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

Abstract—The emerging ad hoc clouds form a new cloud computing paradigm by leveraging untapped local computation and storage resources. An important application over ad hoc clouds is outsourcing computationally intensive problems to nearby cloud agents to solve in a distributed manner. A risk with ad hoc clouds is however the potential cyber attacks, with the security and privacy in distributed outsourcing being a significant challenging issue. In this paper, we consider distributed secure outsourcing of linear algebraic equations (LAE), one of the most frequently used mathematical tool, in ad hoc clouds. The outsourcing client assigns each agent a subproblem; all involved agents then apply a consensus based algorithm to obtain the correct solution in a distributed and iterative manner. We identify a number of security risks in this process, and propose a secure outsourcing scheme which can not only preserve privacy to shield the original LAE parameters and the final solution from the computing agents, but also detect misbehavior based on mutual verifications in a real-time manner. We rigorously prove that the proposed scheme converges to the correct solution of the LAE exponentially fast, has low computation complexity at each agent, and is robust against the identified security attacks. Extensive numerical results are presented to demonstrate the effectiveness of the proposed method.

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

I. INTRODUCTION

Cloud Computing is a revolutionary computing paradigm that can provision users a wide variety of resources, such as computational power, data storage, platform, and software, as a service over the network [1], [2]. It enables resource limited customers to conduct originally impossible complex missions by outsourcing the workload into the cloud. Ad hoc cloud computing, as a neoteric pattern of cloud computing, leverages untapped local computing and storage resources to form a local resource pool [3], [4]. Such an infrastructure will significantly improve the resource utilization of local devices, while providing benefits of conventional client-server cloud computing model over existing heterogeneous hardware.

Outsourcing a computationally intensive problem to the cloud is one major application with cloud computing. While cloud computing brings significant benefits, it also raises a higher risk of security issues than local computation. Since the client has little control over the outsourced data and the behavior of remote cloud entity, it is reasonable for the client to question the trustworthiness of the cloud service [5]. In fact, a cloud service provider may have various motivations to behave dishonestly. For example, the cloud service provider may perform slothfully to save computational resource or power. There also exists the possibility that a cloud service provider is compromised by other malicious attackers, thus intentionally misleading the client to a false computation result. In addition to the integrity of the result, the privacy issue to protect the sensitive information contained in the outsourced problem and the solution of the problem also needs our attention.

Secure outsourcing has been a hot research topic recently [6]–[9]. A secure outsourcing scheme should satisfy three conditions: 1) correctness of the result returned from the cloud can be verified; 2) the private information contained in the problem can be well hidden from the cloud server or ad hoc computing agents; 3) the local computational complexity, including the result verification algorithm and the necessary encryption algorithm for privacy reserving, should not exceed the computational complexity of the original problem.

The linear algebraic equation (LAE) problem is one of the most frequently used mathematical tools for a large variety of real-world engineering and scientific computations. While most of the secure outsourcing works consider outsourcing problems to one or several powerful cloud servers with unlimited computational resources [6], [7], [10], [11], in this paper, we study outsourcing an LAE problem to an ad hoc cloud network comprises of a large number of agents, but each with limited
computation power. While there are already a few studies on distributed computing to solve the LAE problem \[12\], \[13\], the security issues were not considered. In the literature, security issues in distributed outsourcing have not been explicitly studied yet, to the best of our knowledge.

In this paper, we develop a secure outsourcing scheme for solving the LAE problem via a distributed ad hoc cloud network. We first systematically analyze the possible attack models and their effects; such an analysis guide our design of secure outsourcing scheme. Specifically, our basic computing model is based on the consensus based algorithm presented in \[14\], but we design a robust version of the algorithm. Our algorithm can still ensure correct final solution under false updates from malicious agents, as long as the number of false updates is finite. We then apply the robust distributed algorithm to build a secure and practical outsourcing scheme for the LAE problem. The proposed scheme shields the sensitive information contained in the problem parameters by adding random noises. We further design a cooperative verification mechanism for misbehavior detection in the distributed computation. At each iteration step, each agent will double check the update message from a neighbor agent with a probability \(p\). We prove that the cooperative verification algorithm can achieve a misbehavior detection probability close to 1 over our robust distributed algorithm. Moreover, we analyze the computation complexity of the proposed secure outsourcing scheme and prove that the computation cost incurred to a computing agent in the ad hoc cloud is order-level lower than the centralized computation.

Our main contributions can be summarized as follows.

1) We design a robust distributed algorithm for solving LAE, which can ensure the correct solution under a finite number of false updates from misbehaving cloud agents.

2) Based on the robust distributed algorithm, we design a secure outsourcing scheme for LAE in the ad hoc cloud environment, with privacy preserving and misbehavior detection functions.

3) We conduct theoretical analysis of the convergency of the distributed algorithm, the security performance, and the computation complexity.

4) We present extensive numerical results to demonstrate the robust security performance and low computation overhead of the distributed outsourcing scheme, with comparison to centralized computing.

The remainder of this paper is organized as follows. Section II reviews the related work. Section III describes the system model and preliminaries on the distributed algorithm for solving the LAE problem. Section IV presents the attack models and our design goals. Section V presents the detailed secure outsourcing scheme. Section VI presents theoretical analysis of the convergency, security performance, and computation complexity. Section VII presents the numerical results and Section VIII gives further discussions on some related research issues and Section IX concludes this paper.

II. RELATED WORK

A. Ad Hoc Clouds

The revolutionary cloud computing paradigm provides novel service models such as software as a service, platform as a service and infrastructure as a service. As a typical application scenario, offloading expensive computations (e.g., solving large-scale LAE problems) from locally resource-constrained devices to remote cloud servers with high computation capacities has gained interests from both academia and industry, e.g., Google and Amazon provide commercial cloud services \[15\]. However, such cloud computing models require a cluster of expensive and dedicated servers which demand considerable investment and energy. Alternatively, the emerging ad hoc cloud comprise of only a number of resource-constrained devices. It offers attractive services by making use of untapped local computing and storage resources. The ad hoc cloud can be formed by static and/or mobile devices (e.g., smartphones, PDAs) where their spare resources can be used either voluntary or paid. Still evolving, the ad hoc cloud has the potential to gain a great market share. A recent study show that nearly 57,000 UK organizations are potential ad hoc cloud customers \[16\].

In the literature, several related studies have been conducted recently on the architecture, resource allocation and applications of ad hoc clouds \[16\]–\[18\]. For example, studies have shown promising of establishing an ad hoc cloud based on mobile robots for disaster recovery \[19\]. There are very few studies investigating the problem of outsourcing expensive computations to an ad hoc cloud where each participating agent only has limited computation and storage resources. In \[20\], an ad hoc cloud is built over vehicular networks to support that individual vehicles can offload computations to others or the roadside units. Whereas, they focus on general purpose computations, implicitly presuming that the offloading computations have been well-decomposed
into separative tasks. However, outsourcing the LAE problem is more complicated since the problem is hardly decomposed into separate components; it rather requires the participating agents to work collaboratively and the data to be exchanged among them should be carefully designed.

B. Secure Outsourcing in Clouds

Recently, researchers have made steady progress in securing the processes of outsourcing some typical computationally intensive problems such as large-scale linear equations [6], linear programming [7], sequence comparisons [10] and DNA searching [11]. For example, in [9], the authors propose a protocol for secure and private outsourcing of linear algebra computations, especially the problem of multiplying large-scale matrices, to either one or two remote server(s). Their approach is based on the secret sharing scheme proposed in [8], without carrying out expensive cryptographic computations. The problem of secure outsourcing of linear programming (LP) is investigated in [7]. With their approach, malicious behavior can be detected by a computation result verification mechanism where the properties of the dual of the original LP problem is exploited; while the problem confidentiality is preserved by using random matrix and vectors to hide the problem information. Considering a similar problem as studied in this paper, the work in [6] proposed a secure scheme for outsourcing 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. However, they focused on outsourcing this problem to a single (strong) remote cloud server, which is essentially a centralized scheme. Whereas, in this paper, we consider a different application scenario—outsourcing the LAE problem to an ad hoc cloud with the participating agents solving the problem in a completely distributed manner.

C. Solving Large-Scale LAE Problems

Solving LAE problems has received considerable attention in a long history and still remains challenging when the problem size is large. It is well known that direct solutions such as Gaussian elimination and LU factorisation are not well-scalable. Alternative methods featuring iteratively approximating the correct solution with parallel processing capabilities have become the mainstream approach. Among them are Jacobi, Gauss-Seidel, Conjugate Gradient and Markov chain steady-state methods [21], [22]. However, these algorithms either assume special structure of the matrix \( A \) (e.g., sparsity or diagonal-dominance) or require \textit{a priori} estimation of parameters related to matrix \( A \), referring to the surveys provided in [23], [24]. In [12], Lu et al. proposed a distributed approach called subset equalizing for solving LAE (in a different form) using a network of agents, whereby they assume that each agent observes a well-decomposed component of the original problem. However, to apply their method in our case, the outsourcer agent should perform, in its initial step, an expensive decomposition, analogous to the MapReduce procedure [25], in order to dispatch the obtained components to each of the participating agent in the ad hoc cloud. Whereas, the initial step only requires simple matrix manipulations with relatively low complexity.

Recently with the advances in consensus algorithms [26]–[28], a consensus based distributed solution to the LAE has been propose in [13], [14]. Instead of performing problem decomposition, the algorithm assigns each agent one (or possibly multiple) row of \( A \) and \( b \), say \( A_i \) and \( b_i \), respectively. Each agent starts with a feasible solution to the subproblem \( A_i x = b_i \). 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. It has been shown in [13] that, if the network of agents is repeatedly jointly strongly connected over time, the convergence is guaranteed and the correct solution \( A^{-1} b \) 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 susceptible to several malicious attacks ranging from sensitive data probing and disobeying the updating rule during the algorithm running time. With potentially many attack strategies, an adversary can manipulate the final results and even cause the whole algorithm diverge. To the authors’ best knowledge, the security issues of distributed outsourcing problem for solving LAE problems have not been studied.

III. PROBLEM STATEMENT AND PRELIMINARIES

A. Problem Statement

We study the outsourcing problem in an ad hoc cloud which comprises of multiple agent nodes (e.g., static desktops, mobile phones and PDAs), as illustrated in Fig. 1. Each agent is capable of conducting a certain amount of computations within its computational resource limit.
The connection between agents can either be wired or wireless. One of these agents, called the client in the following, is interested in solving a large-scale LAE problem of the form $Ax = b$, which, with respect to the limited computation and storage resources of its own, is computationally intensive or even infeasible for obtaining the solution locally. Notice that solving this problem directly takes time $O(n^3)$, where $n$ is the dimension of the unknown variable $x$. The client dispatch the problem 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.

Based on the distributed algorithm proposed in [13], a solution to the outsourced LAE problem can be eventually reached via collaboration by all the participating agents in the ad hoc cloud. However in most real world application scenarios, there potentially exist malicious agents, aiming to break down the distributed problem solving process by a variety of means. 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. Besides, the parameters and results of the outsourced problem may contain highly sensitive information that the client is not willing to share with other agents.

In order to achieve secure outsourcing of the LAE problem under the aforementioned system model, the outsourcing scheme should have the following properties.

- **Privacy Preserving**: Participating agents, during collaborating with others for solving the outsourced problem, cannot infer client’s sensitive information contained in the problem, i.e., information about the problem parameters $A, b$ and the solution.

- **Misbehavior Detection**: Misbehaving agents can be detected with high probability during the problem solving runtime. The final solution achieved by the agents can be validated.

- **Low complexity**: To ensure practicalness and efficiency, the computation burden on each participating agent, as well as the client, should be kept below $O(n^3)$, i.e., the computation complexity should be less than that of solving the original LAE problem by the client itself.

### B. Preliminaries on Distributively Solving LAE

In order to allow other agents cooperatively to solve the LAE problem, a distributed algorithm which decomposes the LAE problem into smaller subproblems is the foundation of our scheme. Efforts to develop such distributed algorithms have been made by many researchers. In this paper, we build our secure outsourcing scheme based on a consensus-based distributed algorithm proposed by S. Mou in [14]. Compared to other algorithms for solving LAE problem, the aforementioned algorithm has no special requirement on $A$, and achieves an exponential convergence rate. The key idea of Mou’s algorithm is summarized as follows.

We use boldface letters to represent column vectors and matrices. $A_i^T$ denotes the $i$th row of $A$. An LAE problem can be written as $Ax = b$, where $A$ is a $n \times n$ non-singular matrix and $b$ is a $n \times 1$ vector. Let $x^*$ be the exact solution of this problem. Let $[A_i^T \ b_i]$ be a distinct row of the partitioned matrix $[A \ b]$. Assume there are $n$ agents which form a network underlying a connected graph. Each agent is allocated one distinct row and the agent who receives the $i$th row is denoted as agent $i$. At the beginning each agent picks an initial guess of $x^*$, denoted as $x_i(0), i \in \{1, \ldots, n\}$, satisfying $A_i^T x_i(0) = b_i$ where $A_i^T$ is the received row. Let $K_i$ be a matrix whose column span is the kernel of $A_i^T$ such that $A_i^T K_i = 0$. Each agent iteratively updates its guess following an updating rule of the form $x_i(t + 1) = x_i(t) + K_i u_i(t)$ so that, within the iteration process, the approximate solution of each agent always satisfies $A_i^T x_i(0) = b_i$. In order to guarantee the convergence, $u_i(t)$ is chosen as the least square solution to $x_i(t) + K_i u_i(t) = \frac{1}{d_i} \left( \sum_{j \in \mathcal{N}_i} x_j(t) \right)$, where $\mathcal{N}_i$ denotes the set of neighbors of agent $i$ and $d_i$ is the number of neighbors of agent $i$. The update process can be expressed as

$$x_i(t + 1) = x_i(t) - \frac{1}{d_i} P_i(d_i x_i(t) - \sum_{j \in \mathcal{N}_i} x_j(t)) \quad (1)$$

where $P_i = K_i (K_i^T K_i)^{-1} K_i^T$ is the orthogonal projection on the kernel of $A_i$.  

![Fig. 1. Scheme overview.](image-url)
IV. ATTACK MODEL

While outsourcing the LAE problem to the cloud using the aforementioned distributed average consensus based algorithm, the client has little control over other agents participating the algorithm. Without proper outsourcing protocol and 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 misbehaviors possibly conducted by malicious agents, and how these attacks affect the final computation result.

Depending on their purposes, we categorize the behavior of malicious agents into three categories. Malicious agents participating the distributed algorithm may either probe the sensitive information contained in the problem parameters and results, manipulate the computing results, or diverge the algorithm consensus.

- Probing sensitive information: In the preliminary distributed average consensus-based algorithm, each participating agent is assigned with a row of $A$ and the corresponding entry of $b$. This setup process reveals part of information of $A$ and $b$. Besides, malicious agents may collude with each other to obtain more sensitive information. After the consensus is finally reached, all the participating agents will obtain the solution of the outsourced LAE problem, which is highly undesirable. So it is necessary for the outsource client to encrypt the original problem before outsourcing it to the cloud. The encryption should be able to hide the original problem parameters as well as the solution to the outsourced problem.

- Manipulating the results: A malicious agent can mislead the algorithm to a false result by injecting a false update during any iteration of the algorithm, then perform normal updating afterwards. For example, at iteration $k$, malicious agent $i$ sends out a false update $x_i(k)$, which results in $A_i^T x_i(k) = b_i' 
eq b_i$. In the subsequent iterations, according to the update rule in (1), $A_i^T x_i(l) = A_i^T x_i(k) = b_i'$ for $l > k$. The algorithm will finally converge to a false result $x'$, which is the solution to the linear equations $[A, b']$ where $b' = [b_1, b_2, \ldots, b_i', \ldots, b_n]^T$. Thus, by injecting a false update, a malicious agent can mislead the distributed algorithm to an incorrect solution.

- Diverging the algorithm consensus: Without proper defense mechanism, a malicious agent can easily diverge the consensus algorithm by updating random $x_i(t)$ in each iteration, thus launching a denial of service attack. If the malicious agent keep doing this, obviously the distributed consensus algorithm will not converge. A straightforward solution to prevent such attack is during the algorithm setup stage, distribute $A_i^T$ and $b_i$ not only to agent $i$, but also to its neighbor agents, such that neighboring agents can verify each other’s updating value per step by checking whether $A_i^T x_i(t) = b_i$. A randomly chosen $x_i(t)$ by malicious agents will not be likely to satisfy this checking equation and will get detected. However, a “smart” enough malicious agent can still break the convergence of the algorithm by choosing $x_i(t)$ for each iteration within the solution space of $A_i^T x_i(t) = b_i$, for example, resending the initial guess repeatedly. Thus simply checking $A_i^T x_i(t)$ will not prevent the malicious agent from diverging the algorithm consensus. Stronger verification conditions are required.

V. THE PROPOSED SCHEME

In this section, we present our design of a secure outsourcing scheme for solving LAE problem. Our scheme includes three basic components: a robust consensus based algorithm, a problem encryption algorithm, and a cooperative verification and misbehavior detection mechanism. First of all, we clarify some general assumptions about the problem we consider and the multi-agent ad hoc cloud network in which the outsourcing client lies.

We consider solving the LAE problem $Ax = b$, $A \in \mathbb{R}^{n \times n}$ and is non-singular, $b \in \mathbb{R}^n$. For the multi-agent ad hoc cloud network, we envision that each agent in the ad hoc cloud has a unique identity number, e.g., the agents are indexed by $1, 2, \ldots, n$, respectively. We consider a simplified network topology that any pair of neighbor agents has at least one common neighbor, and each message in the network will be authenticated such that one can only read or copy other agents’ message without modification. We further assume that the malicious agents have a sparse distribution, which implies any two attackers are never neighboring with each other. These assumptions are not harsh, since many research papers have made similar assumptions (e.g. see
We also assume that the communications between agents is reliable.

A. Robust Consensus Based Algorithm

Equation (1) provides a collaborative framework for autonomous agents to solve a large-scale LAE problem. However, through our previous analysis on the attack modes of malicious agents, the updating rule in (1) is vulnerable to false messages. Besides, it is hard for single agent to carry out this update process in terms of computation cost and storage requirement. Note that computation overhead and storage requirement could be very high when the problem dimension  is very large. The straightforward calculation of  as involves matrix-matrix multiplications which have  time complexity. In addition,  incurs matrix-vector multiplications which take time  . Storing a large-scale matrix is also a heavy burden for some storage-constrained devices. We are motivated to design a robust and efficient version of the preliminary distributed consensus algorithm.

Observing that  is the orthogonal projection on the kernel of  it can be calculated as

\[ P_i = I - P_{A_i^T} = I - \frac{A_i A_i^T}{A_i^T A_i} \]  (2)

where  is the orthogonal projection on  and can be calculated with time complexity  .

With respect to the storage cost, one straightforward approach is calculating  on-the-fly. However, since  is invariant through the consensus process, calculating it repeatedly is waste of computation. Let \( x_i(t) = \frac{1}{d_i} \sum_{j \in N_i} x_j(t) \) denote the average value of agent  ’s neighbors’ updates. Substituting (2) into (1), the update process can be expressed as

\[
x_i(t+1) = P_{A_i^T} x_i(t) + \bar{x}_i(t) - P_{A_i^T} \bar{x}_i(t)
= A_i A_i^T x_i(t) - \frac{A_i A_i^T}{A_i^T A_i} x_i(t) + \bar{x}_i(t)
= \frac{A_i^T}{\|A_i^T\|^2} A_i - \frac{A_i^T}{\|A_i^T\|^2} A_i + \bar{x}_i(t)
= \frac{b_i}{\|A_i^T\|^2} A_i - \frac{A_i^T}{\|A_i^T\|^2} A_i + \bar{x}_i(t) \quad (3)
\]

where  represents the two-norm of a vector. For convenience of reference, our proposed consensus based algorithm is manifested as

\[ x_i(t+1) = \frac{b_i}{\|A_i^T\|^2} A_i - \frac{A_i^T}{\|A_i^T\|^2} A_i + \bar{x}_i(t) \quad (4) \]

If the initial solution  satisfies  according to (4), we can see that at each update it always maintains  . That is

\[
A_i^T x_i(t+1) = A_i^T \left( \frac{b_i}{\|A_i^T\|^2} A_i - \frac{A_i^T}{\|A_i^T\|^2} A_i + \bar{x}_i(t) \right)
= b_i + A_i^T \bar{x}_i(t) + \frac{A_i^T}{\|A_i^T\|^2} A_i + \bar{x}_i(t)
= b_i - A_i^T \bar{x}_i(t) + \bar{x}_i(t)
= b_i
\]

**Lemma 1.** In each iteration of the proposed robust consensus based algorithm, the computational cost for each agent is  .

Lemma 1 directly follows the algorithm in (4). In (4), the first term of the right side is a fixed constant. Thus, each update only needs to compute the second and third terms of the right side, which can be computed with time complexity  . Besides, each agent only needs to store his own  , which is a  vector.

If an agent doesn’t update its local solution according to the algorithm in (4) (e. g., node  arbitrarily broadcasting an  to its neighbors and  , we term that this node conducts a false update. The proposed algorithm in (4) has a robust performance against the false update, as presented in Theorem 2.

**Theorem 2.** The consensus based algorithm in (4) has robust performance in preventing malicious agents from manipulating the final consensus result through finite times of false updates.

**Proof:** According to (4), the next state of each agent is only determined by the current states of its neighbors. Thus, for the malicious agents who conduct one-time false update, their next update would satisfy  , where  . Since  and  , for normal agents, equation (4) holds regardless of the updates of their neighbors. Therefore, the approximate solutions of all agents will satisfy (4) in their next step. Since they will update correctly afterward, their approximate solutions will converge to the right solution. Because one-time false update of malicious agents cannot lead any agent to violate (4) in the next update, finite times of false update cannot mislead normal agents to converge to an incorrect solution.
B. Problem Encryption

In order to keep confidential the LAE parameters $A$, $b$, and the computation result $x$, the client needs to encrypt the problem before outsourcing it to the cloud. We need to take the computational complexity of the encryption algorithm into consideration, since any computation at the complexity level of $O(n^3)$ required by the client will demotivate the whole outsourcing scheme.

First we consider introducing a random noise $\Delta x$ to hide the solution $x^*$ of the original problem $Ax = b$:

$$A(x^* + \Delta x) = Ax^* + A\Delta x = b + A\Delta x$$

The client generates an $n$ dimensional random vector $\Delta x$, and computes $\Delta b = A\Delta x$. Then the problem to be outsourced transforms into $Ax = b + \Delta b$. After the cloud returns the computation result $x'$ of the transformed problem, the client can derive the solution to the original problem by computing $x' - \Delta x$. The random vector $\Delta x$ is only kept locally with the client, so the participating cloud agents have no information of $x^*$ except its dimension.

Now we consider hiding the problem parameters $A$ and $b$. One straightforward method is to generate a random non-singular $n \times n$ matrix $Q$, and outsource the problem $A'x = b'$, where $A' = QA$ and $b' = Q(b + \Delta b) = Qb + QA\Delta x$. However, such transformation requires the computation of $QA$ with complexity $O(n^3)$, which is unacceptable. So we resort to other method to transform the LAE problem, while maintaining the solution unchanged.

Noticing that performing the same elementary row operations simultaneously on $A$ and $b$ will transform the LAE problem to a new one with the same solution. Elementary row operations include row switching, row multiplication and row addition. Here row switching does not benefit, since in the linear equations problem, usually the index of each single equation does not carry much useful information. To perform row multiplication, the client can multiply each row of the problem by a random number. To perform row addition, for each row the client adds another random chosen row to it. We summarize the key generation and problem encryption process in Algorithm 1 and Algorithm 2, where $Z$ is the set of positive integers.

Algorithm 1 generates the secure key used for the problem encryption. A secure key parameter $\lambda$ will be used for determine the scope for generating random numbers. Through Algorithm 1 the client obtains $\Delta x$ for hiding the solution, an $n$ dimensional vector $\Lambda$ as row multiplier, and an $n$ dimensional vector $\kappa$ as index for row addition. After obtaining the encryption keys, the client encrypts the problem according to Algorithm 2. First, the client transforms $b$ to $b + A\Delta x$. Then, the client uses $\Lambda$ for row multiplication and $\kappa$ as the index for row addition.

Algorithm 1: Encryption Key Generation

**INPUT:** Secure key parameter $\lambda$;
**OUTPUT:** Secure key: $\Delta x$, $\Lambda$, $\kappa$;
**begin**
- generate a random $\Delta x \in \mathbb{R}^{n \times 1}$;
- generate a random $\Lambda \in \mathbb{R}^{n \times 1}$;
- generate $\kappa \in \mathbb{Z}^{n \times 1}$, with $1 \leq \kappa_i \leq n$;
**return** $\Delta x$, $\Lambda$, $\kappa$;

Algorithm 2: Problem Encryption

**INPUT:** $A$, $b$, $\Delta x$, $\Lambda$, $\kappa$;
**OUTPUT:** $A'$, $b'$;
**begin**
- compute $b' = b + A\Delta x$;
  - for $i = 1 : n$ do
    - compute $A'_i = A_i + \Lambda_i \kappa_i$;
    - compute $b'_i = b'_i + \kappa_i$
  - end
**end**
**return** $A'$, $b'$;

Lemma 3. The computational complexity for the client to encrypt the problem is $O(n^2)$.

**Proof:** Through the key generation algorithm the client generates $3n$ random numbers, which is with complexity $O(n)$. The encryption algorithm involves computing $b' = b + A\Delta x$ with complexity $O(n^2)$, and $2n$ elementary row operations, each of which is with complexity $O(n)$. So the total computation complexity for the client is $O(n^2)$.

After running Algorithm 1 and Algorithm 2 the client outsources the encrypted problem $A'x = b'$ to the cloud.
Upon receiving the returned result $x$, the solution to the original problem can be derived by $x^* = x - \Delta x$.

C. Misbehavior Detection

In Section III, we have analyzed the possible misbehaviors taken by malicious agents. A simple updating verification of $A_i^T x_i(k) = b_i$ can only detect malicious agents who update random value in each iteration. For those smart enough malicious agents, a stronger detection condition is needed. An effective approach is to let the ad hoc agents monitor their neighbors’ updates by double checking whether their computation is according to (4) using the information of $F$ where its neighbor $i$ incurs workload $O(d_i n)$. If agent $j$ monitors all its neighbors, the total workload will be $O(\sum d_i^2 d_i n)$. Under a well-connected network topology, for example, a completely network graph with $d_i = n - 1$ for any agent $i$, in a single iteration the verification computational overhead for each monitoring agent is $O(n^3)$, resulting the total computation cost exceeding $O(n^3)$.

To deal with the computation overhead issue, we further set a verification probability $p$: at each iteration, agent $j$ will verify received message with probability $p$. For this probabilistic verification scheme, every agent needs to keep its neighbors’ broadcast information till the end of next iteration: suppose at $(k + 1)$th iteration, agent $j$ receives an alarm message $F_{m,k} = i$ denoting his neighbor agent $i$ performed a false update at $k$th iteration, then agent $j$ will double check $x_i(k)$ with probability 1, and this requires 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 3.

**Theorem 4.** With the proposed misbehavior detection scheme, the probability that a malicious agent sabotages the algorithm without being detected is close to 0.

**Proof:** By Theorem 2 a malicious agent can only sabotage the algorithm by infinitely updating false messages. Suppose agent $i$ intends to sabotage the algorithm, upon each false update, $d_i$ agents will verify its update by probability $p$. The probability that malicious agent $i$ successfully injects $m$ false updates without being caught is $P_m = ((1 - p)^d_i)^m$. As $m$ goes infinity, $P_m$ is close to 0.

D. Main Scheme

We have introduced our proposed robust distributed average consensus based algorithm, privacy preserving problem encryption scheme, and malicious agents misbehavior detection mechanism. Next we will present an integrated scheme for distributed secure outsourcing of LAE problem in a systematic way.

- **Setup Stage:** In this very first phase, the client generates the secure key and performs problem encryption by Algorithm 1 and Algorithm 2. After the problem encryption phase, the client distributes transformed problem parameters $A$ and $b$ to corresponding agents according with the following
convergence of our consensus based algorithm is an
only utilizing the information from its neighbors, the
A. Convergence Analysis

Algorithm 3: Verification process of agent i at iteration k + 1

INPUT: \( \Phi_i(k + 1), \Phi_i(k), A_j, b_j \forall j \in \mathcal{N}_i \); p ;
OUTPUT: \( F_{i,k}, F_{i,k+1} \);
begin:
for \( j \in \mathcal{N}_i \) do
if \( F_{j,k} \neq 0 \) and \( F_{j,k} \in \mathcal{N}_i \) then
compute \( x'_j(k) \) by \( \Phi_i(k) \);
if \( x'_j(k) \neq x_j(k) \) then
\( F_{j,k} = F_{j,k} \)
end
end
randomly choose \( m \) from \((0, 1)\);
if \( m < p \) then
compute \( x'_j(k+1) \);
if \( x'_j(k+1) \neq x_j(k+1) \) then
\( F_{i,k+1} = j \)
end
end
return: \( F_{i,k}, F_{i,k+1} \);

rule: to agent \( i \), the client distributes \( A_i^T, b_i \), and \( \{A_i^T, b_i, \forall j \in \mathcal{N}_i\} \).

- Distributed Computation Stage: At iteration 0, for \( i \in [1, n] \), agent \( i \) picks one initial solution \( x_i(0) \) which satisfies \( A_i^T x_i(0) = b_i \). At iteration \( k + 1, k \geq 0 \), agent \( i \) performs the cooperative verification algorithm Algorithm 3, then computes \( x_i(k+1) \) by \( \Phi_i(k) \), finishes iteration \( k \) by broadcasting its updating message \( \Phi_i(k+1) \) to its neighbors. To determine the termination, each agent checks if

\[
\max_{j \in \mathcal{N}_i} |x_i - x_j|_\infty \leq \varepsilon
\]  

holds for consecutive \( 2L_i \) step, where \(| \cdot |_\infty\) represents the maximum-norm of a vector and \( L_i \) is the eccentricity of agent \( i \) in the underlying graph.

- Final Solution Stage: The distributed average consensus algorithm will achieve a final convergent point \( \bar{x} \), which is the solution of the transformed problem. The client derives the solution to the original problem by computing \( x^* = \bar{x} - \Delta x \).

VI. Performance Analysis

A. Convergence Analysis

Since every agent updates its approximate solution only utilizing the information from its neighbors, the convergence of our consensus based algorithm is an essential concern of our work. The approximate solutions of all the agents need to not only reach a consensus but also converge to the exact solution to the LAE. Furthermore, the convergence rate would be one of the factors which influence the computational efficiency of our scheme. Here, we give a concise proof to demonstrate the correctness of our scheme, which also provides some insights on the convergence rate.

Let \( x^* \) be the exact solution of the LAE. For ease of presentation, the update rule we employ here is in form of \( x_i(t + 1) = P A_i x_i(t) + x_i(t) - P A_i x_i(t) \). Then the error of agent \( i \) after \( k + 1 \) iterations, denoted by \( e_i(k+1) \), can be expressed as

\[
e_i(k+1) = x_i(k+1) - x^*
= P A_i x_i(k) + P_i \frac{\sum_{j \in \mathcal{N}_i} x_j(k)}{d_i} - (P A_i + P_i) x^*
= P_i \frac{\sum_{j \in \mathcal{N}_i} (x_j(k) - x^*)}{d_i}
= P_i \frac{\sum_{j \in \mathcal{N}_i} e_j(k)}{d_i}
\]  

(7)

Let \( m \) be the adjacency matrix corresponding to the underlying graph of the network. Then \( g = \text{diag}(\frac{1}{d_1}, \ldots, \frac{1}{d_n}) m \) is a stochastic matrix corresponding to \( m \). In order to present the error of all agents, concatenate \( e_1, \ldots, e_n \) and let \( C(k+1) = [(e_1(k+1))^T, (e_2(k+1))^T, \ldots, (e_n(k+1))^T]^T \) represent the error of all agents after \( k + 1 \) iterations. Then, according to (7),

\[
C(k+1) = PG C(k)
\]  

(8)

where

\[
P = \begin{pmatrix}
P_1 & \cdots & 0 \\
0 & \ddots & \vdots \\
0 & \cdots & P_n
\end{pmatrix}
\]  

\( \otimes \) denotes the Kronecker product. \( P \in \mathbb{R}^{n^2 \times n^2} \) is a block diagonal matrix which is determined by the matrix \( A \) in the LAE while \( G \in \mathbb{R}^{n \times n^2} \) is characterized by the adjacency of the network. The convergence rate of this algorithm is determined by the spectral radius of the iteration matrix, denoted by \( \rho(PG) \).

Lemma 5. \( \rho(PG) = 1 \) if and only if there exits a vector \( v \in \mathbb{R}^{n} \) such that \( v \) is the eigenvector of every block \( P_i \) in \( P \) corresponding to eigenvalue 1.
Proof: Necessity. If \( \rho(PG) = 1 \), then there exists a vector \( \mathbf{w} \in \mathbb{R}^n \) such that \( P \mathbf{w} = \mathbf{w} \) and \( G \mathbf{w} = \mathbf{w} \). We first consider \( G \mathbf{w} = \mathbf{w} \). If \( [1, \ldots, 1]^T \) is the unique eigenvector of \( G \) corresponding to eigenvalue 1 because the network graph is connected. Since \( G = g \otimes I \), \( \mathbf{w} \) should be in the form of \( \left[ \mathbf{v}_1^T, \ldots, \mathbf{v}_n^T \right]^T \) where \( \mathbf{v} \in \mathbb{R}^n \) is an arbitrary vector. Consider \( P \mathbf{w} = \mathbf{w} \), for every block \( P_i \) in \( P \), \( P_i \mathbf{v} = \mathbf{v} \) must satisfy.

Sufficiency. If there exits a vector \( \mathbf{v} \in \mathbb{R}^n \) such that \( P_i \mathbf{v} = \mathbf{v} \) for every block \( P_i \) in \( P \), then we can construct a vector \( \mathbf{w} = \left[ \mathbf{v}_1^T, \ldots, \mathbf{v}_n^T \right]^T \). It is trivial that \( P \mathbf{w} = \mathbf{w} \) and \( G \mathbf{w} = \mathbf{w} \). Hence, \( P \mathbf{w} = \mathbf{w} \) which implies \( \rho(PG) = 1 \).

Theorem 6. The approximate solutions of all agents will converge to the exact solution exponentially if \( A \) is nonsingular.

Proof: If there exists a vector \( \mathbf{v} \in \mathbb{R}^n \) such that \( P_i \mathbf{v} = \mathbf{v} \) for every block \( P_i \) in \( P \), then \( \mathbf{v} \perp A_i \) for all \( i \), which contradicts the condition that \( A \) is nonsingular. Thus, \( \rho(PG) < 1 \) if \( A \) is nonsingular.

Let \( C(0) \) be the initial error of all the agents. It can be represented as a linear combination of eigenvectors of \( PG \). \( \rho(PG) < 1 \) indicates that every eigenvalue of \( PG \), say \( \lambda \), \( |\lambda| < 1 \). Since each eigenvector decays exponentially with its corresponding eigenvalue during the iteration process, the error of all agents will converge to zero which means the approximate solutions of all agents will converge to the exact solution.

Remark 1. According to the aforementioned analysis, two factors affect the convergence rate of our algorithm. The first one is the condition number of the matrix \( A \). Certain types of matrix, such as diagonally dominant matrix and row-orthogonal matrix, could achieve an acceptable convergence rate based on our experiments. However, an ill-conditioned \( A \), albeit nonsingular, would yield a poor convergence rate, making the consensus process slow. Several preconditioning techniques [31]–[33] have been proposed to transform \( A \) and ensure good condition number. The other one is the connectivity of the network graph. The relation between the convergence rate and the adjacency of the network is discussed in section VII. We also present some numerical results concerning the impact of connectivity on the total computation time of one agent.

B. Computational Complexity Analysis

Low computational complexity, as one of our design goal, is of concern in our scheme. For one thing, due to lacking computation resource, individual agent is not able to support time-consuming computation. For another, if the overall time cost is higher than which the client needs to solve the problem locally, the client would also be reluctant to outsource the problem. Here, we analyze the overall computational complexity for the client and participatory agents throughout the outsourcing process.

Theorem 7. Through the secure outsourcing scheme, the local computational complexity for the client is \( O(n^2) \); the average computational complexity for each agent is \( O(l(dp + 1)dn) \), with \( l \) being the iterations to reach the consensus, \( d \) being the average degree of the network graph.

Proof: For the client, the only computation burden stems from the problem encryption and decryption. By Lemma 3, the encryption complexity is \( O(n^2) \). While decrypting the solution, the client computes \( x' = x - \Delta x \) with complexity \( O(n) \). Thus the total computational complexity for the client is \( O(n^2) \).

With respect to a single agent, it performs two tasks within each iteration: updating their own approximate solutions and probabilistically monitoring their neighbors. Checking the message from their neighbors has the same time complexity as an update process which is \( O(d_i n) \) by Lemma 1. Thus, the complexity expectation within one iteration is \( O((dp + 1)dn) \). Assume it needs \( l \) iterations to reach the consensus, the overall average complexity for each agent is \( O(l(dp + 1)dn) \).

By Theorem 7, the average computational complexity for each agent is related to the network graph and convergence steps. When \( d^2 \ll n \), the computational complexity for each agent approximates to \( O(ln) \). Through simulation we notice that for a diagonal dominant \( A \), the convergence step is bounded by \( O(n) \), resulting in the average complexity for each agent less than \( O(n^2) \).

C. Security Analysis

1) Analysis on Privacy Preservation: We now analyze the privacy guarantee on problem parameters \( A, b, \) and result \( x \). The only information that a malicious agent \( i \) obtains is \( [A'_i]^T b'_j \), \( [A'_j]^T b'_i] \) for \( j \in N_i \) and the solution \( x' \) of the transformed problem. Considering multiple malicious agents may collude, let’s assume malicious agent \( i \) obtains the whole matrix \( A' \) and vector \( b' \). First
of all, the privacy of the solution to the original problem is perfectly protected. Since $x' = x + \Delta x$, with $\Delta x$ can be arbitrary $n$ dimensional vector, the information contained in $x$ is perfectly masked by $\Delta x$. As long as $\Delta x$ being kept locally by the client, no information of the original problem solution can be obtained by agent $i$. Secondly, the privacy of problem parameter $A$ and $b$ are well protected. Since elementary row operation is equivalent to left multiply an elementary matrix, we can express $[A' \ b']$ as $[EA \ E(b + A \Delta x)]$, with $E$ being the multiplication of corresponding elementary matrix. The secret key $\Delta$, $\kappa$ to compute $[A \ b]$ is securely kept by the client, the probability for a malicious agent correctly guessing the secure key is negligible. Besides, offline guessing will not help, since there is no way for a malicious agent to verify its attempt.

2) Analysis on Misbehavior Detection: By Theorem 3, a malicious agent trying to sabotage the algorithm will be detected with probability 1. Now we consider the scenario that a malicious agent sends out a fake alarm identifying agent $j$ updates a false value at iteration $k$, who has in fact honestly performed his duty. A fake alarm message $F_{i,k} = j$ is broadcasted to the common neighboring agents of agent $i$ and $j$. Upon receiving this fake alarm at iteration $k + 1$, the common neighbors of $i$ and $j$ verify $\Phi_j(k)$ with probability 1 and find that $\Phi_j(k)$ is actually correct, then no further alarm message related to agent $j$ on iteration $k$ will be generated. Under the sparse distribution of malicious agents assumption, which implies malicious agents cannot collude, the probability that two fake alarm on the same integrity agent in one iteration is negligible. In summary, we have the following deduction: malicious behaviors 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.

VII. NUMERICAL RESULTS

In this section, we present the numerical results of the proposed scheme, in terms of both efficiency and robustness. We employ a desktop with Intel Core 2 Quad CPU running at 2.34 GHz, 4 GB memory to conduct Problem Encryption process as the client. For the ad hoc clouds, in order to accurately evaluate the computation time of our scheme, the consensus process is simulated in a centralized manner. To meet the memory requirement, this process is performed by a memory optimized instance (r3.2xlarge) on Amazon Elastic Computing Cloud (EC2) [15]. It is worth noting that 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. These simulations are implemented using Python language with the NumPy package extension.

Same as [6], we generate random diagonally dominant matrices to construct the LAE. To model the adjacency of the cloud agents, we use an Erdos-Renyi (ER) random graph $G(n, p)$ [35]. According to the analysis in Section VI, the connectivity of the underlying graph affects the convergence rate of our algorithm. Intuitively, the convergence can be reached at a faster speed underlying a graph with higher connectivity. However, the time complexity per iteration for agent $i$ is $O(d_in)$, which implies that a lower average degree can bring benefit in terms of computation time. Fig. 2 illustrates the trade-off between per iteration time complexity and convergence steps. To guarantee the connectedness of the network graph, we use the union of a $n$-agent cycle and the ER graph generated by $G(n, p)$ model as the network graph. The underlying graph is a cycle with average degree 2 when $p$ equals to 0. Fig shows the relation between the connectivity probability of ER graph and the overall running time for each agent, as well as the convergence steps in a 1000-agent network.

As shown in Fig. 2, although the convergence steps decreases as the connectivity probability increasing, the total running time for each agent is dominated by the time complexity per iteration. This is due to the fact that the computation time rises significantly as the probability increases. The results show that the convergence steps benefit from higher connectivity diminishes exponentially. The marginal reduction of convergence steps can be achieved at $p > 0.1$. Since the average degree of the underlying graph $d \approx np$, the time complexity for
each iteration reduces linearly with a decreasing $p$, which yields a smaller overall computation time. The graph with very small connectivity probability achieves better performance in terms of the computation time compared to a cycle topology when $p = 0$. This is because such a topology could benefit from the dramatical reduction of convergence steps. In the following simulations, we set $p = \frac{\ln n}{n}$. This is a sharp threshold for the connectedness of $G(n, p)$ [35]. Besides, this maps to a relative sparse topology which is reasonable in ad hoc network.

The computational cost of our scheme with respect to the LAE dimension is given in Table I. To illustrate the efficiency of our scheme, we also locally solve the same problem using Jacobi method [34]. For problem size $n = 10000$, the problem encryption only takes less than 2 seconds. Approximate 2.5 hours are needed for solving the LAE locally by Jacobi method, while 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 $10000 \times 1$ vectors, each of which is approximate $10000 \times 8$Bytes $= 80$KB. Considering the computation time gap between the local method and the distributed outsourcing, especially when the scale of problem is very large, the communication overhead of our scheme will be compensated by the running time gain.

In terms of robustness, we adopt two metrics to evaluate the performance of the proposed algorithm — root mean square error (RMSE) and mean standard deviation (MSD). These two metrics can be calculated as

$$RMSE = \frac{1}{n} \sum_{i=1}^{n} \| \mathbf{x}_i(t) - \mathbf{x}^* \|$$  \hspace{1cm} (9)

$$MSD = \frac{1}{n} \| \mathbf{x}_{SD} \|_1$$  \hspace{1cm} (10)

where $\| \cdot \|_1$ represents the one-norm of a vector; $\mathbf{x}_{SD}$ is the standard deviation vector, every entry is the standard deviation of the value in the corresponding entry of all approximate solutions. RMSE characterizes the distance between the exact solution and all the approximate solutions, while the MSD characterizes the extent of consensus for all the agents.

As discussed in section III, without our robust algorithm, a malicious agent can manipulate the final results through a one-time false update. This phenomenon is illustrated in Fig. 3(a). In presence of the malicious agent, final consensus can be reached as shown in Fig. 3(a). However, the RMSE presented in Fig. 3(b) indicates that the algorithm converges to an incorrect solution.

![Fig. 3. Results manipulating attack](image)

For the proposed robust average consensus-based algorithm, similar attacks are conducted at 100th, 300th, 500th iteration respectively. Fig. 4 shows that these three false updates cannot mislead normal agents to converge 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 neighbors of the malicious agent. However, as illustrated in Fig.4(b), these attacks impose little impact on the convergence process. Despite some negligible 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 from honest agents and decay hop by hop.


TABLE I

| Computation Time Comparison |
|-----------------------------|
| Problem Encryption          | 0.038 sec | 0.098 sec |
| Outsourced Problem Solving  | 0.28 sec  | 0.78 sec  |
| Locally Jacobi Method       | 10 sec    | 78 sec    |

VIII. DISCUSSIONS

A. When There Are Less Than n Agents

In real world application scenario, it is unlikely to have exactly n available agents for solving a large scale LAE problem. Now we consider extending this algorithm to work with m agents, for m < n. Instead of obtaining only one row of the problem Ax = b, each agent obtains a set of equations at the problem distribution stage. Consider cloud agent i, who is assigned with a set of equations which can be written as $A_i^{T}x = b_i'$, with $A_i^{T} \in \mathbb{R}^{r \times n}$ and $b_i' \in \mathbb{R}^r$, $r$ is the number of equations assigned to agent i. Now agent i can be viewed as r virtual agents with a connected graph topology. If agent $i$ can run the algorithm as $r$ connected agents and obtain a solution satisfying $A_i^{T}x = b_i'$, then the $m$ agents can apply the proposed algorithm and converge to the exact solution to the outsourced LAE problems. Now we give a proof showing that the proposed algorithm can converge over $r$ agents to a solution satisfying $r$ rows of $A$.

**Corollary 8.** The approximate solutions of $r$ virtual agents could converge into an approximate solution satisfying $r$ rows of $[A \ b]$.

**Proof:** Let $C(k + 1) = [(e_1(k + 1))^T, (e_2(k + 1))^T, \ldots, (e_r(k + 1))^T]^T$ represent the error of these $r$ agents after $k + 1$ iterations. $g_r$ is the stochastic matrix corresponding to the adjacency of these $r$-agent ad hoc network. Then $C(k + 1) = PG_r C(k)$,

where

$$P = \begin{pmatrix} P_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & P_r \end{pmatrix} \quad G_r = g_r \otimes I$$

Since $\rho(PG_r) \leq 1$, $C$ would converge into either zero vector or $w \in \mathbb{R}^{nr}$ such that $PG_r w = w$ depending on the initial solution. For the former case, the approximate solutions of $r$ agents converge into $x^*$ which obviously satisfies $r$ rows of $A$. For the later case, an approximate solution $x' = x^* + v$ would be reached such that $P_i v = v$ for every block $P_i$ in $P$. $v$ is orthogonal to all these $r$ rows and $x'$ is also a solution satisfying $r$ rows of $A$.

**Remark 2.** Note that after these $r$ virtual agents reach consensus, agent i’s update needs to always hold for $A_i^{T}x = b_i'$. To satisfy this condition, the projection matrix of agent i, denoted as $\hat{P}_i$ needs to be the orthogonal projection on the kernel of the row space of $A_i^{T}$. Hence, $\hat{P}_i = \prod_{i=1}^{r} P_i$.

B. Collusion Attack

In this paper, the proposed misbehavior detection scheme based on neighbor monitoring is under the assumption that malicious agents are not neighboring with each other. Such an assumption excludes the situation that malicious agents collude with each other. However, in some cases this assumption may not apply.
For example a malicious agent may hack one or multiple of its neighbors and take control of their computation or communication.

The sufficient condition for a network topology to detect $f$ adversaries is that the network graph is at least $(2f+1)$-connected \cite{36}. Intuitively this condition means there are at least $(2f+1)$-vertex-disjoint paths between any two agents, thus at least $f + 1$ paths cannot be influence by $f$ agents. Therefore, non-local information can be obtained either by relaying or recovering approach. This result provides the theoretical support for a detection mechanism being able to detect multiple malicious agents, whether they collude or not. A promising solution will be each update being kept and relayed for a certain number of steps, thus integrity agents will gather enough non-local information to verify some previous received updates. However, taking into consideration the communication overhead as well as the storage requirement, designing such a detection mechanism is a non-trivial task. We remain the detection scheme for collusion attacks as our future work.

IX. CONCLUSION

In this paper, we proposed a secure outsourcing scheme for solving LAE problem in ad hoc cloud network. Our proposed scheme comprises of a robust distributed average consensus based algorithm, a secret hiding problem transformation technique, and a cooperative verification mechanism. The proposed outsourcing scheme can protect the private information contained in the LAE problem parameters and solutions, and guarantee the correctness of problem solutions. The performance analysis and numerical results demonstrated the proposed scheme is efficient in terms of computational complexity, and robust under variety of malicious behaviors.

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica et al., “A view of cloud computing.”

[2] P. Mell and T. Grance, “The NIST definition of cloud computing,” National Institute of Standards and Technology, 2011.

[3] G. Kirby, A. Dearle, A. Macdonald, and A. Fernandes, “An approach to ad hoc cloud computing,” arXiv preprint arXiv:1002.4738, 2010.

[4] 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, 2014.

[5] C. Rong, S. T. Nguyen, and M. G. Jaatun, “Beyond lightning: a survey on security challenges in cloud computing,” Computers & Electrical Engineering, vol. 39, no. 1, pp. 47–54, 2013.

[6] 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.

[7] 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.

[8] A. Shamir, “How to share a secret,” Communications of the ACM, vol. 22, no. 11, pp. 612–613, 1979.

[9] M. J. Atallah and K. B. Frikken, “Securely outsourcing linear algebra computations,” in Proc. ACM Symposium on Information, Computer and Communications Security, 2010, pp. 48–59.

[10] M. J. Atallah and J. Li, “Secure outsourcing of sequence comparisons,” International Journal of Information Security, vol. 4, no. 4, pp. 277–287, 2005.

[11] M. Blanton and M. Aliashgari, “Secure outsourcing of DNA searching via finite automata,” in Data and Applications Security and Privacy XXIV. Springer, 2010, pp. 49–64.

[12] J. Lu and C. Y. Tang, “Distributed asynchronous algorithms for solving positive definite linear equations over networks-part I: Agent networks,” in Estimation and Control of Networked Systems, vol. 1, no. 1, 2009, pp. 252–257.

[13] S. Mou, J. Liu, and A. S. Morse, “A distributed algorithm for solving a linear algebraic equation,” in Proc. Annual Allerton Conference on Communication, Control, and Computing, 2013, pp. 267–274.

[14] J. Liu, S. Mou, and A. S. Morse, “An asynchronous distributed algorithm for solving a linear algebraic equation,” in Proc. IEEE Conference on Decision and Control, 2013, pp. 5409–5414.

[16] G. A. McGilvary, “Ad hoc Cloud Computing,” Ph.D. dissertation, University of Edinburgh, 2014.

[17] S. K. Pippal, S. Mishra, and D. S. Kushwaha, “Architectural design and issues for ad-hoc clouds,” in Advances in Communication, Network, and Computing. Springer, 2012, pp. 291–296.

[18] A. Malhotra, S. K. Dhandher, and B. Kumar, “Resource allocation in multi-hop mobile ad hoc cloud,” in Proc. IEEE Recent Advances in Engineering and Computational Sciences (RAECS), 2014, pp. 1–6.

[19] J. Straub, R. Marsh, and A. F. Mohammad, “Robotic disaster recovery efforts with ad-hoc deployable cloud computing,” in SPIE Defense, Security, and Sensing, 2013, pp. 87110Q–87110Q.

[20] B. Li, Y. Pei, H. Wu, Z. Liu, and H. Liu, “Computation offloading management for vehicular ad hoc cloud,” in Algorithms and Architectures for Parallel Processing. Springer, 2014, pp. 728–739.

[21] A. Margaritis, “Parallel implementation of the Jacobi linear algebraic system solver,” in Proc. 3rd Balkan Conference in Informatics, 2007.

[22] J. Bylina and B. Bylina, “Merging jacobi and gauss-seidel methods for solving Markov chains on computer clusters,” in Proc. IEEE International Multi-conference on Computer Science and Information Technology, 2008, pp. 263–268.

[23] Y. Saad and H. A. Van Der Vorst, “Iterative solution of linear systems in the 20th century,” Journal of Computational and Applied Mathematics, vol. 123, no. 1, pp. 1–33, 2000.

[24] R. Mehmmood and J. Crowcroft, “Parallel iterative solution method for large sparse linear equation systems,” Computer Laboratory: University of Cambridge, 2005.

[25] J. Dean and S. Ghemawat, “MapReduce: simplified data processing on large clusters,” Communications of the ACM, vol. 51, no. 1, pp. 107–113, 2008.
[26] R. Olfati-Saber, J. A. Fax, and R. M. Murray, “Consensus and cooperation in networked multi-agent systems,” *Proc. of the IEEE*, vol. 95, no. 1, pp. 215–233, 2007.

[27] J. He, J. Chen, P. Cheng, 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, 2013.

[28] 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, 2012.

[29] Q. Yan, M. Li, T. Jiang, W. Lou, and Y. T. Hou, “Vulnerability and protection for distributed consensus-based spectrum sensing in cognitive radio networks,” in *Proc. IEEE INFOCOM*, 2012, pp. 900–908.

[30] J. He, P. Cheng, L. Shi, and J. Chen, “SATS: secure average-consensus-based time synchronization in wireless sensor networks,” *IEEE Trans. Signal Process.*, vol. 61, no. 24, pp. 6387–6400, 2013.

[31] K. Chen, *Matrix preconditioning techniques and applications*. Cambridge University Press, 2005, no. 19.

[32] M. J. Grote and T. Huckle, “Parallel preconditioning with sparse approximate inverses,” *SIAM Journal on Scientific Computing*, vol. 18, no. 3, pp. 838–853, 1997.

[33] M. Benzi, “Preconditioning techniques for large linear systems: a survey,” *Journal of Computational Physics*, vol. 182, no. 2, pp. 418–477, 2002.

[34] Y. Saad, *Iterative methods for sparse linear systems*. SIAM, 2003.

[35] P. Erdős and A. Rényi, “On the evolution of random graphs,” *Publ. Math. Inst. Hungar. Acad. Sci*, vol. 5, pp. 17–61, 1960.

[36] 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, 2013.