Journal of Computer Security 20 (2012) 583–634
DOI 10.3233/JCS-2012-0452
IOS Press

Secure and efficient distributed linear programming

Yuan Hong [a,*], Jaideep Vaidya [a] and Haibing Lu [b]
[a] *CIMIC, Rutgers University, Newark, NJ, USA*
E-mails: {yhong, jsvaidya}@cimic.rutgers.edu
[b] *OMIS, Santa Clara University, Santa Clara, CA, USA*
E-mail: hlu@scu.edu

In today's networked world, resource providers and consumers are distributed globally and locally, especially under current cloud computing environment. However, with resource constraints, optimization is necessary to ensure the best possible usage of such scarce resources. *Distributed linear programming* (DisLP) problems allow collaborative agents to jointly maximize profits or minimize costs with a linear objective function while conforming to several shared as well as local linear constraints. Since each agent's share of the global constraints and the local constraints generally refer to its private limitations or capacities, serious privacy problems may arise if such information is revealed. While there have been some solutions raised that allow secure computation of such problems, they typically rely on inefficient protocols with enormous computation and communication cost.

In this paper, we study the DisLP problems where constraints are *arbitrarily partitioned* and every agent privately holds a set of variables, and propose *secure and extremely efficient* approach based on mathematical transformation in two adversary models – semi-honest and malicious model. Specifically, we first present a *secure column generation (SCG) protocol* that securely solves the above DisLP problem amongst two or more agents without any private information disclosure, assuming semi-honest behavior (all agents properly follow the protocol but may be curious to derive private information from other agents). Furthermore, we discuss potential selfish actions and colluding issues in malicious model (distributed agents may corrupt the protocol to gain extra benefit) and propose an *incentive compatible protocol* to resolve such malicious behavior. To address the effectiveness of our protocols, we present security analysis for both adversary models as well as the communication/computation cost analysis. Finally, our experimental results validate the efficiency of our approach and demonstrate its scalability.

Keywords: Distributed linear programming, security, efficiency, transformation

1. Introduction

Optimization is a fundamental problem found in many diverse fields. As an essential subclass of optimization, where all of the constraints and objective function are *linear*, linear programming models are widely applicable to solving numerous profit-maximizing or cost-minimizing problems such as transportation, commodities, airlines and communication.

*Corresponding author: Yuan Hong, CIMIC, Rutgers University, Newark, NJ 07102, USA. E-mail: yhong@cimic.rutgers.edu.

With the rapid development of computing, sharing and distributing online resources, collaboration between distributed agents always involves some optimization problems. For instance, in the packaged goods industry, delivery trucks are empty 25% of the time. Just four years ago, Land O'Lakes truckers spent much of their time shuttling empty trucks down slow-moving highways, wasting several million dollars annually. By using a web based collaborative logistics service (Nistevo.com), to merge loads from different companies (even competitors) bound to the same destination, huge savings were realized (freight costs were cut by 15%, for an annual savings of $2 million [38]). This required sending all information to a central site. Such complete sharing of data may often be impossible for many corporations, and thus result in great loss of possible *confidential information leakage*. In general, Walmart, Target and CostCo ship millions of dollars worth of goods over the seas every month. These feed into their local ground transportation network. The cost of sending half-empty ships is prohibitive, but the individual corporations have serious problems with disclosing freight information. If it were possible simply to determine what trucks should make their way to which ports to be loaded onto certain ships, e.g., solve the classic transportation problem, without knowing the individual constraints, the savings would be enormous. In all of these cases, complete sharing of data would lead to invaluable savings/benefits. However, since unrestricted data sharing is a competitive impossibility or requires great trust, and since this is a transportation problem which can be modeled through linear programming, a privacy-preserving *distributed linear programming* (DisLP) solution that tightly limits the information disclosure would make this possible without the release of proprietary information.

To summarize, DisLP problems facilitate collaborative agents to jointly maximize global profits (or minimize costs) while satisfying several (global or local) constraints. In numerous DisLP problems, each company hold its own variables to constitute the global optimum decision in the collaboration. Variables are generally not shared between companies in those DisLP problems because collaborators may have their own operations w.r.t. the global maximized profit or minimized cost. In this paper, we study such kind of common DisLP problems where constraints are *arbitrarily partitioned* and every agent privately holds a set of variables. Arbitrary partition implies that some constraints are globally formulated among multiple agents ("vertically partitioned" global constraints – each agent knows only a share of these constraints) whereas some constraints are locally held and known by only one agent ("horizontally partitioned" local constraints – each agent completely owns all these constraints). In this arbitrary partition based collaboration, each agent's share in the global constraints and its local constraints generally refer to such agent's private limitations or capacities, which should be kept private while jointly solving the problem.

Consider the following two illustrative examples.

Example 1 (Collaborative transportation). K Companies $P_1, \ldots, P_K$ share some of their delivery trucks for transportation (seeking minimum cost): the amount of

transportation from location j to location k for company P_i ($i \in [1, K]$) are denoted as $x_i = \{\forall j, \forall k, x_{ijk}\}$, thus P_i holds its own set of variables x_i.

In the collaborative transportation, $P_1, \ldots, P_K$ should have some local constraints (e.g., its maximum delivery amount from location j to k, or the capacities of its non-shared trucks) and some global constraints (e.g., the capacity of its shared trucks for global usage). After solving the DisLP problem, each company should know its delivery amount from each location to another location (the values of x_i in the global optimal solution).

Example 2 (Collaborative production). K companies $P_1, \ldots, P_K$ share some raw materials or labor for production (seeking maximum profits): the amount of company P_i ($i = 1, \ldots, K$)'s product j to be manufactured are denoted as $x_i = \{\forall x_{ij}\}$, thus P_i holds its own set of variables x_i.

Similarly, in the collaborative production, $P_1, \ldots, P_K$ should have some local constraints (e.g., each company's non-shared local raw materials or labor) and some global constraints (e.g., shared raw materials or labor for global usage). After solving the DisLP problem, each company should only know its production amount for each of its products (the values of x_i in the global optimal solution).

Figure 1 demonstrates a simple example for formulating collaborative production problem with the described DisLP model. K companies jointly manufacture products (two for each company) using a shared raw material where $P_1, P_2, \ldots, P_K$ have the amount $15, 10, \ldots, 20$, respectively (the sum of the global profits can be increased via this collaboration since the combined resources are better utilized). They also have their local constraints, e.g., the total labor for producing each company's

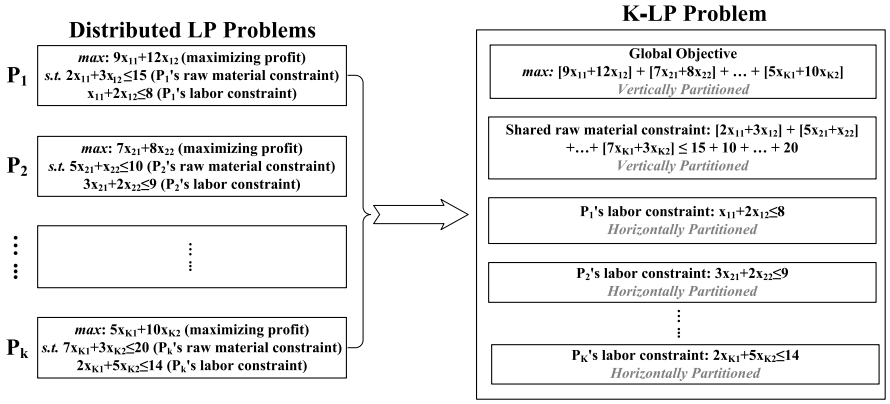


Fig. 1. Illustrative example for distributed LP problem formulation. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/JCS-2012-0452.)

products are bounded with constraints $8, 9, \ldots, 14$, respectively. After solving this DisLP problem, each company should know the (global) optimal production amount for only its products but should not learn anything about the private constraints and solution of other companies.

To simplify the notation, we formally define this kind of DisLP problem as below (due to {min: $c^{\mathrm{T}}x \equiv$ max: $-c^{\mathrm{T}}x$}, we only need to model objective function max: $c^{\mathrm{T}}x$).

Definition 1 (K-agent LP problem (K-LP)). An LP problem is solved by K distributed agents where each agent P_i privately holds n_i variables x_i, its m_i local constraints $B_i x_i \bowtie_i b_i$, a share of the objective vector c_i, and the matrix/vector share A_i/b_0^i in m_0 global constraints $\sum_{i=1}^{K} A_i x_i \bowtie_0 b_0$ (as shown in Eq. (1)[1,2] ($i \in [1, K]$ and $\sum_{i=1}^{K} b_0^i = b_0$).

$$\max c_1x_1 + c_2x_2 + \cdots + c_Kx_K$$
$$\text{s.t.} \left\{ \begin{array}{c} x_1 \in \mathbb{R}^{n_1} \\ x_2 \in \mathbb{R}^{n_2} \\ \vdots \\ x_K \in \mathbb{R}^{n_K} \end{array} \right. : \begin{pmatrix} A_1 \ldots A_K \\ B_1 \\ \quad \ddots \\ \quad\quad B_K \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_K \end{pmatrix} \begin{array}{c} \bowtie_0 \\ \bowtie_1 \\ \vdots \\ \bowtie_K \end{array} \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_K \end{pmatrix}. \tag{1}$$

Indeed, besides collaborative transportation (see Example 1) and production (see Example 2), K-LP problems occur very frequently in real world (e.g., selling the goods in bundles for distributed agents to maximize the global profits, and determining profit-maximized travel packages for hotels, airlines and car rental companies), where the distributed resources can be better utilized via collaboration.

1.1. Contributions

Before providing an efficient privacy-preserving approach for the K-LP problem, we first investigate the security/privacy issues in the arbitrarily partitioned constraints and privately held variables (see Definition 1). Intuitively, securely solving K-LP problem can be regarded as a secure multiparty computation (SMC [15,41]) problem, though we tackle the privacy issues for the K-LP problem based on mathematical transformation rather than SMC techniques (for significantly improved efficiency). Normally, two major adversary models are identified in secure computation – semi-honest and malicious model. In semi-honest model, all distributed agents are honest-but-curious: they are honest to follow the protocol but may be curious to derive private information from other agents. The case becomes worse in malicious model,

[1] $\bowtie$ denotes $\leqslant, =$ or $\geqslant$.

[2] Matrices/vectors: $i \in [1, K]$, $A_i \in \mathbb{R}^{m_0 \times n_i}$, $B_i \in \mathbb{R}^{m_i \times n_i}$, $c_i \in \mathbb{R}^{n_i}$, $b_0 \in \mathbb{R}^{m_0}$ and $b_i \in \mathbb{R}^{m_i}$.

dishonest agents can deviate from the protocol and act in their self-interest, or collude with each other to gain additional information. In this paper, we present efficient protocols to resolve the privacy as well as the dishonesty problems in the above two adversary models. Thus, the main contributions of this paper are summarized as below:

(1) *Efficient privacy-preserving protocol in semi-honest model*. We first introduce a secure and efficient protocol for K-LP problem by revising Dantzig–Wolfe decomposition [7,37] which was originally proposed for efficiently solving large-scale LP problems with block-angular constraint matrix. Specifically, we transform the K-LP problem to a private information protected format and design the secure K-agent column generation (SCG) protocol to solve the transformed LP problem securely and efficiently.
(2) *Efficient incentive compatible protocol in malicious model*. We also analyze most of the major potential malicious actions in K-LP problems. To resolve such malicious behavior, we design an incentive compatible protocol by enforcing honesty based on game theoretic models.
(3) *Detailed analysis*. We provide security analysis for the secure column generation (SCG) protocol in semi-honest model and prove incentive compatibility in malicious model. Moreover, we also present communication and computational cost analysis for our protocols.
(4) *Experimental validation*. Finally, our experimental results validate the efficiency (computational performance) of our approach and demonstrate the scalability of the SCG and the incentive compatible protocols.

The rest of this paper is structured as follows. Sections 2 and 3 introduce related work and preliminaries for our approach respectively. Section 4 presents the security-oriented transformation methodology and shows how to reconstruct each agent's share in the original global optimal solution after solving the transformed problem. In Section 5, we present the secure protocol with security and computation/communication cost analysis in semi-honest model. We resolve malicious behavior and discuss the additional computation/communication costs in Section 6. Finally, we validate the efficiency and the scalability of our protocols in Section 7 and conclude the paper in Section 8.

2. Related work

Research in distributed optimization and privacy-preserving optimization are both relevant to our work.

2.1. Distributed optimization

There is work in distributed optimization that aims to achieve a global objective using only local information. This falls in the general area of distributed decision

making with incomplete information. This line of research has been investigated in a worst case setting (with no communication between the distributed agents) by Papadimitriou et al. [8,27,28]. In [28], Papadimitriou and Yannakakis first explore the problem facing a set of decision-makers who must select values for the variables of a linear program, when only parts of the matrix are available to them and prove lower bounds on the optimality of distributed algorithms having no communication. Awerbuch and Azar [2] propose a distributed flow control algorithm with a global objective which gives a logarithmic approximation ratio and runs in a polylogarithmic number of rounds. Bartal et al.'s distributed algorithm [3] obtains a better approximation while using the same number of rounds of local communication.

Distributed Constraint Satisfaction was formalized by Yokoo [42] to solve naturally distributed constraint satisfaction problems. These problems are divided between agents, who then have to communicate among themselves to solve them. To address distributed optimization, complete algorithms like OptAPO and ADOPT have been recently introduced. ADOPT [24] is a backtracking based bound propagation mechanism. It operates completely decentralized, and asynchronously. The downside is that it may require a very large number of messages, thus producing big communication overheads. OptAPO [21] centralizes parts of the problem; it is unknown a priori how much needs to be centralized where, and privacy is an issue. Distributed local search methods like DSA [18]/DBA [45] for optimization, and DBA for satisfaction [43] start with a random assignment, and then gradually improve it. Sometimes they produce good results with a small effort. However, they offer no guarantees on the quality of the solution, which can be arbitrarily far from the optimum. Termination is only clear for satisfaction problems, and only if a solution was found.

DPOP [29] is a dynamic programming based algorithm that generates a linear number of messages. However, in case the problems have high induced width, the messages generated in the high-width areas of the problem become too large. There have been proposed a number of variations of this algorithm that address this problem and other issues, offering various tradeoffs (see [30,31]). Petcu and Faltings [31] propose an approximate version of this algorithm, which allows the desired tradeoff between solution quality and computational complexity. This makes it suitable for very large, distributed problems, where the propagations may take a long time to complete. However, in general, the work in distributed optimization has concentrated on reducing communication costs and has paid little or no attention to security constraints. Thus, some of the summaries may reveal significant information. In particular, the rigor of security proofs has not been applied much in this area.

2.2. Privacy-preserving distributed optimization

There is also work in secure optimization. Silaghi and Rajeshirke [34] show that a secure combinatorial problem solver must necessarily pick the result randomly

among optimal solutions to be really secure. Silaghi and Mitra [35] propose arithmetic circuits for solving constraint optimization problems that are exponential in the number of variables for any constraint graph. A significantly more efficient optimization protocol specialized on generalized Vickrey auctions and based on dynamic programming is proposed by Suzuki and Yokoo [36], though it is not completely secure under the framework in [34]. Yokoo et al. [44] also propose a scheme using public key encryption for secure distributed constraint satisfaction. Silaghi et al. [33] show how to construct an arithmetic circuit with the complexity properties of DFS-based variable elimination, and that finds a random optimal solution for any constraint optimization problem. Atallah et al. [1] propose protocols for secure supply chain management. However, much of this work is still based on generic solutions and not quite ready for practical use. Even so, some of this work can definitely be leveraged to advance the state of the art by building general transformations or privacy-preserving variants of well-known methods.

Recently, there has been significant interest in the area of *privacy-preserving linear programming*. In general, most of the existing privacy-preserving linear programming techniques follow two major directions – the LP problem transformation approach and the secure multiparty computation (SMC [15,41]) based approach. On one hand, Du [9] and Vaidya [39] transformed the linear programming problem by multiplying a monomial matrix to both the constraint matrix and the objective function, assuming that one party holds the objective function while the other party holds the constraints. Bednarz et al. [4] pointed out a potential attack to the above transformation approach. Mangasarian presented two transformation approaches for horizontally partitioned linear programs [23] and vertically partitioned linear programs [22], respectively. Li et al. [20] extended the transformation approach [23] for horizontally partitioned linear programs with equality constraints to inequality constraints. Furthermore, we presented an transformation approach for the DisLP problem where constraints are arbitrarily partitioned among two or more parties and every variable belongs to only one party in our prior work [16]. More recently, Dreier and Kerschbaum proposed a secure transformation approach for a complex data partition scenario, and illustrated the effectiveness of their approach. On the other hand, Li and Atallah [19] addressed the collaborative linear programming problem between two parties where the objective function and constraints can be arbitrarily partitioned, and proposed a secure simplex method for such problem using some cryptographic tools (homomorphic encryption and scrambled circuit evaluation). Vaidya [40] proposed a secure revised simplex approach which also employs SMC techniques (secure scalar product computation and secure comparison) but improved the efficiency compared with Li and Atallah's approach [19]. Catrina and Hoogh [5] presented a solution to solve distributed linear program based on secret sharing. The protocols utilized a variant of the simplex algorithm and secure computation with fixed-point rational numbers, optimized for such application.

Finally, as for securing distributed NP-hard problems, Sakuma et al. [32] proposed a genetic algorithm for securely solving two-party distributed traveling salesman

problem (NP-hard). They consider the case that one party holds the cost vector while the other party holds the tour vector (this is a special case of multi-party distributed combinatorial optimization). The distributed TSP problem that is completely partitioned among multiple parties has been discussed but not solved in [32]. Hong et al. [17] consider a more complicated scenario in another classic combinatorial optimization model – graph coloring that is completely partitioned among multiple parties. Moreover, a privacy-preserving tabu search protocol was proposed to securely solve the distributed graph coloring problem. Clifton et al. [6] have also proposed an algorithm that find opportunities to swap loads without revealing private information except the swapped loads for practical use (though the vehicle routing optimization problem has been mapped into one dimension using Hilbert space-filling curve). The security and incentive compatibility have been proven in such work. Our work applies to only linear programming but is significantly more efficient than prior solutions and is resilient to both semi-honest and malicious adversaries.

3. Preliminaries

In this section, we briefly review some preliminaries with regard to our work, including some definitions and properties related to LP problems, the security for general secure multiparty computation as well as the K-LP problem in semi-honest model, and the notion of incentive compatibility in malicious model.

3.1. Polyhedra

From the geometrical point of view, the linear constraints of every LP problem can be represented as a polyhedron. We thus introduce some geometrical definitions for LP problems.

Definition 2 (Polyhedron of linear constraints). A polyhedron $P \subseteq \mathbb{R}^n$ is the set of points that satisfy a finite number (m) of linear constraints $P = \{x \in \mathbb{R}^n\colon\ Ax \bowtie b\}$ where A is an $m \times n$ constraint matrix.

Definition 3 (Convex combination). A point $x \in \mathbb{R}^n$ is a convex combination of a set $S \subseteq \mathbb{R}^n$ if x can be expressed as $x = \sum_i \lambda_i x^i$ for a finite subset $\{x^i\}$ of S and $\lambda > 0$ with $\sum_i \lambda_i = 1$.

Definition 4 (Extreme point (or vertex)). A point $x^e \in P$ is an extreme point (or vertex) of $P = \{x \in \mathbb{R}^n\colon\ Ax \bowtie b\}$ if it cannot be represented as a convex combination of two other points $x^i, x^j \in P$.

Definition 5 (Ray in polyhedron). Given a non-empty polyhedron $P = \{x \in \mathbb{R}^n\colon\ Ax \bowtie b\}$, a vector $r \in \mathbb{R}^n, r \neq 0$ is a ray if $Ar \bowtie 0$.

Definition 6 (Extreme ray). A ray r is an extreme ray of $P = \{x \in \mathbb{R}^n\colon\ Ax \bowtie b\}$ if there does not exist two distinct rays r^i and r^j of P such that $r = \frac{1}{2}(r^i + r^j)$.

Note that the optimal solution of an LP problem is always a vertex (extreme point) or an extreme ray of the polyhedron formed by all the linear constraints [7].

Theorem 1 (Optimal solution). *Consider a feasible LP problem* $\max\{c^{\mathrm{T}}x\colon\ x \in P\}$ *with* $\operatorname{rank}(A) = n$. *If it is bounded it has an optimal solution at an extreme point* x^* *of* P; *if it is unbounded, it is unbounded along an extreme ray* r^* *of* P (*and so* $cr^* > 0$).

Proof. See [25,26]. □

3.2. Dantzig–Wolfe decomposition

Assume that we let x^i (size n vector) represent the extreme points/rays in the LP problem. Hence, every point inside the polyhedron can be represented by the extreme points/rays using convexity combination (Minkowski's representation theorem [25, 37]).

Theorem 2 (Minkowski's representation theorem). *If* $P = \{x \in \mathbb{R}^n\colon\ Ax \bowtie b\}$ *is non-empty and* $\operatorname{rank}(A) = n$, *then* $P = P'$, *where* $P' = \{x \in \mathbb{R}^n\colon\ x = \sum_{i\in\Pi}\lambda_i x_i + \sum_{j\in R}\mu_j r^j; \sum_{i\in\Pi}\lambda_i = 1; \lambda, \mu > 0\}$ *and* $\{x^i\}_{i\in\Pi}$ *and* $\{r^j\}_{j\in R}$ *are the sets of extreme points and extreme rays of* P, *respectively.*

Proof. See [25,26,37]. □

Thus, a polyhedron P can be represented by another polyhedron $P' = \{\lambda \in \mathbb{R}^{|E|}\colon\ \sum_{i\in E}\delta_i\lambda_i = 1; \lambda \leqslant 0\}$ where $|E|$ is the number extreme points/rays and

$$\delta_i = \begin{cases} 1 & \text{if } x^i \text{ is a extreme point (vertex),} \\ 0 & \text{if } x^i \text{ is an extreme ray.} \end{cases} \tag{2}$$

As a result, a projection from P to P' is given: $\{\Pi\colon \mathbb{R}^n \mapsto \mathbb{R}^{|E|}; x = \sum_i \lambda_i x^i \mapsto \lambda\}$. If the polyhedron P is block-angular structure, it can be decomposed to smaller-dimensional polyhedra $\forall P_i = \{x_i \in \mathbb{R}^{n_i}\colon\ B_i x_i \bowtie_i b_i\}$ where x_i is a n_i-dimensional vector, $n = \sum_{i=1}^{K} n_i$ (as shown in Eq. (1)). Additionally, we denote polyhedron of the global constraints as $P_0 = \{x \in \mathbb{R}^n\colon\ (A_1\ A_2\ \cdots\ A_K)x \bowtie_0 b_0\}$. We thus have:

$$P = P_0 \cap (P_1 \times P_2 \times \cdots \times P_K). \tag{3}$$

Hence, the original LP problem (Eq. (1)) can be transformed to a master problem (Eq. (4)) using Dantzig–Wolfe decomposition [26]. Note that x_j^i represents the extreme point or ray associated with λ_{ij} (now $\forall x_i^j$ are constants whereas $\forall\lambda_{ij}$ are the

variables in the projected polyhedron P'), and $|E_1|, \ldots, |E_K|$ denote the number of extreme points/rays of the decomposed polyhedra $P_1, \ldots, P_K$, respectively.

$$\max \sum_j c_1^{\mathrm{T}} x_1^j \lambda_{1j} + \cdots + \sum_j c_K^{\mathrm{T}} x_K^j \lambda_{Kj}$$

$$\text{s.t.} \begin{cases} \sum_j A_1 x_1^j \lambda_{1j} + \cdots + \sum_j A_K x_K^j \lambda_{Kj} \bowtie b_0, \\ \sum_j \delta_{1j} \lambda_{1j} = 1, \\ \ddots \\ \sum_j \delta_{Kj} \lambda_{Kj} = 1, \\ \lambda_1 \in \mathbb{R}^{|E_1|}, \ldots, \lambda_K \in \mathbb{R}^{|E_K|}, \delta_{ij} \in \{0, 1\}. \end{cases} \tag{4}$$

As proven in [37], primal feasible points, optimal primal points, an unbounded rays, dual feasible points, optimal dual points and certificate of infeasibility in the *master problem* are equivalent to the *original problem* (also see Appendix B).

3.3. Security in the semi-honest model of distributed computation

All variants of Secure Multiparty Computation (SMC [15,41]), in general, can be represented as: K-distributed agents $P_1, \ldots, P_K$ jointly computes one function $f(x_1, \ldots, x_K)$ with their private inputs $x_1, \ldots, x_K$; any agent P_i ($i \in [1, K]$) cannot learn anything other than the output of the function in the secure computation. The ideal security model is that the SMC protocol (all agents run it without trusted third party) is equivalent to a trusted third party that collects the inputs from all agents, computes the result and sends the result to all agents. The SMC protocol is secure in semi-honest model if every agent's view during executing the protocol can be simulated by a polynomial machine (nothing can be learnt from the protocol) [14].

However, in distributed linear programming, which includes a combination of numerous complex computation functions, the strictly secure SMC protocols generally requires tremendous computation and communication cost, and cannot scale to large-size LP problems. To enhance the efficiency and scalability, we thus employ a relaxed security notion by allowing trivial information disclosure in the protocol and shows that any private information cannot be learnt from the protocol and the transformed data.

3.4. Incentive compatibility

Moreover, we utilize game theoretical approach to resolve the potential malicious behavior occurring in the protocol execution – the problem solving phase. We thus focus on the notion of incentive compatibility in malicious model.

Definition 7 (Incentive compatibility). A protocol is incentive compatible if "honest play" is the strictly dominant strategy[3] for every participant.

In our protocol against malicious agents, all the rational participants (agents) will follow the protocol and deliver correct information to other agents. Note that our incentive compatible protocol guarantees that any malicious agent *gains less payoffs/benefits by deviating from the protocol* under our cheating detection mechanism.

4. Revised Dantzig–Wolfe decomposition

As shown in Eq. (1), K-LP problem has a typical block-angular structure, though the number of global constraints can be significantly larger than each agent's local constraints. Hence, we can solve the K-LP problem using Dantzig–Wolfe decomposition. In this section, we transform our K-LP problem to an anonymized (block-angular) format that preserves each agent's private information. Moreover, we also show that the optimal solution for each agent's variables can be derived after solving the transformed problem.

4.1. K-LP transformation

Du [9,10] and Vaidya [39] proposed a transformation approach for solving two-agent DisLP problems: transforming an $m \times n$ constraint matrix M (and the objective vector c^{T}) to another $m \times n$ matrix $M' = MQ$ (and $c'^{\mathrm{T}} = c^{\mathrm{T}}Q$) by post-multiplying an $n \times n$ matrix Q, solving the transformed problem and reconstructing the original optimal solution. Bednarz et al. [4] showed that to ensure correctness, the transformation matrix Q must be monomial. Following them, we let each agent P_i ($i \in [1, K]$) transform its local constraint matrix B_i, its share in the global constraint matrix A_i and the global objective vector c_i using its privately held monomial matrix Q_i.

We let each agent P_i ($i \in [1, K]$) transform A_i and B_i by Q_i individually for the following reason. Essentially, we extend a revised version of Dantzig–Wolfe decomposition to solve K-LP and ensure that the protocol is secure. Thus, an arbitrary agent should be chosen as the master problem solver whereas all agents (including the master problem solving agent) should solve the pricing problems. For transformed K-LP problem (Eq. (5)), we can let $\forall P_i$ ($i \in [1, K]$) send its transformed matrices/vector A_iQ_i, B_iQ_i, $c_i^{\mathrm{T}}Q_i$ to another agent P_j ($j \in [1, K], j \neq i$) and let P_j solve P_i's transformed pricing problems. In this case, we can show that any agent P_i's share in the LP problem A_i, B_i, c_i cannot be learnt while solving the problems. This is due to the fact that every permuted column in A_i, B_i or number in c_i has been multiplied with an unknown random number in the transformation [4,39]. Note that the

[3] In game theory, strictly dominant strategy occurs when one strategy gains better payoff/benefit than any other strategies for one participant, no matter what other participants play.

attack specified in [4] can be eliminated in our secure K-LP problem). Otherwise, if each agent solves its pricing problem, since each agent knows its transformation matrix, additional information might be learnt from master problem solver to pricing problem solvers in every iteration (this is further discussed in Section 5).

$$\max \sum_{i=1}^{K} c_i^{\mathrm{T}} Q_i y_i$$
$$\text{s.t.} \begin{cases} y_1 \in \mathbb{R}^{n_1} \\ y_2 \in \mathbb{R}^{n_2} \\ \vdots \\ y_K \in \mathbb{R}^{n_K} \end{cases} \begin{pmatrix} A_1Q_1 & \dots & A_KQ_K \\ B_1Q_1 & & \\ & \ddots & \\ & & B_KQ_K \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_K \end{pmatrix} \begin{matrix} \bowtie_0 \\ \bowtie_1 \\ \vdots \\ \bowtie_K \end{matrix} \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_K \end{pmatrix}. \tag{5}$$

The K-LP problem can be transformed to another block-angular structured LP problem as shown in Eq. (5). We can derive the original solution from the solution of the transformed K-LP problem using the following theorem. (Proven in Appendix A.1.)

Theorem 3. *Given the optimal solution of the transformed K-LP problem* $y^* = (y_1^*, y_2^*, \dots, y_K^*)$*, the solution* $x^* = (Q_1y_1^*, Q_2y_2^*, \dots, Q_Ky_K^*)$ *should be the optimal solution of the original K-LP problem.*

4.2. Right-hand side value b anonymization

Essentially, with a transformed matrix, nothing about the original constraint matrix can be learnt from the transformed matrix (every appeared value in the transformed matrix is a scalar product of a row in the original matrix and a column in the transformed matrix, though Q is a monomial matrix) [4,9,39]. Besides protecting each party's share of the global constraint matrix A_i and its local constraint matrix B_i, solving the LP problems also requires the right-hand side constants b in the constraints. Since b sometimes refers to the amount of limited resources (i.e., labors, materials) or some demands (i.e., the amount of one product should be no less than 10, $x_{ij} \geqslant 10$), they should not be revealed. We can anonymize b for each agent before transforming the constraint matrix and sending them to other agents.

Intuitively, in the K-LP problem, each agent P_i ($i \in [1, K]$) has two distinct constant vectors in the global and local constraints: b_0^i and b_i where $b_0 = \sum_{i=1}^{K} b_0^i$. We can create *artificial variables* and *equality constraints* to anonymize both b_0^i and b_i. While anonymizing b_0^i in the global constraints $\sum_{i=1}^{K} A_i x_i \bowtie_0 \sum_{i=1}^{K} b_0^i$, each agent P_i creates a new artificial variable s_{ij} (s_{ij} is always equal to a fixed random number η_{ij}) and a random coefficient α_{ij} of s_{ij} for the jth global constraint ($j \in [1, m_0]$).

Consequently, we expand A_i to a larger $m_0 \times (n_i + m_0)$ matrix as shown in Eq. (6) ($A_i^1, \ldots, A_i^{m_0}$ denote the rows of matrix A_i).

$$A_i x_i = \begin{pmatrix} A_i^1 \\ A_i^2 \\ \vdots \\ A_i^{m_0} \end{pmatrix} x_i \quad \Longrightarrow \quad A_i' x_i' = \left(\begin{array}{c|cccc} A_i^1 & \alpha_{i1} & 0 & \ldots & 0 \\ A_i^2 & 0 & \alpha_{i2} & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_i^{m_0} & 0 & 0 & \ldots & \alpha_{im_0} \end{array}\right) \begin{pmatrix} x_i \\ \hline s_{i1} \\ \vdots \\ s_{im_0} \end{pmatrix}. \tag{6}$$

As a result, b_0^i can be anonymized as $(b_0^i)'$ where $\forall j \in [1, m_0], (b_0^i)_j$ and $(b_0^i)'_j$ denote the jth number in (b_0^i) and $(b_0^i)'$, respectively, and $\forall j \in [1, m_0], \alpha_{ij}$ and η_{ij} are random numbers generated by agent P_i:

$$b_0^i = \begin{pmatrix} (b_0^i)_1 \\ (b_0^i)_2 \\ \vdots \\ (b_0^i)_{m_0} \end{pmatrix} \quad \Longrightarrow \quad (b_0^i)' = \begin{pmatrix} (b_0^i)_1 + \alpha_{i1}\eta_{i1} \\ (b_0^i)_2 + \alpha_{i2}\eta_{i2} \\ \vdots \\ (b_0^i)_{m_0} + \alpha_{im_0}\eta_{im_0} \end{pmatrix}. \tag{7}$$

To guarantee the equivalence of $\forall j \in [1, m_0], s_{ij} = \eta_{ij}$, each agent P_i creates additional m_0 (or more) local *linear independent* equality constraints with artificial variables $s_i = \{\forall j, s_{ij}\}$ and random coefficients r_{ijk}, $k \in [1, m_0]$:

$$\text{s.t.} \begin{cases} \sum_{j=1}^{m_0} r_{ij1} s_{ij} = \sum_{j=1}^{m_0} r_{ij1}\eta_{ij}, \\ \sum_{j=1}^{m_0} r_{ij2} s_{ij} = \sum_{j=1}^{m_0} r_{ij2}\eta_{ij}, \\ \quad \vdots \qquad \vdots \qquad\quad \vdots \\ \sum_{j=1}^{m_0} r_{ijm_0} s_{ij} = \sum_{j=1}^{m_0} r_{ijm_0}\eta_{ij}, \end{cases} \tag{8}$$

where the above $m_0 \times m_0$ constraint matrix is randomly generated (note that the random rows in the constraint matrix can be considered as linearly independent [11]). Thus, $\forall j \in [1, m_0], s_{ij} = \eta_{ij}$ always holds since the rank of the constraint matrix is equal to m_0 (with m_0 linearly independent rows in the constraint matrix; if we generate more than m_0 equality constraint using the random coefficients, the rank of the constraint matrix remains m_0). We thus let agent P_i add those linear independent equality constraints (Eq. (8)) into its local constraints $B_i x_i \bowtie_i b_i$. Therefore, we

have:

- The jth global constraint should be converted to $\sum_{i=1}^{K} A_i^j x_i + \sum_{i=1}^{K} s_{ij} \bowtie_0^j \sum_{i=1}^{K} (b_0^i)'_j$ where $(b_0^i)'_j$ represents the jth number in the length-m_0 vector $(b_0^i)'$ and it can be any random number (thus, $b'_0 = \sum_{i=1}^{K} (b_0^i)'$ can be securely summed or directly summed by K agents).
- Additional local linear independent equality constraints ensure $\sum_{i=1}^{K} A_i x_i \bowtie_0 b_0^i$ for a feasible K-LP problem since $\forall i, s_i = \eta_i$.

Besides b_0^i, we can anonymize b_i using a similar way. P_i can use the same set of artificial variables s_i to anonymize b_i. By generating linear combination of the artificial variables $s_i = \{\forall j \in [1, m_0], s_{ij}\}$ (not required to be linearly independent since the values of $\forall i, s_i$ have been fixed as η_i in the process of b_0^i anonymization), the left-hand side of the wth ($w \in [1, m_i]$) constraint in $B_i x_i \bowtie_i b_i$ can be updated: $B_i^w x_i \leftarrow B_i^w x_i + \sum_{j=1}^{m_0} h_{ijw} s_{ij}$ where h_{ijw} is a random number and the wth value in b_i is updated as $b_i^w \leftarrow b_i^w + \sum_{j=1}^{m_0} h_{ijw} \eta_{ij}$. If anonymizing b_i as above, adversaries may guess m_0 additional local constraints out of $(m_i + m_0)$ total local constraints from P_i's sub-polyhedron. The probability of guessing out the additional linearly independent constraints and calculating the exact values of the artificial variables is $\frac{m_0! m_i!}{(m_i + m_0)!}$ (since we standardize all the operational symbols "$\bowtie_i$" into "$=$" with slack variables while solving the problem, guessing the additional m linear independent equality constraints is equivalent to randomly choosing m_0 from $(m_i + m_0)$ constraints).

Furthermore, since the anonymization process should be prior to the matrix multiplication transformation, even though the adversary knows m_0 linearly independent equations, it is also impossible to figure out $\forall j \in [1, m_0], \alpha_{ij} \eta_{ij}$ in $(b_0^i)'$ and $\forall w \in [1, m_i], \sum_{j=1}^{m_0} h_{ijw} \eta_{ij}$ in b'_i because: (1) the coefficients of variables s_i in all constraints have been multiplied with unknown random numbers (thus the variable values computed from those linearly independent equations are no longer $\forall j \in [1, m_0], \eta_{ij}$), (2) s_i's random coefficients in A'_i: $\{\forall j \in [1, m_0], \alpha_{ij}\}$ and random coefficients in B'_i: $\{\forall w \in [1, m_i], \forall j \in [1, m_0], h_{ijw}\}$ are only known to who implements anonymization (the data owner). Hence, b_i and b_0^i can be secure against semi-honest adversaries. Algorithm 1 introduces the detailed steps of anonymizing b. Note that if any agent P_i would like to have a higher privacy guarantee, P_i can generate more artificial variables and more local equality constraints to anonymize both b_0^i and b_i. This improves privacy, since the probability of guessing the additional linearly independent constraints by the adversaries becomes even lower, but leads to a loss of efficiency, since the problem size increases (which is a typical tradeoff between privacy and efficiency).

4.3. Revised Dantzig–Wolfe decomposition

Dantzig–Wolfe decomposition was originally utilized to solve large-scale block-angular structured LP problems [7,37]. Actually, for all the K-LP problems, we

Algorithm 1: Right-hand side value b anonymization

/* A_i^j, $(A_i^j)'$, B_i^j and $(B_i^j)'$ denote the jth row of A_i, A_i', B_i and B_i', respectively */

1 **forall the** *agent* $P_i, i \in [1, K]$ **do**

2 generates m_0 pairs of random numbers $\forall j \in [1, m_0], \eta_{ij}, \alpha_{ij}$;

3 initializes m_0 artificial variables $s_i = \{s_{i1}, \ldots, s_{im_0}\}$ where $\forall j \in [1, m_0], s_{ij} = \eta_{ij}$;

4 **for** *the* j*th global constraint* ($j \in [1, m_0]$) **do**

5 $(A_i^j)' x_i' \leftarrow A_i^j x_i + \alpha_{ij} s_{ij}$;

6 $(b_0^i)_j' \leftarrow (b_0^i)_j + \alpha_{ij} \eta_{ij}$;

7 **for** *the* w*th local constraint* $B_i^w x_i \bowtie_i^w b_i^w$ *in* $B_i x_i \bowtie_i b_i$ ($w \in [1, m_i]$) **do**

8 generates a linear equation using all the artificial variables $\forall j \in [1, m_0], s_{ij} \in s_i$: $\sum_{j=1}^{m_0} h_{ijw} s_{ij} = \sum_{j=1}^{m_0} h_{ijw} \eta_{ij}$ where $\{\forall j \in [1, m_0], h_{ijw}\}$ are random numbers;

9 $B_i^w x_i \bowtie_i^w b_i^w \longleftarrow B_i^w x_i + \sum_{j=1}^{m_0} h_{ijw} s_{ij} \bowtie_i^w b_i^w + \sum_{j=1}^{m_0} h_{ijw} \eta_{ij}$;

10 generates m_0 (or more) local equality constraints with random coefficients: $\forall k \in [1, m_0], \sum_{j=1}^{m_0} r_{ijk} s_{ij} = \sum_{j=1}^{m_0} r_{ijk} \eta_{ij}$ where $m_0 \times m_0$ random numbers $\forall r_{ijk}$ guarantee m_0 linear independent equality constraints for m_0 artificial variables s_i [11];

11 union $B_i x_i \bowtie_i b_i$ (anonymized in step 10) and m_0 linear independent local equality constraints $\forall k \in [1, m_0], \sum_{j=1}^{m_0} r_{ijk} s_{ij} = \sum_{j=1}^{m_0} r_{ijk} \eta_{ij}$ to get the updated local constraints $B_i' x_i' \bowtie_i b_i'$;

/* for every global constraint, we can create more than one artificial variable to obtain more rigorous privacy guarantee */

can appropriately partition the constraints into block-angular structure (as shown in Eq. (1)). Specifically, we can consider each agent P_i ($i \in [1, K]$)'s horizontally partitioned local constraint matrix B_i as a block. By contrast, every vertically partitioned constraint that is shared by at least two agents can be regarded as a global constraint (each agent P_i holds a share A_i in the constraint matrix of all the vertically partitioned global constraints). Even if A_i may have more rows than B_i, the constraints are still block-angular partitioned.

Furthermore, after locally anonymizing b and transforming the blocks, each agent still has its local constraints block $B_i' Q_i$ and the global constraints share $A_i' Q_i$. Hence, we can solve the transformed K-LP problem using Dantzig–Wolfe decomposition. We thus denote the entire process as revised Dantzig–Wolfe decomposition.

Definition 8 (Revised Dantzig–Wolfe decomposition). A secure and efficient approach to solving K-LP problems that includes the following stages: anonymizing b

by each agent, transforming blocks by each agent and solving the transformed K-LP problem using Dantzig–Wolfe decomposition.

According to Eq. (4), the Dantzig–Wolfe representation of the transformed K-LP problem is:

$$\max \sum_j c_1'^{\mathrm{T}} Q_1 y_1^j \lambda_{1j} + \cdots + \sum_j c_K'^{\mathrm{T}} Q_K y_K^j \lambda_{Kj}$$

$$\text{s.t.} \begin{cases} \sum\limits_{\forall j} A_1' Q_1 y_1^j \lambda_{1j} + \cdots + \sum\limits_{\forall j} A_K' Q_K y_K^j \lambda_{Kj} \bowtie_0 b_0', \\ \sum\limits_{\forall j} \delta_{1j} \lambda_{1j} = 1, \\ \qquad\qquad \ddots \\ \qquad\qquad\qquad \sum\limits_{\forall j} \delta_{Kj} \lambda_{Kj} = 1, \\ \lambda_1 \in \mathbb{R}^{|E_1'|}, \ldots, \lambda_K \in \mathbb{R}^{|E_K'|}, \delta_{ij} \in \{0,1\}, i \in [1,K], \end{cases} \tag{9}$$

where $\forall i \in [1,K], c_i \subseteq c_i', A_i \subseteq A_i', B_i \subseteq B_i'$ (c_i', A_i', B_i' are expanded from c_i, A_i, B_i for anonymizing b, note that the coefficient of all the artificial variables in c_i' is equal to 0), and $|E_1'|, \ldots, |E_K'|$ represent the number of each agent's extreme points/rays in the transformed problem. In sum, Fig. 2 depicts the three steps of revised Dantzig–Wolfe decomposition.

Furthermore, after solving the problem, each agent P_i should obtain an optimal solution $\lambda_i = \{\forall j, \lambda_{ij}\}$. Figure 3 shows the process of deriving each agent's optimal solution for the original K-LP problem. Specifically, in step 1, the optimal solutions for each agent's transformed problem can be derived by computing the convexity combination of all extreme points/rays y_i^j: $y_i = \sum_{\forall j} \lambda_{ij} y_i^j$. In step 2 ($x_i' = Q_i y_i$),[4] the optimal solution of the original problem with anonymized b can be derived by left multiply Q_i for each agent (Theorem 3). In step 3, each agent can extract its individual optimal solution in the K-LP problem by excluding the artificial variables (for anonymizing b) from the optimal solution of x_i'.

At first glance, we can let an arbitrary agent formulate and solve the transformed master problem (Eq. (9)). However, the efficiency and security is not good enough for large-scale problems since the number of extreme points/rays are approximately $\frac{n_i'!}{m_i'!(n_i'-m_i')!}$ (choosing n_i' basis variables out of m_i' total variables) for each agent

[4] Apparently, if $y_i = 0$ and we have $x_i' = Q_i y_i$, x_i' should be 0 and revealed to other agents. However, y_i includes some transformed variables that is originally the value-fixed but unknown artificial variables for anonymizing b. Hence, x_i' cannot be computed due to unknown Q_i and non-zero y_i (the situation when the optimal solution in y_i is 0, is not known to the holder other than P_i), and this possible privacy leakage can be resolved.

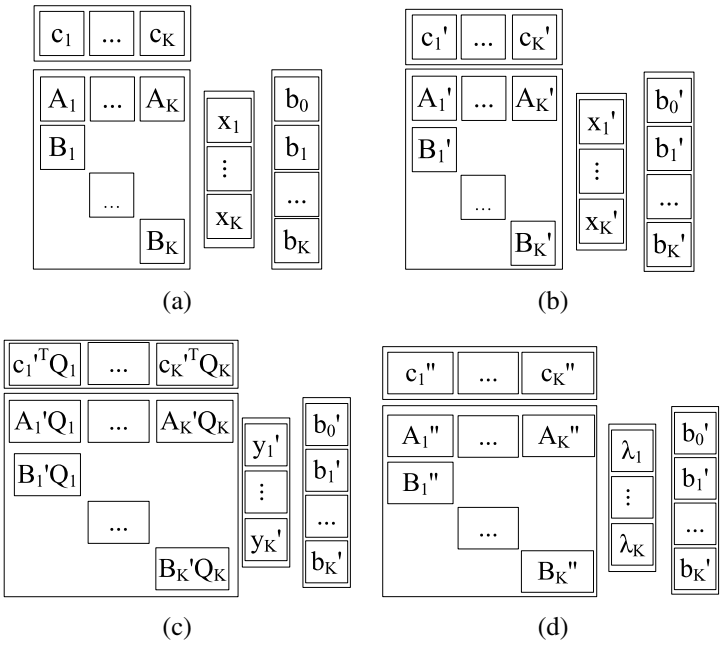


Fig. 2. Transformation steps in revised Dantzig decomposition.

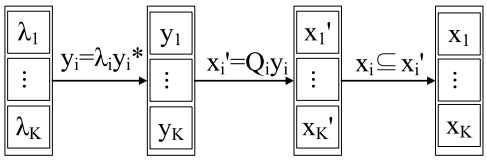


Fig. 3. Optimal solution reconstruction after solving the transformed K-LP problem.

and all the extreme points/rays should be sent to the master problem solver (assuming that the K-LP problem is standardized with slack variables before running Algorithm 1 and transformation). In Section 5, we introduce our secure K-agent Column Generation (SCG) Protocol that is based on the column generation algorithm of Dantzig–Wolfe decomposition [7,37] – iteratively formulating restricted master problems (the master problem/Eq. (9) with some variables in λ equal to 0) and pricing problems (the optimal solutions/columns of the pricing problems are sent to the master problem for improving the objective value) until global optimum of the restricted master problem is achieved (the detailed protocol and analysis are given later on).

To sum up, our revised Dantzig–Wolfe decomposition inherits all the features of the standard Dantzig–Wolfe decomposition. Thus, the advantages of this sort of revised decomposition and the corresponding secure K-agent Column Generation (SCG) Protocol are: (1) the decomposed smaller pricing problems can be formulated and solved independently (this improves the efficiency of solving large-scale LP problems); (2) the master problem solver does not need to get into the details on how the proposals of the pricing problems are generated (this makes it easier to preserve privacy if the large problem could be solved without knowing the precise contents of the pricing problems); (3) if the pricing problems have special structure (e.g., perhaps one is a transportation problem) then those specialized problem solving techniques can be used.

5. Secure protocol in semi-honest model

Solving K-LP by revised Dantzig–Wolfe decomposition and column generation is fair to all K agents. Hence, we assume that an arbitrary agent can be the master problem solver. As discussed in Section 4.3, it is inefficient to solve the full master problem due to huge number of extreme points/rays for each agent. In this section, we present an efficient protocol – the secure column generation (SCG) for revised Dantzig–Wolfe decomposition in semi-honest model. We also analyze the security and computation/communication cost for the secure protocol.

As mentioned in Section 4, the full master problem in the revised Dantzig–Wolfe decomposition includes $\sum_{i=1}^{K} \frac{n_i'!}{m_i'!(n_i'-m_i')!}$ variables. However, it is not necessary to involve all the extreme points/rays simply because a fairly small number of constraints in the master problem might result in many non-basis variables in the full master problem. Hence, restricted master problem (RMP) of the transformed K-LP problem is introduced to improve efficiency.

We let

$$[c_i] = \left(\forall j \in \left[1, \frac{n_i'!}{m_i'!(n_i' - m_i')!} \right], c_i'^{\mathrm{T}} Q_i y_i^j \right)$$

and

$$[A_i] = \left(\forall j \in \left[1, \frac{n_i'!}{m_i'!(n_i' - m_i')!} \right], A_i' Q_i y_i^j \right).$$

For RMP, we denote the coefficients in the master problem restricted to $\mathbb{R}^{|\widehat{E_1}|}, \ldots, \mathbb{R}^{|\widehat{E_K}|}$ ($|\widehat{E_1}|, \ldots, |\widehat{E_K}|$ represent each agent's number of extreme points/rays in current RMP) as $\widehat{c_i}$, $\widehat{A_i}$, $\widehat{y_i}$, $\widehat{\delta}$ and $\widehat{\lambda}$. Specifically, some of the variables in λ for all agents are initialized to non-basis 0. τ_i denotes the number of extreme points/rays in P_i's

pricing problem that has been proposed to the master solver where $\forall i \in [1, K], \tau_i \leqslant \frac{n_i'!}{m_i'!(n_i'-m_i')!}$. Hence, we represent the RMP as below:

$$\max \widehat{c_1}^{\mathrm{T}}\widehat{\lambda_1} + \cdots + \widehat{c_K}^{\mathrm{T}}\widehat{\lambda_K}$$
$$\text{s.t.} \begin{cases} \widehat{A_1}\widehat{\lambda_1} + \cdots + \widehat{A_K}\widehat{\lambda_K} \bowtie_0 b_0', \\ \sum_{j=1}^{\tau_1} \delta_{1j}\lambda_{1j} = 1, \\ \qquad \ddots \\ \sum_{j=1}^{\tau_K} \delta_{Kj}\lambda_{Kj} = 1, \\ \lambda_1 \in \mathbb{R}^{|\widehat{E_1}|}, \ldots, \lambda_K \in \mathbb{R}^{|\widehat{E_K}|}, \delta_{ij} \in \{0, 1\}. \end{cases} \tag{10}$$

In the general form of column generation algorithm [7,37] among distributed agents (without privacy concern), every agent iteratively proposes local optimal solutions to the RMP solver for improving the global optimal value while the RMP solver iteratively combines all the proposed local optimal solutions together and computes the latest global optimal solution. Specifically, the column generation algorithm repeats following steps until global optimum is achieved: (1) the master problem solver (can be any agent, e.g., P_1) formulates the restricted master problem (RMP) based on the proposed optimal solutions of the pricing problems from all agents $P_1, \ldots, P_K$ (the optimal solution of any pricing problem is also a proposed column of the master problem and an new extreme point or ray of the pricing problem); (2) the master problem solver (e.g., P_1) solves the RMP and sends the dual values of the optimal solution to every pricing problem solver $P_1, \ldots, P_K$; (3) every pricing problem solver thus formulates and solves a new pricing problem with its local constraints $B_i x_i \bowtie_i b_i$ (only the objective function varies based on the received dual values) to get the new optimal solution/column/extreme point or ray (which should be possibly proposed to the master problem solver, e.g., P_1 to improve the objective value of the RMP).

To improve the privacy protection, we can let each agent's pricing problems be solved by another arbitrary agent since other agents cannot apply inverse transformation to learn additional information. If this is done, more information can be hidden (e.g., the solutions and the dual optimal values of the real pricing problems) while iteratively formulating and solving the RMPs and pricing problems. Without loss of generality (for simplifying notations), we assume that P_1 solves the RMPs, P_i sends $A_i'Q_i$, $B_i'Q_i$, $c_i'^{\mathrm{T}}Q_i$, $(b_0^i)'$ and b_i' to the next agent $P_{(i \bmod K)+1}$ and $P_{(i \bmod K)+1}$ solves P_i's pricing problems (P_1 solves P_K's pricing problems).

Algorithm 2: Solving the RMP by an arbitrary agent

Input : K agents $P_1, \ldots, P_K$ where $\forall i \in [1, K]$, $P_{(i \bmod K)+1}$ holds P_i's variables y_i, transformed matrices $A'_i Q_i, B'_i Q_i, c'^{\mathrm{T}}_i Q_i$ and anonymized vectors $b'_i, (b^i_0)'$

Output: the optimal dual solution of RMP $(\pi^*, \mu^*_1, \ldots \mu^*_K)$ or infeasibility

/* Agent P_i's τ_i pairs of extreme points/rays ($\forall j \in [1, \tau_i]$, $y^j_i \in \widehat{y_i}$) and variables $\lambda_{ij} \in \widehat{\lambda_i}$ have been sent to P_1 from $P_{(i \bmod K)+1}$ ($s_i \leqslant \frac{n'_i!}{m'_i!(n'_i - m'_i)!}$ and P_1 solves the RMP) */

1 P_1 constructs the objective of the RMP: $\sum_{i=1}^{K} \sum_{j=1}^{\tau_i} c'^{\mathrm{T}}_i Q_i y^j_i \lambda_{ij}$;

2 P_1 constructs the constraints of the RMP: $\sum_{i=1}^{K} \sum_{j=1}^{\tau_i} A'_i Q_i y^j_i \lambda_{ij} \bowtie^j_0 \sum_{i=1}^{K} (b^i_0)'$ and $\forall i \in [1, K]$, $\sum_{j=1}^{\tau_i} \delta_{ij} \lambda_{ij} = 1$ where $b_0 = \sum_{i=1}^{K} (b^i_0)'$ can be securely summed or directly summed by K agents;

3 P_1 solves the above problem using Simplex or Revised Simplex method.

5.1. Solving restricted master problem (RMP) by an arbitrary agent

The detailed steps of solving RMP in revised Dantzig Wolfe decomposition are shown in Algorithm 2. We can show that solving RMP is secure in semi-honest model.

Lemma 1. *Algorithm* 2 *reveals at most*:

- *the revised DW representation of the K-LP problem*;
- *the optimal solution of the revised DW representation*;
- *the total payoffs* (*optimal value*) *of each agent*.

Proof. RMP is a special case of the full master problem where some variables in λ_i are fixed to be non-basis (not proposed to the RMP solver P_1). Hence, the worst case is that all the columns of the master problem are required to formulate the RMP where P_1 can obtain the maximum volume of data from other agents (though this scarcely happens). We thus discuss the potential privacy leakage in this case.

We look at the matrices/vectors that are acquired by P_1 from all other agents P_i where $\forall i \in [1, K]$. Specifically, $[c_i] = (\forall j \in [1, \frac{n'_i!}{m'_i!(n'_i - m'_i)!}], c'^{\mathrm{T}}_i Q_i y^j_i)$ and $[A_i] = (\forall j \in [1, \frac{n'_i!}{m'_i!(n'_i - m'_i)!}], A'_i Q_i y^j_i)$ should be sent from P_i to P_1. At this time, $[c_i]$ is a vector with size $\frac{n'_i!}{m'_i!(n'_i - m'_i)!}$ and $[A_i]$ is a $m_0 \times \frac{n'_i!}{m'_i!(n'_i - m'_i)!}$ matrix. The jth value in $[c_i]$ is equal to $c'^{\mathrm{T}}_i Q_i y^j_i$, and the jth column in matrix $[A_i]$ is equal to $A'_i Q_i y^j_i$.

Since P_1 does not know y_i^j and Q_i, it is impossible to calculate or estimate the (size n_i') vector c_i' and sub-matrices A_i and B_i. Specifically, even if P_1 can construct $(m_0+1)\cdot\frac{n_i'!}{m_i'!(n_i'-m_i')!}$ non-linear equations based on the elements from $[c_i]$ and $[A_i]$, the number of unknown variables in the equations (from c_i', A_i', Q_i and $\forall j \in [1, \frac{n_i'!}{m_i'!(n_i'-m_i')!}], y_i^j$) should be $n_i' + m_0 n_i' + n_i'^2 + n_i' \cdot \frac{n_i'!}{m_i'!(n_i'-m_i')!}$. Due to $n_i' \gg m_0$ in linear programs, we have $n_i' + m_0 n_i' + n_i'^2 + n_i' \cdot \frac{n_i'!}{m_i'!(n_i'-m_i')!} \gg (m_0+1)\cdot\frac{n_i'!}{m_i'!(n_i'-m_i')!}$. Thus, those unknown variables in c_i', A_i', Q_i[5] and $\forall j \in [1, \frac{n_i'!}{m_i'!(n_i'-m_i')!}], y_i^j$ cannot be derived from those non-linear equations ($\forall j \in [1, \frac{n_i'!}{m_i'!(n_i'-m_i')!}], y_i^j$ constitute the private sub-polyhedron $B_i x_i \bowtie_i b_i$). As a result, P_1 learns nothing about A_i, c_i, b_0^i (anonymized) and $B_i x_i \bowtie_i b_i$ from any agent P_i.

By contrast, while solving the problem, P_1 formulates and solves the RMPs. P_i thus knows the primal and dual solution of the RMP. In addition, anonymizing b and transforming c_i, A_i and B_i does not change the total payoff (optimal value) of each agent, the payoffs of all values are revealed to P_1 as well. Since we can let arbitrary agent solve any other agent's pricing problems, the payoff owners can be still unknown to the RMP solver (we can permute the pricing solvers in any order).

Therefore, solving the RMPs is secure – the private constraints and the meanings of the concrete variables cannot be inferred based on the above limited disclosure. □

5.2. Solving pricing problems by peer-agent

While solving the K-LP problem by the column generation algorithm (CGA), in every iteration, each agent's pricing problem might be formulated to test whether any column of the master problem (extreme point/ray of the corresponding agent's pricing problem) should be proposed or not. If any agent's pricing problem cannot propose column to the master solver in the previous iterations, no pricing problem is required for this agent any further. As discussed in Section 4.1, we permute the pricing problem owners and the pricing problem solvers where private information can be protected via transformation. We now introduce the details of solving pricing problems and analyze the potential privacy loss.

Assuming that an honest-but-curious agent $P_{(i \bmod K)+1}$ ($i \in [1, K]$) has received agent P_i's variables y_i, transformed matrices/vector $A_i'Q_i, B_i'Q_i, c_i'^{\mathrm{T}}Q_i$ and the anonymized vectors $b_i', (b_0^i)'$ (as shown in Fig. 2(c)). Agent $P_{(i \bmod K)+1}$ thus formulates and solves agent P_i's pricing problem.

[5] As described in [4], Q_i should be a monomial matrix, thus Q_i has n_i' unknown variables located in $n_i'^2$ unknown positions.

More specifically, in every iteration, after solving RMP (by P_1), P_1 sends the optimal dual solution (π^*, μ_i^*) to $P_{(i \bmod K)+1}$ if the RMP is feasible. The reduced cost d_{ij} of variable λ_{ij} for agent P_i can be derived as:

$$d_{ij} = (c_i'^{\mathrm{T}} Q_i - \pi^* A_i' Q_i) y_i^j - \begin{cases} (\mu_i^*)_j & \text{if } y_i^j \text{ is an extreme point (vertex),} \\ 0 & \text{if } y_i^j \text{ is an extreme ray.} \end{cases} \tag{11}$$

Therefore, $P_{(i \bmod K)+1}$ formulates P_i the pricing problem as below:

$$\max(c_i'^{\mathrm{T}} Q_i - \pi^* A_i' Q_i) y_i \quad \text{s.t.} \begin{cases} B_i' Q_i y_i \bowtie b_i', \\ y_i \in \mathbb{R}^{n_i'}. \end{cases} \tag{12}$$

Algorithm 3 presents the detailed steps of solving P_i's pricing problem by $P_{(i \bmod K)+1}$ which is secure.

Lemma 2. *Algorithm* 3 *reveals* (*to* $P_{(i \bmod K)+1}$) *only*:

- *the feasibility of* P_i*' block sub-polyhedron* $B_i x_i \bowtie_i b_i$;
- *dual optimal values* (π, μ_i) *of the RMP.*

Proof. Since we can let another arbitrary peer-agent solve any agent's pricing problems (fairness property): assuming that $\forall i \in [1, K]$, $P_{(i \bmod K)+1}$ solves P_i's pricing problems. Similarly, we first look at the matrices/vectors acquired by $P_{(i \bmod K)+1}$ from P_i: size n_i' vector $c_i'^{\mathrm{T}} Q$, $m_i' \times n_i'$ matrix $B_i' Q_i$ and $m_0 \times n_i'$ matrix $A_i' Q_i$. The jth value in $c_i'^{\mathrm{T}} Q_i$ is equal to $c_i'^{\mathrm{T}} Q_i^j$ (Q_i^j denotes the jth column of Q_i), and the value of the kth row and the jth column in $A_i' Q_i$ (or $B_i' Q_i$) is equal to the scalar product of the kth row of A_i' (or B_i') and Q_i^j.

Since $P_{(i \bmod K)+1}$ does not know Q_i, it is impossible to calculate or estimate the (size n_i') vector c_i' and matrices A_i' (or A_i) and B_i' (or B_i). Specifically, even if $P_{(i \bmod K)+1}$ can construct $(m_0 + m_i' + 1)n_i'$ non-linear equations based on the elements from $c_i'^{\mathrm{T}} Q_i$, $A_i' Q_i$ and $B_i' Q_i$, the number of unknown variables in the equations (from c_i', A_i', B_i and Q_i) should be $n_i' + m_0 n_i' + m_i' n_i' + n_i'$. Due to $n_i' \gg 0$ in linear programs, we have $n_i' + m_0 n_i' + m_i' n_i' + n_i' \gg (m_0 + m_i' + 1)n_i'$. Thus, those unknown variables in c_i', A_i', B_i and Q_i cannot be derived from the non-linear equations.[6]

[6]Bednarz et al. [4] proposed a possible attack on inferring Q with the known transformed and original objective vectors ($c^{\mathrm{T}} Q$ and c^{T}) along with the known optimal solutions of the transformed problem and the original problem ($y*$ and $x* = Qy*$). However, this attack only applies to the special case of DisLP in Vaidya's work [39] where one party holds the objective function while the other party holds the constraints. In our protocol, P_i sends $c_i'^{\mathrm{T}} Q_i$ to $P_{(i \bmod K)+1}$, but $c_i'^{\mathrm{T}}$ is unknown to $P_{(i \bmod K)+1}$, hence it is impossible to compute all the possibilities of Q_i by $P_{(i \bmod K)+1}$ in terms of Bednarz's approach. In addition, the original solution is not revealed as well. It is impossible to verify the exact Q_i by $P_{(i \bmod K)+1}$ following the approach in [4].

Algorithm 3: Solving pricing problem by peer-agent

Input : An honest-but-curious agent $P_{(i \bmod K)+1}$ holds agent P_i's variables y_i, transformed matrices/vector $A_i'Q_i$, $B_i'Q_i$, $c_i'^{\mathrm{T}}Q_i$ and vectors b_i', $(b_0^i)'$ (anonymized)

Output: New Optimal Solution or Infeasibility of the Pricing Problem

/* P_1 solves the RMPs and sends π^* and μ_i^* to $P_{(i \bmod K)+1}$ */

$P_{(i \bmod K)+1}$ constructs the objective function of the pricing problem of P_i: $z_i = [c_i'^{\mathrm{T}}Q_i - \pi^*(A_i'Q_i)]y_i$;

$P_{(i \bmod K)+1}$ constructs the constraints: $B_i'Q_iy_i \bowtie_i b_i'$;

$P_{(i \bmod K)+1}$ solves the above pricing problem using Revised Simplex algorithm or a specialized algorithm if the problem has specific structure (e.g. transportation problem);

if *the problem is infeasible* **then**

  the original problem is infeasible and return;

switch *the optimal value* z_i^* **do**

  case *is bounded and* $z_i^* > \mu_i$

    /* this optimal extreme point's corresponding variable (λ_{ij}^*) in the master problem is a candidate to enter the basis of the RMP */

    $P_{(i \bmod K)+1}$ sends the DW represented new variable and coefficients $(c_i'^{\mathrm{T}}Q_iy_i^j\lambda_{ij}^*, A_i'Q_iy_i^j\lambda_{ij}^*)$ to P_1;

  case *is unbounded*

    /* this extreme ray's corresponding variable (λ_{ij}^*) in the master problem is a candidate to enter the basis of the RMP */

    $P_{(i \bmod K)+1}$ sends the DW represented new variable and coefficients $(c_i'^{\mathrm{T}}Q_iy_i^j\lambda_{ij}^*, A_i'Q_iy_i^j\lambda_{ij}^*)$ to P_1;

  case $z_i^* \leqslant \mu_i$

    no extra variable (λ_{ij}) from P_i enters the basis of the RMP;

Hence, $P_{(i \bmod K)+1}$ learns nothing about A_i, B_i, c_i, b_0^i (anonymized) and b_i (anonymized) from P_i if $P_{(i \bmod K)+1}$ solves P_i's pricing problems.

By contrast, before solving the pricing problem, $P_{(i \bmod K)+1}$ should acquire the some dual optimal values of the RMP (only π and μ_i). $P_{(i \bmod K)+1}$ thus knows the dual optimal solution of the RMP related to the convexity combination represented global constraints (π) and the constraints $\sum_{\forall j} \delta_{ij}\lambda_{ij} = 1$ (μ_i). However, $P_{(i \bmod K)+1}$ cannot learn the true pricing problem since everything in the K-LP has

been transformed. Furthermore, if the polyhedron $B_i'Q_iy_i \bowtie_i b_i'$ is infeasible, we have: polyhedron $B_i'x_i \bowtie_i b_i'$ is also infeasible (Theorem 4). Hence, the specific agent with the infeasible local constraints should be spotted (indeed, this should be revealed in any case). However, the private constraints and the meanings of the concrete variables cannot be inferred with this information (for more rigorous privacy protection, we can randomly permute the agents).

Hence, solving the pricing problems by another arbitrary agent is secure. □

Theorem 4. *The polyhedra $B_ix_i \bowtie_i b_i$ and $B_iQ_iy_i \bowtie_i b_i$ have the same feasibility where $i \in [1, K]$.* (*Proven in Appendix* A.2.)

5.3. Secure K-agent Column Generation (SCG) protocol

Algorithms 2 and 3 illustrate the detailed steps of one iteration while solving the transformed DW representation of the K-LP problem using column generation algorithm [7,37]. In a K-agent distributed computing environment, the RMP solver will handle the revised DW represented global constraints by asking the pricing problem solvers for proposals. The master problem solver will choose a combination of proposals that maximizes global profits while meeting all the constraints. Algorithm 4 presents the secure K-agent column generation (SCG) protocol for securely solving K-LP problem with revised Dantzig–Wolfe decomposition.

Also, for better illustrating the secure K-agent column generation protocol (Algorithm 4), we present a protocol overview in Fig. 4 where steps 1–4 in the figure represent:

(1) P_1 initializes/formulates and solves a RMP problem.
(2) P_1 distributes dual values (π, μ_i) to $P_{(i \bmod K)+1}$.
(3) $P_{(i \bmod K)+1}$ solves P_i's pricing problems.
(4) $P_{(i \bmod K)+1}$ proposes P_i's optimal solution of the latest pricing problem/ column to P_1 if necessary.

Practically, the main drawback of this approach is in possible convergence problems. Normally, this method gets very good answers quickly, but it generally requires a lot of time to find the exact optimal solution. This is due to the fact that the number of columns/variables after transformation is significantly greater than the number of variables in the original problem. Therefore, after several rounds, the improvement in the solution starts plateauing out. The subproblems may continue to generate proposals only slightly better than the ones before. Thus, we might have to stop with a near-optimal solution for efficiency reasons if necessary [37]. Specifically, if the RMP is feasible and the pricing problems are all feasible and bounded, P_1 can calculate a new upper bound (dual value) of the master problem $\hat{z} = z^* + \sum_{i=1}^{K}(z_i^* - \mu_i)$. If $\hat{z} < \bar{z}^*$, update the best known dual value $\bar{z}^* \leftarrow \hat{z}$. P_1 thus compute the optimal gap $d = \bar{z}^* - z^*$ and the relative optimal gap $d' = \frac{d}{1+|z^*|}$. If the gap is tolerable, we stop the protocol where the optimal solution of the current RMP is near-optimal.

Algorithm 4: Secure K-agent Column Generation (SCG) protocol

/* P_1 solves the RMPs and $P_{(i \bmod K)+1}$ solves P_i's pricing problems where $i \in [1, K]$. Thus, each agent's share in the transformed K-LP problem has been delivered to the corresponding pricing problem solver. */

forall the *agent* $P_{(i \bmod K)+1}, i \in [1, K]$ **do**
- computes all the extreme points/rays and its convexity coefficient λ_{ij} in polyhedron $B_i' Q_i y_i \bowtie_i b_i'$;
- chooses $\bar{E}_i \subseteq E_i$ and initialize the best known dual value of the master problem $\bar{z}^* = \infty$ (a least upper bound);
- computes $A_i' Q_i y_i^j$ and $c_i'^{\mathrm{T}} Q_i y_i^j$ and sends $A_i' Q_i y_i^j \lambda_{ij}$, $c_i'^{\mathrm{T}} Q_i y_i^j \lambda_{ij}$ and $(b_0^i)'$ to P_1;

P_1 forms the RMP and solve it (Algorithm 2, z^* is the objective function value if the RMP is feasible);

if *RMP is feasible* **then**
- P_1 sends π^* and μ_i^* to $P_{(i \bmod K)+1}$ ($\forall i \in [1, K]$);

else
- the K-LP problem is infeasible and return;

forall the *agent* $P_i, i \in [1, K]$ **do**
- solves the pricing problem, tests the optimal improvement and sends a new column (extreme point/ray) to P_1 if necessary (Algorithm 3, the optimal value z_i^* is also sent to P_1 if near-optimal test is specified);

if *a new column is proposed to* P_1 **then**
- go to step 5;

else
- /* the optimal solution of the current RMP is the optimal solution of the master problem */
- P_1 sends the share of the optimal solution $\forall i \in [1, K]$, λ_i to agent P_i;
- **forall the** *agent* $P_i, i \in \{1, 2, \ldots, K\}$ **do**
  - derive its solution x_i (global optimum) by λ_i (as shown in Fig. 3);

/* apply near-optimal solution test if necessary */

In case of near-optimal tolerance, all the optimal values of the pricing problems $\forall i \in [1, K]$, z_i^* should be sent to P_1 along with the proposed column. However, the protocol is still secure in semi-honest model.

Theorem 5. *The K-agent column generation protocol is secure in semi-honest model.*

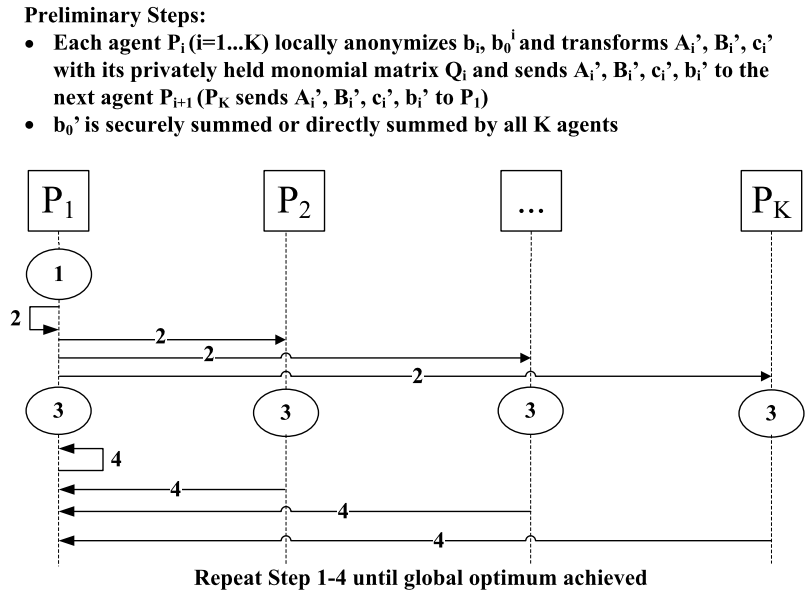


Fig. 4. Secure K-agent column generation protocol.

Proof. As proven in Lemmas 1 and 2, solving RMPs and pricing problems is secure for all K honest-but-curious agents. Since our K-agent column generation protocol the repeated steps of solving transformed RMPs and pricing problems, it is straightforward to show that the protocol is secure against semi-honest adversaries. □

5.4. Computation and communication cost analysis

Our secure column generation protocol is mainly based on local transformation rather than cryptographic encryption that dominates the cost in most of the privacy-preserving distributed linear programming techniques [9,19,39,40]. Hence, our approach significantly outperforms the above work on both computation and communication costs, especially in large-scale K-LP problems.

Computation cost. We discuss the computation cost in two facts: transforming the K-LP problem and solving the transformed problem with column generation. Note that the size of the constraints matrix (all the constraints) should be $(m_0 + \sum_{i=1}^{K} m_i) \times \sum_{i=1}^{K} n_i$. After anonymizing b, the constraint matrix is enlarged to $(m_0 + \sum_{i=1}^{K} m'_i) \times \sum_{i=1}^{K} n'_i$. On one hand, only one-time $(m_0 + m'_i + 1)n'_i$ scalar product computation is required for each agent P_i to transform A'_i, B'_i and c' since anonymizing b only incurs nominal computational cost (generating random numbers and equations). On the other hand, for large-scale block-angular structured LP problems, column generation algorithm has been proven to be more efficient than directly

applying some standard methods to solve the problem (i.e., simplex or revised simplex method) [25,37]. Overall, our secure K-agent column generation (SCG) protocol requires similar computation cost as solving the centralized K-LP problem with column generation (which is extremely efficient).

Communication cost. Similarly, we analyze the communication cost in two facts: sending transformed matrices/vectors and solving the transformed problem with column generation. On one hand, after every agent transforms its shares of the K-LP problem, P_i ($i \in [1, K]$) sends $A_i'Q_i, B_i'Q_i, c'^{\mathrm{T}}Q_i$, $(b_0^i)'$ and b_i' to another agent, thus the communication complexity is $O(K(m_0 + m_i' + 1)n_i' + m_0 + m_i') \approx O(K(m_0 + m_i')n_i')$ if assuming that every number requires $O(1)$ bits. In every iteration, one round communication between the RMP solver and every agent (pricing problem solver) is called to deliver the dual values and the proposed columns, thus the communication complexity is $O(Kt(m_0 + n_i'))$ if assuming the number of iterations as t. Indeed, to seek the near-optimal solution (t is generally bounded with fast convergence), the communication overheads are rather small and can be ignored in practical situations.

6. Incentive compatible protocol in malicious model

The secure K-agent column generation (SCG) protocol protects each agent's private information against honest-but-curious agents. However, in reality, agents involved in the DisLP problem cannot be guaranteed to follow the protocol. For instance, a selfish agent may attempt to modify the protocol (e.g., sending fake messages to other agents) for improving its payoff and sacrificing other agents' payoffs. In this section, we introduce an incentive compatible protocol to securely solve the K-LP problems against malicious agents.

6.1. Malicious behavior in SCG protocol

While executing the secure K-agent column generation (SCG) protocol, the restricted master problem (RMP) solver and the pricing problems solvers might be the potential cheater – feigning the messages in the protocol as shown in Fig. 4. We now discuss some major potential malicious behavior by examining the messages exchanged among the RMP solver and the pricing problem solvers.

(1) *Dishonest RMP solver*. If the restricted master problem (RMP) solver (e.g., P_1) cheats in the SCG protocol, two categories of fake messages can be generated.

First, after the global optimum is achieved, the RMP solver (P_1) may distribute a fake share of the optimal solution (e.g., P_i's optimal solution $\lambda_i = \{\forall j, \lambda_{ij}\}$) to the corresponding owner (e.g., P_i) where P_i's optimal value (payoff) is less than the true payoff. Thus, the RMP solver can create an increased payoff in the global optimal solution for itself (since the RMP solver computes and distributes the shares of the optimal solution to the corresponding owners, it can make arbitrary modification to

the shares in the global optimal solution). This is an explicit way of cheating to gain extra payoff for the RMP solver.

Second, while solving any RMP, the RMP solver (P_1) may distribute a fake dual values (π, μ_i) which is a share of the dual optimal solution of current RMP to $P_{(i \bmod K)+1}$ (note that π and μ_i in the dual optimal solution correspond to the global constraints and P_i's pricing problems, respectively). The fake dual values might result in – no further column is required to propose from P_i's pricing problems. This is an implicit way of reducing the payoff of other agents – even though the fake dual values may deviate the objective function of the pricing problems, any generated fake dual values cannot guarantee other agents' payoff decrease or increase (since the detailed pricing problems are unknown to the RMP solver).

(2) *Dishonest pricing problem solver*. If the pricing problem solver (e.g., $P_{(i \bmod K)+1}$ solves P_i's pricing problems) cheats in the SCG protocol, no matter what $P_{(i \bmod K)+1}$ does can be summarized to one kind of potential cheating, which is sending a fake optimal solution of the pricing problem (which is a proposed column) to the RMP solver (note that all the pricing problem solvers' potential malicious behavior such as modifying the pricing problem or directly modifying the local optimal solution result in the same possible outcome – the proposed optimal solutions are deviated).

(3) *Collusion*. Moreover, the K-agent SCG protocol may also suffer a substantial risk of a typical dishonest behavior – collusion as shown below:

- First, ℓ agents (either RMP solver or pricing problem solvers and $\ell < K$) can collude to infer additional information. For example, agents may collude with each other to learn its online optimal solutions in every iteration of the SCG protocol. Specifically, if P_2 solves P_1's pricing problems, P_2 may send the pricing problems (including the dual values) back to P_1 in every iteration of the SCG protocol. With known transformation matrix Q_1, the proposed columns are then revealed to P_1. This violates the protocol since additional information in the SCG protocol has been revealed.
- Second, ℓ (less than K) agents may realign a dishonest coalition to corrupt the SCG protocol by sending fake solutions/columns to other agents. The probability of detecting cheating from the coalition might be reduced since all the cheating implemented by the agents in the dishonest coalition are known to them (all the agents in the dishonest coalition can send identical fake messages to the recipients, thus it is even difficult to detect such cheating by the agents outside the dishonest coalition).

Our following incentive compatible protocol eliminates the above malicious behavior and enforce honesty while securely solving the K-LP problem.

6.2. Nash equilibrium in SCG protocol

The core idea of establishing an incentive compatible protocol is to force all rational participants to follow the protocol based on a Nash equilibrium – "all partic-

ipants play honestly". In the Nash equilibrium, no participant can gain more payoff by changing its strategy from "honest play" to "cheat".

6.2.1. Dominant strategy for Nash equilibrium

Assuming that any agent P_i ($i \in [1, K]$) might be the malicious agent, we denote the true payoff of P_i in the SCG protocol as T_i. Indeed, if any dishonest play/cheating is caught in a protocol, we can consider its payoff as "$c(T_i)$" (or 0) due to lost cooperation, reputation and so on (clearly, $c(T_i) < T_i$); otherwise, its payoff of the uncaught cheating may achieve a bounded value "$u(T_i)$" (this is bounded – since even if all agents' payoffs have been gained by such agent, the optimal value is still bounded by practical constraints). Moreover, we denote the probability that any cheating in the protocol is caught as ϵ. Thus, if we show that for all agents P_i ($i \in [1, K]$), strategy "honest play" *strictly dominates* "cheat" in the protocol (see Eq. (13)), "all participants play honestly" should be the only pure-strategy Nash equilibrium (in this paper, we only consider *pure strategy* which means choosing exactly one strategy from "honest play" and "cheat"; on the contrary, mixed strategy means a probabilistic strategy that combines "honest play" and "cheat" with a corresponding probability [13]).

$$T_i > \epsilon * c(T_i) + (1 - \epsilon) * u(T_i). \tag{13}$$

Thus, if the probability of cheating detection ϵ satisfies the following inequality, all rational participants will play honestly at the Nash equilibrium.

$$\forall i \in [1, K] \quad \epsilon > \frac{u(T_i) - T_i}{u(T_i) - c(T_i)}. \tag{14}$$

We denote the ratio $\frac{u(T_i)-T_i}{u(T_i)-c(T_i)}$ (less than 1) as "payoff ratio" of agent P_i which will be used to build cheating detection mechanism in the incentive compatible protocol.

6.2.2. Approach to establish Nash equilibrium

As discussed above, we have to ensure that the inequalities $\forall i \in [1, k]$, $T_i > \epsilon * c(T_i) + (1 - \epsilon) * u(T_i)$ hold to establish the honesty enforced Nash equilibrium. Since $\forall u(T_i)$ are bounded, the primary breakthrough point is to enhance the probability of cheating detection in the protocol. A naive approach to detect cheating (or increase the probability of cheating detection ϵ) is solving the K-LP problems by running the SCG protocol multiple times with distinct RMP solvers and pricing problem solvers for all agents in every time (see Example 3).

Example 3. Four agents P_1, P_2, P_3, P_4 jointly solve a K-LP problem. If we run the SCG protocol twice: in the first run, P_1 solves the RMP while P_3 solves P_2's pricing problems; in the second run, P_3 solves the RMP while P_1 solves P_2's pricing problems. If P_2's optimal solution is deviated due to a malicious behavior (by either P_1 or

P_3), P_2 can compare its share in two global optimal solutions and catch the cheating since agents P_1 and P_3 do not know the potential malicious behavior (on P_2's share of the K-LP problem) of each other.

Dishonest coalition. In the above example, the cheating can be discovered if P_1 and P_3 do not collude (the probability of cheating detection $\epsilon = 1$). Since solving the K-LP problem can be regarded as a *multi-player game*, some agents may collude with each other to play dishonestly as a coalition. Thus, the above cheating (by P_1 or P_3) cannot be caught if P_1 and P_3 form a dishonest coalition (the fake information for P_2 can be identical since they are generated by the agents in this dishonest coalition). However, we can increase the number of multiple SCG protocol run with distinct RMP and pricing problem solvers to improve the probability of cheating detection ϵ (since distinct RMP and pricing problem solvers other than the colluding agents P_1 and P_3 might be chosen, which assist P_2 to detect the optimal solution deviation resulting from the dishonest coalition of P_1 and P_3).

Assumption of catching cheating. Essentially, each agent can discover that any cheating exists among all the agents by comparing the optimal solutions in multiple SCG protocol run with distinct RMP/Pricing Problems solvers. However, the agent who plays dishonestly or the dishonest coalition cannot be detected (since every agent can only learn whether the received share in the N optimal solutions from different solvers are identical or not, rather than, which one is deviating). In fact, *catching the cheating as above is sufficient to enforce the honesty for solving the K-LP problem*. More specifically, we assume that if an arbitrary agent P_i, $i \in [1, K]$, discovers that any cheating exists in the system, P_i should quit the collaboration and mark all the remaining agents as potential cheaters. It is unnecessary to apply particular punishment to the actual cheaters for the following reason: as soon as all the agents with deviated share of the optimal solution quit the collaboration, the cheating will not be meaningful for those agents who play dishonestly or the dishonest coalition (no evident payoff can be gained from the cheating), and all the cheaters will definitely lose reputation (we can consider this as a punishment). Thus, even if the dishonest agent cannot be punished directly, the caught payoff $c(T_i)$ should be less than T_i, and we can still enforce the honesty by increasing the cheating detection probability ϵ with multiple SCG protocol run. In this paper, we follow this cheating caught assumption, and the concept "cheating is caught" means "any agent discovers an existing cheating in the protocol".

In remaining part of this section, we will present our approach to eliminate the malicious behavior by establishing the Nash equilibrium of "honest play" for all participants, assuming the largest dishonest coalition with no more than ℓ malicious agents ($\ell < K$).

6.3. Incentive compatible protocol

Two categories of collusion have been discussed in Section 6.1: colluding to infer additional private information, and cheat as a dishonest coalition. We first present an

transformation based on multiple agents to resolve the first collusion issue (prevent learning additional private information), and then present the incentive compatible protocol against malicious participants if the largest dishonest coalition include at most ℓ colluding agents.

6.3.1. Multi-agent transformation

Before running the SCG protocol, each agent P_i ($i \in [1, K]$) locally transforms their shares in the K-LP problem by anonymizing the right-hand side value of the constraints (b_i and b_0^i) and post-multiplying a privately held monomial matrix Q_i to matrices A_i', B_i' and vector $c_i'^{\mathrm{T}}$. Thus, if another agent (e.g., $P_{(i \bmod K)+1}$) solves P_i's pricing problems, any element in A_i', B_i' and $c_i'^{\mathrm{T}}$ cannot be inferred with known $A_i'Q_i$, $B_i'Q_i$ and $c_i'^{\mathrm{T}}Q_i$. Formally speaking, a monomial matrix Q_i is a generalized permutation matrix where there is exactly one non-zero element in each row and each column. The non-zero entry can be randomly generated in our revised DW transformation. For example, if P_i transforms A_i' with Q_i as below:

$$
\begin{aligned}
A_i' &= \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n_i'} \\ a_{21} & a_{22} & \dots & a_{2n_i'} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m_0 1} & a_{m_0 2} & \dots & a_{m_0 n_i'} \end{pmatrix} \in \mathbb{R}^{m_0 \times n_i'}, \\
Q_i &= \begin{pmatrix} 0 & 0 & \dots & 0 & r_{n_i'} \\ r_1 & 0 & \dots & 0 & 0 \\ 0 & r_2 & \dots & 0 & 0 \\ 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \dots & r_{n_i'-1} & 0 \end{pmatrix} \in \mathbb{R}^{n_i' \times n_i'},
\end{aligned} \tag{15}
$$

$$
A_i'Q_i = \begin{pmatrix} a_{12}r_1 & a_{13}r_2 & \dots & a_{11}r_{n_i'} \\ a_{22}r_1 & a_{23}r_2 & \dots & a_{21}r_{n_i'} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m_0 2}r_1 & a_{m_0 3}r_2 & \dots & a_{m_0 1}r_{n_i'} \end{pmatrix}. \tag{16}
$$

Thus, post-multiplying Q_i can be considered as two steps: (1) permute the columns of matrix A_i' (or B_i' and c'^{T}) according to the positions of the non-zero elements in Q_i (same as permutation matrix where there is exactly one non-zero element "1" in each row and each column) and (2) multiply a random number to every column of the column-permuted A_i', B_i' and c'^{T} (for the same column in A_i', B_i' and c'^{T}, e.g., the first column in A_i', B_i' and c'^{T}, the multiplied random numbers are identical; for different columns in A_i', B_i' or c'^{T}, the multiplied random numbers are distinct).

As discussed earlier, if $P_{(i \bmod K)+1}$ solves P_i's pricing problems, $P_{(i \bmod K)+1}$ cannot learn any element in A_i', B_i' and $c_i'^{\mathrm{T}}$ (Bednarz's attack [4] can be also avoided),

and P_i cannot learn any information with regard to its pricing problems ($P_{(i \bmod K)+1}$ cannot learn it either since the pricing problems have been transformed by Q_i). However, if $P_{(i \bmod K)+1}$ colludes with P_i, $P_{(i \bmod K)+1}$ can deliver everything regarding P_i's transformed pricing problems to P_i. Thus, P_i can learn everything in its original pricing problems (e.g., proposed optimal solutions/columns, dual values) since P_i knows Q_i. Even though this additional private information does not directly hurt other agents in the protocol (since the pricing problem owner only learns additional information of its own pricing problems), retaining them unknown to every participant is better while securely solving K-LP problems. Moreover, if P_i cannot learn the proposed column of every pricing problem, it could alleviate the incentive of sending the fake payoff-increased columns to the RMP solver.

To resolve this collusion, we can extend one agent matrix post-multiplication to multi-agent transformation (pre-multiplication and post-multiplication): for every share (block) of the constraints matrix in K-LP problem, e.g. A'_i, B'_i, let *all the agents* post-multiply a monomial matrix, and also pre-multiply a monomial matrix to A'_i or B'_i. Similar to post-multiplying a monomial matrix (see Eqs (15) and (16)), pre-multiplying a monomial matrix to A'_i (or B'_i) can be also considered as two steps [12]: (1) permutate the rows in A'_i (or B'_i) according to the positions of the non-zero elements in the monomial matrix and (2) multiply a random number to every row of the row-permuted A'_i (or B'_i). For example as below:

$$
\begin{aligned}
A'_i &= \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n'_i} \\ a_{21} & a_{22} & \dots & a_{2n'_i} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m_0 1} & a_{m_0 2} & \dots & a_{m_0 n'_i} \end{pmatrix} \in \mathbb{R}^{m_0 \times n'_i}, \\
Q'_i &= \begin{pmatrix} 0 & 0 & \dots & 0 & r'_{m_0} \\ r'_1 & 0 & \dots & 0 & 0 \\ 0 & r'_2 & \dots & 0 & 0 \\ 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \dots & r'_{m_0-1} & 0 \end{pmatrix} \in \mathbb{R}^{m_0 \times m_0},
\end{aligned} \tag{17}
$$

$$
Q'_i A'_i = \begin{pmatrix} a_{m_0 1} r'_{m_0} & a_{m_0 2} r'_{m_0} & \dots & a_{m_0 n'_i} r'_{m_0} \\ a_{11} r'_1 & a_{12} r'_1 & \dots & a_{1 n'_i} r'_1 \\ \vdots & \vdots & \ddots & \vdots \\ a_{(m_0-1)1} r'_{m_0-1} & a_{(m_0-1)2} r'_{m_0-1} & \dots & a_{(m_0-1)n'_i} r'_{m_0-1} \end{pmatrix}. \tag{18}
$$

Therefore, if we let all agents be involved in the transformation of each agent P_i ($i \in [1, K]$)'s share in the (b anonymized) K-LP problem, $A'_i, B'_i, c'^{\mathrm{T}}_i$, $(b^i_0)'$ and b'_i

can be transformed by all agents $\forall j \in [1, K]$, P_j as:[7]

$$\begin{aligned}
&\forall i \in [1, K] \quad A'_i \Longrightarrow \prod_{j=1}^{K} Q_j A'_i \prod_{j=1}^{K} Q_{ij}, \qquad (b_0^i)' \Longrightarrow \prod_{j=1}^{K} Q_j (b_0^i)', \\
&\forall i \in [1, K] \quad B'_i \Longrightarrow \prod_{j=1}^{K} Q'_{ij} B'_i \prod_{j=1}^{K} Q_{ij}, \qquad b'_i \Longrightarrow \prod_{j=1}^{K} Q'_{ij} b'_i, \\
&\forall i \in [1, K] \quad c_i'^{\mathrm{T}} \Longrightarrow c_i'^{\mathrm{T}} \prod_{j=1}^{K} Q_{ij}.
\end{aligned} \tag{19}$$

Note that $(b_0^i)'$ and b'_i should be applied with the same pre-multiplication as A'_i and B'_i, respectively, since the right-hand side values of the constraints should be consistent with the corresponding rows in the constraints matrix in permutation and multiplication. Moreover, first, since every agent P_i's A'_i in the global constraints has the same number of rows and the applied permutation (to $\forall i \in [1, K], A_i$) should be consistent (otherwise, the global constraints cannot be formulated anymore), we should let each agent P_j ($j \in [1, K]$) generate a privately held monomial matrix $Q'_j \in \mathbb{R}^{m_0 \times m_0}$ for all $i \in [1, K], A'_i$; second, every agent P_i's B'_i in its local constraints has different number of rows m'_i, we should let each agent P_j ($j \in [1, K]$) generate a different monomial matrix Q_{ij} for transforming P_i's B'_i.

Why pre-multiply different matrices for A'_i and B'_i. Since matrices $A'_i \in \mathbb{R}^{m_0 \times n'_i}$, $B'_i \in \mathbb{R}^{m'_i \times n'_i}$ have different number of rows, the pre-multiplied monomial matrices should have different size. From the perspective of permutation, we should apply the same column permutation to A'_i and B'_i. However, A'_i is a share in the global constraints while B'_i is the constraints matrix of P_i's local constraints, thus they should be permutated separately.

Why we need both multi-agent matrix pre-multiplication and post-multiplication for A'_i and B'_i against collusion. If only multi-agent post-multiplication is applied (as in the semi-honest model), the collusion of learning additional private information cannot be resolved. Specifically, assume that $P_{(i \bmod K)+1}$ solves P_i's pricing problems and they collude to learn additional information regarding P_i's true pricing problems, thus $P_{(i \bmod K)+1}$ obtains $\{A'_i \prod_{j=1}^{K} Q_{ij}, B'_i \prod_{j=1}^{K} Q_{ij}, c_i'^{\mathrm{T}} \prod_{j=1}^{K} Q_{ij}\}$ (if only multi-agent post-multiplication is applied). Then, $P_{(i \bmod K)+1}$ sends back the transformed matrices/vector to P_i. Since the matrix multiplication can be considered as column permutations to the matrices (and the transpose of the vector) plus a unknown coefficient to every column, P_i can still learn the multi-agent column

[7]Each agent P_j ($j \in [1, K]$) generates three privately held monomial matrices $Q_{ij} \in \mathbb{R}^{n'_i \times n'_i}$, $Q_j \in \mathbb{R}^{m_0 \times m_0}$ and $Q'_{ij} \in \mathbb{R}^{m'_i \times m'_i}$ for transforming P_i's share in the K-LP problem.

permutation by comparing the columns in $A'_i \prod_{j=1}^{K} Q_{ij}$, $B'_i \prod_{j=1}^{K} Q_{ij}$ and $\{A'_i, B'_i\}$ (the corresponding column in the transformed matrix is equal to the original column times a positive number, which can be simply discovered in the transformed matrices; note that P_i cannot compare the columns in $c_i'^{\mathrm{T}}$ and $c_i'^{\mathrm{T}} \prod_{j=1}^{K} Q_{ij}$ since every column is a single number rather than a vector in them, the relation between the original number and the corresponding number in $c_i'^{\mathrm{T}} \prod_{j=1}^{K} Q_{ij}$ cannot be inferred). Thus, besides multi-agent post-multiplication, we have to let all agents permute the rows in the transformed constraints and multiply a random number to every row as well (doing this via pre-multiplication).

In sum, after each agent P_i ($i \in [1, K]$) locally anonymizes $(b_0^i)'$ and b'_i, all agent can jointly transform the K-LP problem into:

$$\max \sum_{i=1}^{K} \left(c_i'^{\mathrm{T}} \prod_{j=1}^{K} Q_{ij} \right) y_i$$

$$\text{s.t.} \begin{cases} y_1 \in \mathbb{R}^{n'_1} \\ y_2 \in \mathbb{R}^{n'_2} \\ \vdots \\ y_K \in \mathbb{R}^{n'_K} \end{cases} \begin{pmatrix} (\prod_{j=1}^{K} Q_j) A'_1 (\prod_{j=1}^{K} Q_{1j}) & \cdots & (\prod_{j=1}^{K} Q_j) A'_K (\prod_{j=1}^{K} Q_{Kj}) \\ (\prod_{j=1}^{K} Q'_{1j}) B'_1 (\prod_{j=1}^{K} Q_{1j}) & & \\ & \ddots & \\ & & (\prod_{j=1}^{K} Q'_{Kj}) B'_K (\prod_{j=1}^{K} Q_{Kj}) \end{pmatrix} \tag{20}$$

$$\times \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_K \end{pmatrix} \begin{matrix} \bowtie_0 \\ \bowtie_1 \\ \vdots \\ \bowtie_K \end{matrix} \begin{pmatrix} (\prod_{j=1}^{K} Q_j) b'_0 \\ (\prod_{j=1}^{K} Q'_{1j}) b'_1 \\ \vdots \\ (\prod_{j=1}^{K} Q'_{Kj}) b'_K \end{pmatrix}.$$

The detailed steps of multi-agent transformation is presented in Algorithm 5.

Theorem 6. *Multi-agent transformation* (*pre-multiplication and post-multiplication*) *does not reveal additional private information against arbitrary number of colluding agents* ($\ell < K$).

Proof. We first look at the correctness of the multi-agent transformation. Since the product of any two permutation matrices remains a permutation matrix [12], the product of any two monomial matrices should be a monomial matrix as well. Thus, $\forall i \in [1, K]$, $\prod_{j=1}^{K} Q_{ij}$ and $\prod_{j=1}^{K} Q'_{ij}$ are also monomial matrices, and transforming A'_i, B'_i and $c_i'^{\mathrm{T}}$ by multiplying $\prod_{j=1}^{K} Q_{ij}$ is correct while solving the K-LP problem (after obtaining P_i's share in the optimal solution of the transformed problem y_i^*, the original share in the global optimal solution x_i^* can be derived as $\prod_{j=1}^{K} Q_{ij} y_i^*$ by all agents). Moreover, since pre-multiplying $\prod_{j=1}^{K} Q_j$ to $\forall i \in [1, K]$, A'_i and b'_0 (the global constraints) and pre-multiplying $\prod_{j=1}^{K} Q'_{ij}$ to $\forall i \in [1, K]$, B'_i and

Algorithm 5: Multi-agent transformation

forall the *agent* $P_i, i \in [1, K]$ **do**
  generates a privately held monomial matrix $Q_i \in \mathbb{R}^{m_0 \times m_0}$ (for pre-multiplication of $\forall i \in [1, K]$, A'_i and $(b^i_0)'$) where non-zero elements are random numbers;
forall the *agent* $P_i, i \in [1, K]$ **do**
  generates two privately held monomial matrices $Q_{ii} \in \mathbb{R}^{n'_i \times n'_i}$ (for post-multiplication of A'_i, B'_i and c'^{T}_i) and $Q'_{ii} \in \mathbb{R}^{m'_i \times m'_i}$ (for pre-multiplication of B'_i and b'_i) where non-zero elements are random numbers;
  transforms $A \leftarrow Q_i A'_i Q_{ii}, B \leftarrow Q'_{ii} B'_i Q_{ii}, c^{\mathrm{T}} \leftarrow c'^{\mathrm{T}}_i Q_{ii}, b_A \leftarrow Q_i (b^i_0)', b_B \leftarrow Q'_{ii} (b^i_0)'$;
  sends $A, B, c^{\mathrm{T}}, b_A$ and b_B to the next agent $P_j, j \in [1, K], j \neq i$;
  forall the *agent* $P_j, j \in [1, K], j \neq i$ **do**
    generates two privately held monomial matrices $Q_{ij} \in \mathbb{R}^{n'_i \times n'_i}$ (for post-multiplication of A'_i, B'_i and c'^{T}_i) and $Q'_{ij} \in \mathbb{R}^{m'_i \times m'_i}$ (for pre-multiplication of B'_i and b'_i) where non-zero elements are random numbers;
    transforms $A \leftarrow Q_j A Q_{ij}, B \leftarrow Q'_{ij} B Q_{ij}, c^{\mathrm{T}} \leftarrow c^{\mathrm{T}} Q_{ij}, b_A \leftarrow Q_j b_A, b_B \leftarrow Q'_{ij} b_B$;
    sends $A, B, c^{\mathrm{T}}, b_A$ and b_B to the next agent;
output the transformed shares of the K-LP problem.

b'_i do not change feasible region and the optimal solution [23] (this can be considered as swapping the order of the constraints and multiplying the same random number to both sides of the constraints). Thus, applying pre-multiplication and post-multiplication from agent to agent generates correct transformation. Finally, each agent P_i ($i \in [1, K]$)'s share in the original optimal solution can be derived by pre-multiplying $\prod_{j=1}^{K} Q_{ij}$ to its share in the optimal solution of the multi-agent transformed problem (inverse row permutation/pre-multiplication to the optimal solution is not necessary [23]).

Clearly, in case that ℓ agents collude where $\ell < K$, since for every agent for P_i ($i \in [1, K]$)'s share of the K-LP problem, three monomial matrices $\prod_{j=1}^{K} Q_j$, $\prod_{j=1}^{K} Q_{ij}$, $\prod_{j=1}^{K} Q'_{ij}$ are unknown to all agents, thus it is impossible to compute the optimal solution/proposed columns of any of P_i's pricing problems without involving all agents (even if the colluding agents know A'_i, B'_i and c'^{T}_i). From the perspective of permutation and multiplication, for any agent P_i ($i \in [1, K]$)'s input matrices/vector $\{A'_i, B'_i$ and $c'_i\}$, every agent P_j (including itself) applies a new column permutation to each of them and multiplies a new random number to every

column (both steps execute as post-multiplying Q_{ij}), and then do the same thing to rows by pre-multiplying Q_j or Q'_{ij}. The overall row and column permutation and multiplied random numbers in the monomial matrices (which are only jointly generated) are unknown to all agents, and without any agent, the true column and row permutation indices and the multiplied random numbers cannot be reconstructed.

Thus, for arbitrary number (ℓ) of colluding agents ($\ell < K$), no additional information can be revealed in the protocol with multi-agent transformation, and this completes the proof. □

6.3.2. Multiple SCG protocol run with random RMP and pricing problem solvers

A naive approach to establish Nash equilibrium has been introduced in Section 6.2.2 – run the SCG protocol for multiple times with distinct RMP solver and pricing problem solvers. We now discuss how it works in two cases: (1) no colluding agents and (2) ℓ (less than K) colluding agents.

(1) *No colluding malicious agents*. Assuming the SCG protocol is run N times, we then describe the detection of all the malicious behavior (except collusion) listed in Section 6.1 (e.g., the RMP solver modifies the final optimal solution or the dual values in every iteration of the SCG protocol, the pricing problem solver proposes fake local optimal solutions/columns to the RMP solver). If all the agents follow the SCG protocol, the optimal solutions derived from N times SCG protocol run with different RMP solvers should be identical:[8]

- In any of the N runs of the SCG protocol, suppose that the RMP solver modifies the final optimal solution or the dual values in every iteration of the SCG protocol. Any difference of the N optimal solutions should be detected by the owner of the share in the optimal solutions (discovering such difference is sufficient to establish Nash equilibrium in the protocol as introduced earlier). Note that if the modification does not cause distortion to the output (this may happen if the RMP solver modifies the dual values which does not result in distinct proposed local optimal solutions/columns), the modification does not affect the protocol and we can ignore it.
- Similarly, in any of the N runs of the SCG protocol, if any pricing problem solver proposes wrong local optimal solutions/columns to the RMP solver (this may distort the optimal solution, but not always succeed since the proposed local optimal solutions/columns might not be the basis in the master problem), the pricing problem owner should detect such modification if the modification leads to deviated output (otherwise, the modification can be also ignored) since

[8]The optimal solution in the original K-LP problem can be reconstructed in each of the N times SCG protocol run even if N different revised Dantzig–Wolfe decomposition are implemented with different monomial transformation matrices and RMP/Pricing problem solvers. Thus, the optimal solutions should be identical in all N times SCG protocol run as all agents follow the protocol (since all of them should be optimal to the original K-LP problem). Note that if the LP problem have multiple optimal solutions, all the optimal solutions should be given as the output, and they should be identical as well.

the pricing problem owner also receives the optimal solution from another agent in the remaining $N - 1$ SCG protocol runs.

To sum up, since any malicious behavior can be caught through multiple SCG protocol run with different RMP and pricing problem solvers, the Nash equilibrium on "honest play" always holds in case of no colluding malicious agents.

(2) *Against ℓ colluding malicious agents*. We now extend the discussion to a more general form: assuming at most ℓ (less than K) malicious agents cheat together as a dishonest coalition, and the number of dishonest coalitions may exceed one.[9] In this case, all the agents inside the same dishonest coalition can implement identical malicious behavior when they solve the RMP problem or the same agent's pricing problems in multiple SCG protocol run, and the cheating from the same dishonest coalition cannot be caught.

We still employ multiple SCG protocol run to build the Nash equilibrium by increasing the probability of detecting cheating from the dishonest coalitions with at most ℓ malicious agents. Initially, we denote N as the number of multiple SCG protocol run with randomly picked RMP and pricing problem solvers. In all N SCG protocol run, we uniformly pick N *different RMP solvers* from K agents and also pick N *different pricing problem solvers* from K agents for solving each agent's pricing problems (clearly, $N \leqslant K$). Note that we allow each agent solving its own transformed pricing problems: the pricing problem owner (collude with less than K agents) cannot learn additional information while solving them with multi-agent transformation (Theorem 6). Otherwise, the malicious behavior cannot be eliminated in the worst case that $\ell = K - 1$ (if the only agent outside the dishonest coalition is not allowed to solve its pricing problems, the cheating from the coalition cannot be caught in any case since all the pricing problems of the only victim agent are solved by the agents from the dishonest coalition, no matter how many times of running the SCG protocol with permuted solvers).

Essentially, for every agent P_i ($i \in [1, K]$) in the K-LP problem, if P_i's N groups of pricing problems are solved by at least two non-colluding agents in N SCG protocol run, and N groups of RMPs are also solved by at least two non-colluding agents, the cheating with regard to P_i's share in the K-LP problem should be caught (the caught cheating covers not only explicit modifications to the pricing problems or the proposed columns/local optimal solution, but also the cheating on other agents' shares in the K-LP problem which results in deviation to P_i's share in the final global optimal solution). Therefore, while uniformly picking N distinct RMP solvers out of K agents and N distinct pricing problem solvers out of K agents for each agent's pricing problems in N SCG protocol run, we can utilize either of the following two conditions to detect cheating:

(1) $N \geqslant \ell + 1$;

[9] We assume that one agent cannot collude with agents in more than one dishonest coalition in the K-LP problem (if this happens, we can consider the union of all the dishonest coalitions involving the same agent as a larger dishonest coalition).

(2) at least one RMP solver is picked from the largest dishonest coalition and at least one RMP solver is picked outside the coalition; for every agent's pricing problems, at least one solver is picked from the largest dishonest coalition and at least one solver is picked outside the coalition.

If either of the above two conditions is satisfied, modifying any agent's share in the K-LP problem will be caught since every agent's share in the global optimal solution are generated by at least two non-colluding agents in N SCG protocol run (note that the agents outside the largest dishonest coalition may collude and form smaller dishonest coalitions as well). Thus, for maintaining efficiency, our incentive compatible protocol should seek the *minimum N satisfying either of the two conditions*.

- Clearly, if condition (1) holds, then every agent's share in the $N \geqslant \ell + 1$ global optimal solutions should be generated by at least two agents from different coalitions since the largest dishonest coalition includes only ℓ participants, and any cheating will be definitely caught (note that it is unnecessary to know which agent plays dishonestly since the Nash equilibrium can be achieved in any case).
- We now look at the probability of satisfying condition (2) if random solvers are picked. Specifically, if picking N distinct RMP solvers out of K agents ($N \leqslant K$), the probability of picking at least one RMP solver outside and at least one RMP solver inside the largest dishonest coalition (ℓ malicious agents collude to modify the protocol) is equal to:

$$\mathit{Prob}_R = 1 - \prod_{i=0}^{N-1} \frac{\ell - i}{K - i} - \prod_{i=0}^{N-1} \frac{(K - \ell) - i}{K - i}. \tag{21}$$

Moreover, since picking N pricing problem solvers (out of K) for all agents are independent and similar to uniformly picking N RMP solvers, for every agent P_i ($i \in [1, K]$)'s pricing problems, the probability of picking at least one solver outside and at least one agent inside the largest dishonest coalition in N SCG protocol run is:

$$\mathit{Prob}_P = 1 - \prod_{i=0}^{N-1} \frac{\ell - i}{K - i} - \prod_{i=0}^{N-1} \frac{(K - \ell) - i}{K - i}. \tag{22}$$

Note that if $N > \max\{K - \ell, \ell\}$, we have $\mathit{Prob}_R = \mathit{Prob}_P = 1$ in Eq. (21) and (22) (such N also satisfies condition (1): $N \geqslant \ell + 1$). Since we are seeking the least N for the incentive compatible protocol, we only need to consider the case $N \leqslant \max\{K - \ell, \ell\}$ for condition (2).

Since the probability of catching the malicious behavior with condition (2) is $\mathit{Prob}_R * \mathit{Prob}_P$ (picking RMP and pricing problem solvers are independent), we consider the probability of detecting cheating as $\epsilon = \mathit{Prob}_R * \mathit{Prob}_P$ for all agents.

In addition, if the probability of detecting cheating for every agent P_i ($i \in [1, K]$) satisfies $\epsilon > \frac{u(T_i)-T_i}{u(T_i)-c(T_i)}$ (Eq. (14)) where P_i's payoff ratio $\frac{u(T_i)-T_i}{u(T_i)-c(T_i)} < 1$, "honest play" should be the strictly dominant strategy for all agents. As a result, if the following K inequalities hold, the protocol should be incentive compatible.

$$\forall i \in [1, K] \quad \left[1 - \prod_{i=0}^{N-1} \frac{\ell - i}{K - i} - \prod_{i=0}^{N-1} \frac{(K - \ell) - i}{K - i}\right]^2 > \frac{u(T_i) - T_i}{u(T_i) - c(T_i)}. \quad (23)$$

Since every agent's payoff ratio $\forall i \in [1, K]$, $\frac{u(T_i)-T_i}{u(T_i)-c(T_i)}$ is fixed in the K-LP problem, if the maximum payoff ratio among all K agents $\max\{\forall i \in [1, K], \frac{u(T_i)-T_i}{u(T_i)-c(T_i)}\}$ is less than $[1 - \prod_{i=0}^{N-1} \frac{\ell-i}{K-i} - \prod_{i=0}^{N-1} \frac{(K-\ell)-i}{K-i}]^2$, the incentive compatibility always holds. Thus, while satisfying condition (2), we obtain N as the least integer to satisfy:

$$\prod_{i=0}^{N-1} \frac{\ell - i}{K - i} + \prod_{i=0}^{N-1} \frac{(K - \ell) - i}{K - i} < 1 - \sqrt{\max\left\{\forall i \in [1, K], \frac{u(T_i) - T_i}{u(T_i) - c(T_i)}\right\}}. \quad (24)$$

Note that $1 - \sqrt{\max\{\forall i \in [1, K], \frac{u(T_i)-T_i}{u(T_i)-c(T_i)}\}}$ is fixed in the range $(0, 1)$ and $\prod_{i=0}^{N-1} \frac{\ell-i}{K-i} + \prod_{i=0}^{N-1} \frac{(K-\ell)-i}{K-i}$ is anti-monotonic on N, we can always find a least integer N to acquire the Nash equilibrium (if $N > \max\{K - \ell, \ell\}$, we have $\prod_{i=0}^{N-1} \frac{\ell-i}{K-i} + \prod_{i=0}^{N-1} \frac{(K-\ell)-i}{K-i} = 0$ and the inequality also holds).

In sum, we can combine two conditions together, and *compute the smallest N to satisfy either of them*. Indeed, condition (1) is more effective to catch cheating for smaller ℓ and condition (2) is more effective to catch cheating for larger ℓ.

Theorem 7. *Running N times SCG protocol with random distinct RMP and pricing problem solvers (N is given as the smaller number of $\ell + 1$ and the least integer in Eq. (24)) is incentive compatible against malicious participants if the largest dishonest coalition include at most ℓ colluding agents.*

Proof. We can simply prove this theorem with an inverse analysis based on the above discussion. Assuming that the largest coalition includes at most ℓ malicious agents, if $N > \ell$ or N satisfies the inequality in Eq. (24), for any agent P_i ($i \in [1, K]$)'s pricing problems (P_i can be an agent inside or outside the dishonest coalition), the probability of receiving P_i's share in N global optimal solutions from at least two non-colluding agents is absolutely greater than $\forall i \in [1, K]$, $\frac{u(T_i)-T_i}{u(T_i)-c(T_i)}$.

Moreover, we assume that the malicious behavior on P_i's share in the K-LP problem is done by another agent $P_j, j \neq i$ (or more agents). Hence, the probability of generating P_i's share in N global optimal solutions by at least one agent which does not collude with P_j (and its colluding agents) is greater than $\frac{\max(T_j)-T_j}{\max(T_j)-\min(T_j)}$. Since such random solvers can catch all the malicious behavior on P_i's share in the K-LP problem, the probability of detecting P_j's cheating ϵ_j is greater than $\frac{u(T_i)-T_i}{u(T_i)-c(T_i)}$. Thus, the strategy "honest play" for P_j strictly dominates the strategy "cheating".

Similarly, we can show that all malicious agents have a strictly dominant strategy of "honest play", though at most ℓ agents can collude to form a dishonest coalition. According to the Nash equilibrium, the incentive compatibility is proved. □

6.3.3. Protocol and analysis

For any K-LP problem, all agents can estimate the maximum payoff of uncaught cheating and determine the payoff of caught cheating with punishment. Thus, the least integer N to establish Nash Equilibrium can be estimated using Eq. (24). The incentive compatible protocol can be given as N SCG protocol run with randomly picked distinct RMP and pricing problem solvers (Algorithm 6).

Computation and communication cost analysis. Since the incentive compatible protocol runs the SCG protocol for N times (N is not required to be greater than

Algorithm 6: Incentive compatible protocol

`/* Each agent` P_i`'s share (`b_0^i `and` b_i`) in the right-hand side values have been anonymized into (`b_0^i`)′ and` b_i'`, respectively. */`

1 **Transform** each agent P_i's share in the K-LP problems $A_i', B_i', c'^{\mathrm{T}}, (b_0^i)'$ and b_i' using Algorithm 5 ($i \in [1, K]$);

2 **Solve** the transformed K-LP problem with the SCG protocol (Algorithm 4) for N times where all the N RMP solvers and each agent's N transformed pricing problem solvers are uniformly picked from agents all agents $P_1, P_2, \ldots, P_K$ (N is given as the *smaller number* of $\ell + 1$ and the least integer in Eq. (24));

3 **Distribute** K shares of the N global optimal solutions to the corresponding pricing problem owners;

4 **Compute** each agent P_i's share in the original optimal solution by (K agents) jointly pre-multiplying $\prod_{j=1}^{K} Q_{ij}$ to the optimal solutions received from the RMP solvers;

`/* It is not necessary to apply inverse row permutation (pre-multiplication) after obtaining the optimal solutions. */`

5 **Report** cheating if any difference among each agent's share in N optimal solutions is found;

$\max\{K - \ell, \ell\}$), the computation and communication complexity of it are approximately $O(N)$ to the complexity of the SCG protocol, which is still efficient.

7. Experimental results

In this section, we present the experimental results to evaluate the performance of the SCG protocol and the incentive compatible protocol using both synthetic data and real data. Specifically, we study: (1) the computational performance of the SCG protocol compared with Secure Transformation (ST) [39], Secure Revised Simplex Method (SRS) [40] and Secure Simplex Method (SS) [19]; (2) the scalability of the SCG protocol and the incentive compatible protocol in terms of computational performance. Note that the communication overheads of the SCG protocol are extremely tiny in the K-LP problem (we thus skip such evaluation).

7.1. Experimental setup

Datasets. Our experiments are conducted on both synthetic datasets and real-world dataset (MovieLens dataset[10]). In the experiments on synthetic data, we generate 10 K-LP problems for every test and average the result. All the random numbers generated in the shares of the constraint matrix $\forall i \in [1, K]$, A_i, B_i are non-negative in the range [0, 10] (with density 30%), the objective vector $c = [c_1\ c_2\ \cdots\ c_K]$ is randomly selected in the range [10, 20] and the right-hand side values $b_0, b_1, \ldots, b_K$ are randomly generated from the range [100, 500]. All the constraints are inequalities with operational symbol "$\leqslant$", which guarantees the feasibility of the LP problems. In addition, we also formulate a K-LP problem by extracting data from the MovieLens dataset with the same problem size as every 10 synthetic data based K-LP problems. Since the MovieLens dataset is a collection of 71,567 users' ratings for 10,681 movies (including 10,000,054 ratings in total, represented as the entries in a *user* $\times$ *movie* matrix), we can extract constraint matrices as the subsets of such large-scale matrix. With the same setting on the right-hand side values, objective vector and the operational symbol as the randomly generated LP problems, the LP problems based on the MovieLens dataset are also feasible. Note that we run the same groups of experiments on both synthetic and real data.

Protocol comparison setting. We compare the computational performance of our SCG protocol with some prior secure DisLP methods. In that experimental setting, we assume that the K-LP problems are jointly solved by two agents, because only two-agent distributed LP problems can be securely solved by Secure Transformation (ST)[39], Secure Revised Simplex Method (SRS) [40] and Secure Simplex Method (SS) [19]. More specifically, we test the computation costs of our SCG protocol and

[10]http://www.grouplens.org.

Table 1
Experimental setting for protocol comparison (two distributed agents)

LP size	Synthetic and real data setting		
	Each agent's number of variables (distinct movies)	Number of global constraints (users with overall ratings)	Each agent's number of local constraints (users with ratings on each agent's movies)
15×20	10	5	5
30×40	20	10	10
45×60	30	15	15
60×80	40	20	20
75×100	50	25	25

the above three methods on a group of two-agent LP problems with varying problem size: 15×20, 30×40, 45×60, 60×80, 75×100 (we choose this groups of medium-scale LP problems because some methods cannot scale well to large-scale problems), where each agent holds $1/2$ of the total variables and $1/3$ of the total constraints (as the local constraints), and the remaining $1/3$ of the total constraints are generated as the global constraints. While creating the constraints matrix (with the synthetic data or the real-world data), we generate elements for specific blocks in the constraints matrix: $\forall i \in [1, K]$, A_i, B_i. For the blocks in the synthetic matrices, the density of the matrix is fixed at 30% and each non-zero entry is uniformly chosen from the range (0, 10] (as described as above); for the blocks in the real matrix, we select some popular movies (the number of distinct movies is equal to the number of variables in every LP problem, and each agent holds $1/2$ of them), create the global constraints matrix by selecting m_0 users' ratings on all the selected movies, and finally select m_i users' ratings for each agent P_i's movies as the elements in the block of local constraints B_i. Table 1 describes the detail of the distributed LP problems for comparing four protocols.

Groups of experiments for evaluating the SCG and incentive compatible protocol. While evaluating the scalability of the SCG and the incentive compatible protocol with regard to the computational performance, we use the same mechanism as above to formulate the K-LP problems with the synthetic and real data. Specifically, we build two groups of distributed LP problems as shown in Tables 2 and 3.

To test the scalability of the SCG protocol, we conduct two groups of experiments:

- fixing the LP problem size as 500×1000 and letting all agents have the same number of variables, we test the computational cost of the SCG protocol on varying number of agents: 2, 5, 10, 20, 50 (see the detail of the K-LP problems in Table 2);
- fixing the number of agents as 20 and letting all agents have the same number of variables, we test the computational cost of the SCG protocol on varying LP size: $100 \times 200, 200 \times 400, 300 \times 600, 400 \times 800, 500 \times 1000$ (see the detail of the K-LP problems in Table 3).

Table 2
K-LP problems with varying number of agents (problem size: 500 × 1000)

Number of agents	Synthetic and real data setting		
	Each agent's number of variables (distinct movies)	Number of global constraints (users with overall ratings)	Each agent's number of local constraints (users with ratings on each agent's movies)
2	500	100	200
5	100	100	80
10	50	100	40
20	25	100	20
50	10	100	8

Table 3
K-LP problems with varying size (20 agents)

LP size	Synthetic and real data setting		
	Each agent's number of variables (movies)	Number of global constraints (users with overall ratings)	Each agent's number of local constraints (users with ratings on each agent's movies)
100 × 200	10	20	4
200 × 400	20	40	8
300 × 600	30	60	12
400 × 800	40	80	16
500 × 1000	50	100	20

To test the scalability of the incentive compatible protocol, we conduct four groups of experiments:

- fixing the LP problem (500 × 1000) and the largest dishonest coalition includes at most $1/2$ of the total agents, test the computational cost of the incentive compatible protocol on varying *number of agents*: 2, 5, 10, 20, 50 (see the detail of the K-LP problems in Table 2);
- fixing the number of agents as 20 (10 of them collude), test the computational cost of the incentive compatible protocol on the varying *LP problem size*: 100 × 200, 200 × 400, 300 × 600, 400 × 800, 500 × 1000 (see the detail of the K-LP problems in Table 3);
- fixing the LP problem (500 × 1000) and the number of agents as 20, test the computational cost of the incentive compatible protocol on the *number of colluding agents in the largest dishonest coalition*: 2, 4, 6, 8, 10, 12, 14, 16, 18.
  Note that in the above three groups of experiments, we use an estimated maximum payoff ratio $\max\{\forall i \in [1, K], \frac{u(T_i)-T_i}{u(T_i)-c(T_i)}\} = \frac{1}{2}$ (this is determined by the real collaborative LP problem as a number in the range $(0, 1)$ and $1/2$ is a common value in such range – e.g., $u(T_i) = 100, T_i = 50, c(T_i) = 0$);

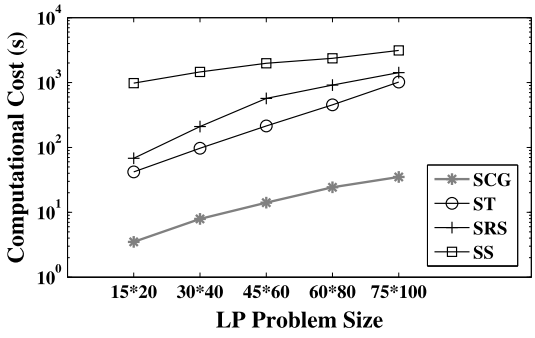
- fixing the LP problem (500×1000), number of agents as 20 (10 of them collude), we also test the computational cost of the incentive compatible protocol on all possible values for the *maximum payoff ratio* $\max\{\forall i \in [1, K], \frac{u(T_i)-T_i}{u(T_i)-c(T_i)}\}$ from 0.1 to 0.9.

Note that in both SCG protocol and the incentive compatible protocol, the near-optimal tolerance parameter for column generation is given as 10^{-6}.

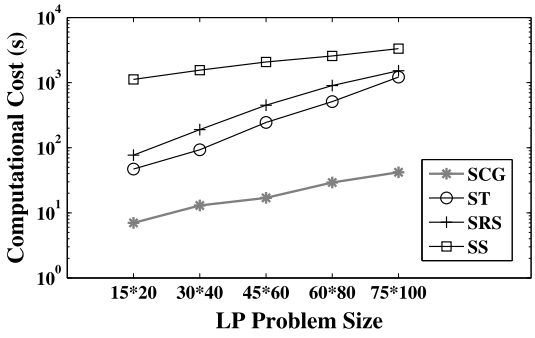
Platform. The algorithms were implemented in Matlab, and all the experiments were carried on an HP machine with Intel Core 2 Duo CPU 3 GHz and 3G RAM.

7.2. Secure K-agent column generation (SCG) protocol

Figure 5 shows the computational performance of four privacy-preserving linear programming protocols: Secure Column Generation (SCG), Secure Transformation (ST), Secure Simplex Method (SS) and Secure Revised Simplex Method (SRS). For

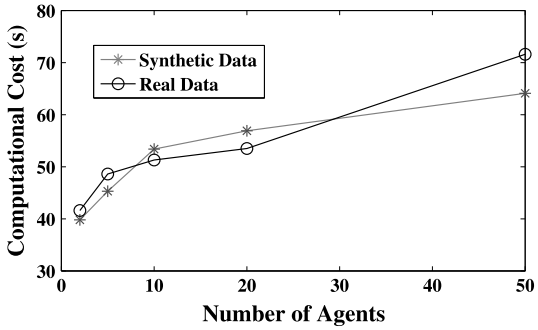


(a)

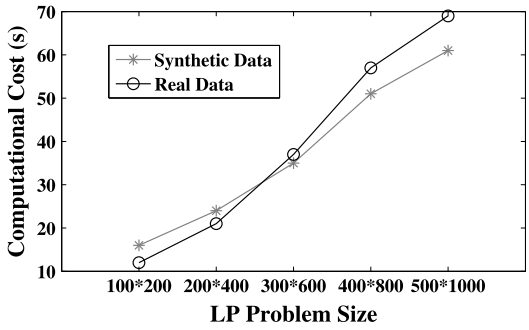


(b)

Fig. 5. Computational performance comparison of secure DisLP protocols (logarithmic scale). (a) Synthetic data. (b) Real data. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/JCS-2012-0452.)



(a)



(b)

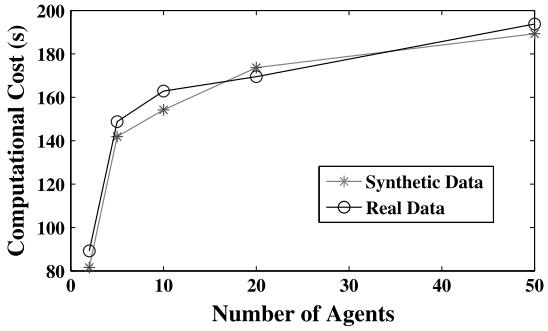
Fig. 6. Computational performance of the SCG protocol. (a) Varying number of agents. (b) Varying LP problem size. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/JCS-2012-0452.)

different size of the LP problems, the SCG protocol requires significantly less runtime to solve the two-agent distributed LP problems formulated by both synthetic data (Fig. 5(a)) and real data (Fig. 5(b)).

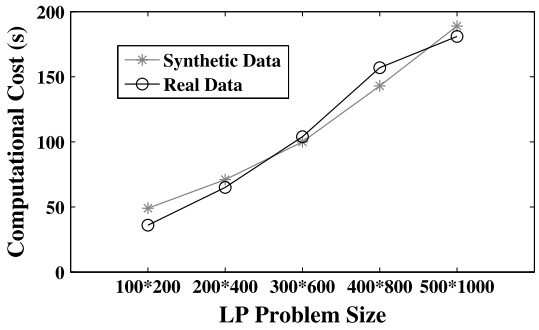
Figure 6(a) illustrates the scalability of the SCG protocol on varying number of agents (fixed LP problem) and varying LP problem size (fixed number of agents). The computation cost of the SCG protocol scales well to (synthetic and real) large size LP problems – the runtime increases approximately in linear trend in both groups of experiments (Fig. 6(a) and (b)).

7.3. Incentive compatible protocol

Similar to the evaluation on the SCG protocol, Fig. 7(a) and (b) illustrates the computational performance of the incentive compatible protocol on varying number of agents (fixed LP problem, percentage of the colluding agents and the maximum payoff ratio) and varying LP problem size (fixed total number of agents, percentage



(a)



(b)

Fig. 7. Computational performance I of the incentive compatible protocol. (a) Varying number of agents. (b) Varying LP problem size. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/JCS-2012-0452.)

Table 4
N in the incentive compatible protocol

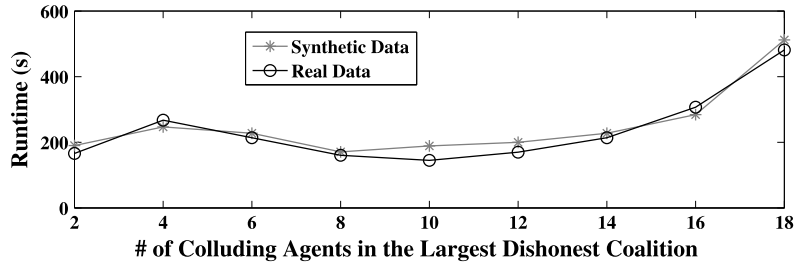
Experiment groups	Experiment results	ℓ	Minimum number of SCG protocol run
1	Fig. 7(a)	$\ell = 1, 2, 5, 10, 25$	$N = 2, 3, 3, 3, 3$
2	Fig. 7(b)	all $\ell = 10$	$N = 3, 3, 3, 3, 3$
3	Fig. 8(a)	$\ell = 2, 4, 6, 8, 10, 12, 14, 16, 18$	$N = 3, 5, 4, 3, 3, 3, 4, 5, 9$
4	Fig. 8(b)	all $\ell = 10$	$N = 2, 2, 2, 2, 2, 3, 3, 4, 4$

of the colluding agents and the maximum payoff ratio). The incentive protocol also scale well to solve large distributed LP problems and distributed LP problems with large number of agents.

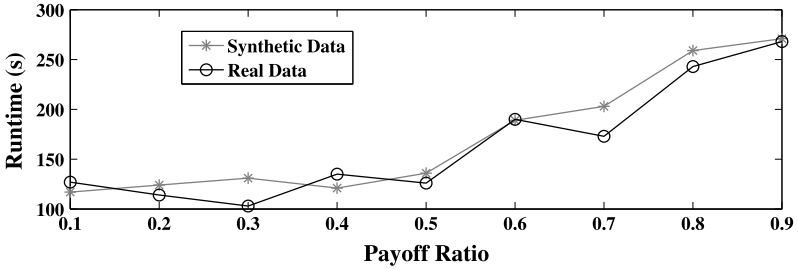
Moreover, we capture the minimum number of SCG protocol run required in every group of experiments (see Table 4). The minimum number of required SCG protocol

run N in the above two groups of experiments does not exceed 3 (especially in the first group of experiments, no matter how many agents exist in the K-LP problem, if only half of them collude and the maximum payoff ratio equal to $1/2$, only 3 times SCG run with random RMP and pricing problem solvers could make the protocol incentive compatible).

In the third group of experiments, we tested the computation cost of the incentive compatible protocol tuning on parameter – the number of colluding agents in the largest dishonest coalition (fixed LP problem, total number of agents and the maximum payoff ratio). As shown in the second row of Table 4, for $\ell = 2$ and 4, the incentive compatible protocol runs $N = \ell + 1 = 3$ and 5 times SCG protocol respectively (condition (1)); for $\ell > 4$, N turns to the value computed from condition (2): 4, 3, 3, 3, 4, 5, 9 (which are less than $\ell + 1$). Another interesting point is that if ℓ is close to half of the entire number of agents $K/2$, for any N, $\prod_{i=0}^{N-1} \frac{\ell-i}{K-i} + \prod_{i=0}^{N-1} \frac{(K-\ell)-i}{K-i}$ (which is approximately symmetric to the value of $N = \lfloor K/2 \rfloor$) becomes smaller (thus N could be small to satisfy condition (2) due to $1 - \sqrt{\max\{\forall i \in [1, K], \frac{u(T_i)-T_i}{u(T_i)-c(T_i)}\}}$ is fixed when N is close to $K/2$). The actually computation cost shown in Fig. 8(a) demonstrates a similar varying trend as the



(a)



(b)

Fig. 8. Computational performance II of the incentive compatible protocol. (a) Varying number of colluding agents (largest dishonest coalition). (b) Varying maximum payoff ratio. (Colors are visible in the online version of the article; http://dx.doi.org/10.3233/JCS-2012-0452.)

minimum number of SCG protocol run required in the incentive compatible protocol N (this matches the above analysis).

Finally, we examine the computation cost of the incentive compatible protocol (and N) on different maximum payoff ratios $\max\{\forall i \in [1, K], \frac{u(T_i)-T_i}{u(T_i)-c(T_i)}\}$ (fixed LP problem, total number of agents and percentage of the colluding agents). We can see that if the maximum payoff ratio increases (probably the payoff of the uncaught cheating $u(T_i)$ or the payoff of the caught cheating $c(T_i)$ in any agent P_i ($i \in [1, K]$)'s payoff ratio increases), N should be enlarged to maintain incentive compatibility. We can observe this in Fig. 8(b) and the fourth row of Table 4.

8. Conclusion and future work

In this paper, we studied the security and privacy issues in the collaborative LP problems where the linear constraints are arbitrarily partitioned among K agents and each agent holds its own set of variables. We introduced an extremely efficient protocol to solve such distributed LP problems with limited disclosure in semi-honest adversary model – assuming that all honest-but-curious agents follow the protocol but may infer private information from each other. Moreover, we discussed some major malicious behavior in the collaboration – every agent may play dishonestly (modifying the protocol) to gain additional benefits/payoffs. In such malicious model, we also presented an efficient incentive compatible protocol which enforce all rational agents to play honestly since honest play gains more payoff under the proposed cheating detection mechanism. We analyzed the security, computation and communication cost of the protocols in both adversary models. Also, we conducted groups of experiments to test the efficiency and scalability of our protocols in both adversary models.

We have tackled the dishonesty issues while solving distributed LP problems. Nevertheless, input cheating (which means changing its inputs to gain additional payoff before executing the protocol) is still a challenging problem in many practical distributed computation scenarios and online applications. For example, if one agent P_i modifies its local constraints in collaborative production – it inputs larger amount of labor to manufacture its products in the local constraints of the distributed LP problem, P_i's payoff computed from the distributed LP problem might be increased (which shows that P_i can utilize more shared raw materials from other agents). On the contrary, if P_i changes the coefficients of its objective function (e.g. increase the selling prices of their products), P_i is supposed to gain more raw materials from other agents in the collaboration as well. In the above cases, since P_i cheats before the collaboration and the protocol will recognize any deviated input as the correct input, it is nearly impossible to catch this cheating in distributed LP problems.

Actually, for the first case, the real-world constraints may restrict each agent's incentive to modify its inputs – P_i may not have the capacity to manufacture the increased amount of production since its practical local labor cannot afford such large

amount of production computed from the distributed LP problem. For the second case, we may attest the objective functions over a period of time (while implementing collaboration after solving the K-LP problem) to prevent such kind of input cheating. Inspired from the above ideas, we will try to figure out the input cheating issue arising in the distributed LP problems in the future work. Furthermore, we may also explore some other game theoretic approaches to establish reputation in specific distributed optimization scenarios, such as outsourcing share of the distributed LP problems to the cloud.

Acknowledgment

The work of Vaidya and Hong is supported in part by the National Science Foundation under Grant No. CNS-0746943.

Appendix A: Proofs

A.1. Proof of Theorem 3

Suppose $x^* = (Q_1 y_1^*, Q_2 y_2^*, \ldots, Q_K y_K^*)$ is not the optimal solution of the original vertical LP problem. In this case, we have another vector $x' = (x'_1, x'_2, \ldots, x'_K)$ such that $c^{\mathrm{T}} x' > c^{\mathrm{T}} x^* \Rightarrow c_1^{\mathrm{T}} x'_1 + \cdots + c_K^{\mathrm{T}} x'_K > c_1^{\mathrm{T}} x_1^* + \cdots + c_K^{\mathrm{T}} x_K^*$ where $Mx' \bowtie b$ and $x' \geqslant 0$. Let $y' = (y'_1, \ldots, y'_K) = (Q_1^{-1} x'_1, \ldots, Q_K^{-1} x'_K)$, thus we have $c_1^{\mathrm{T}} Q_1 y'_1 + \cdots + c_K^{\mathrm{T}} Q_K y'_K = c_1^{\mathrm{T}} Q_1 Q_1^{-1} x'_1 + \cdots + c_K^{\mathrm{T}} Q_K Q_K^{-1} x'_K = c_1^{\mathrm{T}} x'_1 + \cdots + c_K^{\mathrm{T}} x'_K$.

Thus, $c_1^{\mathrm{T}} x'_1 + \cdots + c_K^{\mathrm{T}} x'_K = c_1^{\mathrm{T}} Q_1 y'_1 + \cdots + c_K^{\mathrm{T}} Q_K y'_K > c_1^{\mathrm{T}} x_1^* + \cdots + c_K^{\mathrm{T}} x_K^* \Rightarrow c_1^{\mathrm{T}} Q_1 y'_1 + \cdots + c_K^{\mathrm{T}} Q_K y'_K > c_1^{\mathrm{T}} Q_1 Q_1^{-1} x_1^* + \cdots + c_K^{\mathrm{T}} Q_K Q_K^{-1} x_K^* \Rightarrow c_1^{\mathrm{T}} Q_1 y'_1 + \cdots + c_K^{\mathrm{T}} Q_K y'_K > c_1^{\mathrm{T}} Q_1 y_1^* + \cdots + c_K^{\mathrm{T}} Q_K y_K^*$ (since $Q_1^{-1} x_1^* = y_1^*, \ldots, Q_K^{-1} x_K^* = y_K^*$).

Hence, y' is a better solution than y^* which is a contradiction to that y^* is the optimal solution. Thus, Theorem 3 has been proven.

A.2. Proof of Theorem 4

We prove this equivalence in two facts.

First, suppose that the polyhedron $B_i x_i \bowtie_i b_i$ is feasible and one of its feasible solutions is x_i. Now, we have all the constraints (equalities or inequalities) in B_i that satisfy $B_i x_i \bowtie_i b_i$. Let $x_i = Q_i y_i$, hence $B_i Q_i y_i \bowtie_i b_i$ are all satisfied and the polyhedron $B_i Q_i y_i \bowtie_i b_i$ is feasible.

On the contrary, suppose that the polyhedron $B_i Q_i y_i \bowtie_i b_i$ is feasible and one of its feasible solutions is y_i. Now, we have all the constraints (equalities or inequalities) in $B_i Q_i$ that satisfy $B_i Q_i y_i \bowtie_i b_i$. Let $y_i = Q_i^{-1} x_i$, hence $B_i x_i \bowtie_i b_i$ are all satisfied and the polyhedron $B_i x_i \bowtie_i b_i$ is feasible.

Thus, Theorem 4 has been proven.

Appendix B: Dantzig–Wolfe decomposition: Master problem and original problem

Proposition 1. *If λ is a primal feasible and optimal for the master problem, then $x = \Pi^{-1}(\lambda)$ is primal feasible and optimal for the original problem and has the same objective value. Vice-versa.*

Proof. See [37]. □

Proposition 2. *If (π, v) is dual feasible and optimal for the original problem, then (π, μ) is dual feasible and optimal for the master problem where $u_i = v_i b_i$. Conversely, if (π, μ) is dual feasible and optimal for the master problem, then (π, v) is dual feasible and optimal for the original problem where v_i is a dual solution to the agent P_i's pricing problem.*

Proof. See [37]. □

Proposition 3. *The master problem is equivalent to the original problem in feasibility.*

Proof. See [37]. □

References

[1] M.J. Atallah, V. Deshpande, H.G. Elmongui and L.B. Schwarz, Secure supply-chain protocols, in: *Proceedings of the 2003 IEEE International Conference on E-Commerce*, Newport Beach, CA, June 24–27, 2003, pp. 293–302.

[2] B. Awerbuch and Y. Azar, Local optimization of global objectives: competitive distributed deadlock resolution and resource allocation, in: *Proc. IEEE Symposium on Foundations of Computer Science*, 1994, pp. 240–249.

[3] Y. Bartal, J.W. Byers and D. Raz, Fast, distributed approximation algorithms for positive linear programming with applications to flow control, *SIAM J. Comput.* **33**(6) (2004), 1261–1279.

[4] A. Bednarz, N. Bean and M. Roughan, Hiccups on the road to privacy-preserving linear programming, in: *Proceedings of the 8th ACM Workshop on Privacy in the Electronic Society*, ACM, New York, NY, USA, 2009, pp. 117–120.

[5] O. Catrina and S. de Hoogh, Secure multiparty linear programming using fixed-point arithmetic, in: *ESORICS*, 2010, pp. 134–150.

[6] C. Clifton, A. Iyer, R. Cho, W. Jiang, M. Kantarcioglu and J. Vaidya, An approach to identifying beneficial collaboration securely in decentralized logistics systems, *Manufacturing & Service Operations Management* **10**(1) (2008), 108–125.

[7] G. Dantzig, *Linear Programming and Extensions*, Princeton Univ. Press, Princeton, NJ, USA, 1963.

[8] X. Deng and C.H. Papadimitriou, Distributed decision-making with incomplete information, in: *Proceedings of the 12th IFIP Congress*, Madrid, 1992.

[9] W. Du, A study of several specific secure two-party computation problems, PhD thesis, Purdue University, West Lafayette, IN, USA, 2001.

[10] W. Du and M.J. Atallah, Privacy-preserving cooperative scientific computations, in: *Proceedings of the 14th IEEE Computer Security Foundations Workshop*, 2001, pp. 273–282.
[11] X. Feng and Z. Zhang, The rank of a random matrix, *Appl. Math. Comput.* **185**(1) (2007), 689–694.
[12] J.N. Franklin, *Matrix Theory*, Dover, New York, USA, 2000.
[13] H. Gintis, *Game Theory Evolving: A Problem-Centered Introduction to Modeling Strategic Interaction*, Princeton Univ. Press, Princeton, NJ, USA, 2009.
[14] O. Goldreich, *The Foundations of Cryptography*, Cambridge Univ. Press, Cambridge, 2004.
[15] O. Goldreich, S. Micali and A. Wigderson, How to play any mental game – a completeness theorem for protocols with honest majority, in: *Proceedings of the 19th ACM Symposium on the Theory of Computing*, ACM, New York, NY, USA, 1987, pp. 218–229.
[16] Y. Hong, J. Vaidya and H. Lu, Efficient distributed linear programming with limited disclosure, in: *DBSec*, 2011, pp. 170–185.
[17] Y. Hong, J. Vaidya, H. Lu and B. Shafiq, Privacy-preserving tabu search for distributed graph coloring, in: *SocialCom/PASSAT*, 2011, pp. 951–958.
[18] S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi, Optimization by simulated annealing, *Science* **220**(4598) (1983), 671–680.
[19] J. Li and M.J. Atallah, Secure and private collaborative linear programming, in: *Proc. International Conference on Collaborative Computing: Networking, Applications and Worksharing*, 2006, pp. 1–8.
[20] W. Li, H. Li and C. Deng, Privacy-preserving horizontally partitioned linear programs with inequality constraints, *Optim. Lett.* (2011), DOI: 10.1007/s11590-011-0403-2.
[21] R. Mailler and V. Lesser, Solving distributed constraint optimization problems using cooperative mediation, in: *Proceedings of 3rd International Joint Conference on Autonomous Agents and Multiagent Systems*, IEEE Computer Society, Los Alamitos, CA, USA, 2004, pp. 438–445.
[22] O.L. Mangasarian, Privacy-preserving linear programming, *Optim. Lett.* **5**(1) (2011), 165–172.
[23] O.L. Mangasarian, Privacy-preserving horizontally partitioned linear programs, *Optim. Lett.* **6**(3) (2012), 431–436.
[24] P.J. Modi, W.-M. Shen, M. Tambe and M. Yokoo, An asynchronous complete method for distributed constraint optimization, in: *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems*, ACM Press, New York, NY, USA, 2003, pp. 161–168.
[25] G.L. Nemhauser and L.A. Wolsey, *Integer and Combinatorial Optimization*, Wiley, New York, NY, USA, 1988.
[26] C.H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Upper Saddle River, NJ, USA, 1982.
[27] C.H. Papadimitriou and M. Yannakakis, On the value of information in distributed decision-making (extended abstract), in: *Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing*, ACM Press, New York, NY, USA, 1991, pp. 61–64.
[28] C.H. Papadimitriou and M. Yannakakis, Linear programming without the matrix, in: *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, ACM Press, New York, NY, USA, 1993, pp. 121–129.
[29] A. Petcu and B. Faltings, A scalable method for multiagent constraint optimization, in: *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, 2005, pp. 266–271.
[30] A. Petcu and B. Faltings, Superstabilizing, fault-containing multiagent combinatorial optimization, in: *Proceedings of the National Conference on Artificial Intelligence*, 2005, pp. 449–454.
[31] A. Petcu and B. Faltings, Approximations in distributed optimization, in: *Proc. Workshop on Distributed and Speculative Constraint Processing*, 2005, pp. 802–806.
[32] J. Sakuma and S. Kobayashi, A genetic algorithm for privacy preserving combinatorial optimization, in: *GECCO*, 2007, pp. 1372–1379.
[33] M.-C. Silaghi, B. Faltings and A. Petcu, Secure combinatorial optimization simulating DFS tree-based variable elimination, in: *Proc. 9th Symposium on Artificial Intelligence and Mathematics*, Ft. Lauderdale, FL, USA, January 2006, available at: http://www2.cs.fit.edu/ msilaghi/papers/.

[34] M.-C. Silaghi and V. Rajeshirke, The effect of policies for selecting the solution of a disCSP on privacy loss, in: *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multiagent Systems*, 2004, pp. 1396–1397.

[35] M.-C. Silaghi and D. Mitra, Distributed constraint satisfaction and optimization with privacy enforcement, in: *Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, Beijing, 2004, pp. 531–535.

[36] K. Suzuki and M. Yokoo, Secure generalized Vickrey auction using homomorphic encryption, in: *Proceedings of the 7th Annual Conference on Financial Cryptography*, Lecture Notes in Computer Science, Vol. 2742, Springer, Heidelberg, 2003, pp. 239–249.

[37] J.R. Tebboth, A computational study of Dantzig–Wolfe decomposition, PhD thesis, University of Buckingham, 2001.

[38] E. Turban, R.K. Rainer and R.E. Potter, *Introduction to Information Technology*, 3rd edn, Wiley, New York, NY, USA, 2005.

[39] J. Vaidya, Privacy-preserving linear programming, in: *SAC*, 2009, pp. 2002–2007.

[40] J. Vaidya, A secure revised simplex algorithm for privacy-preserving linear programming, in: *Proceedings of the 23rd IEEE International Conference on Advanced Information Networking and Applications*, 2009, pp. 347–354.

[41] A.C. Yao, How to generate and exchange secrets, in: *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, IEEE Computer Society, Los Alamitos, CA, USA, 1986, pp. 162–167.

[42] M. Yokoo, E.H. Durfee, T. Ishida and K. Kuwabar, Distributed constraint satisfaction for formalizing distributed problem solving, in: *Proceedings of International Conference on Distributed Computing Systems*, 1992, pp. 614–621.

[43] M. Yokoo and K. Hirayama, Distributed breakout algorithm for solving distributed constraint satisfaction problems, in: *Proceedings of the 1st International Conference on Multi-Agent Systems*, V. Lesser, ed., MIT Press, Cambridge, MA, USA, 1995, p. 467.

[44] M. Yokoo, K. Suzuki and K. Hirayama, Secure distributed constraint satisfaction: reaching agreement without revealing private information, in: *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming*, Springer-Verlag, London, UK, 2002, pp. 387–401.

[45] W. Zhang and L. Wittenburg, Distributed breakout algorithm for distributed constraint optimization problems – dbarelax, in: *Proceedings of the International Joint Conference on Autonomous Agents and Multi Agent Systems*, 2003, pp. 185–192.