*)$ and $(\rho_H^*, \mathbf{z}^*, \mathbf{u}_H^*, \mathbf{v}_H^*)$. At the local side, the end-users then check whether $\rho_G^* > 0$ and $\rho_H^* > 0$ and whether the equality $\mathbf{K}_1\mathbf{y}^* + \mathbf{r}_1 = \mathbf{K}_2\mathbf{z}^* + \mathbf{r}_2$ holds.

VI. Evaluation

In this section, we will evaluate the performance of LinSOS. We first compare LinSOS with two existing outsourcing schemes. Then we present some numeric results to show the efficiency of LinSOS.

A. Performance Comparison

We compare the performance of our scheme with two existing schemes in terms of security and computational complexity.

1) Linear Programming: In [4], to transform the linear programming problem described in Equation (1), a secret key $\mathbf{K} = \{\mathbf{Q}, \mathbf{M}, \mathbf{r}, \boldsymbol{\lambda}, \gamma\}$ is generated, where $\mathbf{Q}_{m\times m}$ and $\mathbf{M}_{n\times n}$ are randomly generated non-singular matrices and $\mathbf{r}$ is an $n \times 1$ vector. With this secret key, the original problem is transformed to the following problem

$$\begin{aligned} \text{minimize} \quad & \mathbf{c'^T x} \\ \text{subject to} \quad & \mathbf{A'x}=\mathbf{b'}, \mathbf{D'x} \geq \mathbf{0}, \end{aligned}$$

where $\mathbf{A'} = \mathbf{QAM}, \mathbf{D'} = (\mathbf{D} - \lambda\mathbf{QA})\mathbf{M}, \mathbf{b'} = \mathbf{Q}(\mathbf{b} + \mathbf{Ar})$ and $\mathbf{c'} = \gamma\mathbf{M^T c}$.

In terms of computational complexity, the computational overhead of the outsourcing scheme in [4] as well as our scheme lies primarily in matrix multiplication. As stated in their paper, the overall computational complexity for the scheme proposed in [4] is slightly less than $\mathcal{O}(n^3)$ depending on the algorithm chosen to implement matrix multiplication. For instance, when the Strassen algorithm is adopted, the complexity becomes $\mathcal{O}(n^{2.81})$; while for the Coppersmith-Winograd algorithm the complexity is $\mathcal{O}(n^{2.376})$. However, by carefully selecting the secret key $\mathbf{K}$, our scheme can limit the complexity within $\mathcal{O}(n^2)$.

In terms of security, both schemes can conceal the private information by some disguising techniques, that is to disguise the original matrices by multiplying them with some random matrices. As a consequence, the security they can achieve is in the same level. Actually, the scheme in [4] can be regarded as a special case of our scheme where the secret key $\mathbf{K}$ is randomly generated without possessing any pattens.

Thus while achieving the same security level, LinSOS outperforms in terms of complexity. In addition, LinSOS also provides end-users with the flexibility to select different outsourcing options with different complexity according to their security demands.

2) System of Linear Equations: In [5], the authors investigated outsourcing of system of linear equations $\mathbf{Ax} = \mathbf{b}$ based on iterative method. First, the problem is transformed to $\mathbf{Ay} = \mathbf{b'}$, where $\mathbf{y} = \mathbf{x} + \mathbf{r}$, $\mathbf{b'} = \mathbf{b} + \mathbf{Ar}$ and $\mathbf{r}$ is a random vector. Then the end-user solves the transformed problem iteratively with the aid of cloud servers and an initial guess $\mathbf{y}_0$ from the following iterative equation:

$$\mathbf{y_{k+1}} = \mathbf{T} \cdot \mathbf{y_k} + \mathbf{c'}, \tag{2}$$

where $\mathbf{A} = \mathbf{D} + \mathbf{R}$, $\mathbf{T} = -\mathbf{D^{-1}} \cdot \mathbf{R}$ and $\mathbf{c'} = \mathbf{D^{-1}} \cdot \mathbf{b'}$. The end-user utilizes the cloud servers to compute the most expensive part $\mathbf{T} \cdot \mathbf{y_k}$ based on homomorphic encryption to conceal the private information $\mathbf{T}$.

Table II
PERFORMANCE COMPARISON

	Applicability		Computational Complexity
	LE	LP	
LinSOS	√	√	$\mathcal{O}(n^2)$
[4]		√	$\mathcal{O}(n^{2.376})$
[5]	√		$\mathcal{O}(n^{3+\epsilon})$

As stated above, the computational overhead at the local side primarily lies in the decryption of $\mathbf{T}\cdot\mathbf{y_k}$ in each iteration. Suppose the algorithm terminates after L rounds of iteration, then the end-user has to perform $L \cdot n$ times of decryption. However, the decryption process of public-key cryptosystem is much more expensive than simple multiplication of real numbers since it mainly consists of modular exponentiation of large numbers. For instance, the decryption process [9] adopted in [5] has a complexity of $\mathcal{O}(n^3)$ and a modified version can achieve a complexity of $\mathcal{O}(n^{2+\epsilon})$. Thus, the outsourcing scheme in [5] introduces $\mathcal{O}(n^{3+\epsilon})$ computational overhead at the local side. Further more, this iterative process will introduce extra communication overhead to the end-user. Also, it requires the end-user to be "online" for the process to continue. In comparison, our scheme allows the end-user to be "offline", which means that after outsourcing the transformed problem, the end-user does not need to interact with the cloud servers until the result is sent back.

We summarize the performance comparison for Linear Programming (LP) and system of Linear Equations (LE) in Table II.

B. Numeric Results

In this section, we measure the performance of LinSOS utilizing MATLAB. The computation of both the end-user and the cloud server is simulated using the same computer with an Intel Core 2 Due CPU running at 2.53 GHz with 4GB RAM. In the process of outsourcing, we focus on the overhead of problem transformation, result recovery and the performance gain that they can achieve by outsourcing problems to the cloud. We denote the time for local computation in the outsourcing process as $\mathcal{T}_e$, the time cost without outsourcing as $\mathcal{T}_s$. The performance gain is defined as $\mathcal{I} = \mathcal{T}_s/\mathcal{T}_e$.

We take the outsourcing of system of linear equations $\mathbf{Ax} = \mathbf{b}$ as an example. In our evaluation, we utilize option-3 and let $\mathbf{K}$ be a band matrix with bandwidth W varying from 1 to 31. And we let the dimension n vary from 1000 to 5000. The numeric results are shown in Table III. First, we can learn from the results that when the bandwidth of the banded matrix $\mathbf{K}$ becomes larger, the computational overhead at local side grows and the performance gain decreases. This fact coincides with our analysis on the trade-off between complexity and security. Second, the performance gain increases with the growth of the problem dimension n. This is because our scheme requires the end-users to carry out only simple operations such as addition and multiplication.

Table III
PERFORMANCE EVALUATION FOR SYSTEM OF LINEAR EQUATIONS

Dimension	**Bandwidth**	$\mathcal{T}_e$ **(sec)**	$\mathcal{T}_s$ **(sec)**	$\mathcal{I}$
$n = 1000$	$W = 1$	0.0265	0.2356	8.9
	$W = 7$	0.0265	0.2402	9.1
	$W = 15$	0.0546	0.2356	4.3
	$W = 31$	0.0858	0.2387	2.8
$n = 2000$	$W = 1$	0.0593	1.3962	23.6
	$W = 7$	0.0936	1.4071	15.0
	$W = 15$	0.1248	1.3853	11.1
	$W = 31$	0.1950	1.3494	6.9
$n = 3000$	$W = 1$	0.1170	3.9234	33.5
	$W = 7$	0.1856	3.9281	21.2
	$W = 15$	0.3058	3.8844	12.7
	$W = 31$	0.4867	3.8766	8.0
$n = 4000$	$W = 1$	0.2184	8.5832	39.3
	$W = 7$	0.3416	8.6924	25.4
	$W = 15$	0.7129	8.6565	12.1
	$W = 31$	1.0171	8.6768	8.5
$n = 5000$	$W = 1$	0.3260	15.8138	48.5
	$W = 7$	0.5288	15.9839	30.2
	$W = 15$	1.2683	15.8793	12.5
	$W = 31$	1.8174	15.9698	8.8

VII. CONCLUSION

In this paper, we proposed a cost-aware secure outsourcing scheme (LinSOS) for linear computational problems. We demonstrated that LinSOS can be utilized for secure outsourcing of various computational problems, such as system of linear equations and linear programming. Our scheme also provides mechanisms for the end-users to verify results received from the cloud. We provided security analysis of our proposed scheme on a cost-aware basis. In particular, LinSOS is secure in protecting the input and output of the problem and can partly protect other side information. Our analysis shows that LinSOS can limit the computational overhead at local side to $\mathcal{O}(n^2)$. Since LinSOS is executed off-line, the communication overhead is in the same level as that of outsourcing the original problem itself. We also compared LinSOS with several existing schemes and showed that LinSOS is more efficient and has a wider applicability.

REFERENCES

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

[2] D. Benjamin and M. J. Atallah, "Private and cheating-free outsourcing of algebraic computations," in *Privacy, Security and Trust, 2008. PST'08. Sixth Annual Conference on*, pp. 240–245, IEEE, 2008.

[3] M. J. Atallah and K. B. Frikken, "Securely outsourcing linear algebra computations," in *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, pp. 48–59, ACM, 2010.

[4] C. Wang, K. Ren, and J. Wang, "Secure and practical outsourcing of linear programming in cloud computing," in *INFOCOM, 2011 Proceedings IEEE*, pp. 820–828, IEEE, 2011.

[5] C. Wang, K. Ren, J. Wang, and K. M. R. Urs, "Harnessing the cloud for securely solving large-scale systems of linear equations," in *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, pp. 549–558, IEEE, 2011.

[6] C. E. Shannon, "Communication theory of secrecy systems," *Bell system technical journal*, vol. 28, no. 4, pp. 656–715, 1949.

[7] W. Stallings, *Cryptography and Network Security, 4/E*. Pearson Education, Inc., 2003.

[8] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2009.

[9] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in cryptology-EUROCRYPT*, pp. 223–238, Springer, 1999.