A Distributed Secure Outsourcing Scheme for Solving Linear Algebraic Equations in Ad Hoc Clouds

Wenlong Shen, *Student Member, IEEE*, Bo Yin, *Student Member, IEEE*, Xianghui Cao, *Senior Member, IEEE*, Yu Cheng, *Senior Member, IEEE*, and Xuemin (Sherman) Shen, *Fellow, IEEE*

Abstract—The emerging ad hoc clouds form a new cloud computing paradigm by leveraging untapped local computation and storage resources. An important application of ad hoc clouds is to outsource computational intensive problems to nearby cloud agents. Specifically, for the problem of solving a linear algebraic equation (LAE), an outsourcing client assigns each cloud agent a subproblem, and then all involved agents apply a consensus-based algorithm to obtain the correct solution of the LAE in an iterative and distributed manner. However, such a distributed collaboration paradigm suffers from cyber security threats that undermine the confidentiality of the outsourced problem and the integrity of the returned results. In this paper, we identify a number of such security threats in this process, and propose a secure outsourcing scheme which not only preserves the privacy of the LAE parameters and the final solution from the participating agents, but also guarantees the correctness of the final solution. We prove that the proposed scheme has low computation complexity at each agent, and is robust against the identified security attacks. Numerical and simulation results are presented to demonstrate the effectiveness of the proposed method.

Index Terms—Ad hoc cloud, linear algebraic equations, outsourcing, distributed consensus, security, privacy

1 Introduction

Cloud computing is a revolutionary paradigm of delivering network resources, ranging from computational power and data storage to platform and software, as a service over the network [1]. As mobile devices are equipped with increasing computational capability and memory, a new peer-to-peer cloud computing model is proposed to interconnect nearby devices to form an ad hoc cloud, in which a device can either work as a service provider or a client of a service requester. Such an ad hoc cloud computing model can significantly improve the resource utilization of local devices, while providing benefits of conventional client-server cloud computing model over existing heterogeneous hardware [2]. Ad hoc cloud computing has drawn much research attention on its architecture[3], service model [4], applications [5], and security [6].

Computation outsourcing is one major application of cloud computing, which enables resource limited cloud to conduct originally impossible complex missions by outsourcing the workloads to the cloud. In spite of such a benefit, cloud users should seriously consider the potential risks before resorting to the cloud, since the user has little control over the outsourced data and the behavior of the remote cloud entities. Thus, a secure outsourcing scheme should first have the capability to hide both the sensitive information contained in the problem parameters and the computation results from the participating cloud agents and malicious eavesdroppers as well. In fact, cloud service providers have various motivations to behave dishonestly. For example, the cloud service providers may perform slothfully to save computational resource or power. There also exists the possibility that a cloud service provider is compromised by a malicious attacker, thus intentionally misleading the client to a false computation result. Therefore, for a secure outsourcing scheme, it is important that the correctness of the returned results of the outsourced problem can be verified and guaranteed to be correct. Moreover, the local computation complexity in a secure outsourcing scheme, including that incurred by certain privacy preserving and result verifying computations, should not exceed that by locally solving the original problem alone.

In [7], Gentry constructed the first fully homomorphic encryption (FHE) scheme, which allows computing arbitrary functions with encrypted data. Following [7], many schemes, e.g., [8], [9] were proposed to improve the efficiency of FHE for practical applicability. An important technique for efficient verification of arbitrarily complex computations [10], [11] is interactive proofs, where a powerful prover can

- *W. Shen, B. Yin, and Y. Cheng are with Department of Electrical and Computer Engineering, Illinois Institute of Technology, Chicago, IL 60616. E-mail: {wshen7, byin}@hawk.iit.edu, cheng@iit.edu.*
- *X. Cao is with School of Automation, Southeast University, Jiangsu, Sheng 210018, China. E-mail: xhcao@seu.edu.cn.*
- *X. Shen is with Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON N2L 3G1, Canada. E-mail: xshen@bbcr.uwaterloo.ca.*

Manuscript received 21 Apr. 2016; revised 25 Oct. 2016; accepted 22 Dec. 2016. Date of publication 4 Jan. 2017; date of current version 5 June 2019. Recommended for acceptance by C. Rong.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TCC.2016.2647718

convince a weak verifier of the truth of statements that the verifier could not compute on its own. Homomorphic encryption and computation verification techniques construct the foundations of secure outsourcing scheme in a client-server cloud setting. However, in an ad hoc cloud setting, the aforementioned techniques are no longer applicable. Unlike the traditional cloud computing model which features in one or multiple powerful servers, in the ad hoc cloud network, the resource pool is formed by leveraging untapped resources from local devices, each of which has only limited computational power. Resource limited entities in the ad hoc cloud are often unfordable to perform the FHE schemes and verification protocols.

Solving linear algebraic equations (LAEs) $\mathbf{A}\mathbf{x} = \mathbf{b}$ is one of the most frequently used mathematical tool for a large variety of real-world engineering and scientific computations. Unlike most of other secure outsourcing works in the traditional cloud computing model with one or several powerful cloud servers, in this paper, we study the problem of securely outsourcing the LAE problem in an ad hoc cloud network comprised of multiple agents with limited computation resources. The work in [12] proposes a consensus-based distributed algorithm, which enables multiple agents to solve the LAE problem in a collaborative manner. In this algorithm, each agent is assigned one (or possible multiple) row of $[\mathbf{A}, \mathbf{b}]$, say $[\mathbf{A}_i, \mathbf{b}_i]$. Starting with a feasible solution to its subproblem $\mathbf{A}_i\mathbf{x} = \mathbf{b}_i$, each agent iteratively updates its local solution based on solutions from neighboring agents. Eventually, all agents will agree on a consensus, which is the exact solution of the original LAE problem. Although this algorithm is suitable to outsourcing an LAE problem to an ad hoc cloud network, the collaborative nature makes it vulnerable to deliberate erroneous updates. A malicious agent is able to mislead the final consensus to a wrong solution by misreporting its local solution to neighbors. What's worse, continuous incorrect updates will impede the progress of consensus and even prevent other agents from reaching an agreement.

In this paper, we propose a secure outsourcing scheme for solving LAE problems in ad hoc cloud. Through the analysis of potential security threats, we categorize them into three classes based on the corresponding effects. We define our design goals for a secure outsourcing scheme as to preserve LAE problem privacy and guarantee correct final returned solution. On the basis of the consensus-based algorithm presented in [12], we design a new robust algorithm which can prevent malicious (or compromised) agents from manipulating the final solution by injecting unfaithful intermediate computation results during the consensus process. However a malicious or compromised cloud agent may still diverge the algorithm and hence launch a denial of service attack by continuously injecting unfaithful data during the consensus process. To deal with this issue, we further propose a misbehavior detection mechanism. The main idea of the detection mechanism is that neighboring agents cooperatively verify each other's intermediate computation results at each step of the consensus process with a probability p. Such a detection mechanism, together with the fault tolerance feature of the robust consensus algorithm, can protect the integrity and availability of the solution to the outsourced LAE. In addition, another building block of our proposed secure outsourcing scheme is a privacy disguising technique, which preserves the privacy of the sensitive information contained in both the LAE problem parameters and the final solutions.

The main contributions of this paper can be summarized as follows.

1) We design a robust version of the consensus-based algorithm for distributively solving an LAE problem with fault tolerance.
2) Based on the robust distributed algorithm, we design a secure outsourcing scheme for LAE in the ad hoc cloud environment, with the capabilities of privacy preserving and misbehavior detection. The performance of the scheme is analyzed theoretically.
3) We demonstrate the performance of the proposed outsourcing scheme through both theoretical analysis and numerical results. We also conduct simulations to evaluate the performance of the scheme in WiFi based ad hoc clouds with packet losses.

The remainder of this paper is organized as follows. Section 2 reviews more related work. Section 3 describes the system model and preliminaries on distributed algorithms for solving LAE problems. Section 3.2 presents the attack model and our design goals. Section 4 presents details of the proposed algorithm. Theoretical performance analysis is given in Section 5, followed by numerical results in Section 6. Section 7 discusses collusion attacks and Section 8 concludes this paper.

2 Related Work

Recently, there has been steady progress in the study of securely outsourcing computationally intensive problems such as linear equations [13], linear programming [14], sequence comparisons [15] and DNA searching [16]. For example, the work in [17] proposes a protocol for secure and private outsourcing of linear algebra computations, especially the problem of multiplying large-scale matrices, to either two or one remote server(s). The approach is based on the secret sharing scheme proposed in [18], without carrying out expensive cryptographic computations. In [14], secure outsourcing of a linear programming problem is investigated where malicious behavior can be detected by a computation result verification mechanism by exploiting the properties of the dual of the original LP problem, while the problem confidentiality is preserved by using random matrix and vectors. Based on random scaling and permutation, matrix masking algorithms for secure outsourcing matrix inversion and matrix determinant computation are proposed respectively in [19] and [20].

There are a few works on secure outsourcing of the LAE problem. The work in [21] first introduces several mechanisms for secure outsourcing of scientific computations, which includes the disguising scheme for outsourcing LAE problem. The work in [13] proposes a secure scheme for outsourcing a large-scale LAE problem, where they applied the Jacobi method for solving the LAE problem and preserved the privacy by hiding the problem information based on a homomorphic encryption scheme. A secure outsourcing scheme for LAE problem based on conjugate gradient method (CGM) is presented in [22], and the work in [23] develops a general method for disguising the LAE problem. However, these existing studies focused on outsourcing this

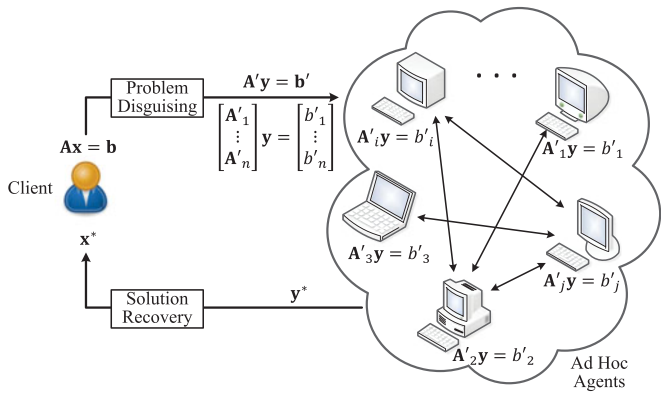

Problem Disguising
$\mathbf{A}'\mathbf{y} = \mathbf{b}'$
$\begin{bmatrix}\mathbf{A}'_1\\ \vdots \\ \mathbf{A}'_n\end{bmatrix}\mathbf{y} = \begin{bmatrix}b'_1\\ \vdots \\ b'_n\end{bmatrix}$
$\mathbf{A}\mathbf{x} = \mathbf{b}$
Client
$\mathbf{x}^*$
Solution Recovery
$\mathbf{y}^*$
$\mathbf{A}'_i\mathbf{y} = b'_i$
$\mathbf{A}'_1\mathbf{y} = b'_1$
$\mathbf{A}'_3\mathbf{y} = b'_3$
$\mathbf{A}'_j\mathbf{y} = b'_j$
$\mathbf{A}'_2\mathbf{y} = b'_2$
Ad Hoc Agents

Fig. 1. Overview of the distributed outsourcing scheme.

problem to a single remote cloud server, which is essentially a centralized scheme. In this paper, we consider a different application scenario—outsourcing the LAE problem to an ad hoc cloud, with the cloud agents involved in solving the problem in a completely distributed manner.

Distributed solutions for large-scale LAE problems have been studied. The mainstream approach is to decompose the original problem into smaller ones which can be solved by parallel processors [24], [25]. However, these parallel algorithms often assume special structure of the matrix $\mathbf{A}$. Recently with the advances in distributed consensus algorithms [26], [27], a consensus-based distributed solution to the LAE has been proposed in [12]. Instead of performing problem decomposition, the algorithm assigns each agent one (or possibly multiple) row of $\mathbf{A}$ and $\mathbf{b}$, say $\mathbf{A}_i$ and b_i, respectively. Each agent starts with a feasible solution to the subproblem. By applying an averaging consensus algorithm where each agent only talks to its neighbors, the solutions obtained by the agents can finally converge to the correct solution of the original problem, and the convergence speed is exponentially fast. Compared to classic algorithms, for example, Jacobi iterations and the classical Kaczmarz method, the consensus-based algorithm does not make special assumptions about $\mathbf{A}$, and does not require the network topology to be strongly complete. It has been shown in [12] that, if the network topology of the agents is repeatedly jointly strongly connected over time, the convergence is guaranteed and the correct solution $\mathbf{A}^{-1}\mathbf{b}$ can be obtained.

The consensus-based algorithm provides an interesting and promising way for outsourcing a large-scale LAE problem to a number of distributed agents, each of which has only limited computation resources. Despite such merit, the algorithm is susceptive to several malicious attacks ranging from sensitive data probing and disobeying the updating rule as the algorithm runs. With potentially many attack strategies, an adversary can manipulate the final results and even cause the whole algorithm diverge. To the best of our knowledge, this paper for the first time systematically studies the security issues in outsourcing an LAE problem to a distributed ad hoc cloud.

3 Problem Statement and Preliminaries

3.1 System Model

In this paper, we study a computation outsourcing problem in an ad hoc cloud system, as illustrated in Fig. 1. The ad hoc cloud comprises of multiple agent nodes, each of which is capable of performing certain computation tasks within its computational resource limit. The physical devices associated with these agents can be desktops, mobile devices, or servers. The connection between agents can either be wired or wireless. Although the consensus-based algorithm we considered works with dynamic network topologies, as long as the network topology is repeatedly jointly strongly connected [12], for ease of presentation, in this paper we only consider the static network topology. One of these agents is interested in solving a large-scale LAE problem in the form $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{A} \in \mathbb{R}^{n\times n}$ is a non-singular constant matrix, $\mathbf{b} \in \mathbb{R}^n$ is a constant vector, and $\mathbf{x} \in \mathbb{R}^n$ is the unknown variable to be solved. Assume that n is large such that solving this LAE problem is computationally intensive considering the limited computing power at one cloud agent (solving this problem directly takes time $\mathcal{O}(n^3)$). To distinguish the problem outsourcer from other agents in the ad hoc cloud, the outsourcer agent of this LAE problem is denoted as the client. Thus, the client resorts to the ad hoc cloud in which other agents who are willing to share their computation power, either voluntary or paid, can collaboratively work towards solving this LAE problem.

Through a distributed algorithm, a solution to the outsourced LAE problem can be eventually reached via collaboration over all the participating agents in the ad hoc cloud. However, in most of the real world application scenarios, there potentially exists malicious agents aiming to break down the problem solving process for a variety of motivations. For example, they may either perform selfishly by claiming the revenue but not fulfilling their tasks, or spitefully preventing the client from deriving the correct solution. Moreover, the parameters and results of the outsourced problem may contain privacy information that the client is not willing to share with other agents. In this paper, the private information in the LAE contains magnitudes and interrelationships of the elements in $\mathbf{A}$, $\mathbf{b}$, and the solution $\mathbf{x}^*$, and the number and positions of zero elements in these matrix/vectors.

3.2 Attack Model

In the multi-agent ad hoc cloud, each agent has a unique identity number, e.g., the agents are indexed by $1, 2, \ldots, n$. All the agents form a connected cloud network and the network-wide time synchronization is always guaranteed. We assume that the connectivity is known by the client. This can be achieved by running a secure neighbor discovery process beforehand [28]. We also assume that each message in the system is authenticated, so that a malicious agent may record and then play back a message but cannot modify it. For the malicious nodes, we assume that they do not collude (the cases with colluding attacks will be discussed in Section 7). We further assume that in each agent's neighborhood, the number of honest agents is greater than the number of malicious ones.[1] The communications between

1. This local honest majority assumption is an f-fraction local model, which has been used in many other literatures [29], [30]. Since the detection mechanism relies on neighbor-checking and runs in a distributed manner based on the majority rule, the above assumption guarantees that the majority rule in each agent's neighborhood does not generate false decisions. In our future work we will study the performance of our scheme in the general honest majority model.

agents are assumed reliable (the cases with packet losses will be evaluated and discussed in Section 6.3).

When an LAE problem is outsourced to the ad hoc cloud, the client will have little control over other agents involved in the computing. Without a proper defense mechanism, a cloud agent may perform dishonestly for a variety of reasons. We assume that malicious agents in the ad hoc cloud, either on their own initiative or compromised, are interested in the information contained within the original problem parameters as well as the problem's final solution. Malicious agents also have the motivation to break down the distributed algorithm, either by misleading the algorithm to a false result or diverging the consensus of the algorithm, thus launching a denial of service attack. In this section, we explicitly analyze the misbehavior possibly conducted by malicious agents and how these attacks affect the final solution.

Depending on their purposes, we specify the behavior of malicious agents into three categories.

- *Probing sensitive information*: In the basic consensus-based algorithm, each participating agent is assigned with a row of $\mathbf{A}$ and the corresponding component of $\mathbf{b}$. This setup process reveals part of information of $\mathbf{A}$ and $\mathbf{b}$. After the consensus is finally reached, all the participating agents will obtain the solution of the outsourced LAE problem, which is highly undesirable, since malicious agents may collude with each other to obtain more information of $\mathbf{A}$ and $\mathbf{b}$. Therefore, it is necessary for the outsourcing client to disguise the original problem before sending it to the cloud. The disguising should be able to hide the original problem parameters as well as the solution to the outsourced problem. How to preserve these privacy information contained in the matrix has been previously studied by many researchers [19], [20], [21]. However, their solutions either reveals partial information or involves $\mathcal{O}(n^3)$ computation complexity.
- *Manipulating the solution*: A malicious agent can mislead the algorithm to a false solution by injecting a false intermediate result during the consensus process. For example, at the kth iteration, malicious agent i sends out a false intermediate result $\mathbf{x}_i(k)$, which results in $\mathbf{A}_i^{\mathrm{T}}\mathbf{x}_i(k) = b_i^{'} \neq b_i$. In the subsequent iterations, according to the update rule in (1), $\mathbf{A}_i^{\mathrm{T}}\mathbf{x}_i(l) = \mathbf{A}_i^{\mathrm{T}}\mathbf{x}_i(k) = b_i^{'}$ for $l > k$. The algorithm will finally converge to a false result $\mathbf{x}'$, which is the solution to $\mathbf{A}\mathbf{x}' = \mathbf{b}^{'}$ where $\mathbf{b}' = [b_1, b_2, \ldots, b_i^{'}, \ldots, b_n]^{\mathrm{T}}$.
- *Diverging the consensus*: Denial of service attack is a common attack in distributed computing systems [26], [31]. Without a proper defense mechanism, a malicious agent can easily diverge the consensus algorithm by randomly updating $\mathbf{x}_i(t)$ in each iteration. If the malicious agent keeps doing this, obviously the distributed consensus algorithm will not converge. Existing detection mechanisms usually require global system information [32] or impose high computational burden to the detector [33]. A straightforward solution to prevent such an attack is resorting to the help from other nodes during the algorithm setup stage: distribute $\mathbf{A}_i^{\mathrm{T}}$ and b_i not only to agent i, but also to its neighbor agents so that neighboring agents can verify each other's updating value per step by checking whether $\mathbf{A}_i^{\mathrm{T}}\mathbf{x}_i(t) = b_i$. A randomly chosen $\mathbf{x}_i(t)$ by malicious agents will not likely to satisfy this checking equation and gets detected by their neighbors as a result. However, a "smart" enough malicious agent can still break the convergence of the algorithm by choosing $\mathbf{x}_i(t)$ for each iteration within the solution space of $\mathbf{A}_i^{\mathrm{T}}\mathbf{x}_i(t) = b_i$, for example, resending the initial guess repeatedly. Thus simply checking $\mathbf{A}_i^{\mathrm{T}}\mathbf{x}_i(t)$ will not prevent the solution process from diverging, which calls for sophisticated mutual verification methods.

According to the aforementioned analysis, securely outsourcing an LAE problem in an ad hoc cloud requires the following properties.

- *Privacy preserving*: Participating agents, during collaborating with each other for solving the outsourced problem, cannot infer the client's privacy information contained in the input $\mathbf{A}$, $\mathbf{b}$, and the solution $\mathbf{x}$.
- *Misbehavior detection*: Misbehaving agents can be detected with a high probability during participating the outsourced computation. The validation of final solutions can be guaranteed.
- *Low complexity*: The computation burden on each participating agent, as well as the client, should be kept below $\mathcal{O}(n^3)$, i.e., less than that of solving the original LAE problem by the client himself.

3.3 Preliminaries on Distributively Solving LAE

In order to allow multiple agents cooperatively solving the LAE problem $\mathbf{A}\mathbf{x} = \mathbf{b}$, a distributed algorithm that can decompose the LAE problem into smaller subproblems is the foundation. In this paper, we build our secure outsourcing scheme over a consensus-based distributed algorithm proposed in [12]. Compared to other algorithms for solving the LAE problem, the consensus-based algorithm has no special requirement on $\mathbf{A}$, and achieves an exponential convergence rate. The key idea of this algorithm is summarized as follows.

We use boldface letters to represent column vectors and matrices. Let $\mathbf{x}^*$ be the exact solution of the LAE problem. Let $\mathbf{A}_i$ be the ith column of the matrix $\mathbf{A}^{\mathrm{T}}$ and $[\mathbf{A}_i^{\mathrm{T}}\ b_i]$ be a distinct row of the partitioned matrix $[\mathbf{A}\ \mathbf{b}]$, where T is the transpose operator. Assume that there are n connected agents that forms a network (the cases when there are less than n agents are discussed in Section 4.2). Each agent is allocated one distinct row, and the agent who receives the ith row is denoted as agent i. To start the algorithm, each agent picks an initial guess of $\mathbf{x}^*$, denoted as $\mathbf{x}_i(0)$, $i \in [1, 2, \ldots, n]$, such that $\mathbf{A}_i^{\mathrm{T}}\mathbf{x}_i(0) = b_i$. Let $\mathbf{K}_i \in \mathcal{R}^{n\times(n-1)}$ be a matrix whose column span is the kernel of $\mathbf{A}_i^{\mathrm{T}}$, i.e., $\mathbf{A}_i^{\mathrm{T}}\mathbf{K}_i = 0$ and $\mathrm{rank}(\mathbf{K}_i) = n-1$. Each agent iteratively updates its guess following an updating rule in the form $\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{K}_i\mathbf{u}_i(t)$ so that, within the iteration process, the local solution of each agent always satisfies $\mathbf{A}_i^{\mathrm{T}}\mathbf{x}_i(t) = b_i$. In order to guarantee the convergence, $\mathbf{u}_i(t)$ is chosen as the least square solution to $\mathbf{x}_i(t) + \mathbf{K}_i\mathbf{u}_i(t) = \frac{1}{d_i}(\sum_{j\in\mathcal{N}_i}\mathbf{x}_j(t))$, where $\mathcal{N}_i$ denotes the set of neighbor agents of agent i ($i \in \mathcal{N}_i$ for convention), and d_i is the number of neighbors of agent i. With $\mathbf{u}_i(t)$ properly determined, the update process can then be expressed as

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) - \frac{1}{d_i}\mathbf{P}_i\left(d_i\mathbf{x}_i(t) - \sum_{j\in\mathcal{N}_i}\mathbf{x}_j(t)\right), \quad (1)$$

where $\mathbf{P}_i = \mathbf{K}_i(\mathbf{K}_i^{\mathrm{T}}\mathbf{K}_i)^{-1}\mathbf{K}_i^{\mathrm{T}}$ is the orthogonal projection on the kernel of $\mathbf{A}_i$.

For a nonsingular $\mathbf{A}$, if the network topology of these agents is a connected graph, all local solutions would reach a consensus, denoted as $\hat{\mathbf{x}}$ [12]. Note that $\mathbf{A}_i^{\mathrm{T}}\hat{\mathbf{x}} = b_i$ holds for all $i \in \{1, 2, \ldots, n\}$, ensuring that $\hat{\mathbf{x}}$ is the solution of this LAE problem. That is, $\mathbf{A}\hat{\mathbf{x}} = \mathbf{b}$.

4 Secure LAE Outsourcing

Our secure outsourcing scheme for LAE consists of three basic components: a robust consensus-based algorithm for distributedly solving the LAE, a privacy preserving algorithm for defeating the probing sensitive information attack, and a cooperative verification and misbehavior detection mechanism for detecting attacks intended to manipulate solution or diverge the consensus.

4.1 Robust Consensus-Based Algorithm

Equation (1) provides a collaborative framework for autonomous agents to solve a large-scale LAE problem. However, the updating rule in (1) is vulnerable to false messages, referring to our analysis in Section 3.2. Furthermore, such an updating process incurs high computation cost and storage requirement when the problem dimension n is very large. Specifically, the straightforward calculation of $\mathbf{P}_i$ as $\mathbf{K}_i(\mathbf{K}_i^{\mathrm{T}}\mathbf{K}_i)^{-1}\mathbf{K}_i^{\mathrm{T}}$ involves matrix-matrix multiplications which have $\mathcal{O}(n^3)$ time complexity. In addition, $\mathbf{P}_i(d_i\mathbf{x}_i(t) - \sum_{j\in\mathcal{N}_i}\mathbf{x}_j(t))$ incurs matrix-vector multiplications which take time $\mathcal{O}(n^2)$. Storing a large-scale matrix is also an expensive burden for some storage-constrained devices. Thus, we are motivated to design a robust and efficient version based on the preliminary consensus algorithm.

Observing that $\mathbf{P}_i$ is the orthogonal projection on the kernel of $\mathbf{A}_i$, it can be calculated as

$$\mathbf{P}_i = \mathbf{I} - \mathbf{P}_{\mathbf{A}_i^{\mathrm{T}}} = \mathbf{I} - \frac{\mathbf{A}_i\mathbf{A}_i^{\mathrm{T}}}{\mathbf{A}_i^{\mathrm{T}}\mathbf{A}_i}, \quad (2)$$

where $\mathbf{I}$ represents the identity matrix of compatible dimension. $\mathbf{P}_{\mathbf{A}_i^{\mathrm{T}}}$ is the orthogonal projection on $\mathbf{A}_i^{\mathrm{T}}$ and can be calculated with time complexity $\mathcal{O}(n^2)$. Let $\bar{\mathbf{x}}_i(t) = \frac{1}{d_i}\sum_{j\in\mathcal{N}_i}\mathbf{x}_j(t)$ denote the average value of agent i's neighbors' updates. Substituting (2) into (1), the updating process can be expressed as

$$\begin{aligned}\mathbf{x}_i(t+1) &= \mathbf{P}_{\mathbf{A}_i^{\mathrm{T}}}\mathbf{x}_i(t) + \bar{\mathbf{x}}_i(t) - \mathbf{P}_{\mathbf{A}_i^{\mathrm{T}}}\bar{\mathbf{x}}_i(t)\\ &= \frac{\mathbf{A}_i\mathbf{A}_i^{\mathrm{T}}}{\mathbf{A}_i^{\mathrm{T}}\mathbf{A}_i}\mathbf{x}_i(t) - \frac{\mathbf{A}_i\mathbf{A}_i^{\mathrm{T}}}{\mathbf{A}_i^{\mathrm{T}}\mathbf{A}_i}\bar{\mathbf{x}}_i(t) + \bar{\mathbf{x}}_i(t)\\ &= \frac{\mathbf{A}_i^{\mathrm{T}}\mathbf{x}_i(t)}{\|\mathbf{A}_i^{\mathrm{T}}\|^2}\mathbf{A}_i - \frac{\mathbf{A}_i^{\mathrm{T}}\bar{\mathbf{x}}_i(t)}{\|\mathbf{A}_i^{\mathrm{T}}\|^2}\mathbf{A}_i + \bar{\mathbf{x}}_i(t)\\ &= \frac{b_i}{\|\mathbf{A}_i^{\mathrm{T}}\|^2}\mathbf{A}_i - \frac{\mathbf{A}_i^{\mathrm{T}}\bar{\mathbf{x}}_i(t)}{\|\mathbf{A}_i^{\mathrm{T}}\|^2}\mathbf{A}_i + \bar{\mathbf{x}}_i(t),\end{aligned} \quad (3)$$

where $\|\cdot\|$ represents the 2-norm of a vector. For convenience of reference, our proposed consensus-based algorithm is

$$\mathbf{x}_i(t+1) = \frac{b_i}{\|\mathbf{A}_i^{\mathrm{T}}\|^2}\mathbf{A}_i - \frac{\mathbf{A}_i^{\mathrm{T}}\bar{\mathbf{x}}_i(t)}{\|\mathbf{A}_i^{\mathrm{T}}\|^2}\mathbf{A}_i + \bar{\mathbf{x}}_i(t). \quad (4)$$

The proposed algorithm (4) has low computation complexity, as specified in Lemma 1.

Lemma 1. *In each iteration of the proposed consensus-based algorithm (4), the computational cost for agent i is $\mathcal{O}(d_in)$.*

Lemma 1 directly follows the algorithm in (4). In (4), the first term of the right-hand side is a constant. Thus, each update only needs to compute the second and the third terms of the right-hand side, which incurs time complexity $\mathcal{O}(d_in)$. Besides, each agent only needs to store his own $[\mathbf{A}_i^{\mathrm{T}}\ b_i]$, which is a $1 \times (n+1)$ vector.

If an agent doesn't update its local solution according to (4) (e.g., agent i broadcasts an $\mathbf{x}_i$ to its neighbors such that $\mathbf{A}_i^{\mathrm{T}}\mathbf{x}_i \neq b_i$), we call that this agent conducts a false update. The proposed algorithm in (4) is robust against a finite number of false updates, as presented in Theorem 1.

Theorem 1. *The revised consensus-based algorithm in (4) is robust against a finite number of false updates. That is, as long as all agents (including malicious ones) update their local solutions according to (4) from a certain moment on, these local solutions will converge to the correct solution.*

Proof. According to (4), we can see that each update always maintains $\mathbf{A}_i^{\mathrm{T}}\mathbf{x}_i(t) = b_i$. That is,

$$\begin{aligned}\mathbf{A}_i^{\mathrm{T}}\mathbf{x}_i(t+1) &= \mathbf{A}_i^{\mathrm{T}}\frac{b_i}{\|\mathbf{A}_i^{\mathrm{T}}\|^2}\mathbf{A}_i - \mathbf{A}_i^{\mathrm{T}}\frac{\mathbf{A}_i^{\mathrm{T}}\bar{\mathbf{x}}_i(t)}{\|\mathbf{A}_i^{\mathrm{T}}\|^2}\mathbf{A}_i + \mathbf{A}_i^{\mathrm{T}}\bar{\mathbf{x}}_i(t)\\ &= b_i - \mathbf{A}_i^{\mathrm{T}}\bar{\mathbf{x}}_i(t)\frac{\mathbf{A}_i^{\mathrm{T}}\mathbf{A}_i}{\|\mathbf{A}_i^{\mathrm{T}}\|^2} + \mathbf{A}_i^{\mathrm{T}}\bar{\mathbf{x}}_i(t)\\ &= b_i - \mathbf{A}_i^{\mathrm{T}}\bar{\mathbf{x}}_i(t) + \mathbf{A}_i^{\mathrm{T}}\bar{\mathbf{x}}_i(t)\\ &= b_i.\end{aligned}$$

Hence, as long as the agents update their local solutions according to (4), our algorithm always ensures that $\mathbf{A}_i^{\mathrm{T}}\mathbf{x}_i(t) = b_i$ regardless of the updates of their neighbors. Moreover, the initial local solution of each agent can be selected randomly. If, from a certain moment on, all the updating follows Equation (4), the past false updates just behave as selecting a different initial value such that those solutions can still converge to the correct solution of the outsourced LAE problem. In contrast, a single false update may impact the final solution of the original consensus-based algorithm, as discussed in Section 3.2 □

4.2 When There are Less Than n Agents

In practice, it is unlikely to have exactly n available agents for solving a large-scale LAE problem. Now we extend our consensus-based algorithm to work with m agents where $m < n$. For the m-agent ad hoc cloud which is assumed connected, we partition the n rows of $[\mathbf{A}\ \mathbf{b}]$ into m groups and assign each agent a distinct group, as shown in Fig. 2. Consider cloud agent i, $1 \leq i \leq m$. It runs r_i instances of algorithm (4) during the consensus process, where r_i

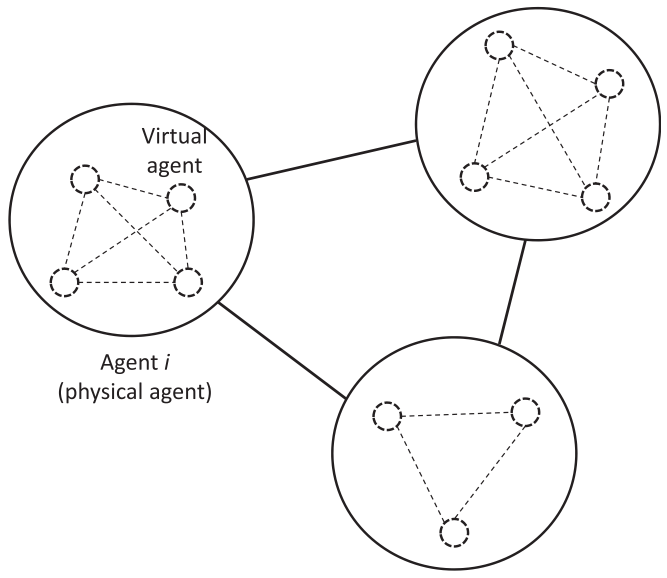


Fig. 2. Proposed scheme with less than n agents.

denotes the number of rows assigned to it. Each of these instances can be viewed as a distinct virtual agent responsible for a single row of $[\mathbf{A}\ \mathbf{b}]$. Since the computations of all virtual agents associated with one physical agent are carried out by the physical agent itself, these virtual agents are considered completely connected. For two virtual agents belonging to two different physical agents, they are considered connected if the two physical agents are also connected. In this way, the whole system can be viewed as a connected n-agent virtual cloud. To update the local solution of each virtual agent, a straightforward strategy for agent i is to send all local solutions of its associated r_i virtual agents to its neighbors. Observing that only the average value of neighbors' local solutions matters in updating the local solution of an agent, the communication overhead can be lessened for agent i by only broadcasting the average value of the r_i local solutions along with r_i. With such information obtained from neighbors, a physical agent can easily calculate $\bar{\mathbf{x}}_i(t)$ can then updates the local solutions for each of its associated virtual agents.

It is worth noting that the network topology of all the virtual agents is still a connected graph, hence the convergence of the algorithm is guaranteed. As long as each honest agent running its virtual agents following the algorithm in (4), the robustness property in Theorem 1 is still valid. For the computational cost, since all the virtual agents in agent i have the same neighbor set (each virtual agent is a neighbor of itself), agent i only needs to compute $\bar{\mathbf{x}}_i(t)$ once in each iteration. The computational cost of $\bar{\mathbf{x}}_i(t)$ is $\mathcal{O}((r_i+d_i)n)$ for agent i, and the computational cost of running r_i virtual agents to compute r_i local solutions is $\mathcal{O}(r_in)$. In sum, the computational cost for agent i in each iteration is $\mathcal{O}((d_i+2r_i)n)$.

4.3 Privacy Preserving

In order to keep confidential the LAE parameters $\mathbf{A}$, $\mathbf{b}$, and the solution $\mathbf{x}^*$, the client needs to disguise the problem before outsourcing it to the cloud. Note that such a disguising algorithm should have low computation complexity, since any computation at the complexity level of $\mathcal{O}(n^3)$ incurred at the client will demotivate the whole outsourcing scheme. In the following, we develop a low-complexity algorithm for disguising the outsourcing problem.

We start with introducing a random noise $\Delta\mathbf{x}$ to mask the solution $\mathbf{x}^*$ of the original problem $\mathbf{A}\mathbf{x}=\mathbf{b}$ as follows. $\Delta\mathbf{x}$ follows uniform distribution whose support is $[-u,u]$, where u is the maximum absolute value of elements in $\mathbf{A}$ and $\mathbf{b}$.

$$\begin{aligned}\mathbf{A}(\mathbf{x}^*+\Delta\mathbf{x}) &= \mathbf{A}\mathbf{x}^*+\mathbf{A}\Delta\mathbf{x}\\ &= \mathbf{b}+\mathbf{A}\Delta\mathbf{x}\\ &= \mathbf{b}+\Delta\mathbf{b}.\end{aligned} \tag{5}$$

Thus, the client generates an n dimensional random vector $\Delta\mathbf{x}$, and computes $\Delta\mathbf{b}=\mathbf{A}\Delta\mathbf{x}$. Then the original problem is transformed into $\mathbf{A}\mathbf{x}=\mathbf{b}+\Delta\mathbf{b}$.

Next, we consider hiding the problem parameters $\mathbf{A}$ and $\mathbf{b}$. One straightforward method is to generate a random non-singular $n\times n$ matrix $\mathbf{Q}$, and outsource the problem $\mathbf{A}'\mathbf{x}=\mathbf{b}'$, with $\mathbf{A}'=\mathbf{Q}\mathbf{A}$ and $\mathbf{b}'=\mathbf{Q}(\mathbf{b}+\Delta\mathbf{b})$. However, the computation of $\mathbf{Q}\mathbf{A}$ commonly has time complexity $\mathcal{O}(n^3)$, which violates the motivation of outsourcing. Hence, we resort to elementary transformations to transform the LAE problem. For one thing, all non-singular matrices with the same size are equivalent under elementary transformations, meaning that $\mathbf{A}$ can be transformed to any non-singular $n\times n$ matrix by a finite sequence of elementary operations. For another, each elementary operation on a matrix takes time $\mathcal{O}(n)$, which enables the client to control the computation complexity of the transformation of the LAE problem.

There are three types of elementary row (resp. column) operations: multiplication, switching and addition. The aggregated multiplication operation can be characterized by a diagonal matrix

$$\mathbf{Q_m}=\begin{pmatrix}\alpha_1 & & & \\ & \alpha_2 & & \\ & & \ddots & \\ & & & \alpha_n\end{pmatrix},$$

where $\alpha_1,\ldots,\alpha_n$ are random non-zero scalars. Left (resp. right) multiplying a matrix by $\mathbf{Q_m}$ is equivalent to multiplying the ith row (resp. column) of that matrix with scalar α_i.

Let $\mathbf{Q}_\pi$ represents the aggregated switching operation,

$$\mathbf{Q}_\pi=\begin{pmatrix}\mathbf{u}_{\pi(1)} & \mathbf{u}_{\pi(2)} & \cdots & \mathbf{u}_{\pi(n)}\end{pmatrix},$$

where π denotes a permutation of n elements; and $\mathbf{u}_i$ denotes a vector of length n with 1 in the ith position and 0 in other positions.

An addition operation which adds row (resp. column) j multiplied by a non-zero scalar β to row (resp. column) i can be denoted by a tuple $\tau\triangleq(i,j,\beta)$, where i and j are two distinct indexes. Let $\mathbf{Q}_\tau$ denote the corresponding elementary matrix of operation τ.

Since an arbitrary non-singular $n\times n$ matrix can be transformed from $\mathbf{A}$ by a finite sequence of elementary operations, e.g., $\mathbf{Q_m}\mathbf{Q}_\pi\mathbf{Q}_{\tau_1}\cdots\mathbf{Q}_{\tau_K}\mathbf{A}$, we transform the LAE problem with two random diagonal matrices $\mathbf{Q_m},\mathbf{Q}'_\mathbf{m}$; two random permutation matrices $\mathbf{Q}_\pi,\mathbf{Q}'_\pi$ and two sequences of random elementary addition operations $\{\mathbf{Q}_{\tau_1},\ldots,\mathbf{Q}_{\tau_K}\}$, $\{\mathbf{Q}'_{\tau_1},\ldots,\mathbf{Q}'_{\tau_{K'}}\}$.

More precisely, we respectively transform $\mathbf{A}$ and $\mathbf{b}$ into $\mathbf{A}'$ and $\mathbf{b}'$, where

$$\mathbf{A}' = \mathbf{Q_m}\mathbf{Q}_\pi\mathbf{Q}_{\tau_1}\cdots\mathbf{Q}_{\tau_K}\mathbf{A}\mathbf{Q}'_\mathbf{m}\mathbf{Q}'_\pi\mathbf{Q}'_{\tau_1}\cdots\mathbf{Q}'_{\tau_{K'}} \tag{6}$$

$$\mathbf{b}' = \mathbf{Q_m}\mathbf{Q}_\pi\mathbf{Q}_{\tau_1}\cdots\mathbf{Q}_{\tau_K}(\mathbf{b}+\Delta\mathbf{b}) \tag{7}$$

If the client outsources the transformed LAE problem $\mathbf{A}'\mathbf{y} = \mathbf{b}'$, let $\mathbf{y}^*$ be the returned solution. One can check that

$$\mathbf{Q}'_\mathbf{m}\mathbf{Q}'_\pi\mathbf{Q}'_{\tau_1}\cdots\mathbf{Q}'_{\tau_{K'}}\mathbf{y}^* = \mathbf{x}^* + \Delta\mathbf{x} \tag{8}$$

which indicates that the client is able to derive the solution of the original LAE problem $\mathbf{x}^*$ from $\mathbf{y}^*$.

For convenience, we hereby summarize the disguising/recovery algorithms for privacy preserving as follows:

1) *Key Generation*: The client generates a random vector $\Delta\mathbf{x}$, two random diagonal matrices $\mathbf{Q_m}, \mathbf{Q}'_\mathbf{m}$; two random permutation matrices $\mathbf{Q}_\pi, \mathbf{Q}'_\pi$ and two sequences of random elementary addition operations $\{\mathbf{Q}_{\tau_1}, \ldots, \mathbf{Q}_{\tau_K}\}, \{\mathbf{Q}'_{\tau_1}, \ldots, \mathbf{Q}'_{\tau_{K'}}\}$.
2) *Problem Disguising*: The client computes $\mathbf{A}'$ and $\mathbf{b}'$ according to (6) and (7), respectively.
3) *Outsourcing*: The client outsources the disguised version of the original problem $\mathbf{A}'\mathbf{y} = \mathbf{b}'$ to the cloud.
4) *Solution Recovery*: After receiving the solution $\mathbf{y}^*$, the client obtains the solution to the original problem by computing $\mathbf{x}^* = \mathbf{Q}'_\mathbf{m}\mathbf{Q}'_\pi\mathbf{Q}'_{\tau_1}\cdots\mathbf{Q}'_{\tau'_K}\mathbf{y}^* - \Delta\mathbf{x}$.

Lemma 2. *If both K and K' are bounded above by $\mathcal{O}(n)$, the computation complexity of the disguising and recovery algorithms for preserving the privacy of the outsourced LAE problem is $\mathcal{O}(n^2)$.*

Proof. Since K and K' are both bounded by $\mathcal{O}(n)$, the computation complexity of the key generation process is $\mathcal{O}(n)$. Computing $\mathbf{A}'$ involves aggregated row and column multiplication; aggregated row and column switching; and K row additions and K' column additions, which takes time $4n^2 + (K+K')n = \mathcal{O}(n^2)$. The complexity of calculating $\Delta\mathbf{b}$ is $\mathcal{O}(n^2)$ since it involves a matrix-vector multiplication. Once obtaining $\mathbf{b}+\Delta\mathbf{b}$, $\mathbf{b}'$ can be computed with complexity $\mathcal{O}(n)$ through K row additions, one aggregated row switching and one aggregated row multiplication. Similarly, to derive the solution, the client computes $\mathbf{x}^* = \mathbf{Q}'_\mathbf{m}\mathbf{Q}'_\pi\mathbf{Q}'_{\tau_1}\cdots\mathbf{Q}'_{\tau_{K'}}\mathbf{y}^* - \Delta\mathbf{x}$ with complexity $\mathcal{O}(n)$. In summary, the computation complexity for disguising and recovery of the LAE problem is $\mathcal{O}(n^2)$. □

Remark 1. In [21], random scaling and permutations are employed to disguise a matrix. Based on a similar idea, matrix masking algorithms for securely outsourcing matrix inversion and matrix determinant computation problems are proposed in [19] and [20], respectively. These methods, however, have two common drawbacks. For one thing, since scaling and permutation cannot mask zero elements, the amount of zero entries remains the same after transformation. For another, the non-zero entries in $\mathbf{A}$ and $\mathbf{A}'$ have the following relationship for any h, i, j, k:

$$\frac{a'_{ij}a'_{hk}}{a'_{ik}a'_{hj}} = \frac{a_{ij}a_{hk}}{a_{ik}a_{hj}}.$$

In our algorithm, these issues can be addressed by elementary addition transformations. The proposed matrix disguising algorithm can also be interpreted as protecting $\mathbf{A}$ by both left-multiplying and right-multiplying two nonsingular matrices, similar as that in [23]. Nevertheless, our procedure has two benefits compared to that used in [23]: i) The mask matrices in [23] are required to be sparse in order to guarantee low complexity. We give a systematic procedure to disguise $\mathbf{A}$ through a series of elementary operations of low complexity, which avoids generating the aforementioned two matrices directly. We hence do not require the equivalent mask matrices in our method, e.g., $\mathbf{M} = \mathbf{Q_m}\mathbf{Q}_\pi\mathbf{Q}_{\tau_1}\cdots\mathbf{Q}'_{\tau_K}$ and $\mathbf{N} = \mathbf{Q}'_\mathbf{m}\mathbf{Q}'_\pi\mathbf{Q}'_{\tau_1}\cdots\mathbf{Q}'_{\tau_{K'}}$ to be sparse. This is because the equivalent mask matrices generated by our method may be dense as the product of extremely sparse matrices can be completely dense [34]. ii) In our method, the client can easily control the computation overhead of problem disguising and solution recovery by adjusting parameters K and K', while the computation complexity of the algorithm in [23] depends on the sparsity of those two mask matrices whose generation method is not given in [23].

4.4 Misbehavior Detection

In Section 3, we have analyzed the possible misbehavior taken by malicious agents. A simple updating verification of $\mathbf{A}_i^{\mathrm{T}}\mathbf{x}_i(k) = b_i$ is effective only for detecting malicious agents who randomly update their values in each iteration. A stronger detection approach is to let the ad hoc agents monitor their neighbors' updates by double checking whether their computation is according to the algorithm in (4). From (4), we notice that at $(k+1)$th iteration, for agent j to verify the update $\mathbf{x}_i(k+1)$ from agent i, agent j requires the knowledge of $\mathbf{A}_i^{\mathrm{T}}$, b_i, and $\bar{\mathbf{x}}_i(k)$. During the problem setup stage, the information distributed to agent j should include not only $[\mathbf{A}_j^{\mathrm{T}}\ b_j]$, but also $[\mathbf{A}_i^{\mathrm{T}}\ b_i]$ for $i \in \mathcal{N}_j$. For each $i \in \mathcal{N}_j$, it can send agent j the set of solution vectors it collected, i.e., $\{\mathbf{x}_l(k) | l \in \mathcal{N}_i\}$, so that agent j can compute $\bar{\mathbf{x}}_i(k)$ and then verify $\mathbf{x}_i(k+1)$.

We set the update process as follows: in $(k+1)$th iteration, agent i broadcasts message

$$\Phi_i(k+1) = \{\mathbf{x}_i(k+1),\ \mathcal{N}_i,\ \{\mathbf{x}_m(k) | m \in \mathcal{N}_i\},\ F_{i,k}\}. \tag{9}$$

where $F_{i,k}$ is the alarm information indicating the malicious agent detected by i at iteration k. $F_{i,k} = 0$ if no misbehavior has been detected; $F_{i,k} = l$ if agent l is detected by agent i as a malicious agent. To verify $\mathbf{x}_i(k+1)$, agent i's neighbor agents recompute $\mathbf{x}_i(k+1)$ according to (4) using the information of $\mathbf{A}_i^{\mathrm{T}}$, $\mathbf{b}_i$ and $\mathbf{x}_m(k)$ contained in $\Phi_i(k+1)$. If agent j detects its neighbor i conducts a malicious behavior, in its next broadcasting message $\Phi_i(k+2)$, it sets $F_{j,k+1} = i$. Based on our assumptions that the majority of agents are honest within any agent's neighborhood, a malicious agent will be monitored by more than half of its neighbors. Thus more than half of its neighbors will generate alarm messages reporting the malicious agent. Once there exists a certain number (half of the number of i's neighbors) of alarm messages denoting agent i's misbehavior in the same iteration, agent i will be confirmed as a malicious agent and eliminated from the cloud. The share of the outsourcing problem originally assigned to i will then be reassigned to one of his neighbors.

It is worth noting that the cooperative detection mechanism introduced above incurs extra computational overhead to each agent. When agent j double checks the computation according to (4) for its neighbor i, it incurs a workload with complexity $\mathcal{O}(d_i n)$. If agent j monitors all its neighbors, the total workload will be $\mathcal{O}(\sum_1^{d_j} d_i n)$. Under a well-connected network topology, for example, a complete network graph with $d_i = n-1$ for any agent i, in a single iteration the verification computational overhead for each monitoring agent is $\mathcal{O}(n^3)$, resulting in the total computation cost exceeding $\mathcal{O}(n^3)$.

To reduce the computation overhead, we further set a verification probability p: at each iteration, agent j will verify a received message from its neighbors with probability p. For this probabilistic verification scheme, every agent needs to keep its neighbors' broadcast information until the end of next iteration. Suppose at $(k+1)$th iteration, agent j receives an alarm message $F_{m,k} = i$, then agent j will double check $\mathbf{x}_i(k)$ with probability 1, thus requiring the knowledge of $\Phi_i(k)$. The system parameter p can be tuned to balance the detection performance and computational overhead. The verification algorithm is summarized in Algorithm 1.

Algorithm 1. Verification Process of Agent i at Iteration $k+1$

```
INPUT: $\Phi_i(k+1)$, $\Phi_i(k)$, $\mathbf{A}_j$, $\mathbf{b}_j \forall j \in \mathcal{N}_i$, $p$;
OUTPUT: $F_{i,k}$, $F_{i,k+1}$;
for $j \in \mathcal{N}_i$ do
    if $F_{j,k} \neq 0$ and $F_{j,k} \in \mathcal{N}_i$ then
        compute $\mathbf{x}'_{F_{j,k}}(k)$ by (4);
        if $\mathbf{x}'_{F_{j,k}}(k) \neq \mathbf{x}_{F_{j,k}}(k)$ then
            $F_{i,k} = F_{j,k}$;
        end
    end
    randomly choose $m$ from $(0,1)$;
    if $m < p$ then
        compute $\mathbf{x}'_j(k+1)$;
        if $\mathbf{x}'_j(k+1) \neq \mathbf{x}_j(k+1)$ then
            $F_{i,k+1} = j$;
        end
    end
end
```

Remark 2. When the network is well connected, the verification process will introduce heavy computational overhead. For example, if the network topology is a complete graph, even with a verification probability p, the computational cost for each agent is $\mathcal{O}(pn^3)$ in each iteration. However, the agents may use a subgraph of the well-connected network topology as their logical network graph to run the algorithm. As long as the logical network graph is connected, our algorithm will converge. In this case, the computational cost for one agent in each iteration will be reduced to $\mathcal{O}(pd'^2 n)$, where d' is its degree in the logical network graph. For example, in our simulations, we use an Erdős-Rényi (ER) random graph $G(n, \frac{\ln n}{n})$, in which the average node degree is $\ln n$. In this case, on average, the computational cost for each agent in a single iteration is $\mathcal{O}(pn(\ln n)^2)$.

When a malicious agent i injects an unfaithful intermediate result, each of its d_i neighbors will verify its update with probability p. The probability that malicious agent i successfully injects a single unfaithful intermediate result without being identified by its neighbors is $P_1 = (1-p)^{d_i}$. By Theorem 1, injecting a finite number of unfaithful intermediate results will not affect the final solution, so the malicious agent should inject unfaithful intermediate results from time to time in order to diverge the algorithm. The probability that malicious agent i successfully injects m unfaithful intermediate results without being caught is $P_m = ((1-p)^{d_i})^m$, which decreases exponentially in m. As m goes to infinity, such a malicious agent will be detected by its neighbors almost surely.

4.5 Main Scheme

In the above, we have developed three important components for secure LAE outsourcing. Here we integrate the components into a complete secure LAE outsourcing system consisting three stages.

- *Setup Stage:* In this very first phase, the client generates the secure key and disguises the problem using the proposed algorithm in Section 4.3. After the problem disguising phase, the client distributes transformed problem parameters $\mathbf{A}'$ and $\mathbf{b}'$ to corresponding agents according to the following rule: to agent i, the client distributes $\mathbf{A}_i'^{\mathrm{T}}, b_i'$, and $\{\mathbf{A}_j'^{\mathrm{T}}, b_j' | \forall j \in \mathcal{N}_i\}$.
- *Distributed Computation Stage:* At iteration 0, each agent i picks one initial solution $\mathbf{x}_i(0)$ randomly. At iteration $k+1, k \geq 0$, agent i performs the cooperative verification algorithm Algorithm 1, updates its consensus value $\mathbf{x}_i(k+1)$ by (4), and then finishes iteration $k+1$ by broadcasting the updating message $\Phi_i(k+1)$ to its neighbors. Agent i will terminate the consensus process if
$$\max_{j \in \mathcal{N}_i} \|\mathbf{x}_i - \mathbf{x}_j\|_\infty \leq \varepsilon \tag{10}$$
holds for consecutive $2L$ step, where $\|\cdot\|_\infty$ represents the infinity-norm of a vector and L is the diameter of the underlying network graph.
- *Final Solution Stage:* The distributed average consensus algorithm will achieve a final convergent point $\bar{\mathbf{x}}$, which is the solution of the transformed problem, i.e., $\mathbf{y}^* = \bar{\mathbf{x}}$. The client can then transform the solution to that of the original problem by computing $\mathbf{x}^* = \mathbf{Q}'_{\mathbf{m}} \mathbf{Q}'_\pi \mathbf{Q}'_{\tau_1} \cdots \mathbf{Q}'_{\tau_{K'}} \mathbf{y}^* - \Delta\mathbf{x}$, referring to Section 4.3.

5 Performance Analysis

5.1 Convergence Analysis

With the distributed consensus-based algorithm, each agent updates its local solution by utilizing the information from its neighbors. A critical issue in the algorithm design is the convergence performance. The local solutions of all the agents need to not only reach a consensus but also converge to the exact solution of the LAE problem. Furthermore, a fast convergence rate is preferred. Without false update, our revised robust consensus-based algorithm yields the same local solutions as the original algorithm, whose correctness has been proved in [12]. According to Theorem 1, our algorithm is able to tolerate finite number of false updates. Besides, the monitoring scheme would prevent malicious agents from sabotaging the consensus process continuously. Therefore, local solutions in our algorithm can eventually converge to the exact solution of the LAE problem. Here, we provide some insights on the convergence rate of our scheme.

Let $\mathbf{x}^*$ be the exact solution of the LAE. For ease of presentation, we conduct analysis based on the updating process $\mathbf{x}_i(t+1) = \mathbf{P}_{\mathbf{A}_i^{\mathrm{T}}}\mathbf{x}_i(t) + \bar{\mathbf{x}}_i(t) - \mathbf{P}_{\mathbf{A}_i^{\mathrm{T}}}\bar{\mathbf{x}}_i(t)$. Suppose that the local solution at agent i, after $k+1$ iterations, deviate from $\mathbf{x}^*$ with a value of $\mathbf{e}_i(k+1)$. Since $\mathbf{P}_{\mathbf{A}_i^{\mathrm{T}}} + \mathbf{P}_i = \mathbf{I}$ according to (2), we have

$$\begin{aligned}\mathbf{e}_i(k+1) &= \mathbf{x}_i(k+1) - \mathbf{x}^*\\ &= \mathbf{P}_{\mathbf{A}_i^{\mathrm{T}}}\mathbf{x}_i(k) + \mathbf{P}_i\frac{\sum_{j\in\mathcal{N}_i}\mathbf{x}_j(k)}{d_i} - (\mathbf{P}_{\mathbf{A}_i^{\mathrm{T}}} + \mathbf{P}_i)\mathbf{x}^*\\ &= \frac{\mathbf{A}_i\mathbf{A}_i^{\mathrm{T}}}{\mathbf{A}_i^{\mathrm{T}}\mathbf{A}_i}(\mathbf{x}_i(k) - \mathbf{x}^*) + \mathbf{P}_i\frac{\sum_{j\in\mathcal{N}_i}\mathbf{x}_j(k)}{d_i} - \mathbf{P}_i\mathbf{x}^*\\ &= \frac{\mathbf{A}_i^{\mathrm{T}}(\mathbf{x}_i(k) - \mathbf{x}^*)}{\mathbf{A}_i^{\mathrm{T}}\mathbf{A}_i}\mathbf{A}_i + \mathbf{P}_i\frac{\sum_{j\in\mathcal{N}_i}\mathbf{x}_j(k)}{d_i} - \mathbf{P}_i\mathbf{x}^*\\ &= \frac{b_i - b_i}{\mathbf{A}_i^{\mathrm{T}}\mathbf{A}_i}\mathbf{A}_i + \mathbf{P}_i\frac{\sum_{j\in\mathcal{N}_i}(\mathbf{x}_j(k) - \mathbf{x}^*)}{d_i}\\ &= \mathbf{P}_i\frac{\sum_{j\in\mathcal{N}_i}\mathbf{e}_j(k)}{d_i}.\end{aligned} \tag{11}$$

Let $\mathbf{D}$ be the adjacency matrix corresponding to the underlying graph of the network and $\mathbf{H} = \mathrm{diag}(\frac{1}{d_1},\ldots,\frac{1}{d_n})\mathbf{D}$. Let $\mathbf{c}(k+1) = [\mathbf{e}_1^{\mathrm{T}}(k+1), \mathbf{e}_2^{\mathrm{T}}(k+1), \ldots, \mathbf{e}_n^{\mathrm{T}}(k+1)]^{\mathrm{T}}$ be the deviation of all agents after $k+1$ iterations. Then, according to (11),

$$\mathbf{c}(k+1) = \mathbf{PGc}(k), \tag{12}$$

where

$$\mathbf{P} = \begin{pmatrix}\mathbf{P}_1 & \cdots & \mathbf{0}\\ \vdots & \ddots & \vdots\\ \mathbf{0} & \cdots & \mathbf{P}_n\end{pmatrix}, \qquad \mathbf{G} = \mathbf{H}\otimes\mathbf{I},$$

with $\otimes$ denoting the Kronecker product. $\mathbf{P}\in\mathbb{R}^{n^2\times n^2}$ is a block diagonal matrix which is determined by the matrix $\mathbf{A}$ in the LAE problem, while $\mathbf{G}\in\mathbb{R}^{n^2\times n^2}$ is determined by the connectivity of the network.

For a square matrix $\mathbf{M}$, let $\rho(\mathbf{M})$ denote its spectral radius, i.e., the maximum modulus of all its eigenvalues. Since every $\mathbf{P}_i$ is a projection matrix, $\rho(\mathbf{P}_i) = 1$ holds for all i, and hence $\rho(\mathbf{P}) = 1$. It is easy to see that $\mathbf{H}$ is a stochastic matrix. Therefore, $\mathbf{G}$ is also a stochastic matrix and $\rho(\mathbf{G}) = 1$. According to (12), the convergence rate of our scheme is bounded above by the spectral radius of the iteration matrix $\rho(\mathbf{PG})$. With connected network topology and nonsingular $\mathbf{A}$, it is proved in [12] that all local solutions would converge to the exact solution exponentially fast, indicating $\rho(\mathbf{PG}) < 1$ such that $\lim_{k\to\infty}(\mathbf{PG})^k = \mathbf{0}$.

Based on aforementioned analysis, two factors affect the convergence rate of our algorithm. The first one is the condition number of the matrix $\mathbf{A}$, which characterizes how inaccurate the solution will be after approximation. The work in [35] investigates the influence of a matrix condition number on the convergence rate of iterative methods such as Jacobi method and Gauss-Seidel method. Let $\hat{\mathbf{x}}$ denote the local solution such that $\mathbf{A}\hat{\mathbf{x}} = \hat{\mathbf{b}}$. If the condition number of $\mathbf{A}$ is large, the deviation between $\hat{\mathbf{x}}$ and $\mathbf{x}^*$ would be large even if $\hat{\mathbf{b}}$ is close to $\mathbf{b}$. Thus, an ill-conditioned $\mathbf{A}$, albeit nonsingular, can yield a poor convergence rate, meaning that $\rho(\mathbf{PG})$ is very close to 1. In order to improve the convergence rate, several preconditioning techniques have been proposed to transform $\mathbf{A}$ with respect to different iterative methods [36], [37], [38]. Certain types of matrices, such as diagonally dominant matrices, could achieve a good convergence rate as indicated in on our simulations. In this work, we focus on the security aspect of the LAE outsourcing problem, analyzing which types of matrices are suitable to our consensus based algorithm or studying the preconditioning techniques for that algorithm is out of scope of this work. Therefore, we employ diagonally dominant matrices to conduct the simulations. The other factor affecting the convergence rate is the connectivity of the network graph. The relationship between the convergence rate and the adjacency of the network will be discussed later, where we present some numerical results concerning the impact of connectivity on the total computation time of one agent.

5.2 Security Analysis

5.2.1 *Privacy Preserving*

Note that the transformation keys $\Delta\mathbf{x}$, $\mathbf{Q_m}$, $\mathbf{Q}'_{\mathbf{m}}$, $\mathbf{Q}_\pi$, $\mathbf{Q}'_\pi$ and $\{\mathbf{Q}_{\tau_1},\ldots,\mathbf{Q}_{\tau_K}\}$, $\{\mathbf{Q}'_{\tau_1},\ldots,\mathbf{Q}'_{\tau_{K'}}\}$ are kept local with the client throughout the updating process. The only information that a malicious agent i could obtain is $[\mathbf{A}'^{\mathrm{T}}_i\ b'_i]$, $[\mathbf{A}'^{\mathrm{T}}_j\ b'_j]$, for some or all $j\in\mathcal{N}_i$, and the solution $\mathbf{y}^*$ of the transformed problem. Considering that the network may be completely connected, let's assume an adversary obtains $\mathbf{A}'$, $\mathbf{b}'$, and the solution to the disguised problem $\mathbf{y}^*$.

We first consider the output privacy, $\mathbf{x}^* = \mathbf{Q}'_{\mathbf{m}}\mathbf{Q}'_\pi\mathbf{Q}'_{\tau_1}\cdots\mathbf{Q}'_{\tau_{K'}}\mathbf{y}^* - \Delta\mathbf{x}$. As $\Delta\mathbf{x}$ is a random n-dimensional vector, the possible attack strategy for an adversary is statistical attack, which takes the advantage of the distribution information of $\Delta\mathbf{x}$ to approximate the solution $\mathbf{x}^*$. In our disguising scheme, since $\mathbf{x}^*$ is also masked by the series of elementary operations, each element in the masked $\mathbf{x}^*$ is a linear combination (with random weights) of all elements in $\mathbf{x}^*$, hence protecting $\mathbf{x}^*$ against statistical attack. In other words, no information of $\mathbf{x}^*$ can be obtained by the adversary. Similarly, an adversary cannot learn $\mathbf{b}$. With respect to $\mathbf{A}$, due to both elementary row and column transformations, each element is also a random combination of all elements. Without the knowledge of those elementary operations, an adversary cannot determine $\mathbf{A}$. However, since the elementary operations do not change invertibility and dimension of the input matrix, our scheme is not indistinguishable under chosen-plaintext attack (IND-CPA). Since all non-singular matrices of the same size are equivalent under elementary transformations, the client can improve the security level of the disguising scheme by increasing K and K'. However, to guarantee $\mathcal{O}(n^2)$ local computation complexity, both K and K' must be bounded above by $\mathcal{O}(n)$.

5.2.2 *Misbehavior Detection*

A malicious agent trying to sabotage the algorithm continuously will almost surely be detected. Now we consider the situation that a malicious agent broadcasts a false alarm message $F_{(i,k)} = j$ accusing an honest agent j for updating a false value at iteration k. Suppose there is a common neighboring agent of agent i and j. Upon receiving this false alarm at iteration $k+1$, this common neighbor verifies $\Phi_j(k)$ with probability 1 and finds that $\Phi_j(k)$ is actually correct. In other words, with honest common neighbors, the malicious agent has no way to cheat others by sending false alarm messages. Since malicious agents are assumed not colluding, the probability that two fake alarms on the same integrity agent in one iteration is negligible. In summary, we have the following deductions: malicious behavior that can sabotage the algorithm can be detected by probability infinitely close to 1; the probability that falsely detecting an integrity agent as a malicious one is negligible.

5.3 Computation Complexity Analysis

Low computation complexity is one of our design goals. For one thing, due to lack of computation resource, an individual agent may not afford time-consuming computation. For another, if the total time cost is much longer than that by solving the LAE problem locally, the client would also be reluctant to outsource the problem. The following theorem gives the total computation complexity for the client and participatory agents throughout the outsourcing process.

Theorem 2. *Through the secure outsourcing process, the local computation complexity for the client is $\mathcal{O}(n^2)$; the average computation complexity for each agent is $\mathcal{O}(l(dp+1)dn)$, where l is the number of iterations to reach the consensus and d is the average degree of the network graph.*

Proof. For the client, the only computation burden stems from the problem disguising and solution recovery. By Lemma 2, the disguising computation complexity is $\mathcal{O}(n^2)$. For recovering the solution, the client computes $\mathbf{x}^* = \mathbf{Q}'_{\mathbf{m}}\mathbf{Q}'_{\pi}\mathbf{Q}'_{\tau_1}\cdots\mathbf{Q}'_{\tau_{K'}}\mathbf{y}^* - \Delta\mathbf{x}$ which incurs complexity $\mathcal{O}(n)$. Thus, the total computation complexity for the client is $\mathcal{O}(n^2)$.

For each agent, it performs two tasks at each iteration—updating its local solution and probabilistically monitors its neighbors. Checking the message from neighbors has the same time complexity as updating its solution which is $\mathcal{O}(d_i n)$ by Lemma 1. Thus, the expected complexity within one iteration for an agent is $\mathcal{O}((dp+1)d_i n)$ and the total average complexity for each agent can be given as $\mathcal{O}(l(dp+1)dn)$. □

Based on Theorem 2, the average computation complexity for each agent is related to the network graph and number of steps to reach consensus. When $d \ll n$, the computation complexity for each agent approximates to $\mathcal{O}(ln)$. Through extensive simulations, we notice that for a diagonally dominant $\mathbf{A}$, the convergent step is roughly bounded by $\mathcal{O}(n)$, resulting in that the average complexity for each agent is less than $\mathcal{O}(n^2)$ which is one-order lower than the computation complexity of solving the LAE problem directly.

5.4 Communication Complexity Analysis

The main communication burden for the client is to distribute the disguised problem. As $\mathbf{A}'^{\mathrm{T}}_i$ and b'_i are distributed not only to agent i, but also to its neighbor agents, on average, the client needs to send $d+1$ copies of disguised LAE problem, incurring $\mathcal{O}(dn^2)$ communication overhead.

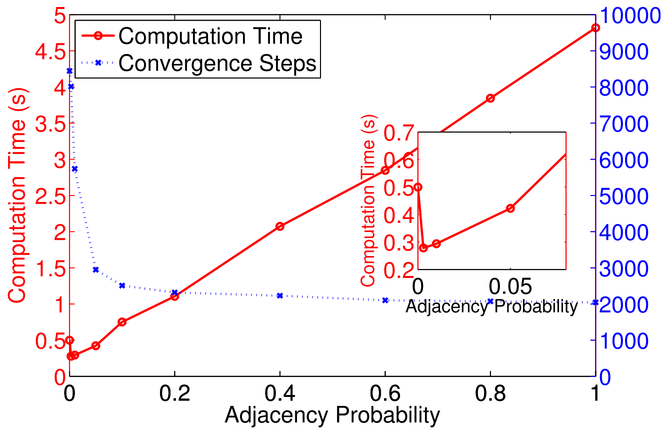


Fig. 3. Total computation time per agent and number of convergence steps versus probability of the ER graph.

For simplicity, we use the amount of data exchanged between agents to characterize the communication complexity for each agent, regardless of the scheduling protocol. At each iteration, agent i broadcasts message Φ_i. To update its local solution, it also needs to receive d_i messages from its neighbors. Since each message contains the current solution of an agent and local solutions of its neighbors in the previous round, the average size of one message is $\mathcal{O}(dn)$. On average, an agent needs to send and receive totally $d+1$ messages. Therefore, the average total communication complexity for each agent can be given as $\mathcal{O}(l(d+1)dn) = \mathcal{O}(ld^2n)$.

6 Numerical Results

In this section, we present numerical results to evaluate the performance of the proposed scheme in terms of both efficiency and robustness. We employ a PC with Intel Core 2 Quad CPU of 2.34 GHz and 4 GB memory to run the problem disguising process at the client. The consensus-based LAE solving process is performed by memory optimized instance (r3.2xlarge) on Amazon Elastic Computing Cloud (EC2). The virtual core of the employed instance is equivalent to an Intel Xeon E5-2670 v2 processor, whose running frequency is comparable to common desktops or mobile devices. We also develop simulation codes using Python with the NumPy package extension.

6.1 Convergence Performance

Similar as in [13], we generate random diagonally dominant matrices to construct the LAE. We use an Erdős-Rényi (ER) random graph $G(n,q)$ [39] to model the connectivity of the cloud agents. According to the analysis in Section 5.1, the connectivity of the underlying graph affects the convergence rate of our algorithm. Intuitively, the convergence can be reached at a faster speed under a graph with better connectivity. However, the computation complexity per iteration for an agent is $\mathcal{O}(d_i n)$, which implies that a lower average degree can bring benefit in terms of computation time. Fig. 3 illustrates the trade-off between the time complexity per iteration and convergence time under different graph connectivity. To guarantee the connectedness of the network topology, we use the union of an n-agent cycle and the ER graph generated by $G(n,q)$ model as the network graph. In an extreme case, the underlying graph is a cycle with average degree 2 when the

TABLE 1
Computation Overhead

Problem size n	1,000	2,000	3,000	4,000	5,000	8,000	10,000
Local Jacobi Method [40]	10 secs	78 secs	250 secs	10 mins	19 mins	78 mins	151 mins
Problem Disguising (our method)	0.066 sec	0.263 sec	0.599 sec	1.089 secs	1.739 secs	4.395 secs	7.225 secs
Problem Solving (our method)	0.28 sec	0.78 sec	1.33 secs	1.93 secs	2.57 secs	4.90 secs	6.84 secs

probability q equals to 0. Fig. 3 shows the relationship between the connectivity probability q and the total running time (equivalently the number of convergence steps) for each agent in a 1,000-agent network.

As shown in Fig. 3, although the number of convergence steps decreases as the network connectivity increases, the total running time for each agent is dominated by the time complexity per iteration. This is due to the fact as in Theorem 2 that the computation time grows in the order of $\mathcal{O}(l(dp+1)dn)$, which is $\mathcal{O}(d^2)$ of the average network degree d but $\mathcal{O}(l)$ of the number of convergence steps l. The results also show that the number of convergence steps decreases exponentially as the connectivity improves. When the probability $q > 0.1$, only marginal reduction of convergence steps can be achieved. Since the average degree of the underlying graph $d \approx nq$, the time complexity per iteration reduces linearly when q decreases. Especially, as compared to a cyclic topology (i.e., $q = 0$), the computation time can be lowered by adding shortcuts to the cycle. This is because such a topology could benefit from the dramatical reduction of convergence steps. In the following simulations, we set $q = \frac{\ln n}{n}$, which is the sharp threshold for the connectedness of $G(n, q)$ and corresponds to a relative sparse topology [39].

The computation cost of our scheme with respect to the LAE problem dimension n is given in Table 1. To illustrate the efficiency of our scheme, we also locally solve the same problem using the existing Jacobi method [40]. For problem disguising, we set the amount of row and column addition operations equal to the problem size, i.e., $K = K' = n$. As shown in the table, for the problem of size $n = 10,000$, the problem disguising algorithm only takes less than 8 seconds. Moreover, for solving the LAE, approximately 2.5 hours are needed by the Jacobi method, while in contrast the computation time required by each agent using the proposed consensus-based algorithm is less than 7 seconds. With respect to the memory occupation, each agent needs to store a couple of $10,000 \times 1$ vectors. Each vector, of size $10,000 \times 8$ Bytes ≈ 80 KB, is significantly shorter than the memory usage by the local Jacobi method which requires at least $10,000^2 \times 8$ Bytes ≈ 800 M.

6.2 Security Performance

We adopt the following two metrics to evaluate the security performance of the proposed scheme—root mean square error (RMSE) and mean standard deviation (MSD):

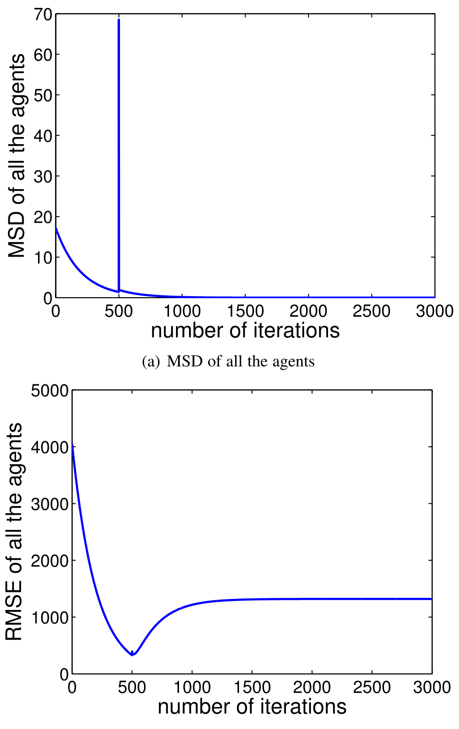
$$RMSE = \frac{1}{n}\sum_{i=1}^{n} \|\mathbf{x}_i(t) - \mathbf{x}^*\|, \quad (13)$$

$$MSD = \frac{1}{n}\|\mathbf{x}_{SD}\|_1, \quad (14)$$

where $\|\cdot\|_1$ represents the l_1-norm of a vector; $\mathbf{x}_{SD}$ is the standard deviation vector with every component being the standard deviation of the value in the corresponding component of all local solutions. RMSE and MSD characterize the error between all local solutions and the exact solution.

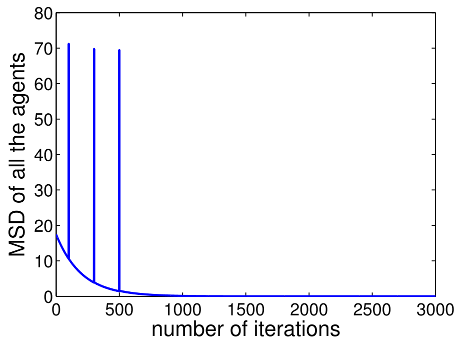
As discussed in Section 3, without our robust algorithm, a malicious agent can manipulate the final results with a one-time false update. This phenomenon is illustrated in Fig. 4, where a 500-agent ad hoc cloud is used in the simulations. At the 500th iteration, a malicious agent injects a false update (see the spike at iteration 500 in Fig. 4a). In presence of the malicious agent, final consensus can be reached as shown in Fig. 4a. However, the RMSE as shown in Fig. 4b indicates that the algorithm eventually converges to an incorrect solution.

When the proposed robust consensus-based algorithm is implemented, we apply the same attack at 100th, 300th and 500th iterations. Fig. 5 shows that these three false updates do not mislead normal agents to an incorrect solution. Since each agent's next update is only determined by the current updates of its neighbors, the false update can only influence the next update of the malicious agent's own neighbors. However, as illustrated in Fig. 5b, these attacks impose little impact on the convergence process. Despite some impulses, all the agents are on the right course of agreeing on the final solution. This is because that these impacts will be averaged out by the correct updates of honest agents and thus decay as time goes.

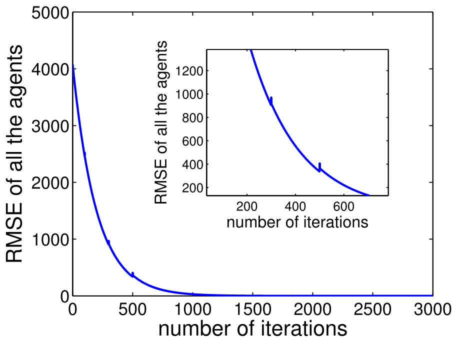


(a) MSD of all the agents
(b) RMSE of all the agents

Fig. 4. The effect of results manipulating attack when without our robust algorithm.

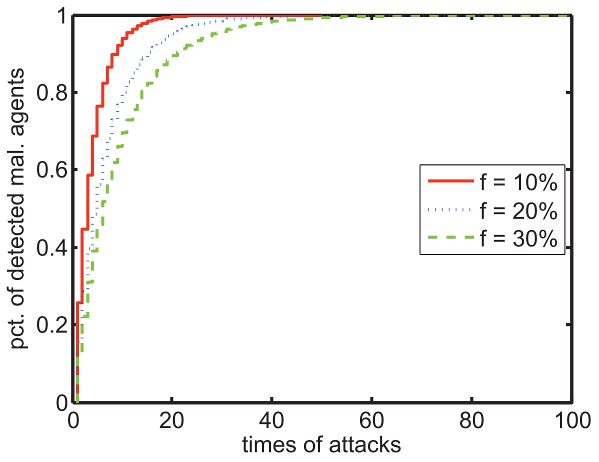


(a) MSD of all the agents

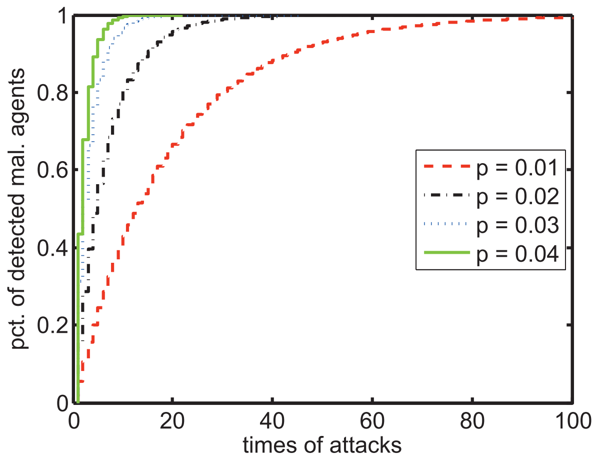


(b) RMSE of all the agents

Fig. 5. The effect of results manipulating attack when with our robust algorithm.



(a) percentage of detected malicious agents under different fraction model



(b) percentage of detected malicious agents under different detection probability

Fig. 6. Performance of the misbehavior detection scheme.

We evaluate the performance of our misbehavior detection scheme with a 1000-agent ad hoc cloud. For simplicity, we employ a complete graph as the network topology of these 1,000 agents. The results shown in Fig. 6 are averaged over 5,000 independent runs. For each agent, the number of malicious agents in its neighborhood is bounded above by a fraction $f \in (0, 1)$. Each malicious agent updates a false result at each iteration. A malicious agent is considered to be detected if there are at least f-fraction of its neighbor agents claiming its misbehavior. For example, when $f = 0.1$, we say a malicious agent has been detected if more than 10 percent of agents in its neighborhood have discovered its misbehaviors. The percentage of detected malicious agents under different fraction models is illustrated in Fig. 6a, where the detection probability is fixed at 0.04. When $f = 0.1$, almost all malicious agents would be detected if they conduct more than 20 times of attack. Fig. 6b shows the percentage of detected malicious agents under different detection probabilities, where the fraction of malicious node is fixed at 0.1. When the detection probability $p \geq 0.02$, more than 80 percent malicious agents would be detected if they conduct more than 10 false updates. For a higher detection probability, e.g., $p = 0.03$ and $p = 0.04$, almost no malicious agent is able to conduct more than 18 times of attack before being detected.

6.3 Performance under WiFi Based Ad Hoc Clouds

In this section, we develop C++ codes within the OMNeT++ discrete event simulation environment to evaluate the performance of the proposed algorithm in a WiFi based wireless ad hoc cloud. We consider that 100 cloud agents are uniformly distributed in a $500\text{m} \times 500\text{m}$ square area. The wireless communications among the agents are carried out based on the IEEE 802.11g standard over the 2.4 GHz channel and with CSMA/CA protocol as the MAC layer protocol. The agents have the same transmit power of 13 dBm and the SINR threshold for successfully decoding a message is 4 dB. The default values for the CSMA/CA protocol parameters are used. In order to run our algorithm in a synchronous manner (to cater for the discrete-time consensus-based algorithm), the time is equally divided into updating windows with each window having the same length. During each window, each agent randomly chooses a time to start the CSMA/CA based contention and only broadcasts its local solution when it succeeds in the contention. Each agent can update the average value $\bar{\mathbf{x}}_i(t)$ upon receiving a local solution from one of its neighbors. At the end of an updating window, each agent calculates its local solution based on the up-to-date $\bar{\mathbf{x}}_i(t)$. A message may get lost due to collision in the contention process. Also, due to channel access delay, if a message arrives beyond the current window, it will be dropped and is considered as a packet loss. This local solution is then stored in the corresponding agent until broadcasted in the next window.

The performance of our consensus-based algorithm with packet loss is illustrated in Fig. 7. The solid line characterizes the relationship between the number of iterations

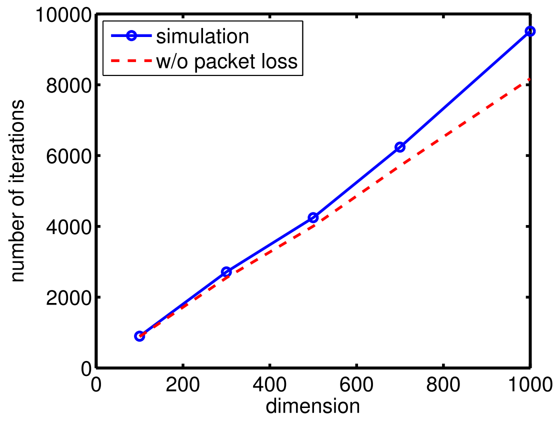


Fig. 7. Performance of consensus-based algorithm with packet loss.

required for convergence and the dimension of the LAE problem. The dotted line represents the results under the same network topology but without packet loss. The scheme described in 4.2 is used when the dimension of the problem is larger than the number of agents, in which case we assign each agent the same number of rows. We focus on LAE problems with $\mathbf{A}$ as a diagonally dominant matrix. Fig. 7 indicates that the convergence steps approximates to $\mathcal{O}(n)$. In addition, compared to the performance when without packet loss, the case with packet loss shows that only a small number of extra iterations is needed to reach consensus, which suggests that the packet loss has no much impact on the consensus process in the simulated scenarios.

6.4 Performance Under LAN Testbed

Through row switching, a diagonally dominant matrix can be transformed to a matrix to which Jacobi method is not applicable. For example, a matrix which does not satisfy the convergence condition of Jacobi method can be constructed by shifting the rows of a diagonally dominant matrix circularly (i.e., the ith row is moved to the $(i-1)$th row while the 1st row is moved to the last). In this section, we show that our proposed algorithm works properly on matrices to which Jacobi method is not applicable, e.g., matrices with the aforementioned structure. We establish a LAN which consists of 5 computers to run the multiple-rows-per-agent version of our proposed algorithm. All computers are linked to a central hub. An LAE problem of size 5,000 $\times$ 5,000 is solved collaboratively by these computers. The total 5,000 rows are distributed evenly to the computers. At each iteration, each computer broadcasts the average of 1,000 local solutions, which is a 5,000-dimension vector. As the number of iterations increases, these vectors will converge to the exact solution of the LAE problem. To illustrate the consensus behavior of these 5 machines, we use the normalized error (normalized deviation from the exact solution) of one dimension of the solution and plot the consensus traces in Fig. 8. As shown in Fig. 8, the normalized errors diminish to zero as the number of iteration increases.

7 Discussions

7.1 Hiding Dimension of the LAE Problem

In order to fully hide the problem dimension, the client needs to have the capability to outsource a modified LAE

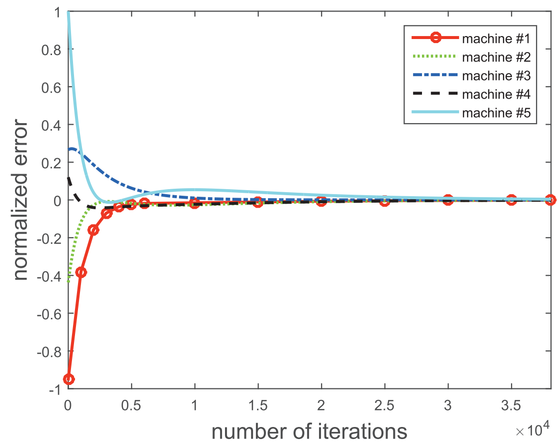


Fig. 8. Consensus traces.

with a different dimension and recover the right solution of the original LAE from the returned solution of the modified LAE. For example, to increase the size of the outsourced problem, the client can first augment the original problem by introducing a random $r\times r$ non-singular matrix $\mathbf{S}$ and an r-dimensional random vector $\mathbf{z}$ and obtain a modified LAE as follows:

$$\begin{pmatrix}\mathbf{A} & \mathbf{0}\\ \mathbf{0} & \mathbf{S}\end{pmatrix}\begin{pmatrix}\mathbf{x}\\ \mathbf{z}\end{pmatrix}=\begin{pmatrix}\mathbf{b}\\ \mathbf{Sz}\end{pmatrix},$$

where r is a random integer. With this augmented problem, the client employs the disguising scheme to mask the problem and then outsources the disguised problem to the ad hoc cloud.

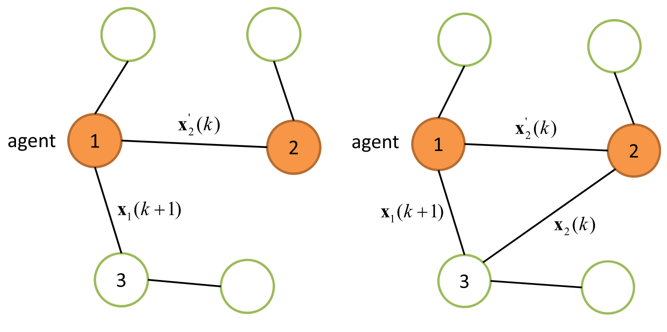
On the other hand, solving LAE problem through outsourcing another (or multiple) computation problem with a smaller dimension is a challenging issue in the distributed context. The work in [21] proposed a method to partition $\mathbf{A}$, which enables the client to solve $\mathbf{A}^{-1}$ by outsourcing several matrix inversion problems with dimensions smaller than n. As the LAE problem $\mathbf{Ax}=\mathbf{b}$ can be solved by computing inversion of $\mathbf{A}$ if $\mathbf{A}$ is non-singular, the partition method in [21] can be directly applied to decrease the dimension of the original LAE problem. To make such method applicable in the scenario of ad hoc cloud, distributed schemes for securely outsourcing matrix inversion and multiplication problem are required. We leave this to our future work.

7.2 Detecting Collusion Attack

So far, we have assumed that malicious agents do not collude. However, in some cases this assumption may not apply. For example, a powerful malicious agent may be able to hack one or multiple of its neighbors and take control of their computation and/or communications. In this section, we discuss possible extensions of our misbehavior detection scheme to deal with collusion attacks.

7.2.1 Collusion Attack Model

For ease of exposition, we use examples to illustrate how two adjacent malicious agents collude and sabotage the consensus-based algorithm. As shown in Fig. 9a, assume that agents 1 and 2 are malicious ones while agent 3 is an honest one. We are to show that the two malicious agents, if they



(a) Attack model (b) Attack detection

Fig. 9. Collusion attack and its detection.

collude, can escape the above proposed misbehavior detection scheme. At some iteration k, agent 2 broadcasts a correct update $\mathbf{x}_2(k)$ to its neighbors, while unicasts an incorrect update $\mathbf{x}'_2(k) \neq \mathbf{x}_2(k)$ to agent 1. Since they collude, agent 1 will move on to use $\mathbf{x}'_2(k)$ to compute its $\mathbf{x}_1(k+1)$ in the next iteration without reporting misbehavior of agent 2. The incorrectness of $\mathbf{x}_1(k+1)$ cannot be detected by agent 3, since agent 1 exactly follows the updating equation (4). Since agent 3 is unaware of the misbehavior of agent 2 and if agent 1 colludes with 2, agent 3 is unable to accuse agent 1. If agent 2 is not caught by its neighbors, it can continuously inject bad data to disturb the consensus process and finally cause the algorithm diverge or converge to a wrong value. Note that this type of attack cannot be done by agent 1 alone, because every message is digitally signed by the generating agent, and the attempt for agent 1 to forge an $\mathbf{x}'_2(k)$ will be detected by agent 3.

7.2.2 Collusion Attack Detection

For an agent to be able to detect the above attack, it has to be the common neighbor of a two colluding agents. For example, as shown in Fig. 9b, if agent 3 is the common neighbor of the colluding agents 1 and 2, it is able to find out that the incorrect $\mathbf{x}'_2(k)$ contained in $\Phi_1(k+1)$ does not match $\mathbf{x}_2(k)$ contained in the message $\Phi_2(k)$. Since messages are signed as mentioned above, agent 3 will notice that agent 1 forwards a wrong message without reporting misbehavior and agent 2 sends wrong messages, thus identifies that the two agents are colluding.

The collusion attack can only be detected by the common neighbor of these colluding agents; however, the common neighbors of a colluding pair can also be malicious ones. To be able to deal with this issue, the network topology has to satisfy that among the common neighbors of a colluding pair, the number of honest agents is larger than the number of malicious ones. We leave the detailed design of a collusion tolerance outsourcing scheme as our future work.

8 Conclusion

In this paper, we have proposed a secure outsourcing scheme for solving LAE problems in ad hoc clouds, which comprises of a robust distributed average consensus-based algorithm, a privacy preserving problem disguising technique, and a cooperative verification mechanism. The proposed scheme can protect the private information contained in the LAE problem parameters and solutions, and guarantee the correctness of the final solution. Performance analysis and numerical results have been provided to demonstrate that the proposed scheme is efficient in terms of computation complexity, and robust against a variety of malicious behaviors. For the future work, we will study the detection and mitigation mechanics for collusion attack, and how to further reduce the communication overhead of the proposed scheme.

Acknowledgments

This work was supported in part by the U.S. National Science Foundation under grants CNS-1117687 and CNS-1320736, the National Natural Science Foundation of China under grant 61573103, and State Key Laboratory of Synthetical Automation for Process Industries of China.

References

[1] M. Armbrust, et al., "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, 2010.
[2] M. Conti and S. Giordano, "Mobile ad hoc networking: Milestones, challenges, and new research directions," *IEEE Commun. Mag.*, vol. 52, no. 1, pp. 85–96, Jan. 2014.
[3] T. Verbelen, P. Simoens, F. De Turck, and B. Dhoedt, "Cloudlets: bringing the cloud to the mobile user," in *Proc. 3rd ACM Workshop Mobile Cloud Comput. Services*, 2012, pp. 29–36.
[4] M. Chen, Y. Hao, Y. Li, C.-F. Lai, and D. Wu, "On the computation offloading at ad hoc cloudlet: Architecture and service modes," *IEEE Commun. Mag.*, vol. 53, no. 6, pp. 18–24, Jan. 2015.
[5] F. Chi, X. Wang, W. Cai, and V. C. Leung, "Ad-hoc cloudlet based cooperative cloud gaming," IEEE Early Access Article, 2016. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7322203
[6] Y. Gong, C. Zhang, Y. Fang, and J. Sun, "Protecting location privacy for task allocation in ad hoc mobile cloud computing," *IEEE Trans. Emerging Topics Comput.*, 2016. IEEE Early Access Article, [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7296638
[7] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. Annu. ACM Symp. Theory Comput.*, 2009, pp. 169–169.
[8] J.-S. Coron, T. Lepoint, and M. Tibouchi, "Scale-invariant fully homomorphic encryption over the integers," in *Public-Key Cryptography*, Berlin, Germany: Springer, 2014, pp. 311–328.
[9] S. Fau, R. Sirdey, C. Fontaine, C. Aguilar-Melchor, and G. Gogniat, "Towards practical program execution over fully homomorphic encryption schemes," in *Proc. 8th Int. Conf. P2P, Parallel, Grid, Cloud Internet Comput.*, 2013, pp. 284–290.
[10] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," in *Proc. IEEE Symp. Secur. Privacy*, 2013, pp. 238–252.
[11] D. Fiore, R. Gennaro, and V. Pastro, "Efficiently verifiable computation on encrypted data," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2014, pp. 844–855.
[12] S. Mou, J. Liu, and A. S. Morse, "A distributed algorithm for solving a linear algebraic equation," *IEEE Trans. Autom. Control*, vol. 60, no. 11, pp. 2863–2878, Nov. 2015.
[13] 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. IEEE ICDCS*, 2011, pp. 549–558.
[14] C. Wang, K. Ren, and J. Wang, "Secure and practical outsourcing of linear programming in cloud computing," in *Proc. IEEE INFOCOM*, 2011, pp. 820–828.
[15] M. J. Atallah and J. Li, "Secure outsourcing of sequence comparisons," *Int. J. Inform. Securi.*, vol. 4, no. 4, pp. 277–287, 2005.
[16] M. Blanton and M. Aliasgari, "Secure outsourcing of DNA searching via finite automata," in *Data and Applications Security and Privacy XXIV*. Berlin, Germany: Springer, 2010, pp. 49–64.
[17] M. J. Atallah and K. B. Frikken, "Securely outsourcing linear algebra computations," in *Proc. ACM Symp. Inform., Comput. Commun. Secur.*, 2010, pp. 48–59.
[18] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[19] X. Lei, X. Liao, T. Huang, H. Li, and C. Hu, "Outsourcing large matrix inversion computation to a public cloud," *IEEE Trans. Cloud Comput.*, vol. 1, no. 1, pp. 1–1, Jan.-Jun. 2013.

[20] X. Lei, X. Liao, T. Huang, and H. Li, "Cloud computing service: The caseof large matrix determinant computation," *IEEE Trans. Serv. Comput.*, vol. 8, no. 5, pp. 688–700, Sep.-Oct. 2015.

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

[22] S. Salinas, C. Luo, X. Chen, and P. Li, "Efficient secure outsourcing of large-scale linear systems of equations," in *Proc. IEEE INFOCOM*, Apr. 2015, pp. 1035–1043.

[23] X. Chen, X. Huang, J. Li, J. Ma, W. Lou, and D. Wong, "New algorithms for secure outsourcing of large-scale systems of linear equations," *IEEE Trans. Inform. Forensics Secur.*, vol. 10, no. 1, pp. 69–78, Jan. 2015.

[24] T. Strohmer and R. Vershynin, "A randomized Kaczmarz algorithm with exponential convergence," *J. Fourier Anal. Appl.*, vol. 15, no. 2, pp. 262–278, 2009.

[25] A. Margaris, "Parallel implementation of the Jacobi linear algebraic system solver," in *Proc. 3rd Balkan Conf. Informat.*, 2007, pp. 161–172.

[26] J. He, J. Chen, P. Chen, and X. Cao, "Secure time synchronization in wireless sensor networks: A maximum consensus based approach," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 4, pp. 1055–1065, Apr. 2014.

[27] Z. Zhang and M.-Y. Chow, "Convergence analysis of the incremental cost consensus algorithm under different communication network topologies in a smart grid," *IEEE Trans. Power Syst.*, vol. 27, no. 4, pp. 1761–1768, Nov. 2012.

[28] M. Fiore, C. E. Casetti, C.-F. Chiasserini, and P. Papadimitratos, "Discovery and verification of neighbor positions in mobile ad hoc networks," *IEEE Trans. Mobile Comput.*, vol. 12, no. 2, pp. 289–303, Feb. 2013.

[29] H. J. LeBlanc, H. Zhang, X. Koutsoukos, and S. Sundaram, "Resilient asymptotic consensus in robust networks," *IEEE J. Sel. Areas Commun.*, vol. 31, no. 4, pp. 766–781, Apr. 2013.

[30] S. M. Dibaji and H. Ishii, "Consensus of second-order multi-agent systems in the presence of locally bounded faults," *Syst. Control Lett.*, vol. 79, pp. 23–29, 2015.

[31] H. Tang, F. R. Yu, M. Huang, and Z. Li, "Distributed consensus-based security mechanisms in cognitive radio mobile ad hoc networks," *IET Commun.*, vol. 6, no. 8, pp. 974–983, 2012.

[32] M. Kefayati, M. S. Talebi, B. H. Khalaj, and H. R. Rabiee, "Secure consensus averaging in sensor networks using random offsets," in *Proc. IEEE Int. Conf. Telecommun. Malaysia Int. Conf. Commun.*, 2007, pp. 556–560.

[33] F. Pasqualetti, A. Bicchi, and F. Bullo, "Distributed intrusion detection for secure consensus computations," in *46th IEEE Conf. Decision Control*, 2007, pp. 5594–5599.

[34] R. Yuster and U. Zwick, "Fast sparse matrix multiplication," *ACM Trans. Algorithms*, vol. 1, no. 1, pp. 2–13, 2005.

[35] A. Pyzara, B. Bylina, and J. Bylina, "The influence of a matrix condition number on iterative methods' convergence," in *Proc. IEEE Federated Conf. Comput. Sci. Inform. Syst.*, 2011, pp. 459–464.

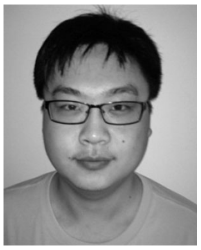
[36] K. Chen, *Matrix Preconditioning Techniques and Applications*. Cambridge, U.K.: Cambridge Univ. Press, 2005.

[37] M. J. Grote and T. Huckle, "Parallel preconditioning with sparse approximate inverses," *SIAM J. Scientific Comput.*, vol. 18, no. 3, pp. 838–853, 1997.

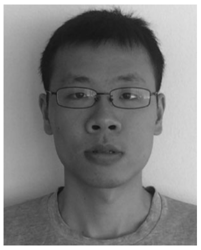
[38] M. Benzi, "Preconditioning techniques for large linear systems: a survey," *J. Comput. Phys.*, vol. 182, no. 2, pp. 418–477, 2002.

[39] P. Erdős and A. Rényi, "On the evolution of random graphs," *Publications Math. Inst. Hungarian Academy Sci.*, vol. 5, no. 1, pp. 17–61, 1960.

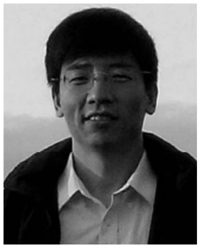
[40] Y. Saad, *Iterative Methods for Sparse Linear Systems*. New Delhi, India: SIAM, 2003.



Wenlong Shen (IEEE S'13) received the BE degree in electrical engineering from Beihang University, Beijing, China, in 2010, and the MS degree in Telecommunications from University of Maryland, College Park, Maryland, in 2012. He is currently working toward the PhD degree in the Department of Electrical and Computer Engineering, Illinois Institute of Technology, Chicago, Illinois. His research interests include vehicular ad hoc networks, mobile cloud computing, and network security.



Bo Yin (IEEE S'15) received the BEng and MEng degrees respectively in 2010 and 2013 from School of Electronic Information Engineering, Beihang University, China. Currently, he is working toward the PhD degree in the Department of Electrical and Computer Engineering, Illinois Institute of Technology. His research interests include mobile cloud computing and network security.



Xianghui Cao (IEEE S'08-M'11-SM'16) received the BS and PhD degrees in control science and engineering from Zhejiang University, Hangzhou, China, in 2006 and 2011, respectively. From December 2007 to June 2009, he was a visiting scholar in the Department of Computer Science, The University of Alabama, Tuscaloosa, Alabama. From July 2012 to July 2015, he was a senior research associate in the Department of Electrical and Computer Engineering, Illinois Institute of Technology, Chicago, Illinois. Currently he is an associate professor with School of Automation, Southeast University, Nanjing, China. His research interests include cyber-physical systems, wireless network performance analysis, wireless networked control and network security. He serves as Publicity Co-chair for ACM MobiHoc 2015, Symposium Co-chair for ICNC 2017 and IEEE/CIC ICCC 2015, and TPC member for a number of conferences. He also serves as an associate editor of several journals, including the *KSII Transactions on Internet and Information Systems*, the *Security and Communication Networks and International Journal of Ad Hoc and Ubiquitous Computing*. He was a recipient of the Best Paper Runner-Up Award from ACM MobiHoc 2014.

Yu Cheng (IEEE S'01-M'04-SM'09) received the BE and ME degrees in electronic engineering from Tsinghua University, Beijing, China, in 1995 and 1998, respectively, and the PhD degree in electrical and computer engineering from the University of Waterloo, Waterloo, Ontario, Canada, in 2003. From September 2004 to July 2006, he was a postdoctoral research fellow in the Department of Electrical and Computer Engineering, University of Toronto, Ontario, Canada. Since August 2006, he has been with the Department of Electrical and Computer Engineering, Illinois Institute of Technology, Chicago, Illinois, where he is now an associate professor. His research interests include next-generation Internet architectures and management, wireless network performance analysis, network security, and wireless/wireline interworking. He received a Best Paper Award from the conferences QShine 2007 and IEEE ICC 2011, and the Best Paper Runner-Up Award from ACM MobiHoc 2014. He received the National Science Foundation (NSF) CAREER AWARD in 2011 and IIT Sigma Xi Research Award in the junior faculty division in 2013. He served as a Co-Chair for the Wireless Networking Symposium of IEEE ICC 2009, a Co-Chair for the Communications QoS, Reliability, and Modeling Symposium of IEEE GLOBECOM 2011, a Co-Chair for the Signal Processing for Communications Symposium of IEEE ICC 2012, a co-chair for the Ad Hoc and Sensor Networking Symposium of IEEE GLOBECOM 2013, and a Technical Program Committee (TPC) Co-Chair for WASA 2011, ICNC 2015, and IEEE/CIC ICCC 2015. He is a founding Vice Chair of the IEEE ComSoc Technical Subcommittee on Green Communications and Computing. He is an associated editor for *IEEE Transactions on Vehicular Technology* and the New Books & Multimedia Column editor for *IEEE Network*. He is a senior member of the IEEE.

Xuemin (Sherman) Shen (IEEE M'97-SM'02-F'09) received the BSc (1982) degree from Dalian Maritime University, China and the MSc (1987) and PhD degrees (1990) from Rutgers University, New Jersey, all in electrical engineering. He is a professor and University Research Chair, Department of Electrical and Computer Engineering, University of Waterloo, Canada. He was the associate chair for Graduate Studies from 2004 to 2008. His research focuses on resource management in interconnected wireless/wired networks, wireless network security, social networks, smart grid, and vehicular ad hoc and sensor networks. He is an elected member of IEEE ComSoc Board of Governor, and the chair of Distinguished Lecturers Selection Committee. He served as the Technical Program Committee chair/co-chair for IEEE Infocom'14, IEEE VTC'10 Fall, the symposia chair for IEEE ICC'10, the Tutorial Chair for IEEE VTC'11 Spring and IEEE ICC'08, the Technical Program Committee Chair for IEEE Globecom'07, the General co-chair for ACM Mobihoc'15, Chinacom'07 and QShine'06, the chair for the *IEEE Communications Society Technical Committee on Wireless Communications*, and the *P2P Communications and Networking*. He also serves/served as the editor-in-chief for the *IEEE Network*, the *Peer-to-Peer Networking and Application*, and the *IET Communications*; a Founding area editor for the *IEEE Transactions on Wireless Communications*; an associate editor for the *IEEE Transactions on Vehicular Technology, Computer Networks*, and the *ACM/Wireless Networks*, etc.; and the guest editor for the *IEEE JSAC*, the *IEEE Wireless Communications*, the *IEEE Communications Magazine*, and the *ACM Mobile Networks and Applications*, etc. He received the Excellent Graduate Supervision Award in 2006, and the Outstanding Performance Award in 2004, 2007, 2010, and 2014 from the University of Waterloo, the Premier's Research Excellence Award (PREA) in 2003 from the Province of Ontario, Canada, and the Distinguished Performance Award in 2002 and 2007 from the Faculty of Engineering, University of Waterloo. Dr. Shen is a registered Professional Engineer of Ontario, Canada, an IEEE fellow, an Engineering Institute of Canada fellow, a Canadian Academy of Engineering Fellow, and a Distinguished Lecturer of IEEE Vehicular Technology Society and Communications Society.