

PRIVACY-PRESERVING COLLABORATIVE OPTIMIZATION

by
Yuan Hong

A dissertation submitted to the
Graduate school-Newark
Rutgers, The State University of New Jersey
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
Graduate Program in Management
Information Technology Major

Written under the direction of
Dr. Jaideep Vaidya
and approved by

Newark, New Jersey
October 2013

© Copyright 2013
Yuan Hong
All Rights Reserved

DISSERTATION ABSTRACT

PRIVACY-PRESERVING COLLABORATIVE OPTIMIZATION

By Yuan Hong

Dissertation Director: Dr. Jaideep Vaidya

With the rapid growth of computing, storing and networking resources, data is not only collected and stored, but also analyzed by different parties. This creates serious privacy problems while inhibiting the use of such distributed data. In turn, this raises the question of whether it is possible to realize value from distributed data without violating security and privacy concerns. Privacy-preserving data analysis has attracted considerable attention in recent years. Specifically, classification, clustering, association rule mining, outlier detection, regression among others, are securely implemented to analyze data privately held by multiple parties. The basic premise of such secure data analysis is that only the data analysis result can be revealed.

As a fundamental problem found in many diverse fields, optimization is the study of problems in which one seeks to minimize or maximize a real function by systematically choosing the values of real or integer variables within an allowed set. Inspired from multiparty data analysis, the ubiquitous collection of data opens even greater opportunities in the optimization problems, applicable to the fields

of operations research, computer science and mathematics. Collaborative optimization, when done properly with distributed data from different organizations, can facilitate them to improve the allocation of global resources without compromising on security. The primary goal of this dissertation is to develop privacy-preserving collaborative optimization techniques that would allow organizations to gain the maximum value from local information without (or with limited) information disclosure. While answering this problem, an inherent aim is to solve these fundamental problems underlying privacy-preserving analysis and secure multiparty computation (SMC) while making it more accessible and applicable.

In this dissertation, we look at fundamental optimization problems such as linear programming, non-linear programming and some classic NP-hard problems. Particularly, we discuss the potential security and privacy concern in the collaborative formulations of them, which occur in real world, such as logistics and scheduling in supply chain management. To securely solve them, we present efficient privacy-preserving methods along with formal security analysis for the proposed privacy notions. In addition, we identify a potential attack to an earlier work and amend the transformation method with enhanced security guarantee. We also address how game theoretic techniques can be used to solve some of the fundamental incentive problems underlying secure multiparty computation in collaborative optimization. The computation/communication cost analysis and the experimental results demonstrate the feasibility, applicability and scalability of the proposed approaches.

PREFACE

Dissertation Committee Members

- Dr. Nabil Adam, Rutgers University
- Dr. Vijayalakshmi Atluri, Rutgers University
- Dr. Christopher W. Clifton, Purdue University
- Dr. Jaideep Vaidya, Rutgers University

ACKNOWLEDGEMENTS

This work signifies the accumulative research contributions in the past five years. I would like to express my deepest gratitude to my esteemed advisor, Prof. Jaideep Vaidya. Without him, this dissertation would be impossible and I would not have gone so far in academia. He helped me a lot not only on guiding my research but also on building my academic career. Also, I really appreciate his generous financial support for my doctorate study.

I am very grateful to Prof. Nabil Adam, Prof. Vijayalakshmi Atluri, and Prof. Christopher Clifton for serving on the committee and helping me improve my dissertation with their highly constructive comments. Center for Information Management, Integration and Connectivity (CIMIC), established by them, provides me excellent equipments, facilities and environment to conduct research. Thank you for all of these.

I would also like to express my special thanks to my prior advisor Prof. Lingyu Wang at Concordia University (Montreal, Canada), Prof. Haibing Lu at Santa Clara University, Prof. Miklos Vasarhelyi at Rutgers Business School (Department of Accounting and Information Systems), and Dr. Mingrui Wu at Microsoft. I learnt quite a lot from the collaboration with them. Meanwhile, they gave me a lot of very helpful suggestions on my job search and career development. Thanks

also go to Prof. Basit Shafiq, Prof. Panagiotis Karras, Prof. Xiaoyun He, Prof. Soon Ae Chun, Dr. Anirban Basu, and also my friends/colleagues at Rutgers: David Lorenzi, Nazia Badar, Emre Uzun, Shengbin Wang, Xin Xu, Victoria Chiu, Hussein Issa, Vincent Ogutu, et al. It was a great pleasure to work with all of them.

For my family, no words can describe my great gratitude. Their endless love and unwavering support helped me immensely. I particularly thank my wife Qi for constantly encouraging and supporting me during the tough time, my daughter Mabel for always bringing me greatest joy and sunshine, and my parents for their high standard of expectation leading to the success. I dedicate this work to all of you.

TABLE OF CONTENTS

ABSTRACT	ii
PREFACE	iv
ACKNOWLEDGEMENTS	v
LIST OF TABLES	x
LIST OF FIGURES	xi
CHAPTER 1. INTRODUCTION	1
1.1 Real-world Business Collaboration	1
1.1.1 Collaborative Logistics	1
1.1.2 Collaborative Production	3
1.1.3 Collaborative Scheduling	4
1.2 Security and Privacy Concern in Collaboration	5
1.2.1 Trusted Third-party	6
1.2.2 Secure Multiparty Computation (SMC)	8
1.3 Summary of the Dissertation	9
CHAPTER 2. RELATED WORK	11
2.1 Secure Multiparty Computation	11
2.2 Privacy-preserving Data Analysis/Mining	13
2.3 Collaborative/Distributed Optimization	15
2.3.1 Privacy-preserving Collaborative Optimization	18

CHAPTER 3. BACKGROUND	21
3.1 Optimization	21
3.1.1 Linear Programming (LP)	22
3.1.2 Non-linear Programming (NLP)	26
3.1.3 NP-hard Problem 1: Graph Coloring	29
3.1.4 NP-hard Problem 2: Traveling Salesman Problem	31
3.2 Secure Multiparty Computation	33
3.2.1 Semi-honest Model	34
3.2.2 Malicious Model – Cryptographic and Economic Notion	36
3.3 Research Challenges of Privacy-preserving Collaborative Optimization	37
CHAPTER 4. SEMI-HONEST COLLABORATIVE LINEAR PROGRAMMING (LP)	42
4.1 An Inference-proof to Horizontally Partitioned LP [75, 90]	42
4.1.1 Inference Attack against the Existing Work [75, 90]	43
4.1.2 Algorithm	48
4.1.3 Computational Results	58
4.2 Arbitrarily Partitioned LP Problem	59
4.2.1 Revised Dantzig-Wolfe Decomposition	63
4.2.2 Secure Column Generation (SCG) Protocol for K-LP Problem ..	73
4.2.3 Experiments	86
CHAPTER 5. MALICIOUS COLLABORATIVE LINEAR PROGRAMMING (LP)	93
5.1 Malicious Behavior in the SCG Protocol	93
5.2 Nash Equilibrium in the SCG Protocol	96
5.2.1 Dominant Strategy for Nash Equilibrium	96
5.2.2 Approach to Establish Nash Equilibrium	97
5.3 Incentive Compatible Protocol	98
5.3.1 Multiparty Transformation	99
5.3.2 Achieving Incentive Compatibility	106
5.3.3 Protocol and Analysis	112
5.4 Experiments	113

5.4.1	Groups of Experiments	113
5.4.2	Performance	115
CHAPTER 6. GOING BEYOND LINEAR PROGRAMMING		118
6.1	Non-linear Programming	118
6.1.1	Secure Transformation and Permutation	119
6.1.2	Algorithm	121
6.1.3	Security and Cost Analysis	123
6.2	Collaborative Graph Coloring	126
6.2.1	Problem Definition	128
6.2.2	Privacy-preserving Tabu Search	129
6.2.3	Security and Cost Analysis	149
6.2.4	Experiments	153
6.3	Collaborative Traveling Salesman Problem	156
6.3.1	Problem Definition	160
6.3.2	Privacy-preserving Simulated Annealing	165
6.3.3	Security and Cost Analysis	171
CHAPTER 7. CONCLUSION AND FUTURE WORK		177
REFERENCES		182

LIST OF TABLES

1.1	Collaborative Production	3
4.1	Experimental Setting for Protocol Comparison (Two Parties)	89
4.2	K-LP Problems with Varying # of Parties (Problem Size: 500×1000)	89
4.3	K-LP Problems with Varying Size (20 Parties)	90
5.1	N in the Incentive Compatible Protocol	116
6.1	Partitioned NLP Problem	119
6.2	Notations in Synchronous Move	145
6.3	k Shipping Companies' Cost Vectors	161
6.4	k Shipping Companies' Cost Vectors (Simplified Notation)	162
6.5	Finer-grained Traveling Route Vector with Route Assignment	162

LIST OF FIGURES

1.1	Collaborative Logistics	2
1.2	Collaborative Scheduling	5
3.1	Piecewise Linear Approximation of Convex Function $\log(x_i + 1)$.	28
3.2	Graph Coloring Problem	29
4.1	Local Transformation in Revised Dantzig Decomposition	71
4.2	Optimal Solution Reconstruction after Solving the Transformed K-LP Problem	72
4.3	Secure K-party Column Generation Protocol	84
4.4	Computational Performance Comparison of Secure Collaborative LP Protocols (Logarithmic Scale)	91
4.5	Computational Performance of the SCG Protocol	92
5.1	Computational Performance I of the Incentive Compatible Protocol	115
5.2	Computational Performance II of the Incentive Compatible Protocol	116
6.1	Collaborative Graph Coloring Problem	127
6.2	Private Information in Collaborative Graph Coloring	130
6.3	Secure Conflict Computation (Secured Sum of the Scalar Products)	133
6.4	An Example for Inference Attack on Secure Comparison [$P_1(n=3)$ has a border vertex that is adjacent to one vertex of $P_0(n=1)$ and two vertices of $P_2(n=2)$]	141
6.5	Privacy-preserving Tabu Search Protocol	147
6.6	P_0, \dots, P_{m-1} 's View (the sign of $\mu(x') - \mu(x)$: 1 is $\mu(x') < \mu(x)$; 0 is $\mu(x') \geq \mu(x)$)	150
6.7	Total Number of Conflicts ($\mu(x)$) in each Iteration	155
6.8	Optimal Results and Computation Cost Testing	155
6.9	Collaborative Shipping Companies	158

6.10 Client E_C 's Traveling Route	159
--	-----

CHAPTER 1

INTRODUCTION

1.1 Real-world Business Collaboration

With the rapid growth of computing, storing and networking resources, the collaboration among different parties occurs frequently in business world. For example, different companies usually collaborate with each other to jointly optimize the operations, such as production, scheduling and delivery. To facilitate the decision making, collaborative optimization problems are generally formulated amongst those companies for seeking the global maximum profit to minimum cost. Essentially, various collaborative optimization models have been widely applied to solve many real-world problems in almost all industries. We now consider three motivating examples in logistics, production and scheduling respectively.

1.1.1 Collaborative Logistics

In the packaged goods industry, delivery vehicles are empty 25% of the time [100]. Just two years ago, Land O'Lakes spent much of their time shuttling empty vehicles 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 bound to the same destination, huge savings were realized.

For instance, General Mills and Land O'Lakes are two food producing companies in the United States [10, 19]. As shown in the left-hand side part of Figure 1.1, General Mills plans to deliver pizza rolls from a warehouse in New York to New Jersey, Pennsylvania, Illinois and Ohio with an optimal route computed by itself. In the meanwhile, Land O'Lakes delivers its goods from Ohio to New Jersey. If they do not collaborate with each other, both companies' delivery routes include a lot of "empty vehicle movement", which means that the vehicles are empty but traveling on the route. Alternatively, the collaborative logistics service enables Land O'Lakes and General Mills to seek their global minimum transportation cost by sharing the vehicles, empty vehicle movement can be significantly reduced as shown in the right-hand side part of Figure 1.1. Then, the transportation costs – both fuel cost and drivers' working hours, can be saved via collaborative optimization.

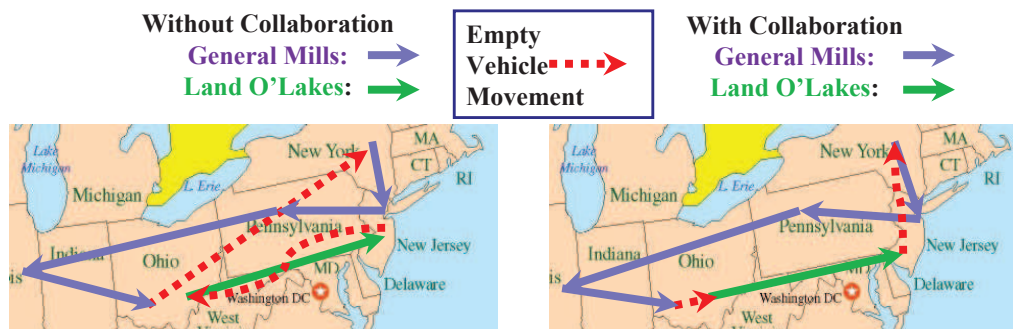


Figure 1.1. Collaborative Logistics

In fact, freight costs of Land O'Lakes were cut by 15% after employing collaborative logistics service, with an annual savings of 2 million [123]). This opens greater opportunities for us to pursue collaborative optimization in logistics.

1.1.2 Collaborative Production

Manufacturing planning and control facilitate decision makers to determine the optimal production activities with limited resources. For example, two factories manufacture two different products – Bowls and Mugs respectively with the same raw material clay. The demand of clay and labor hours to produce every bowl and mug is given in Table 1.1, and each factory has limited amount of clay and limited number of laboring hours. With the unit revenue of these two products, the factories intend to make the optimal production assignment that can maximize their revenue.

Table 1.1. Collaborative Production

Product	Labor hr/unit	Clay lb/unit	Revenue \$/unit
Bowl (Factory 1)	1	4	40
Mug (Factory 2)	2	3	50

Factory 1: 10 hours of labor and 60 pounds of clay every day

Factory 2: 30 hours of labor and 60 pounds of clay every day

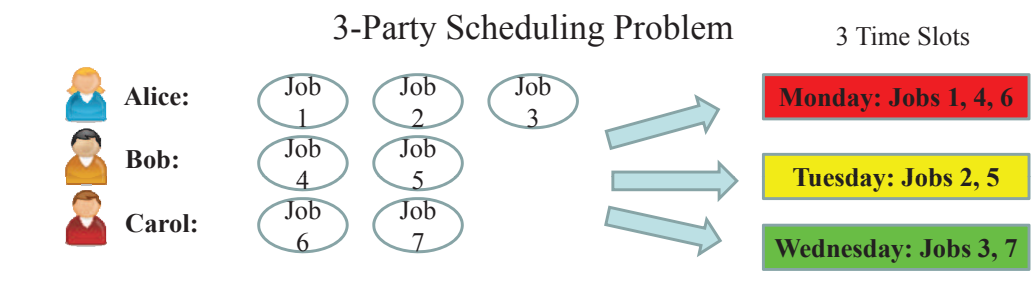
Considering the non-collaborative scenario, each factory can formulate a simple linear programming problem to obtain its maximum revenue respectively, which are 10 bowls (\$400) and 15 mugs (\$750). However, the resources are not well utilized if they derive the optimal solution individually, since Factory 1 does not have sufficient hours of labor compared to its possessed raw materials. Instead, if they collaborate with each other by sharing the raw materials and/or hours of labor, they can formulate a global linear programming problem and obtain the optimal production assignment as 24 bowls and 8 mugs (\$1,360 in total which is clearly greater than $400+750=\$1,150$). Thus, collaborative optimiza-

tion can create more revenue for organizations by better utilizing scarce resources in production.

1.1.3 Collaborative Scheduling

Scheduling covers a wide variety of real-life problems which are applied or extended from optimization models, particularly in supply chain management. A typical scheduling problem seeks the optimal solution of assigning some jobs or tasks to several time slots, while possibly subjecting to some constraints. Similar to transportation and production, many of such problems also involve different parties to jointly optimize the scheduling process. For example in Figure 1.2, Alice, Bob and Carol have their own set of jobs respectively, and they plan to assign their jobs into three time slots: Monday, Tuesday and Wednesday. However, some jobs cannot be assigned into the same time slots, possibly because they share the same machine or need the same technician. Completing such a scheduling problem is to find an optimal job assignment solution without conflicts – the conflicted pair of jobs are assigned to different time slots (the model will be introduced later on in this dissertation).

Compared with scheduling by every party individually, the collaborative scheduling problem does not explicitly improve the performance of the optimization. In other words, the total number of required time slots may not be reduced in the cooperative scenario. However, since Alice, Bob and Carol can share the machine or technician to work on their own jobs, this kind of collaboration can also bring great benefits to different parties if they have limited resources.



Time Conflicts: some pair of jobs **cannot be assigned into the same time slot**

→ jobs share the same machine: (1, 2), (1, 3), (2, 3), (4, 5), (6, 7)

→ jobs need the same technician: (2, 4), (5, 7), (3, 6)

Figure 1.2. Collaborative Scheduling

1.2 Security and Privacy Concern in Collaboration

In every collaborative optimization problem, all parties should formulate the optimization problem with their own local information (e.g., their limited resources), and eventually solve it. However, such local information may involve considerable amount of commercial secret, for example, how many pizza rolls should be delivered from New York to New Jersey for General Mills. Meanwhile, those cooperative companies are competitors in the market, e.g., Land O'Lakes and General Mills. Thus, it is impossible for those participants to completely share their local information.

Innumerable such situations exist, where appropriate safe and secure use of data leads to immense financial and social benefits. For example, consider the case of Ford and Firestone. In 2001, numerous accidents due to tread separation were reported. Initially both companies blamed each other. It turned out that it was only Ford Explorers with Firestone tires from the Decatur, Illinois plant, in specific situations that had these problems. If found out earlier, much loss could

have been avoided. While both companies individually collect a lot of pertinent testing data, this was not shared due to commercial concerns. Moreover, recall that Land O'Lakes saves their transportation cost with collaborative logistics in the packaged goods industry. Though this required sending all information to a central site, there was no alternative if savings were desired. However, this dissertation would make this possible without the release of proprietary information. Finally, Walmart, Target and CostCo ship millions of dollars worth of goods over the seas every month. These feed in to 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 to determine what trucks should make their way to which ports to be loaded onto certain ships, i.e., 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, better solutions must be found.

1.2.1 Trusted Third-party

Most of the commercial companies currently employ a trusted third-party to solve the collaborative optimization problem by completely sharing their local private information to the third-party beforehand. For example, IBM Sterling Transportation Management System, which was formerly Nistevo.com, provides collaborative logistics services for companies in the United States, like General Mills and Land O'Lakes.

Although trusted third-party gives a common solution for contemporary society, it is not always the best choice for commercial companies due to several inherent drawbacks. First, trusted third-party requires high degree of trust in reality, where all participants involved in the collaboration should trust the external third-party. In case that one party doesn't trust the third-party, the collaboration might be in stuck and therefore all participants should find alternative solutions. Second, even though the local private information is not directly shared amongst collaborative companies at this time, such information is disclosed to the third-party. For instance, as competitors, General Mills and Land O'Lakes do not know the constraints and delivery route/amount from each other, yet Nistevo.com knows everything from both companies since it acts the solver for the collaborative optimization problem. For this reason, some companies may be reluctant to share their local information, especially their confidential commercial secrets to external parties, even if sharing them to the third-party with signed agreements would not hurt them directly. Third, from the economic perspective of every collaborative company, the trusted third-party usually charges a lot for their services, e.g., Land O'Lakes pays \$250,000 for the collaborative logistics service to Nistevo.com every year. To some extent, finding a solution without trusted third-party can avoid such a big expenditure. Finally, if a centralized site plays the role of trusted third-party, all the computation regarding the collaborative optimization will be intensively centralized into a single site. This may lead to unaffordable burden to the computational capacity of the trusted third-party.

With the above concerns, it is extremely worth exploring alternative approaches that can securely solve the collaborative optimization problems for different par-

ties without establishing a trusted third-party, which is the primary aim of this dissertation.

1.2.2 Secure Multiparty Computation (SMC)

The field of Secure Multiparty Computation (SMC) addresses exactly this problem. The celebrated results [13, 46, 132] in this area show that any function can be securely computed in a distributed manner efficiently (e.g., in polynomial time with respect to the size of the circuit required to compute the function). However, this can lead to very inefficient solutions for complex functions or for large input sizes. More efficient solutions are necessary for some practical problems with complex functions or large-scale inputs.

Recall that data mining tasks are typically considered as a subcategory of the above practical problems (with complex functions or large-scale inputs). Therefore, privacy-preserving data analysis/mining [3, 80, 81] has attracted considerable attention in recent years – classification [34, 128, 130, 131, 136, 137], clustering [63, 76, 127], association rule mining [68, 126], outlier detection [129] and regression [70, 115] are securely implemented across distributed parties where each party privately holds a share of the global data. The basic premise of such secure data analysis is that only the data analysis result can be revealed.

Optimization is a fundamental problem found in many diverse fields. Similar to data analysis, to solve a real-world profit maximizing or cost minimizing problem, optimization problem is modeled with data in a particular scenario. Inspired from multiparty data analysis, the ubiquitous collection of data while formulating the collaborative optimization problem opens even greater research opportunities,

applicable to the fields of operations research, computer science and mathematics. In this dissertation, we look at the specific problem of how organizations optimize allocation of global resources while preserving the privacy of local information, and tackle the challenges in developing privacy-preserving collaborative optimization techniques. This would greatly increase the value of collected data in real-life collaboration and prevent data warehouses from turning into data cemeteries in any case.

1.3 Summary of the Dissertation

This dissertation will advance this state of the art by developing a suite of privacy-preserving collaborative optimization techniques using secure computation tools and methodologies. Initially, we will start the research problem with linear programming, since it is one of the most important subfields in optimization, applicable to numerous real-life problems. A secure solution for linear programming would be sufficient to solve the transportation and production problem discussed earlier. To have maximum impact, any study must start with linear programming. In addition, we also study the optimization problems beyond linear programming: non-linear programming and combinatorial optimization problems. On one hand, the objective function or constraints in many real world problems are *non-linear*, and non-linear optimization provides fundamental insights into mathematical analysis and is widely used in a variety of fields such as engineering design, regression analysis, inventory control, geophysical exploration, and economics. On the other hand, combinatorial optimization is to find an optimal solution from a *finite* set of objectives. It is a subset of mathematical optimiza-

tion which is related to operations research, algorithm theory, and computational complexity theory. It also has important applications in many fields, including artificial intelligence, machine learning, mathematics, auction theory, and software engineering. Thus, to focus our effort, we develop efficient algorithms for secure optimization in the following areas: (1) linear programming, (2) specific flavors of non-linear programming, and (3) combinatorial optimization, particularly the NP-hard problems.

The rest of this dissertation is organized as follows. We survey the related work of secure multiparty computation, privacy-preserving data analysis and collaborative optimization in Chapter 2. Chapter 3 gives some background knowledge related to this dissertation: some optimization problems and the corresponding solvers, secure multiparty computation, privacy-preserving collaborative optimization and game theory. Moreover, the privacy-preserving approaches under two different adversarial models to collaborative linear programming, are given in Chapter 4 (semi-honest) and 5 (malicious) respectively. Chapter 6 presents our contributions beyond linear programming, and more specifically, we introduce the privacy-preserving techniques to solve non-linear programming problem and NP-hard combinatorial optimization problems. Finally, we conclude the dissertation and discuss the future work in Chapter 7.

CHAPTER 2

RELATED WORK

Privacy is an important problem especially relevant to current times. Several research communities have independently developed solutions which contribute to privacy protection. Much of this work can be leveraged, if appropriately combined. We first discuss the work in the field of secure multiparty computation, and then some application of it to data analysis community. Finally, we overview the relevant work from the field of collaborative optimization.

2.1 Secure Multiparty Computation

The concept of secure multiparty computation (SMC) is especially relevant to this dissertation. The setting of SMC encompasses tasks as simple as coin-tossing and broadcast, and as complex as electronic voting, electronic auctions, electronic cash schemes, contract signing, anonymous transactions, and private information retrieval schemes. The key idea behind secure multiparty computation is that a computation is secure if at the end of the computation, no party learns anything except its own input and the results (and anything that can be inferred from these two pieces). The gold standard is that of a trusted-third party which performs the entire computation. Thus, the key is to achieve the same results without having a trusted-third party. While some communication is obviously required in order

to perform the computation, it is necessary to stick to messages that are useful yet do not reveal anything new. How is this possible? The answer lies in non-determinism. By allowing non-determinism in the exact values sent in the intermediate communication (e.g., encrypt with a randomly chosen key), and proving that a party using its own input and the result can generate a predicted intermediate computation that is as likely as the actual values is sufficient to show that no new information is revealed.

Secure computation has a very rich history. Yao first postulated the two-party comparison problem (Yao's Millionaire Protocol) and developed a provably secure solution [132]. Goldreich et al. [46] generalized this to multiparty computation and proved that there exists a secure solution for any functionality. The approach used is as follows: the function F to be computed is first represented as a combinatorial circuit, and then the parties run a short protocol for every gate in the circuit. Every participant gets random shares of the input and output wires for every gate. This approach, though appealing in its generality and simplicity, means that the number of rounds of the protocol grow with the size of the circuit. This grows with the size of the input. This is highly inefficient for large-scale inputs or complicated circuits, as in optimization. Although this proves secure solutions exist, achieving efficient secure solutions for collaborative optimization is still open.

There has been significant theoretical work in this area. Both [132] and [46] assumed polynomially time bounded passive adversaries. In a line of work initiated by Ben-Or et al. [13], the computational restrictions on the adversary were removed, but users were assumed to be able to communicate in pairs in perfect

secrecy. Ben-Or et al. [13] assume passive adversaries, while Chaum et al. [25] extend this to active adversaries. Ostrovsky and Yung [99] introduce the notion of mobile adversaries, where the corrupt users may change from round to round. Finally, the coercing adversary who can force users to choose their inputs in a way he favors was introduced in the context of electronic elections by Benaloh and Tunistra [15], and generalized to arbitrary multiparty computation by Canetti and Gennaro [21]. Much effort has been devoted to developing crisp definitions of security [20, 47, 95]. However, due to efficiency reasons, it is completely infeasible to directly apply the theoretical work from SMC to form secure protocols for optimization.

Secure multiparty computation does make two key contributions to this dissertation: first, methods for securely computing functions with small inputs (e.g., secure comparison); second, definitions and proof techniques for private and secure computations in a distributed environment. Creating efficient secure variants of different optimization techniques is a non-trivial task requiring much effort. Rather than using SMC directly to build a new solution for every optimization problem, a much better solution is to use SMC tools to build secure generalized methods to transform the data in a way that other techniques could then be applied.

2.2 Privacy-preserving Data Analysis/Mining

Along with electronic auctions [17, 18, 37, 98], another field that has gained a lot of attention in recent years is privacy-preserving data mining (PPDM). PPDM is really the application of secure computation to data mining. Work in PPDM has followed two major directions – the randomization/perturbation approach and the

cryptographic approach. Both of these are of some interest to us.

In the perturbation approach data is locally perturbed by adding “noise” before mining is done. For example, if we add a random number chosen from a gaussian distribution to the real data value, the data miner no longer knows the exact value. However, important statistics on the collection (e.g., average) will be preserved. Special techniques are used to reconstruct the original distribution (not the actual data values). The mining algorithm is modified to work while taking this into consideration. The seminal paper by Agrawal and Srikant [3] introduced this notion to the data mining community – univariate additive noise from either a uniform or gaussian distribution is used to locally perturb the data. Several different algorithms are proposed to reconstruct distributions and learn a decision tree classifier from the perturbed data. A convergence result was proved in [2] for a refinement of this algorithm. There has been work using the same approach for association rule mining [35, 113, 138]. Zhu and Lei [140] study the problem of optimal randomization for privacy-preserving data mining and demonstrate the construction of optimal randomization schemes for density estimation. This model works in the “data warehouse” model of data mining, but trades off privacy for the accuracy of results. However, several problems have been pointed out with the privacy inherent in such an approach [61, 69, 95].

The second approach to PPDM is a direct application of secure computation techniques to the data mining problem. Typically, cryptographic techniques are used to protect privacy. Lindell and Pinkas [80, 81] first proposed this to construct decision trees. There has since been significant work on many techniques in data mining. Specifically, secure methods have been proposed for association rule

mining [68, 126], clustering [63, 76, 127], classification [34, 128, 130, 131, 136, 137], outlier detection [129] and regression [70, 115]. Kantarcioglu and Vaidya [67] proposed an architecture for the mining of information. Clifton et al. [27] proposed the creation of a toolkit of components that could be used to quickly throw together a solution for a new data mining problem. The underlying tools can actually be used for any problem. Thus, some of the techniques and tools developed in this field are equally applicable to the field of optimization, and the results found by this dissertation would be applicable to all other applications of SMC as well.

2.3 Collaborative/Distributed Optimization

Optimization problems occur in all walks of real life. There is work in distributed optimization (viz. collaborative 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 that has been investigated in a worst case setting (with no communication between the distributed agents) by Papadimitriou et al. [29, 104, 105]. In [105], 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 [7] 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 [9] obtains a better approximation while using the

same number of rounds of local communication.

Distributed Constraint Satisfaction was formalized by Yokoo [133] 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 [96] 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 [88] centralizes parts of the problem; it is unknown apriori how much needs to be centralized where, and privacy is an issue. Distributed local search methods like DSA [72] / DBA [139] for optimization, and DBA for satisfaction [134] 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 [110] 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 [106–109, 111, 112]). [106] proposes an approximated 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. There is some work in secure optimization. Silaghi and Rajeshirke [118] show that a secure combinatorial problem solver must necessarily pick the result randomly among optimal solutions to be really secure. Silaghi and Mitra [117] 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 [119], though it is not completely secure under the framework in [118]. Yokoo et al. [135] also propose a scheme using public key encryption for secure distributed constraint satisfaction. Silaghi et al. [116] 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. [6] 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.

2.3.1 Privacy-preserving Collaborative Optimization

Recently, there has been significant interest in the area of privacy-preserving linear programming. We briefly review some of the relevant work. Work in privacy-preserving linear programming has followed two major directions – the problem transformation approach and the SMC based approach.

- **Transformation based approach.** Du [32] and Vaidya [124] 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. [10, 11] pointed out a potential attack to the above transformation approach. To correct the flaw in [124], we revised the transformation and extended the work to the multiparty scenario [54].

In addition, Mangasarian presented two transformation approaches for horizontally partitioned linear programs [90] and vertically partitioned linear programs [89] respectively. Li et al. [75] extended the transformation approach [90] for horizontally partitioned linear programs with equality constraints to inequality constraints. We have identified a potential inference attack to Mangasarian and Li’s transformation based approach, and revised the transformation with significantly enhance security guarantee in [53]. Meanwhile, we extend the above transformation to privacy-preserving non-linear programming, and apply the model to securely anonymize distributed data [58]. Furthermore, we proposed approaches to securely solve the arbitrarily partitioned collaborative linear programming problems in both semi-

honest and malicious adversarial models [55,57]. More recently, Dreier and Kerschbaum proposed a secure transformation approach for a complex data partition scenario, and illustrated the effectiveness of their approach [31].

- **SMC based approach.** Li and Atallah [74] 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 cryptographic tools. Vaidya [125] proposed a secure revised simplex approach also using SMC techniques but improved the efficiency compared with Li and Atallah 's approach [74]. Catrina and Hoogh [24] 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.

Furthermore, as for securing collaborative combinatorial optimization problems (particularly those NP-hard problems), Sakuma et al. [114] proposed a genetic algorithm for securely solving two-party distributed traveling salesman problem (TSP). 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 the multiparty collaborative combinatorial optimization). The TSP that is completely partitioned among multiple parties has been discussed but not solved in [114]. We consider a more complicated scenario in TSP where the problem is partitioned to multiple (more than two) parties, and securely solve it with privacy-preserving simulated annealing protocol in Chapter 6. Meanwhile, we addressed the privacy concern in another NP-hard problem, graph coloring which is partitioned among two or more

parties [59]. Also, we securely solve the graph coloring problem with our privacy-preserving tabu search protocol.

CHAPTER 3

BACKGROUND

This chapter introduces some background knowledge related to this dissertation, including the basic formulations of linear programming, non-linear programming, graph coloring and traveling salesman problem. We also briefly introduce the idea of secure multiparty computation and discuss the research challenges of privacy-preserving collaborative optimization.

3.1 Optimization

Optimization is the study of problems in which one seeks to minimize or maximize a real function by systematically choosing the values of real or integer variables within an allowed set. Formally, given a function $f : S \rightarrow R$ from some set S to the real numbers, we seek an element x_0 in S such that $f(x_0) \leq f(x), \forall x \in S$ (“minimization”) or such that $f(x_0) \geq f(x), \forall x \in S$ (“maximization”). Thus, an optimization problem has three basic ingredients:

- An *objective function* which we want to maximize or minimize.
- A set of *unknowns* or *variables* which affect the value of the objective function.

- A set of *constraints* that allow the unknowns to take on certain values or exclude others.

3.1.1 Linear Programming (LP)

In linear programming, the objective function f is linear and the (solution) set S is specified using only linear equalities and inequalities. Thus, for linear programming, the problem can be easily restated as:

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t. } & Ax \geq b \\ & x \geq 0 \end{aligned}$$

From the geometrical point of view, LP problems can be represented as polyhedra. Some geometrical definitions for LP problems are given as below.

Definition 3.1 (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 $P = \{x \in \mathbb{R}^n : Ax \bowtie b\}$ where M is an $m \times n$ constraint matrix.

Note that \bowtie denotes $<, >, =, \leq$ or \geq .

Definition 3.2 (Convex Set) A set $S \subseteq \mathbb{R}^n$ is convex if for any $x, y \in S$ and $\lambda \in [0, 1]$, $\lambda x + (1 - \lambda)y \in S$.

Proposition 3.1 A polyhedron is a convex set.

Definition 3.3 (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 $\lambda_i > 0$ with $\sum_i \lambda_i = 1$.

Definition 3.4 (Vertex) A point $x^e \in P$ is a vertex of $P = \{x \in \mathbb{R}^n : Ax \preceq b\}$ if it cannot be represented as a convex combination of two other points $x^i, x^j \in P$.

Definition 3.5 (Ray in Polyhedron) Given a non-empty polyhedron $P = \{x \in \mathbb{R}^n : Ax \preceq b\}$, a vector $r \in \mathbb{R}^n, r \neq 0$ is a ray if $Ar \preceq 0$.

Definition 3.6 (Extreme Ray) A ray r is an extreme ray of $P = \{x \in \mathbb{R}^n : Ax \preceq b\}$ if there does not exist two distinct rays r^i and r^j of P such that $r = \frac{1}{2}(r^i + r^j)$.

Theorem 3.1 (Optimal Solution [97] [102]) Consider a feasible LP problem $\{\max : c^T x | x \in P\}$ with $\text{rank}(A) = n$. If it is bounded it has an optimal solution at a vertex x^* of P ; if it is unbounded, it is unbounded along an extreme ray r^* of P (and so $c^T r^* > 0$).

A: Simplex Method

Simplex method was developed by Dantzig in 1947 [28]. It reveals a fact that the optimal solution will occur at a vertex of the polyhedron. The algorithm seeks the optimal solution by moving from vertex to vertex where the objective value would be improved. Although the number of iterations increases exponentially with the number of variables in the worst case, simplex method is considered as a standard solver for LP problems since it is easy to use and it solves LP problems efficiently in most cases.

B: Dantzig-Wolfe Decomposition

For block-angular structured LP, Dantzig-Wolfe Decomposition is more efficient compared to some standard solvers like simplex method, revise simplex method, etc. Consider the following LP problem, which is a typical block-angular structured LP problem, including two kinds of constraints (denoted as global constraints $A_1x_1 + A_2x_2 + \dots + A_Kx_K \bowtie_0 b_0$ and local constraints $B_1x_2 \bowtie_1 b_1, \dots$ where \bowtie represents $\leq, =$ or \geq).

$$\begin{aligned} & \max \quad c_1^T x_1 + c_2^T x_2 + \dots + c_K^T x_K \\ \text{s.t.} \quad & \begin{cases} x_1 \in \mathbb{R}^{n_1} \\ x_2 \in \mathbb{R}^{n_2} \\ \vdots \\ x_K \in \mathbb{R}^{n_K} \end{cases} : \begin{pmatrix} A_1 & \dots & A_K \\ B_1 & & \\ & \ddots & \\ & & B_K \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_K \end{pmatrix} \begin{matrix} \bowtie_0 \\ \bowtie_1 \\ \vdots \\ \bowtie_K \end{matrix} \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_K \end{pmatrix} \end{aligned} \quad (3.1)$$

Assume that we let x^i (size $n = \sum_{i=1}^K n_i$ vector) represent the vertex/extreme ray in the LP problem. Hence, every point inside the polyhedron can be represented by the vertices/extreme rays using convexity combination (Minkowski's Representation Theorem [97, 122]).

Theorem 3.2 (Minkowski's Representation Theorem [97, 102, 122]) *If $P = \{x \in \mathbb{R}^n : Ax \bowtie b\}$ is non-empty and $\text{rank}(A)=n$, then $P = P'$, where $P' = \{x \in \mathbb{R}^n : x = \sum_{i \in \Pi} \lambda_i x_i + \sum_{j \in R} \mu_j r^j; \sum_{i \in \Pi} \lambda_i = 1; \lambda, \mu > 0\}$ and $\{x^i\}_{i \in \Pi}$ and $\{r^j\}_{j \in R}$ are the sets of vertices/extreme rays of P respectively.*

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

$$\delta_i = \begin{cases} 1 & \text{if } x^i \text{ is a vertex} \\ 0 & \text{if } x^i \text{ is an extreme ray} \end{cases} \quad (3.2)$$

As a result, a projection from P to P' is given: $\{\Pi : \mathbb{R}^n \mapsto \mathbb{R}^{|E|}; x = \sum_i \lambda_i x^i \mapsto \lambda\}$. If the polyhedron P is Block-angular structure, it can be decomposed to smaller dimensional polyhedra $\forall P_i = \{x_i \in \mathbb{R}^{n_i} : B_i x_i \bowtie_i b_i\}$ where x_i is a n_i dimensional vector, $n = \sum_{i=1}^K n_i$ (as shown in Equation 3.1). Additionally, we denote polyhedron of the global constraints as $P_0 = \{x \in \mathbb{R}^n : (A_1 \ A_2 \ \dots \ A_K)x \bowtie_0 b_0\}$. We thus have:

$$P = P_0 \bigcap (P_1 \times P_2 \times \dots \times P_K) \quad (3.3)$$

Hence, the original LP problem (Equation 3.1) can be transformed to a master problem (Equation 3.4) using Dantzig-Wolfe Decomposition [102]. Note that x_j^i represents the vertex or extreme ray associated with λ_{ij} (now $\forall x_j^i$ are constants whereas $\forall \lambda_{ij}$ are the variables in the projected polyhedron P'), and $|E_1|, \dots, |E_K|$ denote the number of vertices/extreme rays of the decomposed polyhedra P_1, \dots, P_K respectively.

$$\begin{aligned} & \max \quad \sum_j c_1^T x_1^j \lambda_{1j} + \dots + \sum_j c_K^T x_K^j \lambda_{Kj} \\ & \text{s.t.} \quad \begin{cases} \sum_j A_1 x_1^j \lambda_{1j} + \dots + \sum_j A_K x_K^j \lambda_{Kj} \bowtie b_0 \\ \sum_j \delta_{1j} \lambda_{1j} & = 1 \\ \vdots \\ \sum_j \delta_{Kj} \lambda_{Kj} & = 1 \\ \lambda_1 \in \mathbb{R}^{|E_1|}, \dots, \lambda_K \in \mathbb{R}^{|E_K|}, \delta_{ij} \in \{0, 1\} \end{cases} \end{aligned} \quad (3.4)$$

As proven in [122], 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*.

3.1.2 Non-linear Programming (NLP)

As the name implies, non-linear programming solves the optimization problem where the objective function or (some of) the constraints are *non-linear*. Thus, the general form of the problem can be stated simply as: $\{\min_x f(x) | \text{s.t. } g(x) \bowtie 0\}$ where \bowtie denotes either equality or inequality constraints, and $f(x)$ or $g(x)$ can be either linear or non-linear.

Since either the objective function, the constraints or both could be non-linear, NLP includes a wide variety of mathematical programming problems such as quadratic programming, separable programming, stochastic programming [120]. We particularly introduce separable programming since we will tackle the privacy issues for such problem in this dissertation which has applications in [58].

A function $f(x_1, \dots, x_n)$ is separable if it can be expressed as the sum of n single variable functions $f_1(x_1), \dots, f_n(x_n)$. We thus have $f(x_1, \dots, x_n) = f_1(x_1) + \dots + f_n(x_n)$. We solve the real-world problem in [58] with an optimization problem and convert it to the following *separable* NLP problem.

$$\begin{aligned} \max : \quad & \sum_{i=1}^n [c_i \log(x_i + 1)] \\ \text{s.t.} \quad & \begin{cases} \forall j \in [1, m], \sum_{i=1}^n (x_i \cdot \log t_{ij}) \leq b_j \\ \sum_{i=1}^n x_i = d \\ \forall 0 \leq x_i \leq d \end{cases} \end{aligned} \quad (3.5)$$

Note that, in Chapter 6, we will discuss the collaborative scenario of the above NLP problem and securely solve it amongst different parties. Before doing that, we introduce an approach to solve the non-cooperative one.

Any separable convex problem can be solved by linear approximation [120] with the idea of convexity combination. In this problem, all n non-linear functions $1 \leq x_i \leq n, \log(x_i+1)$ are strictly convex since the derivative $\frac{\partial \log(x_i+1)}{\partial x_i} = \frac{1}{x_i} > 0$. Hence, $1 \leq x_i \leq n, \log(x_i+1)$ can be approximated by a piecewise linear function [120]. Specifically, we can approximate $\forall i \in [1, n], \log(x_i + 1)$ over the interval $[0, d]$ ($\forall i \in [1, n], 0 \leq x_i \leq d$) as follows:

- define $a_s, s = 1, 2, \dots, K$ as the s th breakpoint on interval $[0, d]$ where $a_1 < a_2 < \dots < a_K$.
- $\log(x_i + 1) \approx \sum_{s=1}^K \log(a_s + 1)w_s^i$ where w_s^i is a nonnegative weight associated with the s th breakpoint and variable x_i . Note that $\sum_{s=1}^K w_s^i = 1$ (the non-linear function $\log(x_i + 1)$ is approximated as a convex combination of a set of logarithmic values $1 \leq s \leq K, \log(a_s + 1)$ based on a_s).
- since a_1, \dots, a_K and $\log(a_1 + 1), \dots, \log(a_K + 1)$ are constants in the approximated LP problem, we have $x_i = \sum_{s=1}^K a_s w_s^i$ and $\log(x_i + 1) = \sum_{s=1}^K \log(a_s + 1)w_s^i$ where $1 \leq s \leq K, w_s^i$ are new variables replacing x_i .
- similarly, we can approximate all linear functions (w.r.t. $1 \leq i \leq n, \log(x_i + 1)$) with the same set of breakpoints on $[0, d]$.

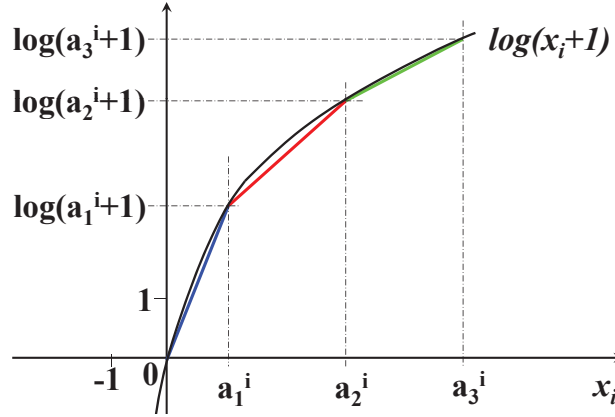


Figure 3.1. Piecewise Linear Approximation of Convex Function $\log(x_i + 1)$

Figure 3.1 shows an example of approximating convex function $\log(x_i + 1)$, three breakpoints excluding the endpoints in x_i 's value range $[0, d]$ are specified as a_1, a_2, a_3 . Therefore, $\log(x_i + 1)$ can be approximately expressed as $\log(a_1 + 1)w_1^i + \dots + \log(a_3 + 1)w_3^i$ (the convex combination of the values $\log(a_1 + 1), \dots, \log(a_3 + 1)$) where $w_1^i + w_2^i + w_3^i = 1$ and $x_i = a_1w_1^i + a_2w_2^i + a_3w_3^i$.

Then, we present the approximated LP problem for Equation 3.5 as below:

$$\begin{aligned} \max : \quad & \sum_{i=1}^n \left[c_i \sum_{s=1}^K w_s^i \cdot \log(a_s + 1) \right] \\ \text{s.t.} \quad & \begin{cases} \forall j \in [1, m], \sum_{i=1}^n (\sum_{s=1}^K a_s w_s^i \cdot \log t_{ij}) \leq b_j \\ \sum_{i=1}^n \sum_{s=1}^K a_s w_s^i = d \\ \forall i \in [1, n], \sum_{s=1}^K w_s^i = 1 \\ \forall i \in [1, n], s \in [1, K], 0 \leq w_s^i \leq 1 \end{cases} \end{aligned} \quad (3.6)$$

Finally, Equation 3.6 can be efficiently solved with Simplex or Revised Simplex method [120], and the optimal solution of the NLP problem (Equation 3.5) can be straightforwardly derived with expressions: $\forall i \in [1, n], x_i = \sum_{s=1}^K a_s w_s^i$.

3.1.3 NP-hard Problem 1: Graph Coloring

Graph coloring covers a set of “coloring” related optimization problem over the graph, such as vertex coloring, edge coloring and total coloring. Note that other coloring problems can be transformed into a vertex coloring version [103]. Thus, the collaborative graph coloring problem in this dissertation refers the vertex coloring formulated by different parties. Specifically, Figure 3.2 gives an illustrative example for graph coloring, where colors are assigned to every vertex of the graph however every pair of adjacent vertices cannot be assigned with the same color.

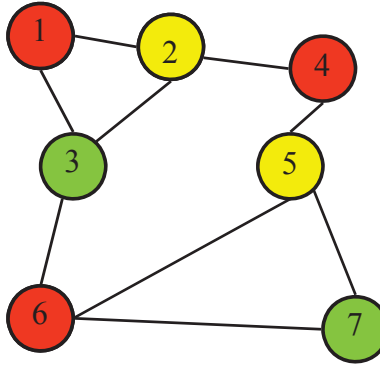


Figure 3.2. Graph Coloring Problem

There are two versions of optimization problems regarding graph (vertex) coloring – the decision problem and optimization problem (finding the chromatic number). On one hand, the decision problem of graph coloring is denoted as k -coloring – does graph G admit a proper vertex coloring with k colors while assigning different colors to adjacent vertices. Seeking such decision is NP-complete [102]. On the other hand, the optimization problem of graph coloring is to seek the minimum k for graph G . Essentially, this NP-hard problem can be converted into a set of k -coloring problems. Thus, we concentrate on the k -coloring

problem in general.

Graph coloring is applicable to model many practical problems, such as scheduling, resource relocation, pattern matching, and radio frequency assignment problem. The details of the application determines different structures of the graph. However, the inherent objective function is to minimize the total conflicts defined in the graph. We now briefly introduce an effective and efficient meta-heuristic developed for this hard problem with large-scale input.

Meta-heuristic: Tabu Search

Tabu search is a meta-heuristic algorithm designed to solve the optimization problems by helping the search process escape the local optimality. It has obtained optimal or near-optimal solutions for a wide variety of classic and practical optimization problems [43]. As mentioned in [43], in many optimization problems, tabu search has found superior solutions than the best solutions previously obtained by the alternative methods. Also, it is easier to implement tabu search with additional considerations/constraints. For instance, the integer programming formulations in Glover and McMillan's employee scheduling problem have 1-4 million variables. However, with tabu search, it needs 22-24 minutes to acquire the solution within 98% of an upper bound on optimality on an IBM PC; In [44], Glover, McMillan, and Novick implemented tabu search to subset clustering problems, a 0-1 mixed integer programs with over 25,000 variables and 50,000 constraints. The solution can be obtained less than one minute with a V77 minicomputer.

Roughly speaking, tabu search uses a memory structure to enhance the performance of a local search method. Once a potential solution has been determined,

it is marked as “taboo”. So the algorithm does not visit that possibility repeatedly. Tabu search has been successfully used for graph coloring. Specifically, given a graph $G = (V, E)$ and k colors, a popular tabu search algorithm for graph coloring – Tabucol [49] is described as follows (note that $\mu(x)$ denotes the total number of conflicts of solution x in G):

1. initialize a solution $x = (\forall v_i \in G, x_i)$ by randomly selecting colors.
2. iteratively generate neighbors x' of x until $\mu(x') < \mu(x)$ by changing the color of an endpoint of an arbitrary conflicted edge $v_i : x_i \rightarrow x'_i$ (if no x' is found with $\mu(x') < \mu(x)$ in a specified number of iterations, find a best x' in current set of neighborhoods where (v_i, x'_i) is not in the tabu list, and let x' be the next solution).
3. if (v_i, x'_i) is not in the tabu list, make current x' (best so far) as the next solution $x = x'$ and add the pair (v_i, x_i) into the tabu list; If (v_i, x'_i) is in the tabu list and $\mu(x')$ is still smaller than $\mu(x)$, make x' as the next solution anyway; Otherwise, discard x' .
4. repeat step 2,3 until $\mu(x') = 0$ or the maximum number of iterations is reached.

3.1.4 NP-hard Problem 2: Traveling Salesman Problem

Traveling salesman problem (TSP) is another computationally difficult (NP-hard) problem in combinatorial optimization, and it has been well studied in literature [103]. Given a set of cities and the distances between every pair of cities, the

optimization solver finds the shortest route to visit all the cities once and back to the original city. Clearly, the overall distance is minimized in such problem.

This optimization model also has many important applications [103], such as logistics, planning, and microchip production, where the distance and city can represent different objects in different scenarios. More generally, the problem can be formulated as: given the cost between every pair of cities (replacing the distance), minimizing the total cost of the traveling route to visit all the cities. Similarly, to solve this NP-hard problem with large-scale input, we also need a meta-heuristic solver such as simulated annealing.

Meta-heuristic: Simulated Annealing

In optimization, the basic idea of Simulated Annealing algorithm originated from thermodynamics, similar to metals cool and anneal [71]. It works based on probabilities that one state (solution) moves to another state (its neighboring solution) in a large search space, especially when the search space is discrete (e.g., traveling salesman problem, graph coloring problem). Bertsimas and Tsitsiklis have summarized some practical applications of simulated annealing in [16] such as image processing. Johnson et al. [64–66] studied the performance of simulated annealing applied to four NP-hard problems: traveling salesman problem [64], graph coloring problem [66], graph partitioning problem [65] and number partitioning problem [66]. In some cases, it performs better than the best known heuristics of those problems, and in other cases, some specialized heuristics performs better. Simulated annealing is an easy to implement and very effective meta-heuristic to produce good solutions for many optimization problems.

In TSP, given the traveling route as x , the objective function of simulated annealing is considered to minimize $f(x)$ and the algorithm iteratively move from solutions to their neighboring solutions. Differently, moving to the next solution or staying in the current solution is determined by a probability derived from two solutions and a sensitivity parameter “Temperature” T . The algorithm is briefly summarized as below:

1. initialize a solution x by randomly selecting a traveling route.
2. randomly pick a neighboring solution x' of x by permutating a segment of the route. For example, if $x = “1 \rightarrow 3 \rightarrow 2 \rightarrow 4”$, then $x = “1 \rightarrow 3 \rightarrow 4 \rightarrow 2”$ is its neighboring solution.
3. decide whether moving to the new solution or not with the probability which is computed from $f(x)$, $f(x')$ and the sensitivity parameter T . If yes, update the current solution.
4. repeat step 2,3 until the maximum number of iterations is reached or a satisfiable solution is obtained.

3.2 Secure Multiparty Computation

Consider that some parties do not trust each other but they wish to jointly compute some functions with their local information, then Secure Multiparty Computation (SMC) can facilitate them to get the result of the functions while keeping their local information private. Since any collaborative optimization problem consists a set of complex functions jointly computed across distributed parties, privacy-

preserving collaborative optimization is indeed a special case of secure multiparty computation.

The first question is to quantify the privacy leakage in secure multiparty computation. Some theoretic studies [13, 45, 46, 132] have been proposed in literature to show that the SMC protocol reveals nothing if all the messages received by every party can be simulated in polynomial time by knowing only the input and output of SMC protocol/functions. Yao first raised the two-party comparison problem (Yao's Millionaire Protocol) and developed a provably secure solution in 1986 [132]. Then Goldreich et al. extended the protocol to multiparty (more than two) computations in 1987 [46], and Ben-Or et al. proposed the SMC protocol under the malicious model in 1988 [13]. To date, SMC has been well developed with the composition theorem that can be used to quantify the privacy leakage of the SMC protocol [45]. We now review some of the different adversarial models in SMC for this dissertation.

3.2.1 Semi-honest Model

Semi-honest model (viz. honest-but-curious model) assumes that all the participants in the protocol are honest to follow the protocol yet curious to derive private information from each other. A formal definition of SMC in semi-honest model is given as below (the formal proof can be found in [45]).

Definition 3.7 (Privacy w.r.t. Semi-honest Behavior [45]) *Given a protocol Π executed among k parties P_1, \dots, P_k , let $D_i (1 \leq i \leq k)$ and Π_i denote P_i 's input and P_i 's result computed from Π respectively, thus Π is secure if every party*

P_i 's execution view $view_i^\Pi$ can be simulated from its input D_i and output Π_i with a polynomial machine.

In other words, if every party's view can be simulated by looking at only its input and output, we can conclude that the protocol reveals nothing. Specifically in the SMC protocol built for collaborative optimization, numerous messages are exchanged among those parties. To prove the security of the protocol, composition theorem plays an essential role, which helps us to create and compose building blocks for the SMC protocol.

Theorem 3.3 (Composition Theorem [45]) *Suppose that g is privately reducible to f and that there exists a protocol for privately computing f . Then there exists a protocol for privately computing g .*

Note that the details of the theorem and the proof are given in [45].

This SMC definition [45, 132] states that a computation is secure if the view of each party during the execution of the protocol can be effectively simulated knowing only the input and the output of that party. This is more about protocol security but not quite the same as saying that all private information is protected against leakage. Indeed, although those simulated messages have been proven to be secure, semi-honest players may further infer private information from the messages and their input/output. Thus, for SMC based approach in this dissertation, we ensure that both SMC based security and ignorable inference attack (which complement each other) are satisfied.

3.2.2 Malicious Model – Cryptographic and Economic Notion

Semi-honest adversarial model assumes that all the participants play honestly in the protocol. In reality, every party may play whatever they like to deviate the protocol or gain additional payoff with their self-interested behavior. At this time, malicious adversarial model is defined as an advanced model beyond assuming semi-honest participants.

Given a general notion, we can consider malicious model in SMC as “any party may behave arbitrary action in the protocol” – they hurt other participants by cheating, colluding, or terminating collaboration without excuse. Specifically, malicious model has been well studies in cryptographic community [45]. To some extent, it is quite difficult to define a model in which all the malicious actions can be resolved, especially when we have to make the protocol efficient and accessible. However, it is very practical to build a relaxed malicious model for a given collaborative optimization problem. For example, assuming malicious participants, parties cannot learn additional information from each other and cannot deviate the protocol, beyond that they can gain with a trust-third party. That is to say, the protocol can defeat all the attacks which using the trusted-third party to solve the collaborative optimization problem does not suffer.

Some malicious behavior, e.g., parties modify their inputs before executing the protocol, cannot be detected by the trusted-third party. Besides cryptography, some other notions – such as incentive compatibility assist us to prevent such kind of cheating.

Definition 3.8 (Incentive Compatibility) *A protocol is incentive compatible if*

“honest play” is the strictly dominant strategy¹ for every participant.

A rational malicious party may evaluate whether it is worth cheating by comparing the payoff they can gain via cheating to the economic loss as well as the risk of being caught. If the protocol can guarantee that the payoff expectation of honest play outperforms cheating, rational players will choose honest play under this economic notion. Note that irrational model is not considered in this dissertation.

3.3 Research Challenges of Privacy-preserving Collaborative Optimization

We now look at some of the fundamental challenges underlying privacy-preserving collaborative computation that must be met to make it accessible and practical.

1) Iteration. Iterative algorithms (viz., iteration) is a significant obstacle to efficient secure computation. Many optimization techniques require iteration in some form or fashion. The effect of iteration on security is enormous. It is possible to write a secure algorithm such that the results of each iteration are also kept secret, but this is certain to lead to severe problems with efficiency. Instead, each iteration could probably be independently made secure much more efficiently. The question then is what to do with the intermediate results? Can they be protected in some form? What do they reveal? What is the threshold for security – how many iterations are to be allowed before a security violation occurs. Indeed, is it possible to define the parameters for a secure violation? The issue of composability of

¹In 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

secure protocols as well as parallelizability has been well studied [22, 23, 73, 77–79]. But the issue of iteration needs to be similarly addressed. That is to say, for formally proving the security of our privacy-preserving optimization techniques, the composition of the secure protocols have to be explored [45].

2) Security Definitions for Difficult Problems. One of the biggest successes of SMC is that it has provided a solid quantifiable mathematical way of proving the security of an algorithm [45]. One can have confidence in the security of an algorithm after it has been proven secure in the SMC framework. However, the computational security definitions in SMC depend on the fact that problems can be solved in polynomial time (i.e., the solutions are “efficient”). What happens when we are considering problems that are exponential in the worst case? A new set of security definitions must be formulated to define and quantify security in this case. Optimization is a good candidate for this since many solutions for non-linear programming and integer programming are exponential in the worst case, and many classic combinatorial optimization problems (e.g., traveling salesman problem, graph coloring problem) are NP-hard in any case. We will explore this issue as we go along.

3) Data Partition between Two or More Parties. There are many ways in which data could be distributed in collaborative optimization problems. Each of the ingredients of the optimization problem could be distributed. For example, the objective function might be known to only one party, or parts of it known to some subsets of the parties. The constraints might also be distributed in some fashion. Different parties might own different constraints or even different parts of the same constraint. Thus, the different ways in which data is distributed give

rise to the following categorization:

- **Horizontal Partitioning:** Here, each constraint would be fully owned by one party. Thus, different parties own different constraints. An example of this would be a collaborative scheduling problem in which several schedulers need to schedule tasks on machines. Each task can be executed by several machines (though not all), and it can be split between several machines, but the fraction of all tasks executed by a machine must under no circumstances exceed its capacity. Each scheduler only knows the tasks that may be executed on its pertinent machines, and based on this information it must decide what fractions of its task to send to which machines. The sum of all fractions is to be maximized. Here, the objective function could be known to a single party or to all of the parties, or even be shared by the parties.
- **Vertical Partitioning:** In this case, each constraint is shared between some subset of the parties. An example of this would be the organization theory problem. A large enterprise has a very extensive set of tasks – say, products manufactured. A fundamental question in Organization Theory is, how are these tasks to be partitioned among managers? Although the profitability and resource requirements of these products may change dynamically with market conditions, the constraint structure, the sparsity pattern of the constraint matrix of the associated linear program, may be fixed. That is, it is known in advance, which products compete for which resources. What are the organizational principles that should guide this assignment of tasks to managers, so that the latter can make more informed decisions. Again, the

objective function might be known to all of the parties, or just to a single party, or be shared by the parties.

- **Arbitrary Partitioning:** Apart from the prior two partitioning methods, the data may also be arbitrary partitioned in some way (some combination of the above). This is more general and subsumes both of the earlier cases. Completely arbitrary partitioning of data is unlikely in practice, though certain specific configurations might easily be found. In any case, solutions for this case will always work for both of the prior cases as well.

For different data partition, the security definition as well as the security protocol might be given in completely different manner. In this dissertation, we also investigate the same optimization problem with different data partition scenario (horizontal partitioning, vertical partitioning, arbitrary partitioning, and even more complex partitioning). Moreover, some prior work presented secure approach for collaborative optimization problems between two parties. This assumption is not widely applicable to many practical problems, we target at finding the secure methods for the optimization problem that is partitioned between not only two parties but also among multiple (more than two) parties.

4) Game Theoretic Approaches to Malicious Adversaries. Significant work is possible at the intersection of game theory and cryptography. Mechanism design (in game theory) and cryptography (in computer science) are both dual and opposites of each other. Cryptography has been concerned with allowing parties to hide information while achieving some particular objective (for example, jointly computing a function). Mechanism design has been concerned with forc-

ing parties to reveal information while achieving a particular objective. This has many ramifications. For one thing, it explains why cryptography has managed for so long to avoid explicit modeling of the parties' utility functions. There is an opportunity to fuse the two perspectives and speak about informational mechanism design (IMD). IMD is characterized by the fact that each party's utility is a function of the informational structure – who knows what. We need to figure out some way to explicitly model utilities. There has been some work at the intersection of cryptography and game theory. Fischer and Wright [37] provide an application of game theoretic techniques to the analysis of a class of multiparty cryptographic protocols for secret bit exchange. Dodis et al. [30] provide a cryptographic protocol to the correlated element selection problem. Teague [121] extends this protocol to work also for non-uniform distribution. Other work that addresses the same problem without help from a third-party mediator includes [5, 8, 14, 39]. Matsuura [92] provides a survey of the emerging interdisciplinary area between information security and economics. Clifton et al. [26] explore secure solutions to the transportation load swapping problem, and show how such a protocol can be incentive compatible, thus protecting from malicious adversaries. This needs to be explored further and much more research is necessary.

CHAPTER 4

SEMI-HONEST COLLABORATIVE LINEAR PROGRAMMING (LP)

As an essential subclass of optimization, linear programs are widely applicable to solving numerous profit-maximizing or cost-minimizing problems in various fields such as transportation, commodities, airlines and communication.

In this chapter, we address the privacy concern of collaborative linear programming in semi-honest model. Specifically, we first identify a potential inference attack to the prior work on securing horizontally partitioned linear programs in semi-honest model [75, 90] and present a revised approach to improve the security guarantee of such model. Then, we present our privacy-preserving model to collaborative linear programs with arbitrarily partitioned constraints, in semi-honest model as well.

4.1 An Inference-proof to Horizontally Partitioned LP [75, 90]

As a fundamental branch of optimization problems, numerous privacy-preserving collaborative linear programming techniques have been proposed in literature with respect to different data partition scenarios for the collaborative linear programs. Specifically, [74] and [125] proposed a secure simplex method and revised simplex method respectively for linear programs co-held by two parties. A matrix transformation approach has been proposed in [32, 124] (some flaws with this ap-

proach were pointed out by Bednarz et al. [11]). In [31, 55, 57, 75, 89, 90], the private information of two or more parties in the linear program are protected via mathematical transformation.

In particular, Mangasarian [90] proposed a constraints transformation based approach to securely solve the horizontally partitioned linear programs among multiple parties, where every party holds its own set of private equality constraints. More recently, Li et al. [75] extended this approach to horizontally partitioned linear programs with inequality constraints. However, the approach is not sufficiently secure – occasionally, the privately owned constraints can still be inferred by adversaries (see Section 4.1.1). In this section, we present an *inference-proof approach* to enhance the security for privacy-preserving horizontally partitioned linear program (PPHPLP) [75, 90] with *arbitrary number of equality and inequality constraints*. Our approach extends the constraint matrix by including artificial constraints and extra slack variables (with random coefficients) to provide significantly more security than the existing work.

4.1.1 Inference Attack against the Existing Work [75, 90]

Privacy-preserving Horizontally Partitioned Linear Program (PPHPLP)

Consider a linear program with equality constraints: $\{\min : c^T x, \text{ s.t. } Ax = b, x \geq 0\}$. In horizontally partitioned linear program [75, 90], the matrix $A \in R^{m \times n}$ and the right-hand side vector $b \in R^m$ in the constraints are partitioned into p horizontal blocks held by p parties respectively: $A_{I_1}, A_{I_2}, \dots, A_{I_p}$ and $b_{I_1}, b_{I_2}, \dots, b_{I_p}$ where A_{I_i}, b_{I_i} and the index set I_i belong to party i (note that $i = 1 \dots p, \bigcup_{i=1}^p I_i = \{1, 2, \dots, m\}$). With privacy concern, every party $i = 1 \dots p$

does not want to share its data block A_{I_i}, b_{I_i} with other parties while solving the problem. Therefore, Mangasarian [90] proposed a transformation approach to address this issue – given a random matrix $B \in R^{k \times m}$ with $k \geq m$ and rank m [36], a new linear program can be formulated as $\{\min : c^T x, \text{ s.t. } BAx = Bb, x \geq 0\}$.

In [90], the original and the transformed linear programs have been proven to have equivalent optimal solution and feasibility if the rank of $B \in R^{k \times m}$ (number of linear independent column vectors in B) is equal to m where $k \geq m$ (which naturally holds for random matrices [36]). Therefore, every party $i = 1 \dots p$ generates a privately held random matrix $B_{\cdot I_i} \in R^{k \times m_i}$ such that:

$$\begin{aligned} BA &= [B_{\cdot I_1}, B_{\cdot I_2}, \dots, B_{\cdot I_p}] \begin{bmatrix} A_{I_1} \\ A_{I_2} \\ \vdots \\ A_{I_p} \end{bmatrix} \\ &= B_{\cdot I_1} A_{I_1} + B_{\cdot I_2} A_{I_2} + \dots + B_{\cdot I_p} A_{I_p} \in R^{k \times n} \end{aligned} \quad (4.1)$$

and

$$\begin{aligned} Bb &= [B_{\cdot I_1}, B_{\cdot I_2}, \dots, B_{\cdot I_p}] \begin{bmatrix} b_{I_1} \\ b_{I_2} \\ \vdots \\ b_{I_p} \end{bmatrix} \\ &= B_{\cdot I_1} b_{I_1} + B_{\cdot I_2} b_{I_2} + \dots + B_{\cdot I_p} b_{I_p} \in R^k \end{aligned} \quad (4.2)$$

In Mangasarian's PPHPLP algorithm [90], every party $i = 1 \dots p$ only discloses publicly the transformed matrix product $B_{\cdot I_i} A_{I_i}$ and the transformed vector $B_{\cdot I_i} b_{I_i}$ for solving the problem. Thus $B_{\cdot I_i}$, A_{I_i} and b_{I_i} can be kept private.

Furthermore, Li et al. [75] considers the horizontally partitioned linear program with inequality constraints as below:

$$\begin{aligned} \min : & \quad c^T x \\ \text{s.t.} \quad & Ax \leq b, x \geq 0 \end{aligned} \tag{4.3}$$

Mangasarian [90] has pointed out that converting the inequality constraints in linear program (4.3) to equality constraints (by adding slack variables) and then transforming the problem per Mangasarian's approach [90] would reveal $B_{.I_i}$. Therefore, Li et al. [75] resolved this issue by letting every party $i = 1 \dots p$ generate a diagonal random matrix $D_{I_i} \in R^{m_i \times m_i}$ besides $B_{.I_i} \in R^{k \times m_i}$ for its slack variables (every inequality requires one slack variable). Consequently, a similar transformation is conducted $Ax + Dx_s = Bb \iff BAx + BDx_s = Bb$ to formulate:

$$\begin{aligned} \min : & \quad c^T x \\ \text{s.t.} \quad & BAx + BDx_s = Bb, x \geq 0 \end{aligned} \tag{4.4}$$

where x_s denotes all the slack variables and $D = \text{diag}(D_{I_1}, D_{I_2}, \dots, D_{I_p}) \in R^{m \times m}$. As a result, if (x^*, x_s^*) is the optimal solution of the transformed linear program (4.4), x^* is also the optimal solution for the linear program (4.3).

Inference Attack

In Li et al. [75]'s extended transformation for linear program (4.3), every party $i = 1 \dots p$ generates a diagonal random matrix $D_{I_i} \in R^{m_i \times m_i}$ to prevent inferring its private random matrix $B_{.I_i}$ from the slack variables. However, such transformation approach is still vulnerable to potential inference attack.

We now look at the illustrative example in [75]: party 1 holds two constraints $x_1 + x_2 + 2x_3 \leq 2$ and $x_1 + 4x_2 - x_3 \leq 1$ while party 2 holds one constraint $x_1 + 2x_2 - 4x_3 \leq 1$. In the transformed linear program, party 1 and 2 generate their privately held random matrices respectively:

$$B_{.I_1} = \begin{bmatrix} 0.4562 & 0.1367 \\ 0.5469 & 0.8739 \\ 0.5633 & 0.7693 \end{bmatrix} \in R^{3 \times 2}, D_{I_1} = \begin{bmatrix} 12.4965 & 0 \\ 0 & 25.6433 \end{bmatrix} \in R^{2 \times 2}$$

$$B_{.I_2} = \begin{bmatrix} 0.3574 \\ 0.7763 \\ 0.6682 \end{bmatrix} \in R^{3 \times 1}, D_{I_2} = [15.7628] \in R^{1 \times 1}$$

Since the inequality constraints require 3 slack variables (this is revealed publicly while solving the linear program), party 1 can learn that party 2 has one inequality constraint ($m_2 = 1$) due to its known $m_1 = 2$. Then, the size of matrices $B_{.I_2}$ and A_{I_2} can be learnt by party 1 as $R^{3 \times 1}$ and $R^{1 \times 3}$. Thus, from party 1's view, the elements in $B_{.I_2} \in R^{3 \times 1}$, $A_{I_2} \in R^{1 \times 3}$, and $b_{I_2} \in R^{1 \times 1}$ can be regarded as real variables $\beta_1, \beta_2, \beta_3, \alpha_1, \alpha_2, \alpha_3$ and δ :

$$A_{I_2} = [\alpha_1 \quad \alpha_2 \quad \alpha_3], B_{.I_2} = [\beta_1, \beta_2, \beta_3]^T, b_{I_2} = [\delta]$$

Note that Mangasarian [90] and Li et al. [75] make public every party i 's transformed matrix $B_{.I_i} A_{I_i}$ and vector $B_{.I_i} b_{I_i}$. Therefore, in this two-party case, party 1 can express the real numbers in $B_{.I_2} A_{I_2} \in R^{3 \times 3}$ and $B_{.I_2} b_{I_2} \in R^{3 \times 1}$ as:

$$B_{.I_2} A_{I_2} = \begin{bmatrix} \beta_1 \alpha_1 & \beta_1 \alpha_2 & \beta_1 \alpha_3 \\ \beta_2 \alpha_1 & \beta_2 \alpha_2 & \beta_2 \alpha_3 \\ \beta_3 \alpha_1 & \beta_3 \alpha_2 & \beta_3 \alpha_3 \end{bmatrix} = \begin{bmatrix} 0.3574 & 0.7148 & -1.4296 \\ 0.7763 & 1.5526 & -3.1052 \\ 0.6682 & 1.3364 & -2.6728 \end{bmatrix}$$

$$B_{\cdot I_2} b_{I_2} = \begin{bmatrix} \beta_1 \delta \\ \beta_2 \delta \\ \beta_3 \delta \end{bmatrix} = \begin{bmatrix} 0.3574 \\ 0.7763 \\ 0.6682 \end{bmatrix}$$

where all the elements in the above matrix and vector have been exactly revealed to party 1. Since $\beta_1 \alpha_1 = 0.3574$, $\beta_1 \alpha_2 = 0.7148$, $\beta_1 \alpha_3 = -1.4296$ and $\beta_1 \delta = 0.3574$ are known to party 1 as constants in $B_{\cdot I_1} A_{I_1}$ and $B_{\cdot I_1} b_{I_1}$ (similarly, other rows in $B_{\cdot I_1} A_{I_1}$ and $B_{\cdot I_1} b_{I_1}$ are also revealed), party 1 can simply compute $\alpha_1 : \alpha_2 : \alpha_3 : \delta = 1 : 2 : (-4) : 1$. Although the exact value of β_1 , α_1 , α_2 , α_3 and δ cannot be learnt by party 1, the ratio $\alpha_1 : \alpha_2 : \alpha_3 : \delta = 1 : 2 : (-4) : 1$ is sufficient to reconstruct the constraint $x_1 + 2x_2 - 4x_3 \leq 1$ for adversaries.

Hence, revealing the number of constraints m in the linear program (and also every party's transformed share per [75,90]) may result in serious privacy leakage: an adversarial party might be able to infer other parties' private constraints by formulating equations with real variables. Indeed, two possible ways of learning m can be identified in the prior work [75,90] as follows:

1. *The number of slack variables* might be the number of constraints m (see above).
2. Since the transformed matrix BA and vector Bb are revealed to public in [75,90], every party can compute the rank of matrix BA . Due to $\text{rank}(B \in R^{k \times m}) = m$ [36] and $\text{rank}(A \in R^{m \times n}) \leq m$, we have $\text{rank}(B) \geq \text{rank}(A)$. Hence, $\text{rank}(BA) = \text{rank}(A)$, and $\text{rank}(A)$ can be inferred by all parties as $\text{rank}(BA)$. However, *the number of constraints is likely equal to $\text{rank}(A)$* , especially when the linear program contains small number of

constraints (this applies to both equality [90] and inequality constraints [75]: in Li et al.'s illustrative example [75], $\text{rank}(BA) = \text{rank}(A) = 3$ can be computed, thus both parties can infer that $m = 3, m_1 = 2, m_2 = 1$). Therefore, the risk of inferring the number of constraints by $\text{rank}(BA)$ is still high in [75, 90].

4.1.2 Algorithm

We now present an inference-proof approach for PPHPLP.

Transformation

We consider a general model of linear programs containing an arbitrary number of equality and inequality constraints. The slack form of the linear program is:

$$\begin{aligned} \min : & \quad c^T x \\ \text{s.t.} \quad & Ax + Mx_s = b, x \geq 0, x_s \geq 0 \end{aligned} \tag{4.5}$$

where $A \in R^{m \times n}$, x_s includes ℓ slack variables (ℓ is the number of inequality constraints in linear program (4.5)) and $M = \begin{bmatrix} 0 \\ I \end{bmatrix} \in R^{m \times \ell}$ ($\ell \leq m$) is not always a diagonal matrix (different from D defined in Li et al. [75]) in this general linear program form: for each *equality constraint*, all the elements in the corresponding row of M are 0; the remaining rows (associated with *inequality constraints*) form a diagonal matrix. In horizontally partitioned linear program, every party $i = 1 \dots p$ holds constraints $A_{I_i}.x + M_{I_i}x_s = b_{I_i}$ ($A_{I_i} \in R^{m_i \times n}$, $b_{I_i} \in R^{m_i}$) where $M_{I_i} \in R^{m_i \times \ell}$ is the rows of M associated with party i 's all equality and inequality constraints, thus:

$$M = \begin{bmatrix} M_{I_1} \\ \vdots \\ M_{I_p} \end{bmatrix} \in R^{m \times \ell}, \ell \leq m$$

First, to prevent learning the number of constraints m from the rank of matrices BA and A , we can let every party $i = 1 \dots p$ locally generate some *artificial constraints* based on its original constraints, and add them to their constraints (we denote all parties' total number of original and artificial constraints as m'). Note that the feasible region of party i ($i = 1 \dots p$)'s all the artificial constraints should be a superset of the feasible region of party i 's original local constraints – *thus adding artificial constraints does not change the feasible region of the linear program*. For instance, if party i owns inequality constraint $x_1 - 3x_2 \leq 9$ where $x \geq 0$, some artificial constraints can be generated as: $x_1 - 4x_2 \leq 9$, $0.9x_1 - 3x_2 \leq 9$, $0.5x_1 - 1.7x_2 \leq 4.5$, etc. And if party i owns equality constraint $x_1 + 3x_2 = 9$ where $x \geq 0$, we can generate: $x_1 + 2x_2 \leq 9$, $0.5x_1 + 1.2x_2 \leq 4.5$, etc. Note that creating artificial equality constraints are not recommended since adding them does not change $\text{rank}(A)$.

As a result, the constraint matrix and right-hand side vector are expanded to $A' \in R^{m' \times n}$ and $b' \in R^{m'}$, and all p parties can only infer $\text{rank}(A')$ rather than $\text{rank}(A)$ by computing $\text{rank}(BA')$ (*it is not difficult for every party to generate artificial constraints with real number coefficients that increase the rank of the global constraint matrix A*). Therefore, the probability of inferring m can be significantly reduced by increasing the number of artificial constraints.

Second, we would also like to reduce the probability of inferring m from the number of slack variables ℓ . We denote the number of inequality constraints in

the artificial constraints-added linear program as ℓ' (indeed, $\ell < \ell' \leq m'$, and the coefficient matrix of slack variables $M \in R^{m \times \ell}$ is expanded to $M' \in R^{m' \times \ell'}$ in the slack form of the artificial constraints-added linear program), thus m cannot be inferred by ℓ anymore. In addition, we can better reduce the inference probability by utilizing *more than one slack variable* to convert each of the inequality constraints to equality constraint in the slack form. Specifically, for the j th inequality constraint ($j = 1 \dots \ell'$) in the artificial constraints-added linear program, we can let the constraint-owning party determine t_j slack variables for converting it into an equality constraint. Thus, the joint linear program can have $t = \sum_{j=1}^{\ell'} t_j \gg \ell'$ slack variables.

We denote every party i 's ($i = 1 \dots p$) total number of slack variables for its all inequality constraints as T_i . In general, T_i can be sufficiently large to better reduce the inference probability (e.g., letting $T_i \gg m'_i$), thus we have $t = \sum_{i=1}^p T_i > m' \geq \ell' > \ell$ and $m' > m \geq \ell$ ($m = \ell$ [75] is the most vulnerable case). Hence, the total number of slack variables t can be completely different from m , and the linear program is:

$$\begin{aligned} \min : \quad & c^T x \\ \text{s.t.} \quad & A'x + D'x'_s = b', x \geq 0, x'_s \geq 0 \end{aligned} \tag{4.6}$$

where $A' \in R^{m' \times n}$, slack variables $x'_s = \{s_1, s_2, \dots, s_t\}$ and $D' \in R^{m' \times t}$ is a coefficient matrix for all slack variables in all constraints: for each *equality constraint*, all the elements in the corresponding row of D' are 0; for the j th *inequality constraint* ($j = 1 \dots \ell'$), in the corresponding row of D' , the coefficients of t_j slack variables are *randomly generated positive real numbers* while the remaining

elements in such row are 0. In horizontally partitioned linear program, every party $i = 1 \dots p$ holds constraints $A'_{I_i}.x + D'_{I_i}.x'_s = b'_{I_i}$ ($A'_{I_i} \in R^{m'_i \times n}, b'_{I_i} \in R^{m'_i}$) where $D'_{I_i} \in R^{m'_i \times t}$ is the rows of D' associated with party i 's all inequality and equality constraints, thus:

$$D' = \begin{bmatrix} D'_{I_1} \\ \vdots \\ D'_{I_p} \end{bmatrix} \in R^{m' \times t}, x'_s = \begin{bmatrix} s_1 \\ \vdots \\ s_t \end{bmatrix}, t > m' > m \geq \ell \text{ and } m' \geq \ell' > \ell \quad (4.7)$$

For example, party 1 owns two constraints $3x_1 + 7x_2 + 2x_3 = 20$ and $x_1 + x_2 \leq 9$ while party 2 owns one constraint $x_1 + 4x_2 + x_3 \leq 17$. Party 1 creates an artificial constraint $2.9x_1 + 6.3x_2 + 2x_3 \leq 20$ while party 2 creates an artificial constraint $0.5x_1 + 2x_2 + 0.3x_3 \leq 8.5$. Two slack variables are determined for converting each of both parties' inequality constraints to equality. Thus, D' can be generated as ($m = 3, \ell = 2, m' = 5, \ell' = 4, t = 8$, only t and m' are possibly disclosed to all parties):

$$D' = \begin{bmatrix} D'_{I_1} \\ D'_{I_2} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2.5 & 3.3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.9 & 1.4 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1.9 & 3.2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 8.5 & 5.7 \end{bmatrix} \in R^{5 \times 8}$$

where party 1 and 2 own the first three rows and last two rows respectively.

Proposition 4.1 *If $(x^*, (x'_s)^*)$ is an optimal solution to linear program (4.6), then x^* is optimal for linear program (4.5).*

Proof. Suppose that x^* is not optimal to linear program (4.5) and $\exists \hat{x}$ such that $c^T \hat{x} < c^T x^*$ in linear program (4.5). Clearly, $x = \hat{x}$ satisfies $Ax + Mx_s = b$. Since

A', M', b' are expanded from A, M, b by adding artificial constraints (without distorting the feasible region), $x = \hat{x}$ also satisfies $A'x + M'x_s = b'$ with the slack variables $x_s = \hat{x}_s$ (assuming that $\hat{x}_s = \{h_1 \dots h_{\ell'}\}$ are the non-negative values to make the slack form $A'\hat{x} + M'\hat{x}_s = b'$ hold based on the satisfied standard form $A'\hat{x} \leq b'$).

Hence, if $\exists M'\hat{x}_s = D'x'_s$, solution $x = \hat{x}$ (and t slack variables x'_s) is also feasible for $A'x + D'x'_s = b'$. Indeed, $M'\hat{x}_s$ and $D'x'_s$ are two length- m' vectors that convert all ℓ' inequalities in $A'x \leq b'$ to equalities by two different sets of slack variables x_s and x'_s (note that $M'\hat{x}_s$ and $D'x'_s$ correspond to the same index of the constraints $A'x \leq b'$). Taking D' and x'_s as below, we can have that $M'\hat{x}_s = D'x'_s$.

First, for all *equalities* in $A'x \leq b'$, the corresponding numbers in $M'\hat{x}_s$ and $D'x'_s$ are 0 (which are equal for any D' and x'_s). Second, for the j th *inequality* ($j = 1 \dots \ell'$) in $A'x \leq b'$, since the corresponding number in $M'\hat{x}_s$ is a non-negative number h_j , taking the j th inequality's t_j slack variables (in x'_s) as any t_j non-negative values such that the linear combination of them (with random positive coefficients in D') equals h_j , can make the corresponding number in $M'\hat{x}_s$ and $D'x'_s$ equal. Similarly, given any D' per (4.7), we can always find x'_s (including all $t = \sum_{j=1}^{\ell'} t_j$ slack variables) to ensure $M'\hat{x}_s = D'x'_s$ for all constraints (since all the ℓ' inequalities do not share any slack variable). Hence, $x = \hat{x}$ is always feasible for $A'x + D'x'_s = b'$.

Actually we can find t non-negative values for x'_s to ensure $M'\hat{x}_s = D'x'_s$: first, the numbers in $M'\hat{x}_s$ and $D'x'_s$ w.r.t. $(m' - \ell')$ equality constraints are both 0 and equal; second, the number in $M'\hat{x}_s$ and $D'x'_s$ w.r.t. the j th ($j = 1 \dots \ell'$)

inequality constraint can be equal by making the sum of t_j slack variables in x'_s with positive random coefficient in D' (which is the number in $D'x'_s$ w.r.t. the j th inequality constraint) equal to $h(j)$, which is the number in vector $M'x_s$ w.r.t. the j th inequality constraint (note that those non-negative values for t_j slack variables in x'_s always exist due to $h(j) \geq 0$).

Thus, x^* is not optimal to linear program (4.6) since $c^T \hat{x} < c^T x^*$ and $x = \hat{x}$ satisfy all the constraints in linear program (4.6). This contradiction completes the proof.

Similarly, we can transform linear program (4.6) to the inference-proof linear program by pre-multiplying a $k \times m'$ random matrix B to both sides of the constraints:

$$\begin{aligned} \min : & \quad c^T x \\ \text{s.t.} \quad & BA'x + BD'x'_s = Bb', x \geq 0, x'_s \geq 0 \end{aligned} \tag{4.8}$$

where the random matrix $B \in R^{k \times m'}$, $k \geq m'$ and $\text{rank}(B) = m'$. Note that the constraint matrix A and D of [75, 90] are replaced by A' and D' respectively (where the size and rank of the matrices are increased without changing the feasible region).

Proposition 4.2 [90] *Let $k \geq m'$ for the random matrix $B \in R^{k \times m'}$. The secure linear program (4.8) is solvable if and only if the linear program (4.6) is solvable in which case the solution sets of the two linear programs are identical.*

Therefore, linear programs (4.5) and (4.8) have the identical optimal solution.

Inference-proof Algorithm

Algorithm 1: Secure Sum Algorithm

Input : Party i holds R_i (matrix, vector or real number) where $i = 1 \dots p$,
and $\forall i, R_i$ has the same size

Output: $R = \sum_{i=1}^p R_i$

```

1  $R \leftarrow 0$ ;
2 for each party  $i = 1 \dots p$  do
3   generates a privately held random (matrix, vector or real number)  $R'_i$ 
   with the same size as  $R_i$ ;
4    $R \leftarrow R + R_i + R'_i$ ;
5 for each party  $i = 1 \dots p$  do
6    $R \leftarrow R - R'_i$ ;
/* Note that  $R_1$  (or  $R_2$ ) might be computed by party
   2 (or 1) if  $p = 2$ , but this does not affect the
   security of Algorithm 2 (see detailed analysis
   later on)
*/
```

We let every party $i = 1 \dots p$ locally create $(m'_i - m_i)$ artificial constraints, convert each of its inequality constraints to equality with one or more slack variables and compute $B_{\cdot I_i} A'_{I_i}$, $B_{\cdot I_i} b'_{I_i}$, $B_{\cdot I_i} D'_{I_i}$ with its privately held matrices $B_{\cdot I_i}$ and D'_{I_i} . To obtain linear program (4.8), we can securely sum $BA' = \sum_{i=1}^p B_{\cdot I_i} A'_{I_i}$, $BD' = \sum_{i=1}^p B_{\cdot I_i} D'_{I_i}$ and $Bb' = \sum_{i=1}^p B_{\cdot I_i} b'_{I_i}$ over p parties with Algorithm 1.

Algorithm 1 ensures that only the sum of the distributed matrix products shares are disclosed. Since $B_{\cdot I_i} A'_{I_i}$, $B_{\cdot I_i} D'_{I_i}$ and $B_{\cdot I_i} b'_{I_i}$ are unknown to other parties (the special case $p = 2$ is discussed later on), the equations with real variables cannot be established. Thus, besides hiding m , the secure sum algorithm (Algorithm 1) can also reduce the inference probability, compared with revealing every share of the matrix products to public in the prior work [75, 90].

Next, we present the detailed steps of the inference-proof algorithm in Algorithm 2. While solving linear program (4.8), party $i = 1 \dots p$ can only learn:

Algorithm 2: Inference-proof Algorithm to PPHPLP

- 1 Each party $i = 1 \dots p$ locally creates some artificial constraints, thus $A_{I_i} \in R^{m_i \times n}$ and $b_{I_i} \in R^{m_i}$ are expanded to $A'_{I_i} \in R^{m'_i \times n}$ and $b'_{I_i} \in R^{m'_i}$ respectively where $m'_i > m_i$ and $\text{rank}(A'_{I_i}) > \text{rank}(A_{I_i})$;
- 2 Each party $i = 1 \dots p$ locally converts each of its inequality constraints into equality with one or more slack variables where the total number of slack variables $T_i \geq m'_i$;
- 3 All p parties securely sum $m' = \sum_{i=1}^p m'_i$ and $t = \sum_{i=1}^p T_i$ where $t > m' > m$ (Algorithm 1);
- 4 Each party $i = 1 \dots p$ generates its privately held random matrix $D'_{I_i} \in R^{m'_i \times t}$ for the slack variables generated in Step 2 (as discussed in Section 4.1.2), where m'_i is the number of rows held by party i in D' as shown in (4.7);
- 5 All p parties agree on an integer value for $k \geq m'$, where k is the number of rows of the random matrix $B \in R^{k \times m'}$ as defined in (4.1) and (4.2);
- 6 Each party $i = 1 \dots p$ generates its privately held random matrix $B_{\cdot I_i} \in R^{k \times m'_i}$, where m'_i is the number of columns held by party i in $B = [B_{\cdot I_1}, B_{\cdot I_2}, \dots, B_{\cdot I_p}] \in R^{k \times m'}$;
- 7 Each party $i = 1 \dots p$ locally computes the matrix products $B_{\cdot I_i} A'_{I_i}$, $B_{\cdot I_i} D'_{I_i}$ and $B_{\cdot I_i} b'_{I_i}$;
- 8 All p parties securely sum the matrix products (Algorithm 1) to get the constraint matrix of the inference-proof linear program (4.8):

$$BA' = [B_{\cdot I_1} A'_{I_1} + B_{\cdot I_2} A'_{I_2} + \dots + B_{\cdot I_p} A'_{I_p}] \in R^{k \times n}$$

10

$$BD' = [B_{\cdot I_1} D'_{I_1} + B_{\cdot I_2} D'_{I_2} + \dots + B_{\cdot I_p} D'_{I_p}] \in R^{k \times t}$$

11 and the right hand side vector for linear program (4.8):

12

$$Bb' = [B_{\cdot I_1} b'_{I_1} + B_{\cdot I_2} b'_{I_2} + \dots + B_{\cdot I_p} b'_{I_p}] \in R^k$$

- 13 Solve LP (4.8) to get the optimal solution $(x^*, (x'_s)^*)$, thus x^* is optimal for LP (4.5);
-

- the value m' (securely summed in Step 3), which can be far greater than m .
- the value of $\text{rank}(BA') = \text{rank}(A')$, which can be far greater than m and $\text{rank}(A)$ ($m = \text{rank}(A)$ if all the rows in A are linearly independent).

- all p parties' total number of slack variables t (securely summed in Step 3), which can be far greater than m and ℓ ($m = \ell$ if all the constraints are inequalities).

Note that in two-party case ($p = 2$), even though $B_{\cdot I_1} A'_{I_1}$, $B_{\cdot I_1} D'_{I_1}$ and $B_{\cdot I_1} b'_{I_1}$ can be computed from Algorithm 1 by party 2 (or vice-versa), party 1's number of constraints m_1 can be only inferred as any number no greater than $m' - m'_2$ or $t - m'_2$ for party 2 in Algorithm 2 since party 2 knows at most m' , t and m'_2 (m' and t can be sufficiently large). With unknown m_1 , party 2 cannot further infer party 1's privately held constraints (or vice-versa). Thus, Algorithm 2 is still secure when $p = 2$.

Overall, Algorithm 2 reveals significantly less information than [75, 90]: m, ℓ , $\{i = 1 \dots p, m_i, B_{\cdot I_i} A_{I_i}, B_{\cdot I_i} D_{I_i}, B_{\cdot I_i} b_{I_i}\}$, BA, BD, Bb and $\text{rank}(A)$ etc are not disclosed to public. Moreover, every party's private constraints cannot be inferred by formulating equations with real variables since the number of constraints are unknown anymore. To further reduce the inference risk, every party $i = 1 \dots p$ can locally increase m'_i and T_i by adding more artificial constraints and slack variables.

Efficiency and Security Tradeoff Discussion

Recall that the transformation approach in [75, 90] tends to suffer high risk of inference attack in some vulnerable cases (e.g., $p = 2$). Indeed, Algorithm 2 can resolve the inference attack for any horizontally partitioned linear program held by p parties, but expands the problem size and thus trades off some efficiency

for enhanced security. We now discuss this efficiency and security tradeoff for different PPHPLPs.

First, if $p = 2$ (two-party), clearly, we should utilize Algorithm 2 to tackle the potential inference attack since every party's number of constraints m_1 or m_2 are explicitly disclosed to each other in [75, 90].

Second, if $p > 2$ (multiparty), Algorithm 2 is also effective to enhance security. In this case, since the probability of inferring any party i 's number of constraints m_i by another party $j \neq i$ with its known m_j is quite low: m_i can be any integer in $[1, m - m_j - (p - 2)]$ (assuming that each of the remaining $p - 2$ parties should include at least 1 constraint), the probability of inferring party i 's constraints is even lower. Note that the prior approach [75, 90] remains applicable with better efficiency.

However, for any $p > 2$, there exist some special cases which are still vulnerable if directly applying the approach in [75, 90]: e.g., $p = 4$, $m = 9$, $m_j = 6$, party j can infer that each of the remaining three parties has exactly only 1 constraint, and then their constraints can be reconstructed by the inference attack described in Section 4.1.1 if their transformed share are explicitly revealed [75, 90]. Therefore, we should adapt our secure sum into the prior PPHPLP algorithm (simply replacing step (III) in Algorithm 3.1 in [90] with Algorithm 1) for keeping every party's transformed share private (note that this has not resolved the inference on every party's number of constraints yet).

In summary, if stronger security is desired and the slightly declined efficiency is tolerable, it would be better to apply our inference-proof algorithm (Algorithm

2) for PPHPLP co-held by any number of parties p . If more efficiency is required and every party does not care the potential inference on its number of constraints when $p > 2$, we can integrate Algorithm 1 and the prior PPHPLP algorithm [75, 90].

4.1.3 Computational Results

We conduct experiments similar to those in [90] to demonstrate the computation cost with MATLAB on a 6G-RAM PC. Specifically, we generate $m = 400$ constraints for $n = 1000$ variables ($\ell = 200$ inequality constraints) for linear program (4.5): $A \in R^{400 \times 1000}$ is randomly generated where every element in A is uniformly distributed in the interval $[-50, 50]$; $b \in R^{400 \times 1}$ is randomly generated in the interval $[300, 500]$. The linear program is horizontally partitioned among 3 parties: $m_1 = 100, m_2 = 100, m_3 = 200$ (half of each party's constraints are inequalities).

To formulate linear program (4.8), we generate 100, 100 and 200 artificial constraints for 3 parties respectively: given an equality or inequality constraint, we reduce a positive coefficient or increase a negative coefficient of the constraint while retaining the same righthand side value (this is not the only way to create artificial constraints). Thus, we have $m'_1 = 200, m'_2 = 200, m'_3 = 400$ and $\ell' = 600$. We let every party utilize two slack variables to convert each of its local inequality constraints to equality, then $t = 1200$. Every party generates $D'_{I_1} \in R^{200 \times 1200}, D'_{I_2} \in R^{200 \times 1200}$ and $D'_{I_3} \in R^{400 \times 1200}$ where the positive random numbers in them are uniformly distributed in the interval $(0, 1]$. We let $k = 1000 > m' = 800$ and generate three random matrices $B_{.I_1} \in R^{1000 \times 200}$,

$B_{.I_2} \in R^{1000 \times 200}$ and $B_{.I_3} \in R^{1000 \times 400}$ with elements uniformly distributed in the interval $[0, 1]$. Thus, we have $A' \in R^{800 \times 1000}$, $D' \in R^{800 \times 1200}$, $B \in R^{1000 \times 800}$ and $b' \in R^{800 \times 1}$.

We solve linear programs (4.5) and (4.8), and compare the optimal solutions of them – which are identical. The computation time was 16.583s for the inference-proof linear program (4.8) which is comparable to 8.34s for linear program (4.5).

4.2 Arbitrarily Partitioned LP Problem

Recall that Chapter 1 has given a story in the packaged goods industry – using a web-based collaborative logistics service (Nistevo.com), the freight costs can be cut by 15%, for an annual savings of \$2 million [123]). 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 efficiencies. Since transportation problem can be modeled with LP, a *Collaborative LP* solution that tightly limits the information disclosure would make this possible without the release of proprietary information. Specifically, such optimization problem can facilitate cooperative parties to jointly maximize global profits (or minimize costs) while satisfying several global and/or local linear constraints. Since each party's share of the global constraints and the local constraints generally refer to its private limitations or capacities and the optimal solution represents its decision, limited disclosure should prevent revealing those commercial secrets in this distributed computing scenario.

While completely arbitrary partitioning of constraints and variables is possible, in many realistic collaborative LP problems, each company holds its own

variables: the values for which together constitute the global optimum decision. Variables are generally not shared between companies because collaborators may have their own operations w.r.t. a maximized profit or minimized cost. Consider the LP model for the collaborative production example in Table 1.1:

$$\begin{aligned} & \max \quad 40x_1 + 50x_2 \\ \text{s.t.} \quad & \begin{cases} x_1 \geq 0 \\ x_2 \geq 0 \end{cases} : \begin{cases} 4x_1 + 3x_2 \leq 60 + 60 \\ x_1 \leq 10 \\ 2x_2 \leq 30 \end{cases} \end{aligned} \quad (4.9)$$

As shown above, the constraints are *arbitrarily partitioned*. Arbitrary partition implies that some constraints are globally formulated among multiple parties (“vertically partitioned” global constraints – each party knows only a share of these constraints) whereas some constraints are locally held and known by only one party (“horizontally partitioned” local constraints – each party completely owns all these constraints). After solving this problem, Factory 1 should only know the number of bowls to be produced while Factory 2 should only know the number of mugs to be produced in the collaboration. However, they should not learn anything about the private constraints and the share of the optimal solution as well as the objective function from each other. A generalized form of such story (K factories share raw materials for manufacturing more products) is given:

Example 4.1 (Collaborative Production) K factories $P_1 \dots P_K$ share some raw materials for production (maximizing profits): the number of company P_i ’s ($i \in [1, K]$) product j to be manufactured are denoted as x_{ij} , thus P_i holds $x_i = \{\forall j, x_{ij}\}$.

In this general form, $P_1 \dots P_K$ 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 problem, each company should only know its production assignment for each of its products (the values of x_i in the global optimal solution).

Similarly, we can also model the collaborative transportation problem with this category of collaborative LP problem:

Example 4.2 (Collaborative Transportation) *K Companies $P_1 \dots P_K$ 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, $P_1 \dots P_K$ 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 problem, each company should know its delivery amount from each location to another location (the values of x_i in the global optimal solution).

To summarize, we formally define the problem as below:

Definition 4.1 (K-Party LP Problem (K-LP)) *An LP problem is solved by K distributed parties where each party 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 Equation 4.10^{2,3} ($i \in [1, K]$ and $\sum_{i=1}^K b_0^i = b_0$).

$$\begin{aligned} & \max \quad c_1^T x_1 + c_2^T x_2 + \cdots + c_K^T x_K \\ \text{s.t.} \quad & \begin{cases} x_1 \in \mathbb{R}^{n_1} \\ x_2 \in \mathbb{R}^{n_2} \\ \vdots \\ x_K \in \mathbb{R}^{n_K} \end{cases} : \begin{pmatrix} A_1 & \cdots & A_K \\ B_1 & & \\ & \ddots & \\ & & B_K \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_K \end{pmatrix} \begin{matrix} \bowtie_0 \\ \bowtie_1 \\ \vdots \\ \bowtie_K \end{matrix} \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_K \end{pmatrix} \end{aligned} \quad (4.10)$$

Besides collaborative production and transportation, K-LP problems occur very frequently in reality, e.g., selling the goods in bundles for distributed parties to maximize the global profits, and determining profit-maximized travel packages for hotels, airlines and car rental companies.

In this section, first, we introduce a secure and efficient protocol for K-LP problem by revising Dantzig-Wolfe decomposition [28, 122] 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-party column generation (SCG) protocol to solve the transformed LP problem securely and efficiently. Second, we also give privacy analysis by showing that the probability of inferring any private information from each other is nearly impossible. Finally, we present communication and computation cost analysis for our protocol.

² \bowtie denotes $\leq, =$ or \geq

³Matrices: $i \in [1, K]$, $A_i \in \mathbb{R}^{m_0 \times n_i}$, $B_i \in \mathbb{R}^{m_i \times n_i}$, $c_i \in \mathbb{R}^{n_i}$, $b_0 \in \mathbb{R}^{m_0}$ and $b_i \in \mathbb{R}^{m_i}$

4.2.1 Revised Dantzig-Wolfe Decomposition

As shown in Equation 3.1, K-LP problem has a typical Block-angular structure, though the number of global constraints can be significantly larger than each party'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 data-hidden format that preserves each party's private information. Moreover, we also show that the optimal solution for each party's variables can be derived after solving the transformed problem.

K-LP Transformation

Du [32, 33] and Vaidya [124] proposed a transformation approach for solving two-party collaborative LP problems: transforming an $m \times n$ constraint matrix M (and the objective vector c^T) to another $m \times n$ matrix $M' = MQ$ (and $c'^T = c^T Q$) by post-multiplying an $n \times n$ matrix Q , solving the transformed problem and reconstructing the original optimal solution. Bednarz et al. [11] showed that to ensure correctness, the transformation matrix Q must be monomial. Following them, we let each party 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 party 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 party should be chosen as the master problem solver whereas all parties (including the master problem solving party) should solve the pricing prob-

lems. For transformed K-LP problem (Equation 4.11), we can let $\forall i \in [1, K]$, P_i send its transformed matrices/vector $A_i Q_i$, $B_i Q_i$, $c_i^T Q_i$ to another party 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 no private information can be learnt while solving the problems (the attack specified in [11] can be eliminated in our secure K-LP problem). Otherwise, if each party solves its pricing problem, since each party 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 4.2.2).

$$\begin{aligned} & \max \quad \sum_{i=1}^K c_i^T Q_i y_i \\ \text{s.t.} \quad & \begin{cases} y_1 \in \mathbb{R}^{n_1} \\ y_2 \in \mathbb{R}^{n_2} \\ \vdots \\ y_K \in \mathbb{R}^{n_K} \end{cases} \begin{pmatrix} A_1 Q_1 & \dots & A_K Q_K \\ B_1 Q_1 & & \\ & \ddots & \\ & & B_K Q_K \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_K \end{pmatrix} \begin{matrix} \bowtie_0 \\ \bowtie_1 \\ \vdots \\ \bowtie_K \end{matrix} \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_K \end{pmatrix} \end{cases} \quad (4.11) \end{aligned}$$

The K-LP problem can be transformed to another block-angular structured LP problem as shown in Equation 4.11. We can derive the original solution from the solution of the transformed K-LP problem using the following theorem.

Theorem 4.1 *Given the optimal solution of the transformed K-LP problem $y^* = (y_1^*, y_2^*, \dots, y_K^*)$, the solution $x^* = (Q_1 y_1^*, Q_2 y_2^*, \dots, Q_K y_K^*)$ should be the optimal solution of the original K-LP problem.*

Proof. Suppose $x^* = (Q_1 y_1^*, Q_2 y_2^*, \dots, Q_K y_K^*)$ is not the optimal solution of the original vertical LP problem. In this case, we have another vector $x' =$

$(x'_1, x'_2, \dots, x'_K)$ such that $c^T x' > c^T x^* \implies c_1^T x'_1 + \dots + c_K^T x'_K > c_1^T x_1^* + \dots + c_K^T x_K^*$ where $Mx' \preceq b$ and $x' \geq 0$. Let $y' = (y'_1, \dots, y'_K) = (Q_1^{-1}x'_1, \dots, Q_K^{-1}x'_K)$, thus we have $c_1^T Q_1 y'_1 + \dots + c_K^T Q_K y'_K = c_1^T Q_1 Q_1^{-1} x'_1 + \dots + c_K^T Q_K Q_K^{-1} x'_K = c_1^T x'_1 + \dots + c_K^T x'_K$.

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

Hence, y' is a better solution than y^* which is a contradiction to that y^* is the optimal solution. Thus, Theorem 4.1 has been proven.

Hiding Righthand Side Value b

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) [11, 32, 124]. 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 righthand side constants b in the constraints. Since b sometimes refers to the amount/quantity of limited resources (e.g., labors, materials) or some demands (e.g., the quantity of one product should be no less than 10, $x_{ij} \geq 10$), they should not be revealed. We can hide b for each party before transforming the constraint matrix and sending them to other parties.

Intuitively, in the K-LP problem, each party $i \in [1, K]$, P_i 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 hide both b_0^i and b_i . While hiding 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 party 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 j th 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 Equation 4.12 ($A_i^1, \dots, A_i^{m_0}$ denote the rows of matrix A_i).

$$A_i x_i = \begin{pmatrix} A_i^1 \\ A_i^2 \\ \vdots \\ A_i^{m_0} \end{pmatrix} x_i \implies A'_i x'_i = \left(\begin{array}{c|cccc} A_i^1 & \alpha_{i1} & 0 & \dots & 0 \\ A_i^2 & 0 & \alpha_{i2} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_i^{m_0} & 0 & 0 & \dots & \alpha_{im_0} \end{array} \right) \begin{pmatrix} x_i \\ s_{i1} \\ \vdots \\ s_{im_0} \end{pmatrix} \quad (4.12)$$

As a result, b_0^i can be hidden as $(b_0^i)'$ where $\forall j \in [1, m_0], (b_0^i)_j$ and $(b_0^i)'_j$ denote the j th 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 party P_i :

$$b_0^i = \begin{pmatrix} (b_0^i)_1 \\ (b_0^i)_2 \\ \vdots \\ (b_0^i)_{m_0} \end{pmatrix} \implies (b_0^i)' = \begin{pmatrix} (b_0^i)_1 + \alpha_{i1}\eta_{i1} \\ (b_0^i)_2 + \alpha_{i2}\eta_{i2} \\ \vdots \\ (b_0^i)_{m_0} + \alpha_{im_0}\eta_{im_0} \end{pmatrix} \quad (4.13)$$

To guarantee the equivalence of $\forall j \in [1, m_0], s_{ij} = \eta_{ij}$, each party 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 $r_{ijk}, k \in [1, m_0]$:

$$s.t. \begin{cases} \sum_{j=1}^{m_0} r_{ij1} s_{ij} = \sum_{j=1}^{m_0} r_{ij1} \eta_{ij} \\ \sum_{j=1}^{m_0} r_{ij2} s_{ij} = \sum_{j=1}^{m_0} r_{ij2} \eta_{ij} \\ \vdots \quad \quad \quad \vdots \\ \sum_{j=1}^{m_0} r_{ijm_0} s_{ij} = \sum_{j=1}^{m_0} r_{ijm_0} \eta_{ij} \end{cases} \quad (4.14)$$

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 [36]). 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 party P_i add those linear independent equality constraints (Equation 4.14) into its local constraints $B_i x_i \bowtie_i b_i$. Therefore, we have:

- the j th 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 j th 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 parties).
- 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 $\forall i, s_i = \eta_i$.

Apart from b_0^i , we can hide b_i using a similar way. P_i can use the same set of artificial variables s_i to hide b_i . By generating linear combination (not required to be linear independent at this time) of the artificial variables $s_i = \{\forall j \in [1, m_0], s_{ij}\}$, the left-hand side of the w th ($w \in [1, m_i]$) 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 w th value in b_i is updated as $b_i^w \leftarrow b_i^w + \sum_{j=1}^{m_0} h_{ijw} \eta_{ij}$. If hiding 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 linear independent constraints and calculating the exact values of the artifi-

cial 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).

Algorithm 3: Hiding Righthand Side Value b

```

/*  $A_i^j$ ,  $(A_i^j)'$ ,  $B_i^j$  and  $(B_i^j)'$  denote the  $j$ th row of  $A_i$ ,
    $A_i'$ ,  $B_i$  and  $B_i'$  respectively */
1 forall party  $P_i, i \in [1, K]$  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  $s_i = \{s_{i1}, \dots, s_{im_0}\}$  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      $B_i^w x_i \bowtie_i^w b_i^w \leftarrow B_i^w x_i + \sum_{j=1}^{m_0} h_{ijw} s_{ij} \bowtie_i^w b_i^w + \sum_{j=1}^{m_0} h_{ijw} \eta_{ij}$ ;
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  $s_i$  [36];
11  union  $B_i x_i \bowtie_i b_i$  (hidden 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 */

```

Furthermore, since hiding b should be implemented 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 hides b (the data owner). Hence, b_i and b_0^i can be secure against semi-honest adversaries. Algorithm 3 introduces the detailed steps of hiding b . Note that if any party P_i pursues higher degree of privacy guarantee, P_i can generate more artificial variables and more local equality constraints for hiding both b_0^i and b_i (which is a typical tradeoff between privacy and efficiency).

Revised Dantzig-Wolfe Decomposition

Dantzig-Wolfe decomposition was originally utilized to solve large-scale block-angular structured LP problems [28, 122]. Indeed, we can appropriately partition the constraints of every K-LP problem into block-angular structure (as shown in Equation 3.1). Specifically, we can consider each party $P_i (i \in [1, K])$'s horizontally partitioned local constraint matrix B_i as a block. However, treating the vertically partitioned constraints that are shared by at least two parties as global constraints among all parties, each party P_i holds a share A_i in the global constraint matrix. Even if A_i may have more rows than B_i , the constraints are still block-angular partitioned.

Furthermore, the structure of the local constraints block $B'_i Q_i$ and the global constraints share $A'_i Q_i$ will not change after locally hiding b and transforming the blocks. Hence, we can apply Dantzig-Wolfe decomposition to the transformed K-LP problem. We thus summarize the complete procedure as Revised Dantzig-

Wolfe Decomposition:

Definition 4.2 (Revised Dantzig-Wolfe Decomposition) *A secure and efficient approach to solving K-LP problems that includes the following stages: locally hiding b by every party, locally transforming blocks by every party, locally decomposing the transformed K-LP problem with Dantzig-Wolfe decomposition by every party, and finally solving the decomposed problem.*

According to Equation 3.4, the Dantzig-Wolfe representation of the transformed K-LP problem is:

$$\begin{aligned}
 & \max \quad \sum_j c_1^T Q_1 y_1^j \lambda_{1j} + \cdots + \sum_j c_K^T Q_K y_K^j \lambda_{Kj} \\
 & s.t. \quad \begin{cases} \sum_{\forall j} A'_1 Q_1 y_1^j \lambda_{1j} + \cdots + \sum_{\forall j} A'_K Q_K y_K^j \lambda_{Kj} \preceq_0 b'_0 \\ \sum_{\forall j} \delta_{1j} \lambda_{1j} = 1 \\ \vdots \\ \sum_{\forall j} \delta_{Kj} \lambda_{Kj} = 1 \\ \lambda_1 \in \mathbb{R}^{|E'_1|}, \dots, \lambda_K \in \mathbb{R}^{|E'_K|}, \delta_{ij} \in \{0, 1\}, i \in [1, K] \end{cases} \quad (4.15)
 \end{aligned}$$

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 hiding b , note that the coefficient of all the artificial variables in c'_i is equal to 0), and $|E'_1|, \dots, |E'_K|$ represent the number of each party's extreme points/rays in the transformed problem. In sum, Figure 4.1 depicts the three steps of Revised Dantzig-Wolfe Decomposition.

Furthermore, after solving the transformed problem, each party P_i should obtain an optimal solution $\lambda_i = \{\forall j, \lambda_{ij}\}$. Figure 4.2 shows the steps of deriving each party's optimal solution for the original K-LP problem. Specifically,

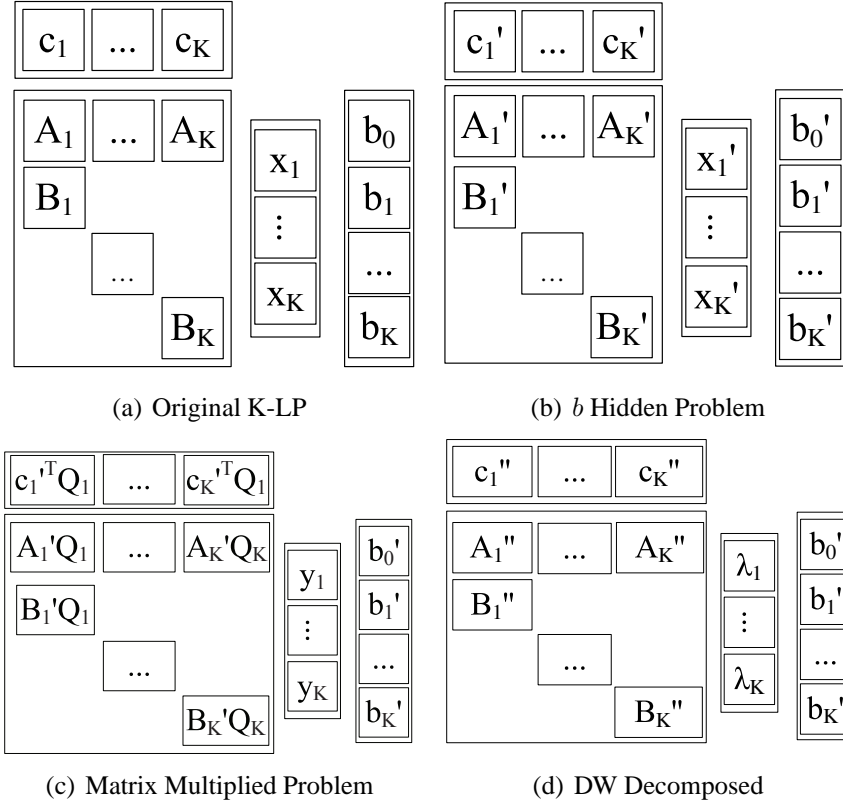


Figure 4.1. Local Transformation in Revised Dantzig Decomposition

in step 1, the optimal solutions for each party's transformed problem can be obtained by computing the convexity combination of all vertices/extreme 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 hidden b can be derived by left multiply Q_i for each party (Theorem 4.1). In step 3, each party can extract its individual optimal solution in the K-LP problem by excluding the artificial variables (for hiding b) from the optimal solution of x'_i .

⁴Apparently, if $y_i = 0$ and we have $x'_i = Q_i y_i$, x'_i should be 0 and revealed to other parties. However, y_i includes some transformed variables that is originally the value-fixed yet unknown artificial variables for hiding 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.

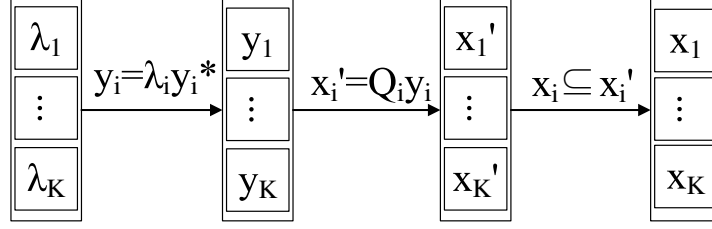


Figure 4.2. Optimal Solution Reconstruction after Solving the Transformed K-LP Problem

At first glance, we can let an arbitrary party formulate and solve the transformed master problem (Equation 4.15). However, the efficiency and security is not good enough for large-scale problems since the number of vertices/extreme 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 party and all the vertices/extreme rays should be sent to the master problem solver (assuming that the K-LP problem is standardized with slack variables before executing Algorithm 3 and the transformation). In Section 4.2.2, we introduce our secure K-party Column Generation (SCG) Protocol that is based on the column generation algorithm associated with Dantzig-Wolfe decomposition [28, 122] – iteratively formulating restricted master problems (the master problem/Equation 4.15 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 will be given later on).

The advantages of this sort of decomposition and column generation 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.

4.2.2 Secure Column Generation (SCG) Protocol for K-LP Problem

Solving K-LP by revised Dantzig-Wolfe decomposition and column generation is fair to all K parties. Hence, we assume that an arbitrary party can be the master problem solver. As discussed in Section 4.2.1, it is inefficient to solve the full master problem since every party holds a huge number of vertices/extreme rays, derived from their local constraints. In this section, we present an efficient protocol – the secure column generation (SCG) for revised Dantzig-Wolfe decomposition in semi-honest model. We also give the analysis of privacy and computation/communication cost for the protocol.

As mentioned in Section 4.2.1, 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 vertices/extreme 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 $[c_i] = (\forall j \in [1, \frac{n'_i!}{m'_i!(n'_i-m'_i)!}], c_i^T Q_i y_i^j)$ and $[A_i] = (\forall j \in [1, \frac{n'_i!}{m'_i!(n'_i-m'_i)!}], A'_i Q_i y_i^j)$. For RMP, we denote the coefficients in the master problem restricted to

$\mathbb{R}^{|\widehat{E}_1|}, \dots, \mathbb{R}^{|\widehat{E}_K|}$ ($|\widehat{E}_1|, \dots, |\widehat{E}_K|$ represent each party's number of vertices/extreme 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 parties are initialized to non-basis 0. τ_i denotes the number of vertices/extreme rays in P_i 's pricing problem proposed to the master solver where $\forall i \in [1, K], \tau_i \leq \frac{n'_i!}{m'_i!(n'_i - m'_i)!}$. Hence, we represent the RMP as below:

$$\begin{aligned}
 & \max \quad \widehat{c}_1^T \widehat{\lambda}_1 + \dots + \widehat{c}_K^T \widehat{\lambda}_K \\
 \text{s.t.} \quad & \begin{cases} \widehat{A}_1 \widehat{\lambda}_1 + \dots + \widehat{A}_K \widehat{\lambda}_K \preceq_0 b'_0 \\ \sum_{j=1}^{\tau_1} \delta_{1j} \lambda_{1j} = 1 \\ \vdots \\ \sum_{j=1}^{\tau_K} \delta_{Kj} \lambda_{Kj} = 1 \\ \lambda_1 \in \mathbb{R}^{|\widehat{E}_1|}, \dots, \lambda_K \in \mathbb{R}^{|\widehat{E}_K|}, \delta_{ij} \in \{0, 1\} \end{cases} \quad (4.16)
 \end{aligned}$$

In the general form of column generation algorithm [28, 122] among distributed parties (without privacy concern), every party 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 party, e.g., P_1) formulates the restricted master problem (RMP) based on the proposed optimal solutions of the pricing problems from all parties P_1, \dots, P_K (the optimal solution of any pricing problem is also a proposed column of the master problem and an new vertex or extreme ray of the pricing problem). 2) the master problem solver (w.l.o.g., P_1) solves the RMP and sends the dual values of the optimal solution to every pricing problem solver P_1, \dots, P_K . 3) every pricing problem solver thus formulates and solves a new pricing problem with its local constraints $B_i x_i \preceq_i b_i$ (only the ob-

jective function varies based on the received dual values) to get the new optimal solution/column/vertex or extreme ray (which should be possibly proposed to the master problem solver P_1 to improve the objective value of the RMP).

To improve the degree of privacy protection, we can let each party's pricing problems be solved by another arbitrary party since other parties cannot apply inverse transformation to learn additional information. If this is done, more information can be hidden ⁵ while iteratively formulating and solving the RMPs and the pricing problems. Without loss of generality, we assume that P_1 solves the RMPs, P_i sends $A'_i Q_i$, $B'_i Q_i$, $c_i'^T Q_i$, $(b_0^i)'$ and b_i' to the next party $P_{(i \bmod K)+1}$ and $P_{(i \bmod K)+1}$ solves P_i 's pricing problems (P_1 solves P_K 's pricing problems).

Solving Restricted Master Problem (RMP) by an Arbitrary Party

Algorithm 4: Solving the RMP by an Arbitrary Party

Input : K parties P_1, \dots, P_K where $\forall i \in [1, K]$, $P_{(i \bmod K)+1}$ holds P_i 's variables y_i , transformed matrices $A'_i Q_i$, $B'_i Q_i$, $c_i'^T Q_i$ and transformed vectors b_i' , $(b_0^i)'$

Output: the optimal dual solution of RMP $(\pi^*, \mu_1^*, \dots, \mu_K^*)$ or infeasibility

/ Party P_i 's τ_i pairs of vertices/extreme rays $(\forall j \in [1, \tau_i], y_i^j \in \hat{y}_i)$ and variables $\lambda_{ij} \in \hat{\lambda}_i$ have been sent to P_1 from $P_{(i \bmod K)+1}$ ($s_i \leq \frac{n_i'!}{m_i'!(n_i'-m_i')!}$ and P_1 solves the RMP) */*

- 1 P_1 constructs the objective of the RMP: $\sum_{i=1}^K \sum_{j=1}^{\tau_i} c_i'^T Q_i y_i^j \lambda_{ij}$;
 - 2 P_1 constructs the constraints of the RMP: $\sum_{i=1}^K \sum_{j=1}^{\tau_i} A'_i Q_i y_i^j \lambda_{ij} \preceq_0^j \sum_{i=1}^K (b_0^i)'$ and $\forall i \in [1, K]$, $\sum_{j=1}^{\tau_i} \delta_{ij} \lambda_{ij} = 1$ where $b_0 = \sum_{i=1}^K (b_0^i)'$ can be securely summed or directly summed by K parties;
 - 3 P_1 solves the above problem using Simplex or Revised Simplex method;
-

The detailed steps of solving RMP in revised Dantzig Wolfe decomposition

⁵The solutions and the dual optimal values of the true pricing problems

are shown in Algorithm 4. We can show that solving RMP is secure in semi-honest model.

Lemma 4.1 *Algorithm 4 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 party;*

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 parties (though this scarcely happens). We thus discuss the potential privacy leakage in this case.

We look at the matrices/vectors acquired from all other parties $\forall i \in [1, K]$, P_i by P_1 . Specifically, $[c_i] = (\forall j \in [1, \frac{n'_i!}{m'_i!(n'_i-m'_i)!}], c_i'^T Q_i y_i^j)$ and $[A_i] = (\forall j \in [1, \frac{n'_i!}{m'_i!(n'_i-m'_i)!}], A_i' Q_i y_i^j)$ 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 j th value in $[c_i]$ is equal to $c_i'^T Q_i y_i^j$, and the j th column in matrix $[A_i]$ is equal to $A_i' Q_i y_i^j$.

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/blocks 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 $n'_i + m_0 n'_i + n'^2_i + n'_i \cdot \frac{n'_i!}{m'_i!(n'_i-m'_i)!}$. Due to $n'_i \gg m_0$ in linear programs, we have $n'_i + m_0 n'_i + n'^2_i + n'_i \cdot \frac{n'_i!}{m'_i!(n'_i-m'_i)!} \gg (m_0 + 1) \cdot \frac{n'_i!}{m'_i!(n'_i-m'_i)!}$. Thus, those unknown variables in c'_i , A'_i , Q_i ⁶ 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 (hidden) and $B_i x_i \bowtie_i b_i$ from any party 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, hiding b and transforming c_i , A_i and B_i do not change the total payoff (optimal value) of each party, the payoffs of all values are revealed to P_1 as well. Since we can let arbitrary party solve any other party'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 with limited information disclosure.

Solving Pricing Problems by Peer-party

While solving the K-LP problem by the column generation algorithm, in every iteration, each party's pricing problem might be formulated to test whether any column of the master problem (extreme point/ray of the corresponding party's

⁶As described in [11], Q_i should be a monomial matrix, thus Q_i has n'_i unknown variables located in n'^2_i unknown positions.

pricing problem) should be proposed or not. If any party's pricing problem cannot propose column to the master solver in the previous iterations, no pricing problem is required for this party any further. As discussed in Section 4.2.1, 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 party $P_{(i \bmod K)+1}$ ($i \in [1, K]$) has received party P_i 's variables y_i , transformed matrices/vector $A'_i Q_i$, $B'_i Q_i$, $c_i'^T Q_i$ and the transformed vectors b'_i , $(b_0^i)'$ (as shown in Figure 4.1(c)). Party $P_{(i \bmod K)+1}$ thus formulates and solves party P_i 's pricing problem.

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 party P_i can be derived:

$$d_{ij} = (c_i'^T Q_i - \pi^* A'_i Q_i) y_i^j - \begin{cases} (\mu_i^*)_j & \text{if } y_i^j \text{ is an extreme point (vertex)} \\ 0 & \text{if } y_i^j \text{ is an extreme ray} \end{cases} \quad (4.17)$$

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

$$\begin{aligned} \max \quad & (c_i'^T Q_i - \pi^* A'_i Q_i) y_i \\ \text{s.t.} \quad & \begin{cases} B'_i Q_i y_i \preceq b'_i \\ y_i \in \mathbb{R}^{n'_i} \end{cases} \end{aligned} \quad (4.18)$$

Algorithm 5 presents the detailed steps of solving P_i 's pricing problem by $P_{(i \bmod K)+1}$ which is secure.

Lemma 4.2 *Algorithm 5 reveals (to $P_{(i \bmod K)+1}$) only:*

Algorithm 5: Solving Pricing Problem by Peer-party

Input : An honest-but-curious party $P_{(i \bmod K)+1}$ holds party P_i 's variables y_i , transformed matrices/vector $A'_i Q_i$, $B'_i Q_i$, $c_i'^T Q_i$ and vectors b'_i , $(b_0^i)'$

Output: New Optimal Solution or Infeasibility of the Pricing Problem

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

- 1 $P_{(i \bmod K)+1}$ constructs the objective function of the pricing problem of P_i :
 $z_i = [c_i'^T Q_i - \pi^*(A'_i Q_i)] y_i$;
- 2 $P_{(i \bmod K)+1}$ constructs the constraints: $B'_i Q_i y_i \bowtie_i b'_i$;
- 3 $P_{(i \bmod K)+1}$ solves the above pricing problem using Revised Simplex algorithm or a specialized algorithm if the problem has specific structure (e.g., transportation problem) ;
- 4 **if the problem is infeasible then**
- 5 | the original problem is infeasible and return;
- 6 **switch the optimal value z_i^* do**
- 7 | **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 $(c_i'^T Q_i y_i^j \lambda_{ij}^*, A'_i Q_i y_i^j \lambda_{ij}^*)$ to P_1 ;
- 8 | **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 $(c_i'^T Q_i y_i^j \lambda_{ij}^*, A'_i Q_i y_i^j \lambda_{ij}^*)$ to P_1 ;
- 9 | **case $z_i^* \leq \mu_i$**
- 10 | | no extra variable (λ_{ij}) from P_i enters the basis of the RMP;

- 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-party solve any party'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 from P_i by $P_{(i \bmod K)+1}$: size n'_i vector $c_i'^T Q_i$, $m'_i \times n'_i$ matrix $B'_i Q_i$ and $m_0 \times n'_i$ matrix $A'_i Q_i$. The j th value in $c_i'^T Q_i$ is equal to $c_i'^T Q_i^j$ (Q_i^j denotes the j th column of Q_i), and the value of the k th row and the j th column in $A'_i Q_i$ (or $B'_i Q_i$) is equal to the scalar product of the k th 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 $c_i'^T Q_i$, $A'_i Q_i$ 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.⁷

Hence, $P_{(i \bmod K)+1}$ learns nothing about A_i , B_i , c_i , b_0^i and b_i 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 combina-

⁷Bednarz et al. [11] proposed a possible attack on inferring Q with the known transformed and original objective vectors ($c^T Q$ 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 happens to the special case of the collaborative LP problem in Vaidya's work [124] where one party holds the objective function while the other party holds the constraints. In our protocol, P_i sends $c_i'^T Q_i$ to $P_{(i \bmod K)+1}$, but $c_i'^T$ 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 [11].

tion 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 $B'_i Q_i y_i \bowtie_i b'_i$ is infeasible, we have: polyhedron $B'_i x_i \bowtie_i b'_i$ is also infeasible (Theorem 4.2). Hence, the specific party 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 parties).

Hence, solving the Pricing Problems by another arbitrary party is secure.

Theorem 4.2 *The polyhedra $B_i x_i \bowtie_i b_i$ and $B_i Q_i y_i \bowtie_i b_i$ have the same feasibility where $i \in [1, K]$.*

Proof. 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 4.2 has been proven.

Secure K-party Column Generation (SCG) Protocol

Algorithm 4 and 5 illustrate the detailed steps of one iteration while solving the transformed DW representation of the K-LP problem using column generation algorithm [28, 122]. In a K-party 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 6 presents the secure K-party column generation (SCG) protocol for securely solving the K-LP problem with revised Dantzig-Wolfe decomposition.

Also, for better illustrating the secure K-party column generation protocol (Algorithm 6), we present a protocol overview in Figure 4.3 where step 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 requires a lot of time to find the optimal solution. The subproblems may continue to generate

Algorithm 6: Secure K-party 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
   party's share in the transformed K-LP problem
   has been delivered to the corresponding pricing
   problem solver. */
1 forall party  $P_{(i \bmod K)+1}, i \in [1, K]$  do
2   | computes all the extreme points/rays and its convexity coefficient  $\lambda_{ij}$  in
   | polyhedron  $B'_i Q_i y_i \bowtie_i b'_i$ ;
3   | 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);
4   | computes  $A'_i Q_i y_i^j$  and  $c_i^T Q_i y_i^j$  and sends  $A'_i Q_i y_i^j \lambda_{ij}$ ,  $c_i^T Q_i y_i^j \lambda_{ij}$  and
   |  $(b_0^i)'$  to  $P_1$ ;
5  $P_1$  forms the RMP and solve it (Algorithm 4,  $z^*$  is the objective function
   value if the RMP is feasible);
6 if RMP is feasible then
7   |  $P_1$  sends  $\pi^*$  and  $\mu_i^*$  to  $P_{(i \bmod K)+1}$  ( $\forall i \in [1, K]$ );
8 else
9   | the K-LP problem is infeasible and return;
10 forall party  $P_i, i \in [1, K]$  do
11   | solves the pricing problem, tests the optimal improvement and sends a
   | new column (extreme point/ray) to  $P_1$  if necessary (Algorithm 5, the
   | optimal value  $z_i^*$  is also sent to  $P_1$  if near-optimal test is specified);
12 if a new column is proposed to  $P_1$  then
13   | go to Step 5;
14 else
   | /* the optimal solution of the current RMP is
   |    the optimal solution of the master problem
   |    */
15    $P_1$  sends the share of the optimal solution  $\forall i \in [1, K], \lambda_i$  to party  $P_i$ ;
16   forall party  $P_i, i \in \{1, 2, \dots, K\}$  do
17     | derive its solution  $x_i$  (global optimum) by  $\lambda_i$  (as shown in Figure
     | 4.2);
   /* apply near-optimal solution test if necessary
   */

```

Preliminary Steps:

- Each party $P_i (i=1\dots K)$ locally hides b_i, b_0^i , transforms A_i', B_i', c_i' with its privately held monomial matrix Q_i and sends A_i', B_i', c_i', b_i' to the next party P_{i+1} (P_K sends A_K', B_K', c_K', b_K' to P_1)
- b_0' is securely summed or directly summed by all K parties

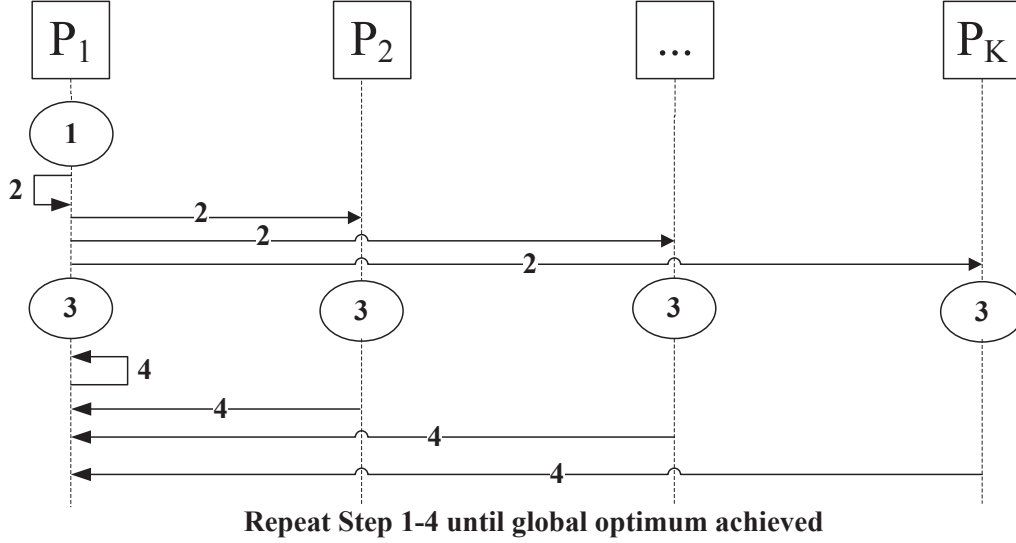


Figure 4.3. Secure K-party Column Generation Protocol

proposals only slightly better than the ones before. Thus, we might have to stop with a near-optimal solution for efficiency reasons if necessary [122]. 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. 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 4.3 *The K-party Column Generation Protocol is secure in Semi-honest*

model.

Proof. As proven in Lemma 4.1 and 4.2, solving RMPs and pricing problems is secure for all K honest-but-curious parties. Since our K -party Column Generation Protocol involves repeated steps of solving transformed RMPs and pricing problems, it is straightforward to show that the protocol is secure against semi-honest adversaries.

Computation and Communication Cost Analysis

Our secure column generation protocol is mainly based on local transformation rather than cryptographic tools that dominates the cost in most of the privacy-preserving collaborative linear programming techniques [32, 74, 124, 125]. 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 locally hiding 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 party P_i to transform A'_i, B'_i and c'_i since hiding b only incurs nominal computation 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 (e.g., simplex or revised simplex method) [97, 122]. In summary, our secure K -party 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 party transforms its shares of the K-LP problem, $i \in [1, K]$, P_i sends $A'_i Q_i$, $B'_i Q_i$, $c'^T Q_i$, $(b_0^i)'$ and b'_i to another party, 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 party (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.

4.2.3 Experiments

In this section, we present the experimental results to evaluate the performance of the SCG protocol using both synthetic data and real data. Specifically, we study: (1) the computational performance of the SCG protocol compared with Secure Transformation (ST) [124], Secure Revised Simplex Method (SRS) [125] and Secure Simplex Method (SS) [74]; (2) the scalability of the SCG 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).

Experimental Setup

Datasets. Our experiments are conducted on both synthetic datasets and real-world dataset (MovieLens dataset ⁸). 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 $c = [c_1 \ c_2 \ \dots \ c_K]$ is randomly selected in the range $[10, 20]$ and the righthand side values b_0, b_1, \dots, b_K are randomly generated from the range $[100,500]$. All the constraints are inequalities with operational symbol “ \leq ”, 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 71567 users’ ratings for 10681 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 righthand 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 collaborative LP methods. In that experimental setting, we assume that the K-LP problems are jointly solved by

⁸<http://www.grouplens.org>

two parties, because only two-party collaborative LP problems can be securely solved by Secure Transformation (ST) [124], Secure Revised Simplex Method (SRS) [125] and Secure Simplex Method (SS) [74]. More specifically, we test the computation costs of our SCG protocol and the above three methods on a group of two-party LP problems with varying problem size: $15 \times 20, 30 \times 40, 45 \times 60, 60 \times 80, 75 \times 100$ (we choose this groups of medium-scale LP problems because some methods cannot scale well to large-scale problems), where each party 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 party 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 party P_i 's movies as the elements in the block of local constraints B_i . Table 4.1 describes the detail of the collaborative LP problems for comparing four protocols.

Groups of Experiments for Evaluating the SCG 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

Table 4.1. Experimental Setting for Protocol Comparison (Two Parties)

	Synthetic and Real Data Setting		
LP Size	Each party's # of Variables (Distinct Movies)	# of Global Const. (Users with Overall Ratings)	Each party's # of Local Const. (Users with ratings on Each party'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

groups of collaborative LP problems as shown in Table 4.2 and 4.3.

Table 4.2. K-LP Problems with Varying # of Parties (Problem Size: 500×1000)

	Synthetic and Real Data Setting		
# of Parties	Each party's # of Variables (Distinct Movies)	# of Global Const. (Users with Overall Ratings)	Each party's # of Local Const. (Users with ratings on Each party's Movies)
2	500	100	200
5	100	100	80
10	50	100	40
20	25	100	20
50	10	100	8

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

1. fixing the LP problem size as 500×1000 and letting all parties have the same number of variables, we test the computation cost of the SCG protocol on varying number of parties: 2, 5, 10, 20, 50 (see the detail of the K-LP problems in Table 4.2);
2. fixing the number of parties as 20 and letting all parties have the same num-

Table 4.3. K-LP Problems with Varying Size (20 Parties)

LP Size	Synthetic and Real Data Setting		
	Each party's # of Variables (Distinct Movies)	# of Global Const. (Users with Overall Ratings)	Each party's # of Local Const. (Users with ratings on Each party'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

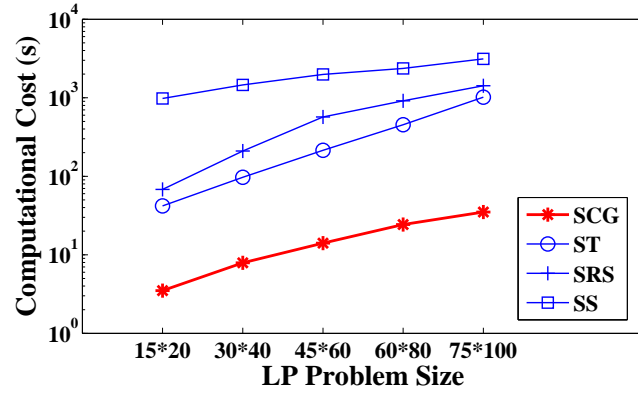
ber of variables, we test the computation cost of the SCG protocol on varying LP size: 100×200 , 200×400 , 300×600 , 400×800 , 500×1000 (see the detail of the K-LP problems in Table 4.3).

Note that the near-optimal tolerance parameter for column generation is given as 10^{-6} .

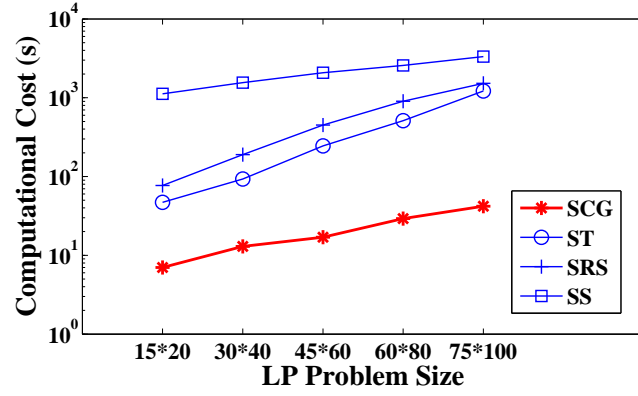
Platform. The algorithms were implemented in Matlab, and all the experiments were carried on an HP machine with Intel Core 2 Duo CPU 3GHz and 3G RAM.

Secure K-party Column Generation (SCG) Protocol

Figure 4.4 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 different size of the LP problems, the SCG protocol requires significantly less runtime to solve the two-party collaborative LP problems formulated by both synthetic data (Figure 4.4(a)) and real data (Figure 4.4(b)).



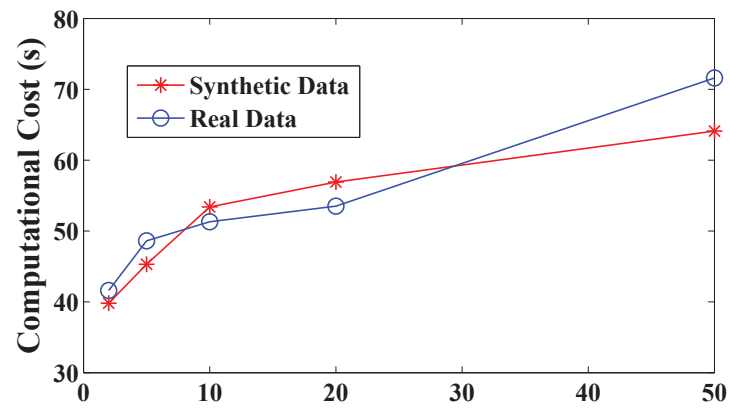
(a) Synthetic Data



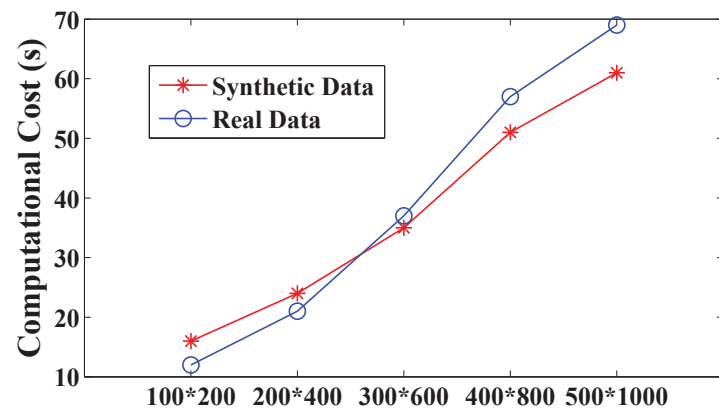
(b) Real Data

Figure 4.4. Computational Performance Comparison of Secure Collaborative LP Protocols (Logarithmic Scale)

Figure 4.5(a) illustrates the scalability of the SCG protocol on varying number of parties (fixed LP problem) and varying LP problem size (fixed number of parties). 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 (Figure 4.5(a) and 4.5(b)).



(a) Varying Number of Parties



(b) Varying LP Problem Size

Figure 4.5. Computational Performance of the SCG Protocol

CHAPTER 5

MALICIOUS COLLABORATIVE LINEAR PROGRAMMING (LP)

Chapter 4 introduced the secure K-party column generation (SCG) protocol which protects each party's private information against honest-but-curious parties. However, in reality, parties involved in the collaborative LP problem cannot be guaranteed to follow the protocol. For instance, a selfish party may attempt to modify the protocol (e.g., sending fake messages to other parties) for increasing its payoff and sacrificing other parties' payoffs. In this Chapter, we introduce an incentive compatible protocol to securely solve the K-LP problems against malicious parties, and also demonstrate the efficiency (computational performance) and scalability of the protocol with experimental results.

5.1 Malicious Behavior in the SCG Protocol

While executing the secure K-party 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 Figure 4.3. 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 (w.l.o.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. 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 parties – even though the fake dual values may deviate the objective function of the pricing problems, any generated fake dual values cannot guarantee other parties' payoff decrease or increase (simply because 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 category 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 would lead to the same possible outcome –

the proposed optimal solutions are deviated).

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

- First, ℓ parties (either RMP solver or pricing problem solvers and $\ell < K$) can collude to infer additional information. For example, parties 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) parties may realign a dishonest coalition to corrupt the SCG protocol by sending fake solutions/columns to other parties. The probability of detecting cheating from the coalition might be reduced since all the cheating implemented by the parties in the dishonest coalition are known to them: all the parties in the dishonest coalition can send the identical fake messages to the recipients, thus it is even difficult to detect such cheating by the parties outside the dishonest coalition.

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

5.2 Nash Equilibrium in the SCG Protocol

The essence of establishing an incentive compatible protocol is to force all rational participants to follow the protocol based on a Nash Equilibrium – “all participants play honestly”. In the Nash Equilibrium, no participant can gain more payoff by changing its strategy from “honest play” to “cheat”.

5.2.1 Dominant Strategy for Nash Equilibrium

Assuming that any party P_i ($i \in [1, K]$) might be the malicious party, 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 parties’ payoffs have been gained by such party, 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 parties P_i ($i \in [1, K]$), strategy “honest play” *strictly dominates* “cheat” in the protocol (see Equation 5.1), “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 [40]).

$$T_i > \epsilon * c(T_i) + (1 - \epsilon) * u(T_i) \quad (5.1)$$

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

$$\forall i \in [1, K], \epsilon > \frac{u(T_i) - T_i}{u(T_i) - c(T_i)} \quad (5.2)$$

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

5.2.2 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 parties in every time (see Example 5.1).

Example 5.1 *Four parties 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 parties 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 parties 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 parties 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 parties 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).

In the remaining part of this chapter, we will present our approach to prevent the malicious behavior by establishing the Nash Equilibrium of “honest play” for all participants, assuming that the largest dishonest coalition includes no more than ℓ malicious parties ($\ell < K$).

5.3 Incentive Compatible Protocol

Two categories of collusion have been discussed in Section 5.1: colluding to infer additional private information, and cheating as a dishonest coalition. We first present a multiparty based transformation to resolve the first collusion issue, and then propose the incentive compatible protocol against malicious participants if the largest dishonest coalition include at most ℓ colluding parties.

5.3.1 Multiparty Transformation

Before running the SCG protocol, each party $P_i (i \in [1, K])$ locally transforms their shares in the K-LP problem by hiding 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 $c_i'^T$. Thus, if another party (e.g., $P_{(i \bmod K)+1}$) solves P_i 's pricing problems, any element in A'_i , B'_i and $c_i'^T$ cannot be inferred with known $A'_i Q_i$, $B'_i Q_i$ and $c_i'^T Q_i$. 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:

$$A'_i = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n'_i} \\ a_{21} & a_{22} & \dots & a_{2n'_i} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m_0 1} & a_{m_0 2} & \dots & a_{m_0 n'_i} \end{pmatrix} \in \mathbb{R}^{m_0 \times n'_i} \quad (5.3)$$

$$Q_i = \begin{pmatrix} 0 & 0 & \dots & 0 & r_{n'_i} \\ r_1 & 0 & \dots & 0 & 0 \\ 0 & r_2 & \dots & 0 & 0 \\ 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \dots & r_{n'_i-1} & 0 \end{pmatrix} \in \mathbb{R}^{n'_i \times n'_i} \quad (5.4)$$

$$A'_i Q_i = \begin{pmatrix} a_{12}r_1 & a_{13}r_2 & \dots & a_{11}r_{n'_i} \\ a_{22}r_1 & a_{23}r_2 & \dots & a_{21}r_{n'_i} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m_0 2}r_1 & a_{m_0 3}r_2 & \dots & a_{m_0 1}r_{n'_i} \end{pmatrix} \quad (5.5)$$

Thus, post-multiplying Q_i can be considered as two steps – 1) permute the columns of matrix A'_i (or B'_i and $c_i'^T$) according to the positions of the non-zero elements in Q_i : the 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'^T (Bednarz's attack [11] 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., the proposed optimal solutions/columns and the dual values) since P_i knows Q_i . Even though this additional private information does not directly hurt other parties in the protocol, 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 mitigate the incentive of sending the fake payoff-increased columns to the RMP solver.

To resolve this collusion, we can extend the matrix post-multiplication based transformation in Chapter 4 to multiparty transformation by integrating both 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 parties* 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 Equation 5.4 and 5.5), pre-multiplying a monomial matrix to A'_i (or B'_i) can be also considered as two steps [38]: 1) per-

mutate 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:

$$A'_i = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n'_i} \\ a_{21} & a_{22} & \dots & a_{2n'_i} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m_0 1} & a_{m_0 2} & \dots & a_{m_0 n'_i} \end{pmatrix} \in \mathbb{R}^{m_0 \times n'_i} \quad (5.6)$$

$$Q'_i = \begin{pmatrix} 0 & 0 & \dots & 0 & r'_{m_0} \\ r'_1 & 0 & \dots & 0 & 0 \\ 0 & r'_2 & \dots & 0 & 0 \\ 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \dots & r'_{m_0-1} & 0 \end{pmatrix} \in \mathbb{R}^{m_0 \times m_0} \quad (5.7)$$

$$Q'_i A'_i = \begin{pmatrix} a_{m_0 1} r'_{m_0} & a_{m_0 2} r'_{m_0} & \dots & a_{m_0 n'_i} r'_{m_0} \\ a_{11} r'_1 & a_{12} r'_1 & \dots & a_{1 n'_i} r'_1 \\ \vdots & \vdots & \ddots & \vdots \\ a_{(m_0-1)1} r'_{m_0-1} & a_{(m_0-1)2} r'_{m_0-1} & \dots & a_{(m_0-1) n'_i} r'_{m_0-1} \end{pmatrix} \quad (5.8)$$

Therefore, if we involve all parties in transforming party $P_i (i \in [1, K])$'s share, $A'_i, B'_i, c_i'^T, (b_0^i)'$ and b'_i can be transformed by all parties $\forall j \in [1, K], P_j$:⁹

$$\begin{aligned} \forall i \in [1, K], A'_i &\implies \prod_{j=1}^K Q_j A'_i \prod_{j=1}^K Q_{ij}, & (b_0^i)' &\implies \prod_{j=1}^K Q_j (b_0^i)' \\ \forall i \in [1, K], B'_i &\implies \prod_{j=1}^K Q'_{ij} B'_i \prod_{j=1}^K Q_{ij}, & b'_i &\implies \prod_{j=1}^K Q'_{ij} b'_i \\ \forall i \in [1, K], c_i'^T &\implies c_i'^T \prod_{j=1}^K Q_{ij} \end{aligned} \quad (5.9)$$

⁹Each party $P_j (j \in [1, K])$ 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

Note that $(b_0^i)'$ and b_i' should be implemented with the same pre-multiplication as A_i' and B_i' respectively since the righthand side values of the constraints should be consistent with the corresponding rows in the constraints matrix in permutation. Moreover, first, since every party 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 party $P_j (j \in [1, K])$ 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 party P_i 's B_i' in its local constraints has different number of rows m_i' , we should let each party $P_j (j \in [1, K])$ 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 whereas B_i' is the constraints matrix of P_i 's local constraints, thus the row permutation should be separated.

Why we need pre-multiplication and post-multiplication involving all parties while transforming A_i' and B_i' to tackle Collusion. If only multiparty post-multiplication is applied (as shown 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 $\{A_i' \prod_{j=1}^K Q_{ij}, B_i' \prod_{j=1}^K Q_{ij}, c_i^T \prod_{j=1}^K Q_{ij}\}$ (if only multiparty 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 an unknown coefficient to every column, P_i can still learn the multiparty column 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 $c_i'^T$ and $c_i'^T \prod_{j=1}^K Q_{ij}$ since every column is a single number rather than a vector in them, the relation between the original number and the corresponding number in $c_i'^T \prod_{j=1}^K Q_{ij}$ cannot be inferred). Thus, besides multiparty post-multiplication, we have to let all parties permute the rows in the transformed constraints and multiply a random number to every row as well (doing this via pre-multiplication).

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

$$\begin{aligned}
 & \max \quad \sum_{i=1}^K (c_i'^T \prod_{j=1}^K Q_{ij}) y_i \\
 \text{s.t.} \quad & \begin{cases} y_1 \in \mathbb{R}^{n'_1} \\ y_2 \in \mathbb{R}^{n'_2} \\ \vdots \\ y_K \in \mathbb{R}^{n'_K} \end{cases} \begin{pmatrix} (\prod_{j=1}^K Q_j) A'_1 (\prod_{j=1}^K Q_{1j}) & \cdots & (\prod_{j=1}^K Q_j) A'_K (\prod_{j=1}^K Q_{Kj}) \\ (\prod_{j=1}^K Q'_{1j}) B'_1 (\prod_{j=1}^K Q_{1j}) & & \\ & \ddots & \\ & & (\prod_{j=1}^K Q'_{Kj}) B'_K (\prod_{j=1}^K Q_{Kj}) \end{pmatrix} \\
 & \times \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_K \end{pmatrix} \begin{matrix} \bowtie_0 \\ \bowtie_1 \\ \vdots \\ \bowtie_K \end{matrix} \begin{pmatrix} (\prod_{j=1}^K Q_j) b'_0 \\ (\prod_{j=1}^K Q'_{1j}) b'_1 \\ \vdots \\ (\prod_{j=1}^K Q'_{Kj}) b'_K \end{pmatrix} \end{cases} \quad (5.10)
 \end{aligned}$$

The detailed steps of multiparty transformation are presented in Algorithm 7.

Theorem 5.1 *Multiparty transformation does not reveal additional private information against arbitrary number of colluding parties.*

Algorithm 7: Multiparty Transformation

```

1 forall party  $P_i, i \in [1, K]$  do
2   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  $(b'_0)'$ ) where non-zero
   elements are random numbers;
3 forall party  $P_i, i \in [1, K]$  do
4   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;
5   transforms  $A \leftarrow Q_i A'_i Q_{ii}, B \leftarrow Q'_{ii} B'_i Q_{ii}, c^T \leftarrow c'^T_i Q_{ii}, b_A \leftarrow$ 
    $Q_i (b'_0)', b_B \leftarrow Q'_{ii} (b'_i)'$ ;
6   sends  $A, B, c^T, b_A$  and  $b_B$  to the next party  $P_j, j \in [1, K], j \neq i$ ;
7   forall party  $P_j, j \in [1, K], j \neq i$  do
8     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;
9     transforms  $A \leftarrow Q_j A Q_{ij}, B \leftarrow Q'_{ij} B Q_{ij}, c^T \leftarrow c^T Q_{ij}, b_A \leftarrow$ 
      $Q_j b_A, b_B \leftarrow Q'_{ij} b_B$ ;
10    sends  $A, B, c^T, b_A$  and  $b_B$  to the next party;
11 output the transformed shares of the K-LP problem;

```

Proof. We first look at the correctness of the multiparty transformation. Since the product of any two permutation matrices remains a permutation matrix [38], 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'^T_i 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 parties). Moreover, 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 b'_i do not change feasible region and the optimal solution [90] – 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 party to party generates correct transformation. Finally, each party $P_i (i \in [1, K])$'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 multiparty transformed problem (inverse row permutation/pre-multiplication to the optimal solution is not necessary [90]).

Clearly, in case that ℓ parties collude where $\ell < K$, since for every party 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 parties, it is impossible to compute the optimal solution/proposed columns of any of P_i 's pricing problems without involving all parties (even if the colluding parties know A'_i , B'_i and $c'_i{}^T$). From the perspective of permutation and multiplication, for any party $P_i (i \in [1, K])$'s input matrices/vector $\{A'_i, B'_i \text{ and } c'_i\}$, every party 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 apply similar transformation 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 parties, and without the participation of any party, the true column and row permutation indices and the multiplied random numbers cannot be reconstructed.

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

5.3.2 Achieving Incentive Compatibility

A naive approach to establish Nash Equilibrium has been introduced in Section 5.2.2 – 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 parties, and 2) ℓ (less than K) colluding parties.

1) No Colluding Malicious Parties. Assuming that the SCG protocol runs N times, we then describe the detection of all the malicious behavior (except collusion) listed in Section 5.1 (e.g., the RMP solver modifies the final optimal solution or the dual values in every iteration of the SCG protocol, and the pricing problem solver proposes fake local optimal solutions/columns to the RMP solver). If all the parties follow the SCG protocol, the optimal solutions derived from N times SCG protocol run with different RMP solvers should be identical¹⁰.

- In any N runs of the SCG protocol, suppose that the RMP solver modifies the final optimal solution or the dual values in an 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. 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 N runs of the SCG protocol, any pricing problem solver

¹⁰If the LP problem has multiple optimal solutions, all the optimal solutions should be given as the output, and they should be identical as well.

may propose wrong local optimal solutions/columns to the RMP solver – this distorts the optimal solution. At this time, the pricing problem owner can detect such modification if the modification leads to deviated output, since the pricing problem owner also receives the optimal solutions from parties 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 parties.

2) Against ℓ Colluding Malicious Parties. We now extend the discussion to a more general form: assuming at most ℓ (less than K) malicious parties cheat together as a dishonest coalition, and the number of dishonest coalitions may exceed one¹¹. In this case, all the parties inside the same dishonest coalition can behave the identical malicious action in the multiple SCG protocol run (e.g., send the same wrong message to the corresponding party), and the cheating from the same dishonest coalition cannot be explicitly caught by comparing the results from different parties.

We still employ the 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 parties. 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 parties

¹¹We assume that one party cannot collude with parties 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 party as a larger dishonest coalition)

and also pick N *different pricing problem solvers* from K parties for solving each party's pricing problems (clearly, $N \leq K$). Note that we allow each party solving its own transformed pricing problems: the pricing problem owner (collude with less than K parties) cannot learn additional information while solving them with multiparty transformation (Theorem 5.1). Otherwise, the malicious behavior cannot be eliminated in the worst case that $\ell = K - 1$ – if the only party 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 party are solved by the parties from the dishonest coalition, no matter how many times of running the SCG protocol with permuted solvers.

Essentially, for every party $P_i (i \in [1, K])$ in the K-LP problem, if P_i 's N groups of pricing problems are solved by at least two non-colluding parties in the N times SCG protocol run, and N groups of RMPs are also solved by at least two non-colluding parties, the cheating with regard to P_i 's share in the K-LP problem should be caught. Note that the caught cheating here covers all kinds of message modification while solving the problem. Therefore, while uniformly picking N distinct RMP solvers out of K parties and N distinct pricing problem solvers out of K parties for each party's pricing problems in N SCG protocol run, we can utilize either of the following two criteria to detect cheating:

1. $N \geq \ell + 1$;
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 party'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 criteria is satisfied, modifying any party's share in the K-LP problem will be caught since every party's share in the global optimal solution are generated by at least two non-colluding parties in the N SCG protocol run. Thus, for maintaining efficiency, our incentive compatible protocol should seek the *minimum N satisfying either of the two criteria*.

◊. Clearly, if *Criterion 1* holds, then every party's share in the $N \geq \ell + 1$ global optimal solutions should be generated by at least two parties since the largest dishonest coalition includes only ℓ participants, and any cheating will be definitely caught.

◊. We now look at the probability of satisfying *Criterion 2* if random solvers are picked. Specifically, if picking N distinct RMP solvers out of K parties ($N \leq K$), the probability of picking at least one RMP solver outside and at least one RMP solver inside the largest dishonest coalition (ℓ malicious parties collude to modify the protocol) is equal to:

$$Prob_R = 1 - \prod_{i=0}^{N-1} \frac{\ell - i}{K - i} - \prod_{i=0}^{N-1} \frac{(K - \ell) - i}{K - i} \quad (5.11)$$

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

$$Prob_P = 1 - \prod_{i=0}^{N-1} \frac{\ell - i}{K - i} - \prod_{i=0}^{N-1} \frac{(K - \ell) - i}{K - i} \quad (5.12)$$

Note that if $N > \max\{K - \ell, \ell\}$, we have $Prob_R = Prob_P = 1$ in Equation 5.11 and 5.12 (such N also satisfies Criterion 1: $N \geq \ell + 1$). Since we are seeking the least N for the incentive compatible protocol, we only need to consider the case $N \leq \max\{K - \ell, \ell\}$ for Criterion 2.

Since the probability of catching the malicious behavior with Criterion 2 is $Prob_R * Prob_P$ (picking RMP and pricing problem solvers are independent), we consider the probability of detecting cheating as $\epsilon = Prob_R * Prob_P$ for all parties. In addition, if the probability of detecting cheating for every party $P_i (i \in [1, K])$ satisfies $\epsilon > \frac{u(T_i) - T_i}{u(T_i) - c(T_i)}$ (Equation 5.2) 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 parties. As a result, if the following K inequalities hold, the protocol should be incentive compatible.

$$\forall i \in [1, K], [1 - \prod_{i=0}^{N-1} \frac{\ell - i}{K - i} - \prod_{i=0}^{N-1} \frac{(K - \ell) - i}{K - i}]^2 > \frac{u(T_i) - T_i}{u(T_i) - c(T_i)} \quad (5.13)$$

Since every party'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 parties $\max\{\forall i \in [1, K], \frac{u(T_i) - T_i}{u(T_i) - c(T_i)}\}$ 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 Criterion 2, we obtain N as the least integer to satisfy:

$$\prod_{i=0}^{N-1} \frac{\ell - i}{K - i} + \prod_{i=0}^{N-1} \frac{(K - \ell) - i}{K - i} < 1 - \sqrt{\max\{\forall i \in [1, K], \frac{u(T_i) - T_i}{u(T_i) - c(T_i)}\}} \quad (5.14)$$

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 summary, we can combine two criteria together, and *compute the smallest N to satisfy either of them*. Indeed, Criterion 1 is more effective to catch cheating for smaller ℓ and Criterion 2 is more effective to catch cheating for larger ℓ .

Theorem 5.2 *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 Equation 5.14) is incentive compatible against malicious participants if the largest dishonest coalition include at most ℓ colluding parties.*

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 parties, if $N > \ell$ or N satisfies the inequality in Equation 5.14, for any party P_i ($i \in [1, K]$)'s pricing problems (P_i can be a party 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 parties 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 party $P_j, j \neq i$ (or more parties). Hence, the probability of generating P_i 's share in N global optimal solutions by at least one party which does not collude with P_j (and its colluding parties) 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 parties have a strictly dominant strategy of “honest play”, though at most ℓ parties can collude to form a dishonest coalition. According to the Nash Equilibrium, the incentive compatibility is proved.

Algorithm 8: Incentive Compatible Protocol

```

/* Each party  $P_i$ 's share  $(b_0^i$  and  $b_i)$  in the righthand
   side values have been hidden into  $(b_0^i)'$  and  $b_i'$ 
   respectively. */
1 Transform each party  $P_i$ 's share in the K-LP problems  $A'_i, B'_i, c'^T, (b_0^i)'$ 
   and  $b_i'$  using Algorithm 7 ( $i \in [1, K]$ );
2 Solve the transformed K-LP problem with the SCG protocol (Algorithm 6)
   for  $N$  times where all the  $N$  RMP solvers and each party's  $N$  transformed
   pricing problem solvers are uniformly picked from parties all parties
    $P_1, P_2, \dots, P_K$  ( $N$  is given as the smaller number of  $\ell + 1$  and the least
   integer in Equation 5.14);
3 Distribute  $K$  shares of the  $N$  global optimal solutions to the corresponding
   pricing problem owners;
4 Compute each party  $P_i$ 's share in the original optimal solution by (K
   parties) 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 party's share in  $N$  optimal
   solutions is found;

```

5.3.3 Protocol and Analysis

For any K-LP problem, all parties 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 Equation

(5.14). The incentive compatible protocol can be given as N times SCG protocol run with randomly picked distinct RMP and pricing problem solvers (Algorithm 8).

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

5.4 Experiments

In this section, we present the experimental results to evaluate the performance of the incentive compatible protocol using the same experimental setup as the semi-honest model (Section 4.2.3).

5.4.1 Groups of Experiments

While evaluating the scalability of the incentive compatible protocol with regard to the computational performance, we use the same mechanism as Section 4.2.3 to formulate the K-LP problems with the synthetic and real data. Specifically, with two groups of collaborative LP problems as shown in Table 4.2 and 4.3, we conduct four groups of experiments to test the scalability of the incentive compatible protocol:

1. fixing the LP problem (500×1000) and the largest dishonest coalition includes at most $1/2$ of the total parties, test the computation cost of the in-

centive compatible protocol on varying *number of parties*: 2, 5, 10, 20, 50 (see the detail of the K-LP problems in Table 4.2);

2. fixing the number of parties as 20 (10 of them collude), test the computation cost of the incentive compatible protocol on the varying *LP problem 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 4.3);
3. fixing the LP problem (500×1000) and the number of parties as 20, test the computation cost of the incentive compatible protocol on the *number of colluding parties 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. $u(T_i) = 100, T_i = 50, c(T_i) = 0$).

4. fixing the LP problem (500×1000), number of parties as 20 (10 of them collude), we also test the computation 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.

Identically, the near-optimal tolerance parameter for column generation is given as 10^{-6} .

5.4.2 Performance

Similar to the evaluation on the SCG protocol in Chapter 4, Figure 5.1(a) and 5.1(b) illustrate the computational performance of the incentive compatible protocol on varying number of parties (fixed LP problem, percentage of the colluding parties and the maximum payoff ratio), and varying LP problem size (fixed total number of parties, percentage of the colluding parties and the maximum payoff ratio). The incentive protocol also scale well to solve collaborative LP problems with large size or number of parties.

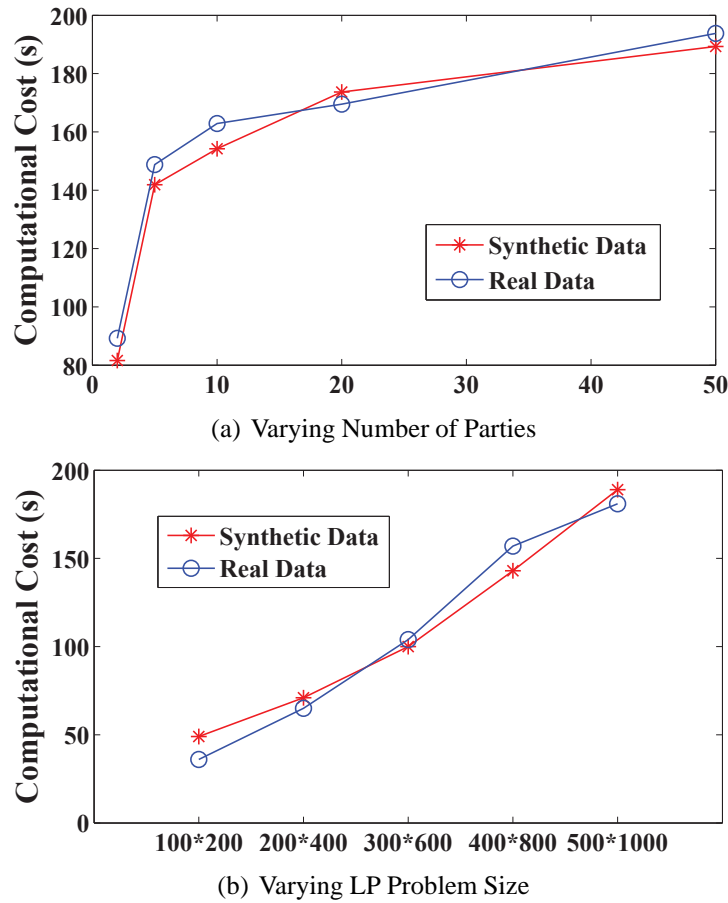


Figure 5.1. Computational Performance I of the Incentive Compatible Protocol

Moreover, we capture the minimum number of SCG protocol run required in

Table 5.1. N in the Incentive Compatible Protocol

Exp. Groups	Exp. Results	ℓ	Minimum Number of the SCG Protocol Run
1)	Figure 5.1(a)	$\ell = 1, 2, 5, 10, 25$	$N = 2, 3, 3, 3, 3$
2)	Figure 5.1(b)	all $\ell = 10$	$N = 3, 3, 3, 3, 3$
3)	Figure 5.2(a)	$\ell = 2, 4, 6, \dots, 16, 18$	$N = 3, 5, 4, 3, 3, 3, 4, 5, 9$
4)	Figure 5.2(b)	all $\ell = 10$	$N = 2, 2, 2, 2, 2, 3, 3, 4, 4$

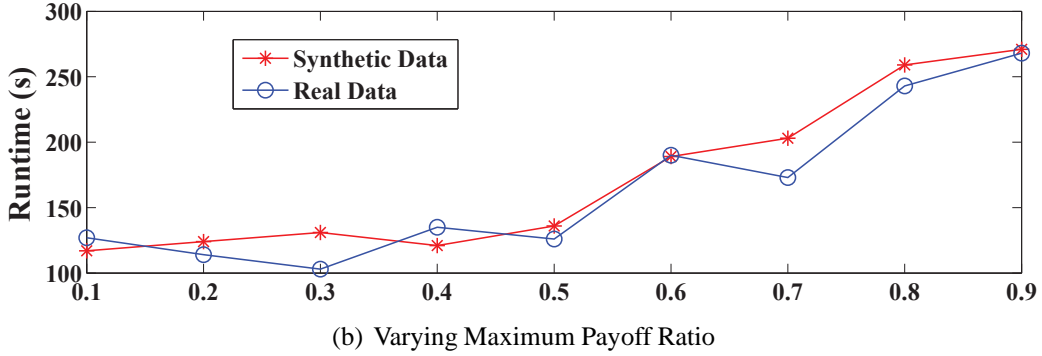
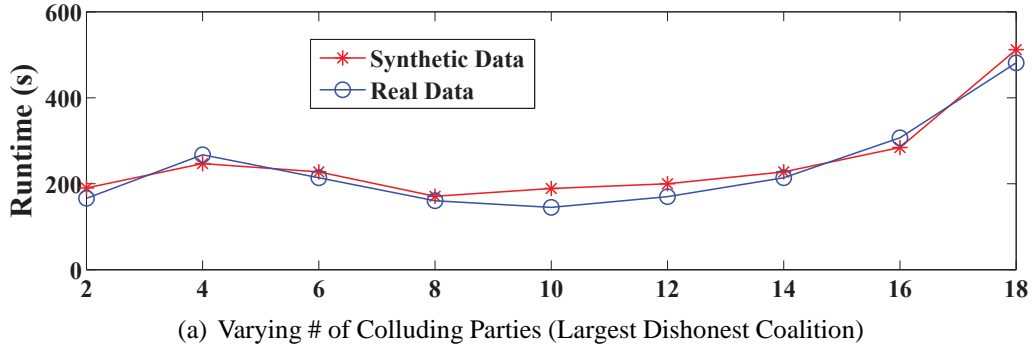


Figure 5.2. Computational Performance II of the Incentive Compatible Protocol

every group of experiments (see Table 5.1). 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 parties 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 test the computation cost of the incen-

tive compatible protocol tuning on parameter – the number of colluding parties in the largest dishonest coalition (fixed LP problem, total number of parties and the maximum payoff ratio). As shown in the second row of Table 5.1, for $\ell = 2$ and 4, the incentive compatible protocol runs $N = \ell + 1 = 3$ and 5 times SCG protocol respectively (Criterion 1); for $\ell > 4$, N turns to the value computed from Criterion 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 parties $K/2$, 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 Criterion 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 practical computation cost shown in Figure 5.2(a) demonstrates a similar varying trend as the 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 parties and percentage of the colluding parties). We can see that if the maximum payoff ratio increases (perhaps the payoff of the uncaught cheating $u(T_i)$ or the payoff of the caught cheating $c(T_i)$ in any party $P_i(i \in [1, K])$'s payoff ratio increases), N should be enlarged to maintain incentive compatibility. We can observe this in Figure 5.2(b) and the fourth row of Table 5.1.

CHAPTER 6

GOING BEYOND LINEAR PROGRAMMING

Apart from collaborative linear programming, sharing data in many other collaborative optimization problems is unacceptable as well. In this chapter, we address the security and privacy issues for optimization models beyond linear programming. Specifically, privacy-preserving techniques for non-linear programming (NLP), graph coloring and traveling salesman problem are given in the following three sections respectively.

6.1 Non-linear Programming

Discussed in Chapter 3, non-linear programming (NLP) problem covers a wide variety of problem formulations, widely applicable to real-world problems in engineering, transportation, machine learning, etc. We particularly look at the NLP problem (Equation 3.5) which is horizontally partitioned amongst several parties to solve a real world problem in [58]. To reformulate Equation 3.5 among r parties with horizontal constraints partition, on one hand, every party $P_k (1 \leq k \leq r)$ holds a set of private linear inequality constraints. On the other hand, all parties know: the non-linear objective function, variables, and the public constraint $\sum_{i=1}^n x_i = d$. Table 6.1 gives the partitioned shares of the NLP problem among r parties, where $1 \leq i \leq n, 1 \leq j \leq m_k, 1 \leq k \leq r$ (we give a more general form

of such problem and ignore the public constraint $\sum_{i=1}^n x_i = d$).

Table 6.1. Partitioned NLP Problem

All r Parties Know	max : $\sum_{i=1}^n [c_i \log(x_i + 1)]$
P_1 Holds	$\forall j \in [1, m_1], \sum_{i=1}^n t_{ij1} x_i \leq b_{j1}$
P_2 Holds	$\forall j \in [1, m_2], \sum_{i=1}^n t_{ij2} x_i \leq b_{j2}$
\vdots	\vdots
P_r Holds	$\forall j \in [1, m_r], \sum_{i=1}^n t_{ijr} x_i \leq b_{jr}$

Following the linear approximation introduced in Section 3.1.2, although we approximate the (separable) objective function by projecting the variables $x = (x_1, \dots, x_n)$ to $\{1 \leq i \leq n, 1 \leq s \leq K, w_s^i \in [0, 1]\}$, the partition scenario of the projected constraints in the approximated LP problem remains identical to that of the original NLP problem.

6.1.1 Secure Transformation and Permutation

Mangasarian's transformation approach [90] pre-multiplies a random matrix to both sides of the horizontally partitioned constraints, where the transformed linear constraints are equivalent to the original ones. However, the privacy leakage cannot be quantified in their transformation approach – additional information might be learnt by adversaries [53]. Instead, we present a secure transformation approach to solve the NLP problem (indeed an LP problem after approximation) among r different parties with quantified information disclosure under SMC definition.

First, we let each party $P_k (1 \leq k \leq r)$ standardize its private linear *inequality constraints* into linear *equality constraints*: for party P_k 's j th inequality con-

straint, $\sum_{i=1}^n t_{ijk}x_i \leq b_{jk}$, we add a slack variable y_{jk} with a positive random coefficient ¹² f_{jk} :

$$s.t. \begin{cases} \forall j \in [1, m_1], \sum_{i=1}^n t_{ij1}x_i + f_{j1}y_{j1} = b_{j1} \\ \forall j \in [1, m_2], \sum_{i=1}^n t_{ij2}x_i + f_{j2}y_{j2} = b_{j2} \\ \vdots \\ \forall j \in [1, m_r], \sum_{i=1}^n t_{ijr}x_i + f_{jr}y_{jr} = b_{jr} \end{cases} \quad (6.1)$$

We denote the standard form of all the linear constraints as $Tx + Fy = b$ where the constraint matrix $T = \begin{bmatrix} T_1 \\ \vdots \\ T_r \end{bmatrix}$ and T, T_1, \dots, T_r are $m \times n, m_1 \times n, \dots, m_r \times n$ matrices and diagonal matrix $F = \text{diag}[F_1, F_2, \dots, F_r]$, where F_1, \dots, F_r are diagonal matrices with positive entries $\{f_{11}, \dots, f_{m_11}\}, \dots, \{f_{1r}, \dots, f_{m_rr}\}$ in the diagonals respectively.

Second, we extend the transformation in [75, 90] to a secured version (enabled by Homomorphic Encryption [101]) and implement an index permutation to the variables $x = (x_1, \dots, x_n)$.

Specifically, Mangasarian [90] has shown that pre-multiplying an $\ell \times m$ random matrix A to both side of the linear constraints can guarantee correctness and security for horizontally partitioned linear constraints: $Tx + Fy = b \iff ATx + AFy = Ab$, where A is vertically partitioned as $A = [A_1 \ A_2 \ \dots \ A_r]$ and A_1, \dots, A_r are random matrices generated by parties P_1, \dots, P_r respectively (b is horizontally partitioned as $b = \begin{bmatrix} B_1 \\ \vdots \\ B_r \end{bmatrix}$, and $1 \leq k \leq r, B_k$ is a vector with length m_k).

¹²Using coefficient f_{jk} for slack variable y_{jk} is equivalent as simply adding the slack variable with coefficient 1, and f_{jk} can prevent learning the transformation matrix from the horizontally partitioned linear constraints [75].

In our secure transformation, we follow the idea of $Tx + Fy = b \iff ATx + AFy = Ab$ and utilize Homomorphic Cryptosystem [101] to securely pre-multiply random matrix to the share of the constraint matrices. One primary advantage of this process is that we can let every party $P_k (1 \leq k \leq r)$ permute the columns of the transformed constraint matrix AT (via permutating the ciphertexts) with its own permutation (or considered as permutating the variables of the optimization problem). Thus no party can learn the true matrix column/variables index and the transformation for other parties' share. See the overall transformation as below:

$$\begin{array}{ll}
 P_1 : & A_1 T_1 x + A_1 F_1 y = A_1 B_1 \\
 \vdots & \vdots \\
 P_r : & A_r T_r x + A_r F_r y = A_r B_r \\
 \implies & \sum_{k=1}^r A_k (T_k x + F_k y) = \sum_{k=1}^r A_k B_k
 \end{array} \tag{6.2}$$

where the feasible region of the linear constraints remains invariable in this linear constraints transformation. Note that the $\ell \times n$ matrix $\sum_{k=1}^r A_k T_k$ leaks even less information after implementing the column/variable index permutation.

6.1.2 Algorithm

The detailed steps of the collaborative NLP problem are presented in Algorithm 9. To improve the security and efficiency of the protocol, we can employ an external party or the cloud P_0 to solve the transformed optimization problem. Then, we can let it generate the public-privacy key pair (pk, sk) and send the public key pk to the data holders P_1, \dots, P_r for encryption.

Consequently, every party first encrypts all the required scalar products in the

Algorithm 9: Privacy-preserving Collaborative NLP

Input : Horizontally partitioned constraint matrix T_k held by P_k
 $(1 \leq k \leq r)$, P_0 is an external party

Output: Optimal Solution $x = (x_1, \dots, x_n)$

/ all party agree on an large integer value $\ell \geq m$*
**/*

- 1 every party P_k ($1 \leq k \leq r$) generates an $\ell \times m_k$ random matrix A_k ;
- 2 P_0 generates a pair of public-private key (pk, sk) and sends pk to P_1, \dots, P_r ;
- /* Homomorphic Encryption: a random nonce is chosen for each encryption* **/*
- 3 **for** $k = 1, \dots, r$ **do**
- 4 P_k encrypts all the entries in A_k, T_k, F_k with pk : $Enc_{pk}(A_k) = A'_k$,
 $Enc_{pk}(T_k) = T'_k$, and $Enc_{pk}(F_k) = F'_k$;
- 5 **for** each row $s \in [1, \ell]$ of A'_k and each column $i \in [1, n]$ of T'_k **do**
- 6 P_k computes $Enc_{pk}[A_k T_k]_{si} = \prod_{j=1}^{m_k} [A'_k]_{sj}^{(T'_k)'_{ji}}$;
- 7 P_k computes $Enc_{pk}[A_k F_k]$ and $Enc_{pk}[A_k B_k]$ similar to Step 5-6;
- 8 P_1, \dots, P_r jointly computes $Enc_{pk}[AT] = \prod_{k=1}^r Enc_{pk}[A_k T_k]$,
 $Enc_{pk}[AF] = \prod_{k=1}^r Enc_{pk}[A_k F_k]$, and $Enc_{pk}[Ab] = \prod_{k=1}^r Enc_{pk}[A_k B_k]$;
- 9 an arbitrary party encrypts the vector in the objective function
 $\vec{c} = (c_1, \dots, c_n)$ with pk : $Enc_{pk}[\vec{c}]$;
- /* Step 10-12: Variables Permutation* **/*
- 10 **for** $k = 1, \dots, r$ **do**
- 11 P_k applies index permutation π_k to $Enc_{pk}[AT]$ and $Enc_{pk}[\vec{c}]$, and
sends them to the next party;
- /* the last party sends $Enc_{pk}[\pi_r(\dots\pi_1(AT)\dots)]$, $Enc_{pk}[Ab]$,
 $Enc_{pk}[AF]$ and $Enc_{pk}[\pi_r(\dots\pi_1(\vec{c})\dots)]$ to P_0* **/*
- 12 P_0 decrypts $Enc_{pk}[\pi_r(\dots\pi_1(AT)\dots)]$, $Enc_{pk}[AF]$, $Enc_{pk}[Ab]$ and
 $Enc_{pk}[\pi_r(\dots\pi_1(\vec{c})\dots)]$ with sk to obtain $\pi_r(\dots\pi_1(AT)\dots)$, AF , Ab and
 $\pi_r(\dots\pi_1(\vec{c})\dots)$;
- 13 P_0 solve the transformed NLP problem: linear constraints
 $\pi_r(\dots\pi_1(AT)\dots)x + AFy = Ab$, and non-linear objective function
 $\sum_{i=1}^n [c_i \log(x_i + 1)]$ (note that x has been permuted here) using linear
approximation. Then, the permuted optimal solution $\pi_r(\dots\pi_1(x)\dots)$ is
obtained;
- /* all parties get the optimal solution in the
true index by applying their inverse
permutations $\pi_k^{-1}(k = r, \dots, 1)$ to $\pi_r(\dots\pi_1(x)\dots)$ in
order* **/*

matrix multiplication $A_k T_k$, $A_k F_k$, $A_k B_k$ using pk (Step 3-7). Second, all parties jointly compute the cipher-texts of $\sum_{k=1}^r A_k T_k$, $\sum_{k=1}^r A_k F_k$ and $\sum_{k=1}^r A_k B_k$ with homomorphic property (Step 8). Note that the vector \vec{c} in the objective function should also be applied with the same permutation as matrix AT since the coefficients for variables (x_1, \dots, x_n) in the objective function are derived from $\vec{c} = (c_1, \dots, c_n)$, where \vec{c} should be encrypted before permutation. Third, every party $P_k (k = 1, \dots, r)$ applies its column/index permutation π_k to the cipher-texts $Enc_{pk}(\vec{c})$ and $Enc_{pk}[AT]$ respectively (Step 10-11), and then they send all the cipher-texts to P_0 . After receiving the cipher-texts from r data holders (P_1, \dots, P_r) , P_0 decrypts the share of the collaborative NLP problem with its private key sk , and solve the transformed problem with linear approximation (Step 12-13).

Finally, every party $P_k (k = r, \dots, 1)$ can apply their inverse permutations $k = r, \dots, 1, \pi_k^{-1}$ to the permuted optimal solution $\pi_r(\dots \pi_1(x) \dots)$ in order, and acquire $x = (x_1, \dots, x_n)$ with the true index.

6.1.3 Security and Cost Analysis

Security Analysis. In Algorithm 9, most of the steps are locally executed by every party.

Theorem 6.1 *In Algorithm 9, P_1, \dots, P_r learns only P_0 's public key pk and the permuted optimal solution $\pi_r(\dots \pi_1(x) \dots)$ while P_0 learns only the transformed matrices/vector $\pi_r(\dots \pi_1(AT) \dots)$, AF , Ab and $\pi_r(\dots \pi_1(\vec{c}) \dots)$.*

Proof. P_1, \dots, P_r 's view: every party $P_k (1 \leq k \leq r)$ first encrypts the scalar products in its transformed matrix/vector $A_k T_k$, $A_k F_k$, and $A_k B_k$ with the public key pk (Step 3-7), and there is no communication occurring in this stage. This can be simulated by running these steps by each party where random nonce are chosen. In addition, step 8 calls a secure sum subprotocol with the inputs of the distributed matrices/vectors (the messages can be simulated per the proof in [127]). In the stage of permutation (Step 10-12), P_1, \dots, P_r can run an inverse permutation algorithm to the cipher-texts with permutation $\pi_r(\dots\pi_1(\cdot)\dots)$ in linear time. Thus, P_1, \dots, P_r 's view can be simulated in polynomial time, and they learn only P_0 's public key pk and the final output x .

P_0 's view: P_0 receives following messages from r parties (P_1, \dots, P_r), $Enc_{pk}[\pi_r(\dots\pi_1(AT)\dots)]$, $Enc_{pk}[AF]$, $Enc_{pk}[Ab]$ and $Enc_{pk}[\pi_r(\dots\pi_1(\vec{c})\dots)]$ in only one round communication. P_0 learns $\pi_r(\dots\pi_1(AT)\dots)$, AF and vectors Ab and $\pi_r(\dots\pi_1(\vec{c})\dots)$ to solve the transformed NLP problem. Thus, the messages can be simulated by encrypting $\pi_r(\dots\pi_1(AT)\dots)$, AF and vectors Ab and $\pi_r(\dots\pi_1(\vec{c})\dots)$ with its own public key pk . This simulator is clearly constructed in linear time.

This completes the proof.

Mangasarian [90] and Li et al. [75] have shown that it is nearly impossible to learn matrices A , F and vector b with the known transformed matrices AT , AF and vector Ab . Besides the matrix multiplication based transformation, we let all parties jointly permute the variables in the NLP problem. Thus, P_0 can only formulate a random NLP problem which reveals nothing about the original problem.

Optimization Computation Cost. In the protocol, an NLP problem with linear constraints is securely transformed by P_1, \dots, P_r and solved by P_0 . Such NLP problem is formulated with n variables and m private linear constraints where m linear constraints are securely transformed into ℓ new linear constraints ($\ell \geq m$). Since P_0 solves the transformed problem by linear approximation with K intervals in $[0, d]$, the final LP problem consists of Km variables and ℓ linear constraints. Indeed, simplex method or revised simplex method can seek the optimal solution for such LP problem in a very short time, which can be ignored in the protocol.

Encryption and Decryption Cost. Every party P_k first encrypts the scalar products of $\ell(n + m + 1)$ pairs of length- m_k vectors (one time length- n vector encryption is also required for objective function). The secure sum then executes with $\ell(n + m + 1)$ entries amongst r parties. Note that the computation cost of permutation can be ignored compared to encryption and decryption. Finally, P_0 runs one time decryption for $\ell(n + m + 1) + n$ entries.

Communication Cost. The external party (or cloud) P_0 first sends its public key to P_1, \dots, P_k . Consequently, the secure sum and variables permutation call $(r - 1)$ rounds of communication among P_1, \dots, P_r in total while the outsourcing calls one-time communication between P_r and P_0 . Thus, the total bit cost for this step is $r * pk_size + 2(r - 1)(\ell * n + \ell * m + \ell) + (\ell * n + \ell * m + \ell + n) \approx O(r\ell(m+n))$ bits (the communication cost of inverse permutation on the length- n optimal solution can be ignored due to tiny overheads).

6.2 Collaborative Graph Coloring

Collaborative graph coloring problem can be used to solve the multiparty scheduling with conflicted resources, networked resource allocation and so on [91]. For instance, we consider a simple multiparty job scheduling problem:

Example 6.1 *As discussed in Section 1.1.3, Alice, Bob and Carol need to assign their 7 different jobs into 3 time slots. Figure 1.2 shows that Alice, Bob and Carol has job $\{1, 2, 3\}$, $\{4, 5\}$ and $\{6, 7\}$ respectively; some jobs cannot be assigned into the same time slot due to resource conflicts (e.g., sharing the same machine); is it possible to complete such scheduling without sharing information?*

If Alice, Bob and Carol do not care security and thus share their information, the above scheduling problems can be typically modeled by a k -coloring problem: the decision problem of graph coloring. Specifically, given $k = 3$ different colors (k time slots), an undirected graph $G = (V, E)$ is provided in Figure 6.1(a) to specify all the conflicts among jobs where the vertex and edge in the graph represent the job and the conflict, respectively. Each vertex in the graph should be assigned with one color out of k colors – choosing one time slot out of k time slots for each job. However, the endpoints of each edge should not be assigned with the same color – every two jobs that have a conflict should not be scheduled into the same time slot. In the above circumstances, we seek a feasible coloring solution for all vertices without violating any conflict, which is the job scheduling solution.

However, G is partitioned as follows: each party holds some vertices and the edges related to its own vertices. As shown in Figure 6.1(a), Alice, Bob and

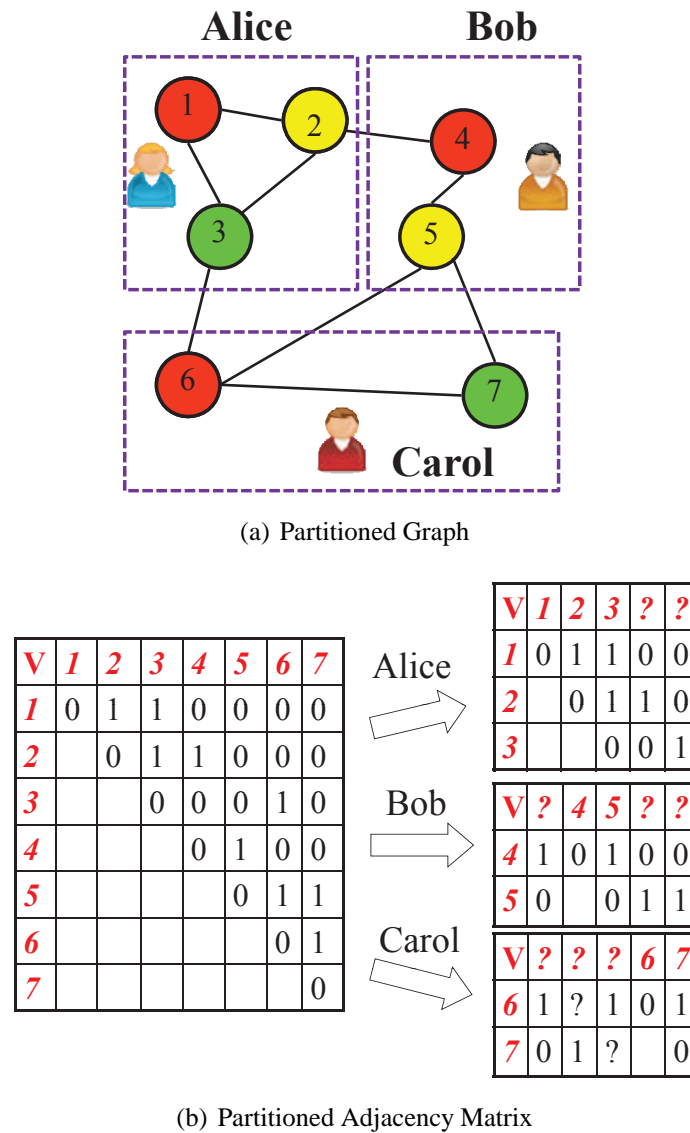


Figure 6.1. Collaborative Graph Coloring Problem

Carol have 3, 2 and 2 jobs, respectively. The graph is thus partitioned and each party only knows a share of the graph (see the adjacency matrix in Figure 6.1(b)). Since each party's partition in the graph represents its preferences, constraints or limitations, Alice, Bob and Carol are not always willing to reveal them to each other. Also, the assigned time slots for every party's private jobs are given as the colors in the graph coloring problem. Therefore, those distributed parties would

be eager to find an approach to securely solve such optimization problem.

Furthermore, since graph coloring and its decision problem (k -coloring) have been proven to be NP-hard and NP-complete respectively, exhaustive search is not feasible to solve them regardless of security concern. Thus, securely solving graph coloring problem should be even more challenging due to the computational complexity posed by both security and optimization. Note that some meta-heuristics (e.g., tabu search) are given as generic approaches to solve the graph coloring problem in literature [41, 42, 49]. Even though meta-heuristics cannot always guarantee the optimal solution and polynomial time complexity, they can obtain near-optimal outputs in an acceptable running time. In this section, we thus propose a secure and efficient solver for collaborative graph coloring problem based on the tabu search meta-heuristic.

6.2.1 Problem Definition

In combinatorial optimization, the decision problem of graph coloring is denoted as k -coloring – does graph G admit a proper vertex coloring with k colors (NP-complete [102]). Since the optimization problem of graph coloring is to seek the minimum k (NP-hard) for a given graph and it can be converted into a set of k -coloring problems, we work on the collaborative k -coloring problems in this dissertation.

While solving either the k -coloring problem or the chromatic number k seeking problem, the number of adjacent vertices that have the same color should be minimized to 0. We thus denote any two adjacent vertices with the same color as a “conflict” in a colored graph. Then, given k different colors, if we let μ represent

the total number of conflicts in a colored G , we thus have: if $\min(\mu) = 0$, G is k -colorable.

Given k colors, we can represent the color of any vertex in the graph using a length- k boolean vector x (domain $\Phi(k)$: all the length- k boolean vectors include only one “1” and $k - 1$ “0”s). If the i th number in the boolean vector is 1, the current vertex is colored with the i th color where $i \in [1, k]$. Thus, we let $x = \{x_i, \forall v_i \in G\}$ be a coloring solution in a k -coloring problem where x_i represents vertex v_i ’s color. In addition, given the colors of two adjacent vertices $x_i, x_j \in \Phi(k)$, we can represent the conflict of v_i and v_j ’s colors as the scalar product of x_i and x_j :

$$\mu_{ij}(x) = \begin{cases} 0 & \text{if } x_i \cdot x_j = 0 \text{ (} v_i \text{ and } v_j \text{ are not conflicted in solution } x \text{)} \\ 1 & \text{if } x_i \cdot x_j = 1 \text{ (} v_i \text{ and } v_j \text{ are conflicted in solution } x \text{)} \end{cases} \quad (6.3)$$

where $\mu(x) = \sum_{\forall e_{ij} \in G} \mu_{ij}(x)$. We thus formulate the k -coloring problem as $\{\min : \sum_{\forall e_{ij} \in G} x_i \cdot x_j, s.t. \forall v_i, v_j \in G, x_i, x_j \in \Phi(k)\}$. If and only if the optimal value is 0, the problem is k -colorable. To seek the chromatic number k , we can solve the above problem using different k and find the minimum k .

6.2.2 Privacy-preserving Tabu Search

In the partitioned graph shown in Figure 6.1, we define two different edges: 1. *internal edge*: any edge e_{ij} ’s two endpoints (v_i and v_j) belong to the same party; 2. *external edge*: any edge e_{ij} ’s two endpoints (v_i and v_j) belong to two different parties.

While m different parties are collaboratively solving the coloring problem,

each party does not want to reveal its partition in G and its vertex colors to each other. For instance, suppose we color the graph in Figure 6.1(a) using $k = 3$ colors, a 3-colored solution is given in Figure 6.1(a). In current solution, Alice, Bob and Carol could only know the information shown in Figure 6.2 respectively.

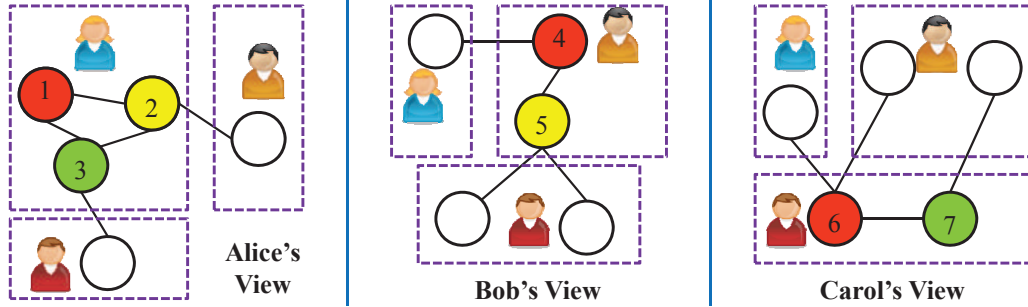


Figure 6.2. Private Information in Collaborative Graph Coloring

We have two security goals while solving the above optimization problem: first, protecting each party's substructure of the graph including its vertices and the internal edges (note that the external edges and their endpoints should be inevitably known by the involved two parties); second, protecting the colors of each party's all vertices (including the endpoints of the external edges) in every iteration. As discussed in Chapter 1, a secure communication protocol is indispensable to replace the trusted third-party for solving the problem without private information disclosure, assuming that all parties are honest-but-curious. We follow the same line of research as the collaborative LP problem.

As mentioned in Section 3.1.3, a typical tabu search algorithm for graph coloring problem includes numerous repeated computation as: computing the total number of conflicts for a solution, searching better neighborhoods for a specific solution, verifying a new vertex color pair with all the taboos and updating the best solution and tabu list if necessary. We now introduce how to secure them.

Fundamental Cryptographic Building Blocks

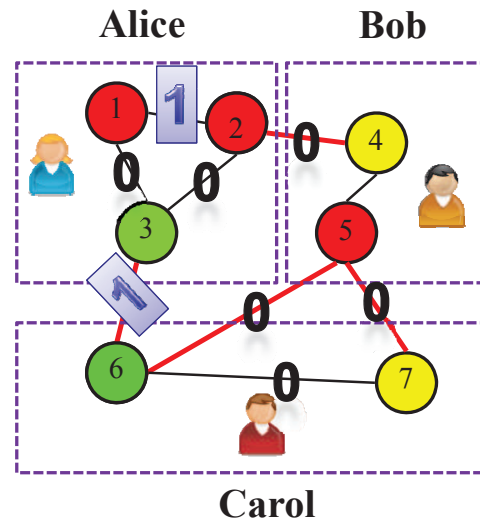
While composing the secure communication protocol for any complex SMC function such as solving the graph coloring problem with tabu search, some common fundamental protocols are iteratively integrated as building blocks of the overall protocol. Specifically, they are summarized as following.




- Homomorphic Cryptosystem for Secure Conflict Computation.** Given any solution (a colored graph) in the collaborative graph coloring problem, the key criterion to justify its optimum is the total number of conflicts. As introduced earlier, the total number of conflicts in a solution $x - \mu(x)$ can be represented as the sum of all the scalar products of every pair of adjacent vertices' coloring vectors. To securely compute the scalar product of two vectors, we utilize a public-key cryptosystem with homomorphic property. Goethals et al. [48] propose a simple and provably secure method to compute the scalar product using Homomorphic Encryption. The problem is defined as follows: P_a has a length- n vector \vec{X} while P_b has another length- n vector \vec{Y} . At the end of the protocol, P_a should get vector vector $\vec{X} \cdot \vec{Y} + r_b$ where r_b is a random number chosen from a uniform distribution and is known only to P_b . The key idea behind the protocol is to use a homomorphic encryption system such as the Paillier cryptosystem [101]. Homomorphic encryption is a semantically-secure public-key encryption which has the additional property that given any two encryptions $E(A)$ and $E(B)$, there exists an encryption $E(A * B)$ such that $E(A) * E(B) = E(A * B)$, where $*$ is either addition or multiplication (in some abelian group). The

cryptosystems mentioned above are additively homomorphic (thus the operation $*$ denotes addition). Using such a system, it is quite simple to create a secure scalar product computation protocol. If P_a encrypts its vector \vec{X} and sends $Enc_{pk}(\vec{X})$ with the public key pk to P_b , P_b can use the additive homomorphic property to compute the encrypted sum of scalar product and its random number: $Enc_{pk}(\vec{X} \cdot \vec{Y} + r_b)$. Finally, P_b sends $Enc_{pk}(\vec{X} \cdot \vec{Y} + r_b)$ back to P_a . Thus, P_a decrypts and obtain $\vec{X} \cdot \vec{Y} + r_b$ (r_b and $\vec{X} \cdot \vec{Y}$ are unknown to P_a). Since P_b cannot decrypt $Enc_{pk}(\vec{X} \cdot \vec{Y} + r_b)$ with a public key, it cannot learn the scalar product either.

Therefore, we can adopt the above cryptographic building block to securely compute the total number of conflicts $\mu(x)$ in solution x (Secure Conflict Computation). Note: while computing $\mu(x)$, it is not necessary to securely compute the number of each party's conflicted internal edges in x . Alternatively, we can let each party independently count them, and sum the number with its shares in secure scalar product computation. The detailed steps are shown in Algorithm 10.

Figure 6.3(b) and 6.3(a) provide a simple example of Secure Conflict Computation. For each external edge, such as e_{24} , e_{36} , e_{56} and e_{57} , an homomorphic encryption is implemented between two involved parties. As shown in Figure 6.3(b), the returned number in secure scalar product computation is the sum of $\mu_{ij}(x)$ and a random number r_{ij} generated by the other party (Note that the scalar product of every external edge is securely computed only once). To generate each party's output, each party thus sums all the returned numbers and its number of conflicted internal edges, and finally

(a) Conflicts in G

			
Color Conflicts: 0, 1	Alice	Bob	Carol
E_{12} : 1 (Alice)	1	N/A	N/A
E_{13} : 0 (Alice)	0	N/A	N/A
E_{23} : 0 (Alice)	0	N/A	N/A
E_{24} : 0 (Alice & Bob)	45	-45	N/A
E_{36} : 1 (Alice & Carol)	100	N/A	-99
E_{45} : 0 (Bob)	N/A	0	N/A
E_{57} : 1 (Bob & Carol)	N/A	77	-77
E_{56} : 0 (Bob & Carol)	N/A	-98	98
E_{67} : 0 (Carol)	N/A	N/A	0
Shares of the Sum	146	-66	-78
Total # of Conflicts	$2 = 146 - 66 - 78$		

(b) Each Party's Share in $\mu(x)=2$

Figure 6.3. Secure Conflict Computation (Secured Sum of the Scalar Products)

minus all its generated random numbers. At this time, the sum of all parties' outputs should be $\mu(x)$. Clearly, this protocol is provably secure.

- **Secure Comparison.**

Additionally, given two solutions x and x' , we can let each party P_a generate

Algorithm 10: Secure Conflict Computation

Input : $G = (V, E)$ is colored as $x \in \Phi(k)$ where the color of $\forall v_i \in G$ is denoted as $x_i \in x$

Output: each party P_a 's share $\mu_a(x)$ in $\mu(x)$ where $\mu(x) = \sum_{\forall a} \mu_a(x)$

- 1 **for** each party P_a **do**
- 2 count the total number of its conflicted internal edges as $\mu_a(x)$;
- 3 **forall** external edges $e_{ij} \in G$ **do**
 - /* Suppose that v_i and v_j belong to P_a and P_b , respectively */
 - 4 W.l.o.g. P_a creates a public and private key pair (pk, sk) ;
 - 5 P_a encrypts its color vector x_i as $x'_i = Enc_{pk}(x_i)$ and sends x'_i and pk to P_b ;
 - 6 P_b generates a random integer r_{ij} , encrypts the scalar product as: $x''_i = Enc_{pk}(x_i \cdot x_j) * Enc_{pk}(r_{ij})$ with the public key pk , and sends x''_i back to P_a ;
 - 7 P_a decrypts x''_i with its private key sk and obtain $s_{ij} = x_i \cdot x_j + r_{ij}$;
- 8 **for** each party P_a **do**
- 9 $\mu_a(x) \leftarrow \mu_a(x) + \sum \forall s_{ij}$ (received from other parties) $- \sum \forall r_{ij}$ (generated by P_a);

its shares $\mu_a(x)$ and $\mu_a(x')$ in $\mu(x)$ and $\mu(x')$ respectively, ensuring that $\mu(x) = \sum_{\forall a} \mu_a(x)$ and $\sum_{\forall a} \mu_a(x')$. However, $\mu(x)$ and $\mu(x')$ cannot be revealed to any party because they may lead to potential privacy breach in generating neighborhoods (we will discuss it later on).

To securely compare $\mu(x')$ and $\mu(x)$ using all parties' partitioned shares, we can employ Yao's secure comparison protocol [132] that compares any two integers securely. Yao's generic method is one of the most efficient methods known. Recently, FairplayMP [12] is developed for secure multiparty computation where m untrusted parties with private inputs x_1, \dots, x_m wish to jointly compute a specific function $f_i(x_1, \dots, x_m)$ and securely obtain each party's output. Specifically, FairplayMP is implemented by writing functions with a high-level language – Secure Function Definition Language

(SFDL) 2.0 and describing the participating parties using a configuration file. The function is then compiled into a Boolean circuit, and a distributed evaluation of the circuit is finally performed without revealing anything. In our secure comparison, each party has the shares in $\mu(x)$ and $\mu(x')$ respectively; the function should be $\mu(x') - \mu(x)$ (computed by all m parties); the output $\mu(x') < \mu(x)$ or $\mu(x') \geq \mu(x)$ is generated for designated parties.

Furthermore, we have to compare $\mu(x)$ and 0 for testing whether the optimization should be terminated or not. With the same protocol, we can let all parties securely compare $\mu(x)$ and 1 with inputs from all parties (the shares of $\mu(x)$ and 1). The output $\mu(x) < 1$ or $\mu(x) \geq 1$ is similarly generated for corresponding parties.

Analysis

1) Distributed Tabu List. In each iteration of the Tabucol algorithm, a set of neighborhoods is generated for current solution x in local search. While generating a new neighborhood x' , if the new vertex color in x' has already been considered as a taboo in the previous iterations, x' should be abandoned only except that $\mu(x') < \mu(x)$ (aspiration criteria). Thus, checking whether the tabu list T includes a move $[x \rightarrow x']$ or not (we call the color change for only one vertex as a “move”) should be a key step in privacy-preserving tabu search.

Similar to Hertz’s work [49], we define every taboo in T as a visited color for a specific vertex. Moreover, if all parties share a global tabu list, the taboos in T may reveal private information (each parties’ vertices and unallowed colors). To protect such information, we can let each party $P_a, a \in [0, m - 1]$ establish a

tabu list T_a to record the latest color of one of its vertices that leads to a critical move ($\mu(x)$ is reduced). We thus have $T = \bigcup_{\forall a} T_a$. Obviously, since each party maintains its own tabu list, distributed tabu list does not breach privacy in any case.

2) Inference Attack from the Outputs of Local Search. Since tabu search is a local search based technique, the computation in local search – generating neighborhoods should be secured. However, even if computing the number of conflicts and comparing $\mu(x)$ and $\mu(x')$ are guaranteed to be secure by SMC, potential inference attack still exists in the outputs of secure computation adapted local search.

In a partitioned graph (as shown in Figure 6.1(a)), an edge is either internal (belongs to one party) or external (belongs to two parties). Similarly, we can divide the vertices into two complementary groups: 1. *Inner Vertex*: a vertex that is not on any external edge (i.e., v_1 in Figure 6.3(a)), and 2. *Border Vertex*: a vertex that is on at least one external edge (might be on both of external edges and internal edges, e.g., v_2 in Figure 6.3(a)).

We first assume that no secure comparison is applied and v_i is the color changed vertex in generating x 's neighborhood x' ($\mu(x)$ is then revealed in each iteration). If v_i is an inner vertex, since v_i is disconnected with other parties' vertices, nothing is revealed to each other. However, if v_i is a border vertex, we first look at a simple case: v_i is an endpoint of only one external edge (e.g., v_2 in Figure 6.3(a)). Assume that P_a and P_b hold v_i (color x_i) and v_j (color x_j) respectively where e_{ij} is the external edge. While changing v_i 's color from x_i to x'_i , since the number of P_a 's conflicted internal edges in x and x' are known to P_a and

the total number of conflicts on the remaining edges (including all the external edges except e_{ij} and other parties' internal edges) are identical in x and x' , the difference between $\mu_{ij}(x)$ (e_{ij} has a conflict or not in solution x) and $\mu_{ij}(x')$ (e_{ij} has a conflict or not in solution x') can be inferred by P_a . We thus discuss the potential privacy breach in three cases where $\delta = \mu_{ij}(x') - \mu_{ij}(x)$:

1. $\delta = 1$. P_a can infer $x_j = x'_i$. The color of v_j is thus learnt by P_a .
2. $\delta = 0$. P_a can infer $x_j \neq x_i$ and $x_j \neq x'_i$. It is secure for large k . For small k , P_a can guess x_j as any color out of the remaining $k - 2$ colors except x_i and x'_i .
3. $\delta = -1$. P_a can infer $x_j = x_i$. The color of v_j is also learnt by P_a .

More generally, if v_i is adjacent to n border vertices from other parties and no secure comparison is implemented in tabu search, moving v_i to generate neighborhoods is under the risk of inference attack:

Lemma 6.1 *Given a coloring solution x , we generate x 's neighborhood x' by changing the color of a border vertex v_i of party P_a which is adjacent to n vertices $\{\forall j \in [1, n], v_j\}$ from other parties. If no secure comparison is implemented in tabu search (P_a knows $\delta = \sum_{j \in [1, n]} [\mu_{ij}(x') - \mu_{ij}(x)]$), precisely one of the following holds:*

1. if $\delta \geq 0$, P_a can guess the color of v_j as $x_j = x'_i$ with $\text{Prob}[x_j = x'_i] = \frac{\delta + \lfloor \frac{n+\delta}{2} \rfloor}{2n}$ and guess $x_j = x_i$ with $\text{Prob}[x_j = x_i] = \frac{\lfloor \frac{n-\delta}{2} \rfloor}{2n}$ where $\delta \in [0, n]$;
2. if $\delta < 0$, P_a can guess the color of v_j as $x_j = x_i$ with $\text{Prob}[x_j = x_i] = \frac{-\delta + \lfloor \frac{n-\delta}{2} \rfloor}{2n}$ and guess $x_j = x'_i$ with $\text{Prob}[x_j = x'_i] = \frac{\lfloor \frac{n+\delta}{2} \rfloor}{2n}$ where $\delta \in [-n, -1]$.

Proof. Assuming that v_i is an endpoint of n external edges $e_{ij}, j \in [1, n]$, we represent that the colors of e_{ij} 's endpoints in x (and x') are conflicted or not as $\mu_{ij}(x) = 0$ or 1 (and $\mu_{ij}(x') = 0$ or 1).

If $\mu(x)$ and $\mu(x')$ are revealed in each iteration, while P_a changes the color of v_i (from x_i to x'_i), $\sum_{\forall j \in [1, n]} [\mu_{ij}(x') - \mu_{ij}(x)]$ (denoted as δ) is known to P_a . Specifically, $\delta \in [-n, n]$ has following cases:

- **Case 1:** if $\delta = n$, P_a learns: no edge from $\forall j \in [1, n]$, e_{ij} has a conflict in x , but all the n edges have conflicts in x' (thus, the colors of all n border vertices from other parties are identical to x'_i);
- **Case 2:** if $\delta = -n$, P_a learns: all the edges have conflicts in x but all the conflicts are eliminated in x' (thus, the colors of all n border vertices from other parties are identical to x_i);
- **Case 3:** more generally, if $-n < \delta < n$, every edge has a probability to have conflict. We denote the probability that a specific edge e_{ij} has no conflict in x but has conflict in x' (and has conflict in x but has no conflict in x') as $Prob[\mu_{ij}(x \rightarrow x') : 0 \rightarrow 1]$ (and $Prob[\mu_{ij}(x \rightarrow x') : 1 \rightarrow 0]$) where $\delta \in (-n, n)$ (since inferring that e_{ij} has conflict in either x or x' should result in learning v_j 's color by P_a).

◇. We first consider that $\delta \geq 0$ (if $\delta < 0$, we can obtain a similar result). We denote the number of edges (out of n) where $\mu_{ij}(x) \rightarrow \mu_{ij}(x') : 0 \rightarrow 1$ as δ_1 ; denote the number of edges (out of n) where $\mu_{ij}(x) \rightarrow \mu_{ij}(x') : 1 \rightarrow 0$ as δ_0 ; denote the number of edges (out of n) where $\mu_{ij}(x) \rightarrow \mu_{ij}(x') : 0 \rightarrow 0$ as δ_c

($1 \rightarrow 1$ is impossible since x_i is changed). Thus, we have $\delta_1 + \delta_0 + \delta_c = n$. Since $\delta_1 - \delta_0 = \delta$ (overall difference), we have $\delta_1 \leq \lfloor \frac{n+\delta}{2} \rfloor$ and $\delta_0 \leq \lfloor \frac{n-\delta}{2} \rfloor$ (if $n - \delta$ is an odd number, due to $\delta_c \geq 0$, we have $\delta_1 < \frac{n+\delta}{2}$). Furthermore, it is clear that $\delta_1 \geq \delta$. We now discuss all possible values of (δ_1, δ_0) .

Specifically, given δ and n , (δ_1, δ_0) has following possible pairs of values: $(\delta, 0), (\delta + 1, 1), \dots, (\lfloor \frac{n+\delta}{2} \rfloor, \lfloor \frac{n-\delta}{2} \rfloor)$ (equally possible for all pairs). Hence, the probability of valuing each pair is $\frac{1}{\lfloor \frac{n-\delta}{2} \rfloor + 1}$. As a consequence, for a given edge, $Prob[\mu_{ij}(x \rightarrow x') : 0 \rightarrow 1] = \frac{1}{\lfloor \frac{n-\delta}{2} \rfloor + 1} \times [\frac{\delta}{n} + \frac{\delta+1}{n} + \dots + \frac{\lfloor \frac{n+\delta}{2} \rfloor}{n}] = \frac{\delta + \lfloor \frac{n+\delta}{2} \rfloor}{2n}$ and $Prob[\mu_{ij}(x \rightarrow x') : 1 \rightarrow 0] = \frac{1}{\lfloor \frac{n-\delta}{2} \rfloor + 1} \times [\frac{0}{n} + \frac{1}{n} + \dots + \frac{\lfloor \frac{n-\delta}{2} \rfloor}{n}] = \frac{\lfloor \frac{n-\delta}{2} \rfloor}{2n}$. We thus have:

1. since the probability $Prob[\mu_{ij}(x \rightarrow x') : 0 \rightarrow 1] = Prob[x_j = x'_i]$ with a known fixed δ , party P_a can guess that $x_j = x'_i$ and the endpoint colors of e_{ij} are conflicted in solution x' with a probability of $\frac{\delta + \lfloor \frac{n+\delta}{2} \rfloor}{2n}$.
2. since the probability $Prob[\mu_{ij}(x \rightarrow x') : 1 \rightarrow 0] = Prob[x_j = x_i]$ with a known fixed δ , party P_a can guess that $x_j = x_i$ and the endpoint colors of e_{ij} are conflicted in solution x with a probability of $\frac{\lfloor \frac{n-\delta}{2} \rfloor}{2n}$.

◇. Similarly, if $\delta < 0$, we can deduce that $Prob[x_j = x_i] = Prob[\mu_{ij}(x \rightarrow x') : 1 \rightarrow 0] = \frac{-\delta + \lfloor \frac{n-\delta}{2} \rfloor}{2n}$ and $Prob[x_j = x'_i] = Prob[\mu_{ij}(x \rightarrow x') : 0 \rightarrow 1] = \frac{\lfloor \frac{n+\delta}{2} \rfloor}{2n}$.

Finally, we verify Case 1 and 2 in the general probability computation: if $\delta = n$, we have $Prob[\mu_{ij}(x \rightarrow x') : 0 \rightarrow 1] = 1$ and $Prob[\mu_{ij}(x \rightarrow x') : 1 \rightarrow 0] = 0$, hence the color of v_j is determined as x'_i ; if $\delta = -n$, we have

$Prob[\mu_{ij}(x \rightarrow x') : 0 \rightarrow 1] = 0$ and $Prob[\mu_{ij}(x \rightarrow x') : 1 \rightarrow 0] = 1$, hence the color of v_j is determined as x_i .

This completes the proof.

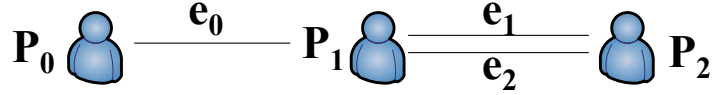
Note that we do not discuss the probability of guessing x_j as any color out of remaining $k - 2$ colors except x_i and x'_i since the probability is fairly small in general. Hence, if $\mu(x)$ is revealed in each iteration (no secure comparison protocol is applied), the probability of guessing any other parties' border vertex color is high. For instance, assuming that $x_i = \text{Blue}$, $x'_i = \text{Red}$, $\delta = 3$ and $n = 5$, P_a can guess that x_j is either *Red* with a probability 0.7 or *Blue* with a probability 0.1. If P_a iteratively moves x_i and learns the probabilities in such process, the color of v_j should be almost determined. Furthermore, if δ is close to either n or $-n$, the probability of inferring privacy is extremely high, especially in case that $\delta = n$ or $-n$ (the worst privacy breach case for any border vertex on n external edges).

Alternatively, if we utilize the secure comparison protocol to securely compare $\mu(x')$ and $\mu(x)$ in each iteration, such probabilities can be significantly reduced – especially in case that δ is close to n or $-n$, simply because δ could be any integer in $[-n, n]$.

Lemma 6.2 *In the same circumstance of Lemma 6.1, if we securely compare $\mu(x)$ and $\mu(x')$ (does not reveal $\mu(x)$ and $\mu(x')$), the risk of inference attack can be reduced in general but still exists in two worst cases with known δ_a ¹³ as below:*

1. If $\delta_a^{13} = n - 1$ and $\mu(x') < \mu(x)$ is the output of the secure comparison, P_a learns $\forall j \in [1, n], x_j = x_i$;

2. If $\delta_a^{13} = -n$ and $\mu(x') \geq \mu(x)$ is the output of the secure comparison, P_a learns $\forall j \in [1, n], x_j = x'_j$.

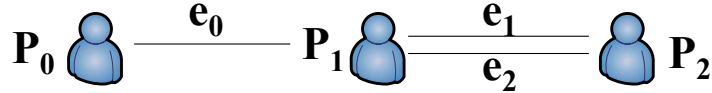


$P_0(n=1)$: if $\delta_0 = n-1=0$ but $\mu(x) \downarrow$, e_0 should be conflicted in x and not conflicted in x' (This happens very often)

$P_1(n=3)$: if $\delta_1 = n-1=2 \uparrow$ but $\mu(x) \downarrow$, δ should be less than -2, thus all e_0, e_1, e_2 should be conflicted in x and not conflicted in x'

$P_2(n=2)$: if $\delta_2 = n-1=1 \uparrow$ but $\mu(x) \downarrow$, δ should be less than -1, thus all e_1, e_2 should be conflicted in x and not conflicted in x'

(a) Worst Case 1



$P_0(n=1)$: if $\delta_0 = -n=-1 \downarrow$ but $\mu(x') \geq \mu(x)$, e_0 should be conflicted in x' and not conflicted in x (This happens very often)

$P_1(n=3)$: if $\delta_1 = -n=-3 \downarrow$ but $\mu(x') \geq \mu(x)$, δ should be no less than 3, thus all e_0, e_1, e_2 should be conflicted in x' and not conflicted in x

$P_2(n=2)$: if $\delta_2 = -n=-2 \downarrow$ but $\mu(x') \geq \mu(x)$, δ should be no less than 2, thus all e_1, e_2 should be conflicted in x' and not conflicted in x

(b) Worst Case 2

Figure 6.4. An Example for Inference Attack on Secure Comparison [$P_1(n=3)$ has a border vertex that is adjacent to one vertex of $P_0(n=1)$ and two vertices of $P_2(n=2)$]

Proof. We let $\mu_a(x)$ and $\mu_a(x')$ be the number of P_a 's conflicted internal edges in x and x' respectively, and denote δ_a as $\mu_a(x') - \mu_a(x)$. If only $\mu(x') < \mu(x)$ or $\mu(x') \geq \mu(x)$ are revealed in each iteration, while P_a changes the color of v_i (from x_i to x'_i), P_a knows only $\delta + \delta_a < 0$ or $\delta + \delta_a \geq 0$ ($\delta = \sum_{\forall j \in [1, n]} (\mu_{ij}(x') -$

¹³ δ_a denotes the difference between the number of P_a 's conflicted internal edges in solution x and x' , which is known to P_a in every iteration

$\mu_{ij}(x))$). We now try to compute the probability $Prob[x_j = x']$ and $Prob[x_j = x]$. Essentially, if δ is known and fixed, $Prob[x_j = x']$ and $Prob[x_j = x]$ can be computed in terms of Lemma 6.1. Hence, we discuss the possible privacy breaches for all cases. Since P_a knows δ_a and the relationship between $\delta + \delta_a$ and 0, we look at two cases:

- **Case 1:** if $\delta + \delta_a < 0$, we have $\delta < -\delta_a$. Since δ_a is known to P_a and it can be any integer, we discuss all possible δ_a as following.

1. if $\delta_a \geq n$, it is impossible to obtain $\delta + \delta_a < 0$ because $\delta \in [-n, n]$;
2. if $\delta_a < -n$, δ can be any integer in $[-n, n]$ to generate output $\mu(x') < \mu(x)$, hence, probabilities $Prob[x_j = x'_i]$ and $Prob[x_j = x_i]$ is minimized (no additional information is revealed);
3. if $-n \leq \delta_a \leq n - 1$, P_a can learn that $-n \leq \delta < -\delta_a$ (regularly, $-n \leq \delta \leq n$ is known to P_a). With the greater $-\delta_a$, P_a learns less additional information. The worst privacy breach case should be $\delta_a = n - 1$ such that $\delta = -n$ is inferred. At this time, P_a can infer x_j with probability $Prob[x_j = x_i] = 1$.

- **Case 2:** if $\delta + \delta_a \geq 0$, we have $\delta \geq -\delta_a$. Similarly, we discuss following cases:

1. if $\delta_a < -n$, it is impossible to obtain $\delta + \delta_a \geq 0$ because $\delta \in [-n, n]$;
2. if $\delta_a \geq n$, δ can be any integer in $[-n, n]$ to generate output $\mu(x') \geq \mu(x)$, hence, probabilities $Prob[x_j = x'_i]$ and $Prob[x_j = x_i]$ is minimized (no additional information is revealed);

3. if $-n \leq \delta_a < n$, P_a can learn that $-\delta_a \leq \delta \leq n$ (regularly, $-n \leq \delta \leq n$ is known to P_a). With the smaller $-\delta_a$, P_a learns less additional information. The worst privacy breach case should be $\delta_a = -n$ such that $\delta = n$ is inferred. At this time, P_a can infer x_j with probability $Prob[x_j = x'_i] = 1$.

In sum, with greater $-\delta_a$ in Case 1 and smaller $-\delta_a$ in Case 2, it should be more difficult to guess δ and infer $x_j = x_i$ or $x_j = x'_i$ in most cases, compared with known δ (to P_a). Hence, the risk of inference attack has been reduced in general. However, we have two worst cases that still breaches privacy: 1. As $\delta_a = n - 1$ and $\mu(x') < \mu(x)$ are the output to P_a in an iteration, thus P_a learns $\forall j \in [1, n], x_j = x_i$; 2. As $\delta_a = -n$ and $\mu(x') \geq \mu(x)$ are the output to P_a in an iteration, thus P_a learns $\forall j \in [1, n], x_j = x'_i$.

This completes the proof.

As described in Lemma 6.2, the worst privacy breach cases cannot be completely resolved by the secure comparison protocol. We show an example for two worst cases in Figure 6.4. While changing the color of P_0 , P_1 , or P_2 's one border vertex to generate neighborhoods, possible privacy breach occurs in these worst cases. For example, P_0 moves its border vertex: if the number of P_0 's conflicted internal edges has not been changed in the move $x \rightarrow x'$ and $\mu(x') < \mu(x)$ is the comparison result (both are known to P_0), the only external edge e_0 should be conflicted in x and not conflicted in x' . Thus P_0 can infer that the other endpoint's color (P_1 's one border vertex) is the same as its border vertex's color in x (before move). This happens very often.

Therefore, we will try to mitigate such inference attack and further reduce the guessing probability as follows.

3) Synchronous Move in Local Search. Note that the inference attack still exists in the worst cases if we reveal the secure comparison result, and it is extremely difficult to build an efficient protocol for such a complex function without any information disclosure. Thus, we alternatively reduce the probability of inferring information from the secure comparison results by slightly modifying local search.

More specifically, if any border vertex v_i is chosen for move while generating neighborhoods for one party, we can let another party locally change the color of one of its vertices (either inner or border) or “does not move”. We denote this mechanism as *synchronous move*. Then, the potential inferences can be mitigated by greatly reducing the probability of guessing something from the comparison results.

Lemma 6.3 *Synchronous move significantly mitigates the inference attack stated in Lemma 6.1 and 6.2.*

Proof. We first present some notations in synchronous move in Table 6.2. With a synchronous move, $\mu(x') < \mu(x)$ or $\mu(x') \geq \mu(x)$ is revealed. δ_a and δ_b are known to P_i and P_j respectively. We now discuss the potential inferences on the above revealed information.

First, if $\mu(x') < \mu(x)$, we thus have $\delta_a + \delta + \delta_b + \delta' < 0$. P_a knows δ_a and $\delta \in [-n, n]$. Since δ_b and δ' can be any integer, no additional information other

Table 6.2. Notations in Synchronous Move

v_i and v_j	P_a and P_b 's vertex for move
$\mu_a(x)$ and $\mu_a(x')$	P_a 's number of conflicted internal edges in x and x' respectively
$\mu_b(x)$ and $\mu_b(x')$	P_b 's number of conflicted internal edges in x and x' respectively
δ_a and δ_b	$\delta_a = \mu_a(x') - \mu_a(x)$, $\delta_b = \mu_b(x') - \mu_b(x)$
$\forall s, e_{is}$	n external edges with endpoint v_i
$\forall t, e_{jt}$	n' external edges with endpoint v_j
$\forall s, \mu_{is}(x)$ and $\mu_{is}(x')$	e_{is} is conflicted or not in x and x'
$\forall t, \mu_{jt}(x)$ and $\mu_{jt}(x')$	e_{jt} is conflicted or not in x and x'
δ	$\delta = \sum_{\forall s} [\mu_{is}(x') - \mu_{is}(x)]$
δ'	$\delta' = \sum_{\forall t} [\mu_{jt}(x') - \mu_{jt}(x)]$

than $\delta < -\delta_a - \delta_b - \delta'$ can be inferred (as soon as δ_a is very large, δ can still be any integer in $[-n, n]$ because $\delta_b + \delta'$ may be negative and sufficiently small to anonymize δ). Similarly, if v_j is a border vertex of P_b , P_b cannot learn any additional information other than $\delta_a + \delta + \delta_b + \delta' < 0$ and $\mu(x') < \mu(x)$.

Second, if $\mu(x') \geq \mu(x)$, we thus have $\delta_a + \delta + \delta_b + \delta' \geq 0$. For the same reason, δ is an unknown integer between $[-n, n]$ to P_i , and δ' is an unknown integer between $[-n', n']$ to P_j . No additional information can be inferred.

In a special case, P_a 's v_i is adjacent to a vertex of P_b (this is unknown to P_b). P_b knows that P_a is moving a border vertex and the result of securely comparing $\mu(x)$ and $\mu(x')$. However, P_b cannot infer that P_a is moving a border vertex that is adjacent to P_b without colluding with other parties (since P_a may have many border vertices, adjacent to many parties). Hence, synchronous move guarantees security for this special case.

Thus, the inference attack stated in Lemma 6.1 and 6.2 can be significantly mitigated. This completes the proof.

Essentially, some generated neighborhoods may include two moves. In other words, the number of moves is at most 2, thus it is still effective to maintain the performance of local search. To improve the search performance, the color changed vertex (either inner vertex or border vertex) can be *preferentially chosen as the endpoints of edges with conflicts*. Since only the conflicted internal edges are known to corresponding parties, we can iteratively let each party change the color of the endpoints of its conflicted internal edges. If moving an inner vertex, that party can independently compare two unknown numbers $\mu(x')$ and $\mu(x)$ (by comparing its shares in $\mu(x')$ and $\mu(x)$) and check its tabu list along with specific operations on its vertices colors in x' – update x and tabu list with x' , or abandon x' ; Otherwise, moving a border vertex, securely computing all scalar products and securely comparing $\mu(x')$ and $\mu(x)$ are required. Hence, while changing a border vertex color by one party, our privacy-preserving tabu search protocol ¹⁴ calls synchronous move, secure conflict computation and secure comparison. The result of secure comparison is only revealed to two involved parties, they thus check their own tabu lists along with specific operations on their vertices colors in x' .

Protocol

In our secure tabu search protocol, we can iteratively let every party (one by one) execute independent tabu search to eliminate all the conflicted internal edges ¹⁵. Each party individually moves inner vertices or jointly/synchronously move bor-

¹⁴Since no party knows the conflicted external edges, they cannot generate neighborhoods by directly changing the endpoint color of conflicted external edge. So the search performance should be slightly compromised for ensuring privacy protection (this is inevitable).

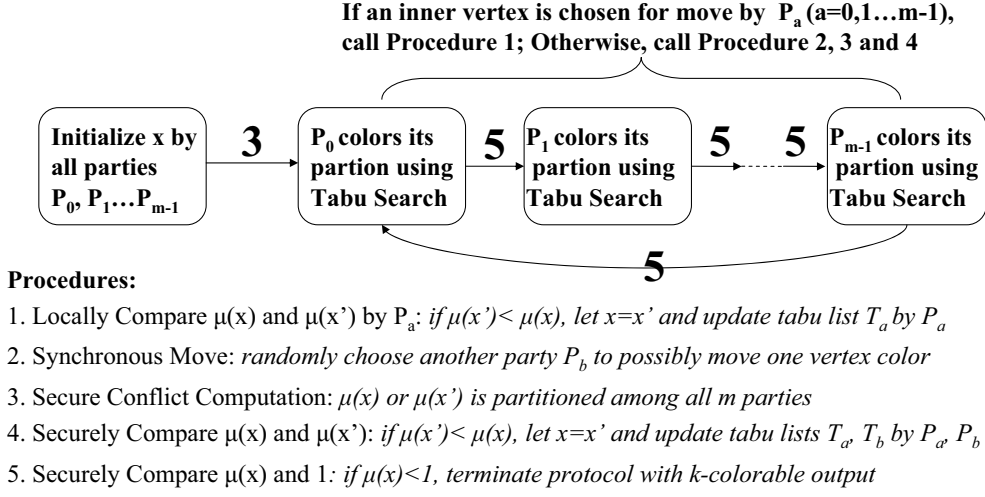


Figure 6.5. Privacy-preserving Tabu Search Protocol

der vertices in local search. If the selected party does not have any conflicted internal edge (local k -coloring is attained), this party can still propose moves on one of its border vertices while triggering the synchronous move, or skip its local search and jump into the next party. In addition, in each move, tabu list is locally checked and the aspiration criteria are applied to every solution (locally for inner vertex move or globally for synchronous move in two parties' tabu lists). Finally, we have to call the secure comparison protocol to securely compare $\mu(x)$ and 0 when any local k -coloring is achieved. Note that since the output of secure comparison is divided into $<$ and \geq , we compare $\mu(x)$ with 1. If $\mu(x) = 0$, our secure tabu search is terminated with k -colorable output. If $\mu(x) \geq 1$ after a given number of iterations, our privacy-preserving tabu search is terminated with an opposite output. The detailed steps are presented in Figure 6.5 and Algorithm 11.

¹⁵Each party's partition in the graph is a subset of G . Hence, if the graph partition is not k -colorable, we can conclude that G is not k -colorable either.

¹⁶Aspiration Criteria: if $\mu(x') < \mu(x)$, $(v_i, x'_i) \in T_a$ is allowed; Otherwise, abandon x' .

¹⁷Aspiration Criteria: if $\mu(x') < \mu(x)$, $(v_i, x'_i) \in T_a$ and $(v_j, x'_j) \in T_b$ are allowed; Otherwise, abandon x' .

Algorithm 11: Privacy-preserving Tabu Search

Input : m parties P_0, \dots, P_{m-1} , Graph $G = (V, E)$;

- 1 initialize a solution x : randomly coloring all vertices by all parties;
- 2 call Secure Conflict Computation: each party hold a share of $\mu(x)$:
 $\forall a \in [0, m-1], \mu_a(x)$;
- 3 **for** party $P_a, a \in [0, m-1]$ **do**
- 4 **while** the number of P_a 's conflicting internal edges is greater than 0 **do**
- 5 P_a generates a neighborhood x' by changing the color of vertex v_i
that is the endpoint of one of its conflicting internal edges: $x_i \rightarrow x'_i$;
- 6 **if** v_i is an inner vertex **then**
- 7 P_a locally compare its shares in $\mu_a(x)$ and $\mu_a(x')$ and check its
tabu list T_a ;
- 8 **if** $\mu_a(x') < \mu_a(x)$ ¹⁶ **then**
- 9 let $x = x'$, update the tabu list T_a and continue;
- 10 **else**
- 11 pick a random party P_b ($b \neq a$) to possibly change the color of
vertex v_j : $x_j \rightarrow x'_j$ (x' is generated by synchronous move);
- 12 call Secure Conflict Computation and Secure Comparison for
 $\mu(x')$ and $\mu(x)$;
- 13 **if** $\mu(x') < \mu(x)$ ¹⁷ **then**
- 14 let $x = x'$, update the tabu list T_a, T_b by P_a, P_b respectively
and continue;
- 15 if x 's *rep* neighborhoods are generated (*rep* is given), choose a best
neighborhood x' where $(v_i, x'_i) \notin T_a$, let $x = x'$ and update T_a (v_i is
an inner vertex of P_a);
- 16 if P_a 's partition in G cannot be k -colored in maximum number of
iterations, terminates protocol with output: G is not k -colorable;
- 17 if P_a 's partition is k -colored, P_a can move its border vertices to
generate new k -coloring solutions (call synchronous move, secure
conflict computation and secure comparison);
- 18 call Secure Comparison to compare $\mu(x)$ and 1 among all m parties;
- 19 if $\mu(x) < 1$, terminates protocol with output: G is k -colorable by
solution x ;
- 20 if $\mu(x) \geq 1$ after a given number of iterations, terminates protocol with
output: G is not k -colorable;
- 21 $a = (a + 1) \bmod m$;

6.2.3 Security and Cost Analysis

Security Analysis

The communication of the PPTS protocol occurs primarily in two sub-protocols: secure scalar product computation and secure comparison. Thus, the security of the PPTS protocol highly relies on the security of these two building blocks. Given these secure protocols, the security proof comes down to demonstrating that the outputs of those sub-protocols can be simulated while knowing one's own input and the final results. The composition theorem [45] then enables completing the proof of security.

Theorem 6.2 *PPTS is secure against semi-honest behavior.*

Proof. We first look at the view of each party (received messages in the protocol) during executing PPTS. While one party $P_a, a \in [0, m-1]$ is chosen to color its partition (this is iteratively done in PPTS), two cases occur.

First, if P_a changes the color of its inner vertices (Step 6-9), P_a learns only $\mu(x') < \mu(x)$ or $\mu(x') \geq \mu(x)$ by locally comparing its shares in $\mu(x')$ and $\mu(x)$. Other parties learn nothing either: the number of iterations required for coloring P_a 's partition is also unknown to other parties. Hence, these can be simulated by simply executing those steps.

Second, if P_a moves one of its border vertices, a synchronous move is triggered. At this time, P_a and P_b ($b \in [0, m-1], b \neq a$) know $\mu(x') < \mu(x)$ or $\mu(x') \geq \mu(x)$. We now examine each party's view in Step 11-14 and possibly Step 17, and build a polynomial simulator that runs for the known synchronous moves.

Steps in PPTS	P ₀ 's view	P ₁ 's view	P ₂ 's view	...	P _{m-1} 's view
P ₀ colors its partition	0, 1, 0, 1, 1, 0, 1...	1 (Syn. Move)			
	1 (Syn. Move)				
	1, 1, 0, 1...	0 (Syn. Move)			
	0 (Syn. Move)				
	0, 1, 1...				
	1 (Syn. Move)				
	1 (Syn. Move)				
P ₁ colors its partition		1, 1, 0, 1, 0, 1...		...	
P ₂ colors its partition			0, 1, 0, 0, 1, 0, 1...	...	
		1 (Syn. Move)	1 (Syn. Move)		
...

Figure 6.6. P_0, \dots, P_{m-1} 's View (the sign of $\mu(x') - \mu(x)$: 1 is $\mu(x') < \mu(x)$; 0 is $\mu(x') \geq \mu(x)$)

P_a 's view: in order to simulate P_a 's view, the polynomial simulator must satisfy the following condition: given P_a 's partition (input) and the final solution (output), simulate the views of secure conflict computation and secure comparison in every synchronous move (the shares in secure scalar product computation and a sequence of results of secure comparison as shown in Figure 6.6). The simulator is built as below:

- The result of secure scalar product computation $s_{ij} = x_i \cdot x_j + r_{ij}$ (w.l.o.g., we let P_a be the receiver of s_{ij}), can be simulated by generating a random from the uniform probability distribution over \mathcal{F} (Note: we assume that s_{ij} is scaled to fixed precision over a closed field, enabling such a selection). Thus, $\text{Prob}[T = t] = \text{Prob}[s_{ij} = t] = \text{Prob}[r_{ij} = t - s_{ij}] = \frac{1}{\mathcal{F}}$.
- To simulate the sequence of secure comparison results for P_a , we can utilize the inverse steps of tabu search on P_a 's border vertices (we denote this

process as inverse optimization). A key result from inverse optimization is that if the original problem is solvable in polynomial time, the reverse problem is also solvable in polynomial time [4]. Specifically, letting the cost function be maximizing $\mu(x)$ (x' remains the neighborhood of x), we start from the final colorable solution (only in P_a 's known part of the graph)¹⁸, generate neighborhoods by changing the color of its border vertices (color is randomly chosen), and compare $\mu(x')$ and $\mu(x)$. If $\mu(x') > \mu(x)$, the simulator outputs an “1”, otherwise it outputs a “0”. Since the comparison signs in moving these vertices in the simulator are deterministic and known to P_a , the comparison signs of synchronous move in original PPTS protocol can be simulated by this machine. Although the colors and the vertices might be different from the original one, the sign of comparison is identical.

P_b 's view: First, P_b knows the sign of $\mu(x') - \mu(x)$ in all synchronous move in which it is involved. Similarly, we can use the simulator to generate the signs of comparison as above. Second, in all secure scalar product computation, the random number r_{ij} can be generated again using the same uniform distribution in secure conflict computation. Since the outputs of other parties are similar to P_b , we can use the same simulator.

In the remaining Steps of PPTS, each party redefines new current solution and manipulates the tabu list in local computation, thus we can simulate them by executing those steps. Therefore, PPTS is secure against semi-honest behavior,

¹⁸Since P_a knows the existence of some adjacent border vertices from other parties, we can let P_a color them at the beginning of executing the simulator. Thus, moving P_a 's any vertex is deterministic in the simulator.

assuming that all the parties follow the PPTS protocol.

This completes the proof.

There are several remaining issues. First, since our simulator is also a meta-heuristic, the worst case running time is not polynomial. This is a problem-dependent issue: if the problem is always solved by tabu search in polynomial time, then the simulator is guaranteed to be polynomial, and the protocol is secure. Since we have defined a maximum number of iterations, we can consider that PPTS is polynomial in general and therefore PPTS is secure. Second, some additional minor information is probably leaked. For example, in the output of PPTS, each party (e.g., P_a) has a k -coloring solution with respect to its vertices. Thus, any other party's border vertex that is adjacent to P_a can be known (to P_a) with a different color from P_a 's border vertices. This is unavoidable in collaborative graph coloring problem – even if we build a trusted third-party, such information should be shared. Moreover, the number of iteratively calling different parties to color its partition and the fact that $\mu(x) > 0$ until the last step are also known to all parties. These information are trivial and do not impact the security of the PPTS protocol.

Communication and Computation Cost Analysis

Given a partitioned graph with n vertices and n_e external edges, in PPTS protocol, we assume that m parties are repeatedly called to locally color their partition for c times and ℓ synchronous moves are executed while one party is coloring its partition. We thus discuss the communication and computation cost.

Communication Cost. In PPTS protocol, only secure conflict computation

and secure comparison require communication among parties. Since computing the conflict of every external edge requires two communication messages, the communication cost of total secure conflict computation in PPTS is $2c*(n_e*\ell+1)$ messages of bit communication where $O(n_e) = O(n^2)$. Thus, the communication complexity of the total secure scalar product computation in PPTS is $O(cn^2\ell)$. Moreover, every secure comparison requires $O(m)$ parties to compute the output. Since the number of communication messages in every secure comparison is equal to the number of computing parties [62] and PPTS includes $c*(\ell+1)$ secure comparison, the communication complexity of the total secure comparison in PPTS is $O(cml)$.

Computation Cost. Since each party calls ℓ synchronous moves while coloring its partition, $c*\ell+1$ secure conflict computations and $c*(\ell+1)$ secure comparisons are required in PPTS. If we denote the runtime for a single secure scalar product computation and a single secure comparison as t_s and t_c respectively, the additional cost can be estimated as $(c*\ell*n_e+1)*t_s + c*(\ell+1)*t_c$. Thus, the computation complexity of PPTS is $O(cn^2\ell)$ on secure scalar product plus $O(c\ell)$ on secure comparison.

6.2.4 Experiments

Experiments Setup

In our experiments, we randomly generate graphs with the number of vertices N and the density of the graph ρ : the probability of presence of each edge. Each generated graph is partitioned by 10 parties (P_0, P_1, \dots, P_9) , we let each party hold $N/10$ vertices. In each test, we generate 10 graphs with the same N and ρ ,

and average the results.

We compare our privacy-preserving tabu search (PPTS) algorithm with the Tabucol [49] using the same group of graphs in every test. Both algorithms are terminated if no k -colorable solution can be found in 10^5 iterations. We let each party hold a tabu list with length $N/10$ in PPTS while the overall tabu list in Tabucol is established with tenure $N/10$. In the synchronous move of PPTS algorithm, we assume that the second party P_b does not move the color of any vertex with probability 50%. Furthermore, in PPTS algorithm, Paillier cryptosystem [101] with 512-bit and 1024-bit key is used for homomorphic encryption while FairplayMP [12] with sufficiently large modulus length for 10 parties (i.e., 128 bits) is used for secure comparison. All the experiments are performed on an HP machine with Intel Core 2 Duo CPU 3GHz and 6G RAM.

We aim at validating two facts: 1. the optimal results (the chromatic number) of solving the collaborative graph coloring problem using PPTS are close to that of Tabucol. This indicates that our protocol is as effective as the general meta-heuristics; 2. the runtime of the PPTS is acceptable (linear or polynomial to Tabucol). Thus, for the first validation, we compute the chromatic number of the partitioned graphs with varying number of vertices $\#V = \{100, 200, 300, 500, 1000\}$ and two different density $\rho = 10\%$ and 30% using both PPTS and Tabucol (for each pair of $\#V$ and ρ , we generate 10 partitioned graphs; 512-bit key is used here). For the latter test, first, we compute the runtime of 512-bit key based PPTS and 1024-bit key based PPTS and the Tabucol with varying $\#V = \{100, 200, 300, 500, 1000\}$ and fixed $\rho = 20\%$; second, we compute the runtime for them with varying $\rho = \{2\%, 5\%, 10\%, 20\%, 30\%\}$ and fixed $\#V = 500$.

Experimental Results

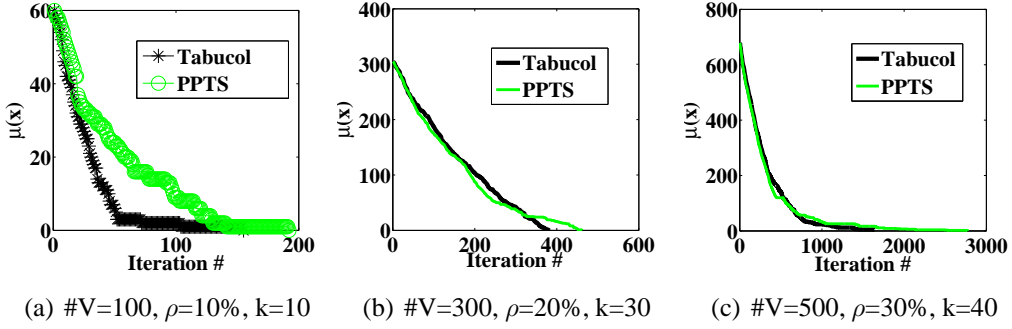


Figure 6.7. Total Number of Conflicts ($\mu(x)$) in each Iteration

Quality of the Result. In the optimal result testing, we first demonstrate the number of total conflicts $\mu(x)$ and the number of iterations required for PPTS and Tabucol for three different graphs in Figure 6.7 (note that k is slightly greater than the chromatic number). While solving the collaborative graph coloring problem on these three graphs, each iteration's improvements of the optimal solution (reduced $\mu(x)$) are quite similar for both PPTS and Tabucol. Note that PPTS generally requires more iterations compared to Tabucol. However, this is unavoidable because the conflict status of all the external edges are unknown to every party at any time, and the search performance of PPTS is indeed acceptable.

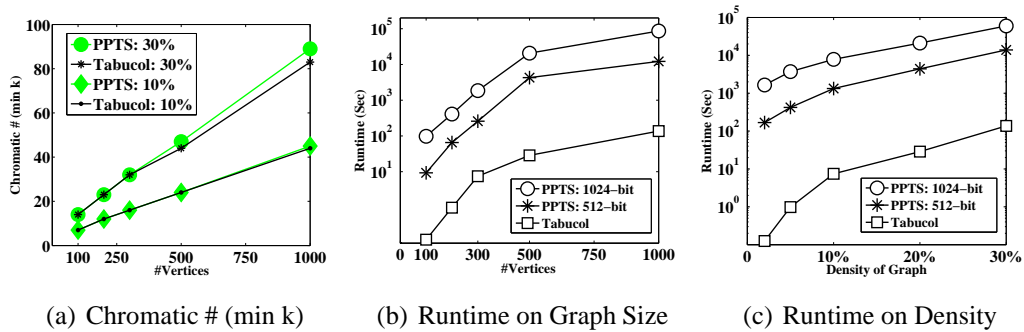


Figure 6.8. Optimal Results and Computation Cost Testing

Moreover, Figure 6.8(a) shows the chromatic number ($\min k$) for the partitioned graphs with different size and density – for each pair of $\#V$ and ρ , we randomly generate 10 graphs, solve them with two algorithms and different k , and choose the $\min k$. The chromatic numbers are almost identical for two algorithms.

Computation Cost. As shown in Figure 6.8(b) and 6.8(c), the runtime of PPTS with 512-bit and 1024-bit are acceptable with polynomial trend, though it consumes approximately 30 hours to securely solve a 10-party collaborative graph coloring problem with 1000 vertices and density 20%. Recall that not every iteration in PPTS requires SMC functions, the efficiency of PPTS can be guaranteed.

6.3 Collaborative Traveling Salesman Problem

Similar to many other optimization problems, the NP-hard problem traveling salesman problem (TSP) also has its collaborative version with *privacy concern*. Sakuma and Kobayashi first studied a two-party collaborative TSP problem in [114]:

Example 6.2 [114] *Two shipping companies E_A and E_B offer different delivery costs between cities. A client E_C wants to decide which shipping company will be employed to deliver their goods to a list of cities in terms of a lower overall transportation cost. However, E_C is reluctant to let both E_A and E_B know its list of cities before signing the contract, and E_A and E_B also do not want to share their delivery cost information to other parties. How can E_C make decision?*

The solution provided in [114] is to let E_C securely solve the two-party TSP problem with E_A and E_B respectively, and then make decision based on the opti-

mal solutions obtained from E_A and E_B . Now, we consider a variant of Example 6.2:

Example 6.3 *Three shipping companies Alice, Bob and Carol provide transportation services between 7 cities with their own shipping methods (Figure 6.9(a)). In reality, they ask for different delivery costs between cities, as shown in Figure 6.9(b), 6.9(c) and 6.9(d). In order to get the minimum freight cost, client E_C can find an optimal route for delivering its goods to a list of cities by working with all the shipping companies. Then, Alice, Bob and Carol will be responsible for only a share of delivery route (since each party charges the fewest for its share, see Figure 6.10). In this case, E_C would like to know the cheapest route and the total cost if possible, however E_C cannot reveal its list of cities to any other party before contracting with the others. Also, three shipping companies would not disclose their cost information to any other party, especially their competitors in any case. Therefore, how can E_C securely get the cheapest traveling route and the total cost from three shipping companies with limited information disclosure?*

More specifically, in the service provider end, Alice, Bob and Carol offer different freight costs, and each party has its own cost for every pair of cities, which is given in its cost vector in Figure 6.9. This is very common because different shipping companies may select different combinations of shipping methods from ground transportation, airplanes, trains and cargo for different cities. In the other end, the client E_C would like to find its best supply chain network for its list of cities by optimizing the entire operation with more than one shipping company. Assume that E_C plan to ship goods to cities: 1, 2, 3, 5, 6 and a solution is given

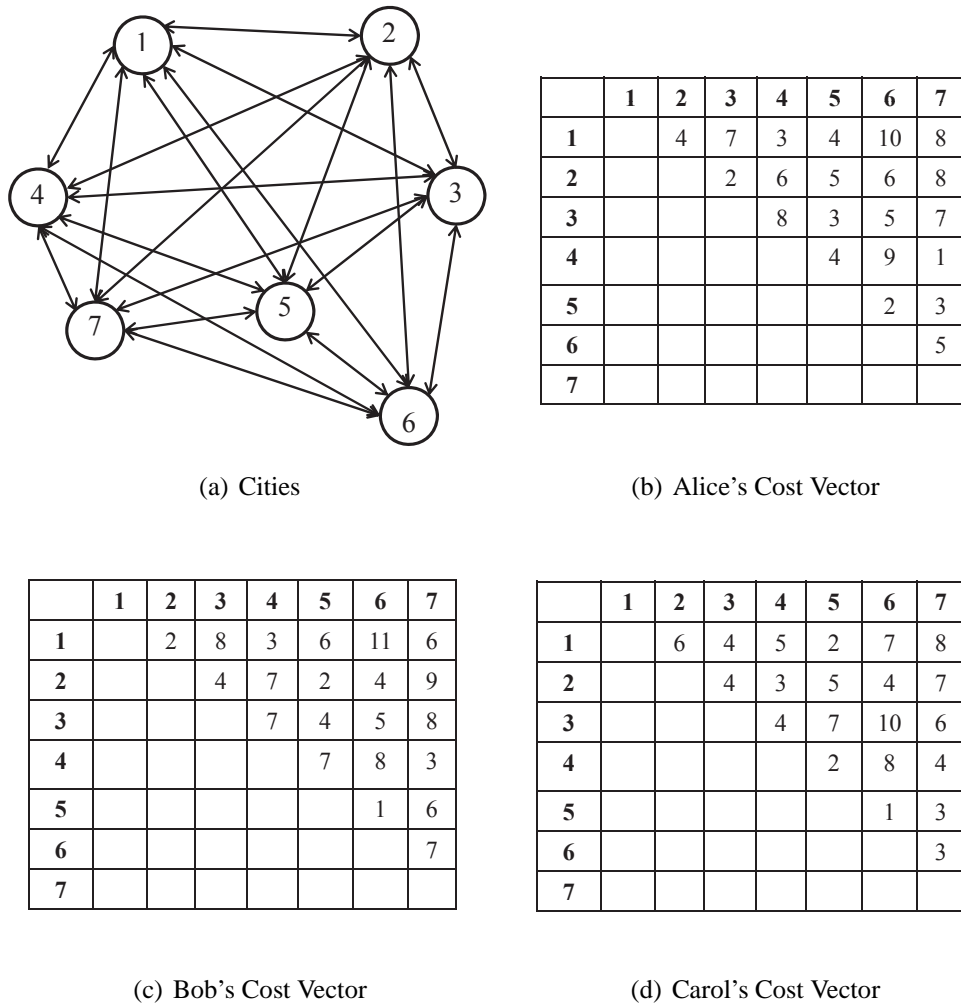


Figure 6.9. Collaborative Shipping Companies

in Figure 6.10(a) as: $1 \rightarrow 5 \rightarrow 6 \rightarrow 2 \rightarrow 3 \rightarrow 1$. In each segment of the route, the shipping company with the minimum freight cost for that segment is chosen. Note that if the minimum shipping cost of any two cities is co-held by two companies, either company will be selected as the service provider by E_C . For instance, Carol serves $1 \rightarrow 5$, Bob or Carol serves $5 \rightarrow 6$, Bob or Carol serves $6 \rightarrow 2$, Alice serves $2 \rightarrow 3$, and Carol serves $3 \rightarrow 1$, as shown in Figure 6.10(a).

Essentially if the client finally employs only one shipping company (Alice, Bob or Carol) to deliver the goods to all the cities on its list, then Example 6.3 becomes Example 6.2. Thus, the problem Sakuma and Kobayashi studied in [114] is a special case of our multiparty collaborative TSP by limiting the supply chain network to be derived from exactly one shipping company. The shipping cost of E_C is indeed not minimized from its economic point of view. Preferably, to additionally cut the cost, E_C can collaborate with as many as three shipping companies and design a supply chain network for its list of cities where each shipping company is responsible for a part of the traveling route.

In sum, we primarily extend the research on collaborative TSP along two dimensions in this section. First, the secure communication protocol for the collaborative TSP is extended to multiparty scenario – can be two or more parties where the client E_C can minimize the overall transportation cost with further co-operation. Second, we build the protocol based on another meta-heuristic solver – Simulated Annealing heuristic, which is more secure against inference attack.

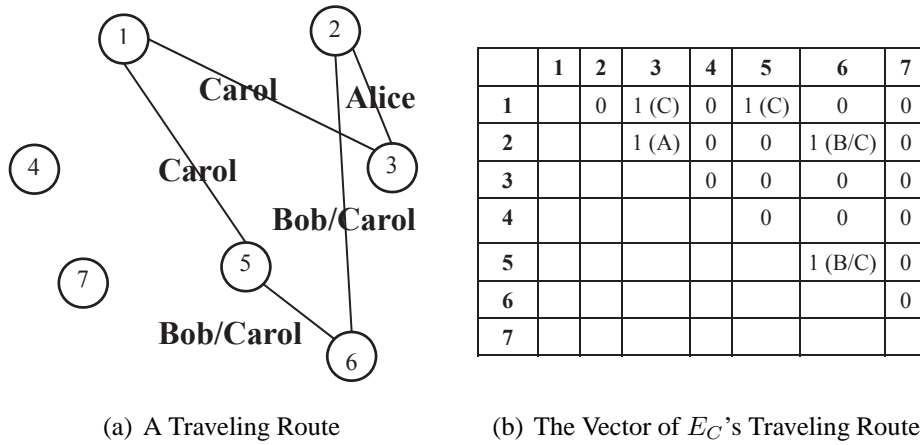


Figure 6.10. Client E_C 's Traveling Route

6.3.1 Problem Definition

In a TSP, $G = (V, E)$ is a complete undirected graph where V and E represent the set of cities and weighted edges (with cost between cities) respectively. Recall that Sakuma and Kobayashi denoted the traveling route as a boolean vector x and the cost of each pair of cities as a non-negative vector y in [114]. Then, the objective function of TSP is expressed as the scalar product of those two vectors: $x \cdot y$.

In the prior work [114], Sakuma and Kobayashi studied their two-party TSP and defined it as $(1, 1)$ -TSP, which includes one client and one shipping company (see Example 6.2). They also discussed two other models without presenting secure solutions: $(k, 1)$ -TSP and $(1, k)$ -TSP in which k means k shipping companies and k clients in two models respectively. More specifically, in $(k, 1)$ -TSP, the cost vector y is partitioned to k vectors for k shipping companies, where the cost vector is given as the sum as the shares $y = \sum_{i=1}^k y_i$. And in $(k, 1)$ -TSP, the traveling route vector x is partitioned to k shares, where $x = \bigcup_{i=1}^k x_i$.

Motivated from Example 6.3 and Figure 6.9, we particularly look at a multi-party TSP model with k shipping companies and 1 client. However, it is different from the $(k, 1)$ -TSP in [114]. We assume that y is not simply partitioned to k shares. Instead, each shipping company holds its own cost vector y_i for n cities. The optimal traveling route will be jointly obtained by k shipping companies and 1 client based on the global minimum freight cost derived from y_1, \dots, y_k and the client's list of cities. Now we formally define the problem, namely $(k, 1)_{\min}$ -TSP as below.

Definition 6.1 ($(k, 1)_{\min}$ -TSP) *Given n cities, k shipping companies provide dif-*

ferent cost vectors for every pair of cities, and the client E_C has a subset of cities that should be visited once (finally visit the original city with a Hamiltonian Cycle). Then, E_C finds the optimal traveling route and the total cost for its cities, such that each shipping company delivers goods for a share of the traveling route and the corresponding delivery cost of each share is the cheapest among all k shipping companies.

Note that if $k = 1$, $(k, 1)_{\min}$ -TSP becomes $(1, 1)$ -TSP [114] since the minimum cost for every share of the route is given by only one shipping company. Thus, we study a more general form of the collaborative TSP than Sakuma and Kobayashi's work [114] where k can be any number. Now, we mathematically model the problem and present a secure solver for the problem.

1) Vectors in $(k, 1)_{\min}$ -TSP.

Given n cities and k shipping companies, we can represent the shipping companies' cost vectors y_1, \dots, y_k for $n(n-1)/2$ pairs of cities as k vectors:

Table 6.3. k Shipping Companies' Cost Vectors

P_1	$y_1 = (y_{(1,2)}^1, \dots, y_{(1,n)}^1, y_{(2,3)}^1, \dots, y_{(2,n)}^1, \dots, y_{(n-1,n)}^1)$
P_2	$y_2 = (y_{(1,2)}^2, \dots, y_{(1,n)}^2, y_{(2,3)}^2, \dots, y_{(2,n)}^2, \dots, y_{(n-1,n)}^2)$
\vdots	\vdots
P_k	$y_k = (y_{(1,2)}^k, \dots, y_{(1,n)}^k, y_{(2,3)}^k, \dots, y_{(2,n)}^k, \dots, y_{(n-1,n)}^k)$

Also, the client E_C 's traveling route vector can be expressed as $x = (x_{(1,2)}, \dots, x_{(1,n)}, x_{(2,3)}, \dots, x_{(2,n)}, \dots, x_{(n-1,n)})$ where

$$x_{(i,j)} = \begin{cases} 1 & e_{ij} \text{ is included in } E_C \text{'s traveling route} \\ 0 & \text{Otherwise} \end{cases} \quad (6.4)$$

Note that the length of vectors x and y_1, \dots, y_k is $d = n(n-1)/2$. For simplicity of notations, we use $j = 1, \dots, d$ to indicate the index of the elements in all the $k+1$ vectors. Therefore, we have a simplified representation for all k shipping companies' cost vectors:

Table 6.4. k Shipping Companies' Cost Vectors (Simplified Notation)

P_1	$y_1 = (y_{11}, y_{12}, y_{13}, \dots, y_{1d})$
P_2	$y_2 = (y_{21}, y_{22}, y_{23}, \dots, y_{2d})$
\vdots	\vdots
P_k	$y_k = (y_{k1}, y_{k2}, y_{k3}, \dots, y_{kd})$

2) Cost Function and Solution (Traveling Route Vector).

As discussed in [114], the cost function of TSP is given as the scalar product in the two-party scenario: $f(x) = x \cdot y = \sum_{i=1}^d x_i y_i$. However, in our model, every "1" in x is assigned to a particular party who provides the same service between two cities yet asks for the minimum cost (we denote this process as "Route Assignment"). Clearly, x can be drilled down to k boolean vectors for k different shipping companies. For example, the traveling route vector in Figure 6.10(b) is drilled down to x_1, x_2, x_3 as shown in Table 6.5.

Table 6.5. Finer-grained Traveling Route Vector with Route Assignment

Overall	$x = (0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0)$
Alice	$x_1 = (0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$
Bob	$x_2 = (0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0)$
Carol	$x_3 = (0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$

In a general fashion, we use $x_{ij} \in \{0, 1\}$ ($i = 1, \dots, k$ and $j = 1, \dots, d$) to denote whether shipping company P_i delivers goods for the j th corresponding

pair of cities or not.

One point is worth noting that only the client E_C would know the traveling route vector in our secure $(k, 1)_{\min}$ -TSP model. For every traveling route, E_C can choose a particular shipping company to deliver its goods for every segment of the route. Hence, E_C splits length- d vector x to k length- d vectors x_1, \dots, x_k according to the selection of the shipping companies. And $\{x_1, \dots, x_k\}$ is then referred as the solution of our $(k, 1)_{\min}$ -TSP model, which is finer-grained than x .

Also, $\forall i \in [1, k], x_i$ is the solution vector (and the variables) assigned for shipping company P_i . Given $\{x_1, \dots, x_k\}$, the cost function of the $(k, 1)_{\min}$ -TSP can be derived as $f(x_1, \dots, x_k) = \sum_{i=1}^k x_i \cdot y_i = \sum_{i=1}^k (\sum_{j=1}^d x_{ij} y_{ij})$. All parties jointly minimize it.

3) Complexity and Heuristics.

We now study the complexity of the $(k, 1)_{\min}$ -TSP.

Theorem 6.3 *The decision problem of $(k, 1)_{\min}$ -TSP is NP-complete.*

Proof. Since Hamiltonian Circuit problem is a special case of the original TSP, the decision problem of the original TSP can be proven to be NP-complete by building the complete graph $G = (V, E)$ for all cities/vertices based on the graph for the Hamiltonian Circuit problem $G' = (V, E')$ (letting cost of one edge be 1 if such edge exists in G' ; otherwise, let the cost of the edge be 2). Thus, whether there exists a route to visit all the vertices once in the G with a total cost cheaper than n (the number of cities/vertices) is equivalent to if there exists a Hamiltonian circuit in G' [103]. Similarly, for any cost threshold for the decision, G can be

formulated with different weights/costs to obtain the equivalence between TSP and Hamiltonian Circuit problem.

For $(k, 1)_{\min}$ -TSP, the complete graph $G = (V, E)$ is the same as the original TSP. The only difference is that every edge in G has k possible weights/costs. If we determine the weight/cost as any number out of the k choices in G , the problem turns into the original TSP. Thus, we can also prove that the decision problem of $(k, 1)_{\min}$ -TSP is NP-complete using the Hamiltonian circuit problem.

Theorem 6.4 $(k, 1)_{\min}$ -TSP is NP-hard.

Proof. Assume that the maximum weight/cost of the complete graph $G = (V, E)$ is max (out of k choices for every edge). Finding the traveling route with minimum overall cost in the original TSP can be completed by solving $O(\log(max)) + O(n^2)$ instances of the decision problem [1]. This also applies to $(k, 1)_{\min}$ -TSP because we can bound the total cost. Thus, $(k, 1)_{\min}$ -TSP is NP-hard since it can be polynomially reducible to the NP-complete problem. The proof is completed.

Apparently, while solving $(k, 1)_{\min}$ -TSP, efficiency is our primary concern besides security since the security enforcement requires large amounts of computation already. It is not so worthwhile to securely finding the “exact” optimal solution by sacrificing too much efficiency. We thus build protocol based on the meta-heuristic – *Simulated Annealing* to get a sufficiently good solution quickly for two reasons. First, simulated annealing has been shown to be an effective and efficient solver for TSP. Second, simulated annealing is more secure against inference attack than other meta-heuristics such as tabu search since it updates the

current solution with not only better solutions but also worse solutions – it depends on the probability derived at each state. Thus, it is more difficult for every party to infer information from different messages received by them. The details will be give later on.

6.3.2 Privacy-preserving Simulated Annealing

As stated above, we build the secure solver for the $(k, 1)_{\min}$ -TSP on the basis of simulated annealing. Due to the meta-heuristic nature, simulated annealing iteratively searches neighboring solutions and updates the current solution until a near-optimal solution is found. Note that the criterion of moving to another solution is the probability derived from the current solution and the neighboring solution in traditional simulated annealing. Thus, every iteration in the simulated annealing over $k + 1$ collaborative parties is implemented as below:

Statement 6.1 *In every iteration, client E_C proposes a solution $x = \{x_1, \dots, x_k\}$ and its neighboring solution $x' = \{x'_1, \dots, x'_k\}$ to all parties, then all parties jointly compute the energy of two solutions $f(x_1, \dots, x_k)$ and $f(x'_1, \dots, x'_k)$, and finally compute the probability of moving the solution from x to x' :*

$$Prob = \min\{1, \exp(-\frac{f(x'_1, \dots, x'_k) - f(x_1, \dots, x_k)}{T})\} \quad (6.5)$$

The entire process is secured in privacy-preserving simulated annealing, where no party knows such probability.

Building Blocks

Consequently, similar to the SMC protocol presented in the previous sections, we first discuss the fundamental building blocks which need to be frequently called.

- **Secure Scalar Product.** Since the cost function of $(k, 1)_{\min}$ -TSP is given as $f(x_1, \dots, x_k) = \sum_{i=1}^k x_i \cdot y_i$ where client E_C holds x_1, \dots, x_k and party $P_i, i = 1, \dots, k$ holds y_i . With the secure scalar product protocol in Algorithm 12, $f(x_1, \dots, x_k)$ can be securely computed and let each party hold a share of the output.

Algorithm 12: Secure Scalar Product

Input : Client E_C 's finer-grained solution x_1, \dots, x_k , k shipping companies P_1, \dots, P_k 's cost vectors y_1, \dots, y_k

Output: $f(x_1, \dots, x_k) = \sum_{i=1}^k x_i \cdot y_i$ co-held by all $k + 1$ parties

- 1 Client E_C creates a public and private key pair (pk, sk) ;
- 2 E_C encrypts x_1, \dots, x_k to $Enc_{pk}(x_1), \dots, Enc_{pk}(x_k)$ with is pk ;
- 3 E_C sends $Enc_{pk}(x_1), \dots, Enc_{pk}(x_k)$ and pk to all remaining k parties P_1, \dots, P_k respectively;
- 4 **for each party** $P_i, i = 1, \dots, k$ **do**
- 5 P_i generates a random integer r_i , encrypts the scalar product as:
 $Enc_{pk}(x_i \cdot y_i) * Enc_{pk}(r_i)$ with the public key pk , and sends
 $Enc_{pk}(x_i \cdot y_i) * Enc_{pk}(r_i)$ back to E_C ;
- 6 E_C decrypts $Enc_{pk}(x_i \cdot y_i) * Enc_{pk}(r_i)$ with its private key sk and obtain a random share $s_i = x_i \cdot y_i + r_i$;

/* E_C privately holds $\forall i \in [1, k], s_i = x_i \cdot y_i + r_i$, and
 $\forall i \in [1, k], P_i$ privately holds $-r_i$. Then, the sum
of all the shares is $f(x_1, \dots, x_k) = \sum_{i=1}^k x_i \cdot y_i$. */

- **Secure Comparison.** At current temperature T , Equation 6.5 is utilized to determine whether the solution $x = \{x_1, \dots, x_k\}$ should be moved to $x' = \{x'_1, \dots, x'_k\}$ in simulated annealing. Similar to privacy-preserving tabu search, we still employ FairplayMP [12] to securely compare the out-

put of two functions. More specifically, $\min\{1, \exp(-\frac{f(x'_1, \dots, x'_k) - f(x_1, \dots, x_k)}{T})\}$ and a random number $\eta \in [0, 1)$ are requested to be compared while implementing simulated annealing with Metropolis-Hastings algorithm [94]: if $\min\{1, \exp(-\frac{f(x'_1, \dots, x'_k) - f(x_1, \dots, x_k)}{T})\} > \eta$, then move $x = \{x_1, \dots, x_k\}$ to $x' = \{x'_1, \dots, x'_k\}$; else, do not move. We thus need to securely compare: $f(x_1, \dots, x_k) - f(x'_1, \dots, x'_k) > T \log \eta$ with FairplayMP [12], where the inputs for $f(x_1, \dots, x_k)$ and $f(x'_1, \dots, x'_k)$ are provided by all $k + 1$ parties. Equivalently, if $f(x_1, \dots, x_k) - f(x'_1, \dots, x'_k) > T \log \eta$, the client E_C moves $x = \{x_1, \dots, x_k\}$ to $x' = \{x'_1, \dots, x'_k\}$ no matter x' is better than x or not; else, do not move.

Note that if the temperature T is lowered, simulated annealing algorithm only accepts moving from x to worse solution x' with closer $f(x'_1, \dots, x'_k)$ and $f(x_1, \dots, x_k)$. This improves the performance of the meta-heuristic.

Top-down Simulated Annealing in $(k, 1)_{\min}$ -TSP

Another important point is to define the neighboring solution in $(k, 1)_{\min}$ -TSP. Different from the traditional TSP, $(k, 1)_{\min}$ -TSP has two categories of neighboring solutions since k possible candidate weights (or costs) are available for every edge (or every pair of cities) in G . First, like the well-known TSP, we can find the neighboring solution x' based on the permutation of the route, e.g., $x = 1 \rightarrow 2 \rightarrow 3$ and $x' = 1 \rightarrow 3 \rightarrow 2$. Second, for every solution, e.g., $x = 1 \rightarrow 2 \rightarrow 3$, each route segment ($1 \rightarrow 2$ and $2 \rightarrow 3$) can be assigned to Alice, Bob or Carol (“Route Assignment”). Then, $x = 1 \rightarrow 2 \rightarrow 3$ (with a particular route assignment) has $3 \times 3 - 1 = 8$ neighboring solutions by choosing different

combinations of service providers in the finer-grained solution x_1, \dots, x_k . Indeed, if y_1, \dots, y_k are known to the client E_C , E_C can simply complete the route assignment by selecting the shipping company with the lowest cost on each segment of the route. However, in secure $(k, 1)_{\min}$ -TSP model, such information is prohibitive to disclose. Therefore, we have to run meta-heuristics again to find the optimal route assignment for every solution (traveling route) traversed in simulated annealing.

In summary, while running privacy-preserving simulated annealing, for every new solution x , bottom-level simulated annealing will be conducted to search the neighborhoods in “Route Assignment”. As soon as a near-optimal solution of “Route Assignment” for any solution is found, such solution will be updated as the traveling route vector with its optimal route assignment. Meanwhile, for x ’s neighboring solution x' (a permuted traveling route), top-level simulated annealing is required to search the optimal traveling route vector while another bottom-level simulated annealing is required for the corresponding route assignment. Therefore, the entire search method is referred as “Top-down simulated annealing” in our approach.

Secure Communication Protocol

The secure communication protocol guarantees that all parties jointly implement the top-down simulated annealing without revealing any private information. Algorithm 13 shows the detailed steps of the secure communication protocol – privacy-preserving simulated annealing, and we summarize some of its conceptual characteristics as below:

Algorithm 13: Privacy-preserving Simulated Annealing

Input : Client E_C 's initial traveling route x ; P_1, \dots, P_k 's cost vectors y_1, \dots, y_k ;
 Initial temperature T_1, T_2 ; Cooling coefficient ρ_1, ρ_2 ; $\eta \in [0, 1]$

Output: Near-optimal route $best\{x\}$ and the route assignment $best\{x_1, \dots, x_k\}$

```

1  $iter_1 \leftarrow 0$ ;  $best\{x\} \leftarrow x$ ;  $best\{x_1, \dots, x_k\} \leftarrow \{x_1, \dots, x_k\}$ ;
  /* Top-level simulated annealing for searching the
    optimal route with optimal route assignment */
2 while  $iter_1 < max\_iter_1$  do
3    $iter_2 \leftarrow 0$ ;
  /* Bottom-level simulated annealing for the optimal
    route assignment of  $best\{x\}$  */
4   Client  $E_C$  drills down  $best\{x\}$  with a random route assignment:  $\{x_1, \dots, x_k\}$ ;
5    $best_1\{x_1, \dots, x_k\} \leftarrow \{x_1, \dots, x_k\}$ ;
6   while  $iter_2 < max\_iter_2$  and  $route\_assigned(best\{x\}) = 0$  do
7     Client  $E_C$  gets  $best_1\{x_1, \dots, x_k\}$ 's random neighboring route
      assignment:  $\{x'_1, \dots, x'_k\}$ ;
8     Call Algorithm 12 twice to securely compute  $f(x_1, \dots, x_k) = \sum_{i=1}^k x_i \cdot y_i$ 
      and  $f(x'_1, \dots, x'_k) = \sum_{i=1}^k x'_i \cdot y_i$  among all  $k + 1$  parties;
9     Call secure comparison (FairplayMP [12]) to compare
       $f(x_1, \dots, x_k) - f(x'_1, \dots, x'_k)$  and  $T_2 \log \eta$  among all  $k + 1$  parties;
10    if  $f(x_1, \dots, x_k) - f(x'_1, \dots, x'_k) > T_2 \log \eta$  then
11       $best_1\{x_1, \dots, x_k\} \leftarrow \{x'_1, \dots, x'_k\}$ ;
12     $iter_2++$ ;
13     $T_2 \leftarrow \rho_2 T_2$  (cooling down after several iterations);
14     $route\_assigned(best\{x\}) = 1$ ;
15    Client  $E_C$  gets a random neighboring route of  $best\{x\}$ :  $x'$ ;
  /* Bottom-level simulated annealing for the optimal
    route assignment of  $x'$  */
16    All  $k + 1$  parties repeat Step 3-13 to obtain the near-optimal route assignment
      for  $x'$ :  $best_2\{x_1, \dots, x_k\}$  and  $route\_assigned(x') = 1$ ;
17    Call Algorithm 12 twice to securely compute  $f(best_1\{x_1, \dots, x_k\})$  and
       $f(best_2\{x_1, \dots, x_k\})$  among all  $k + 1$  parties;
18    Call secure comparison (FairplayMP [12]) to compare  $f(best_1\{x_1, \dots, x_k\}) -$ 
       $f(best_2\{x_1, \dots, x_k\})$  and  $T_1 \log \eta$  among all  $k + 1$  parties;
19    if  $f(best_1\{x_1, \dots, x_k\}) - f(best_2\{x_1, \dots, x_k\}) > T_1 \log \eta$  then
20      Client  $E_C$  update:  $best\{x\} \leftarrow x'$ ;  $route\_assigned(best\{x\}) = 1$ ;
21       $best\{x_1, \dots, x_k\} \leftarrow best_2\{x_1, \dots, x_k\}$ ;
22     $iter_1++$ ;
23     $T_1 \leftarrow \rho_1 T_1$  (cooling down after several iterations);
24 return  $best\{x\}$ ,  $best\{x_1, \dots, x_k\}$  and  $f(best\{x\})$  (securely summed);

```

- The top-level simulated annealing searches the optimal traveling route vector where the embedded bottom-level simulated annealing is called to search the optimal route assignment for every solution. More specifically, for any traveling route x and its neighboring solution x' (permuted route) in the top-level simulated annealing, their corresponding optimal route assignments are found by bottom-level simulated annealing respectively. Thus, the current best route (with its optimal route assignment) moves towards the optimal traveling route (with the optimal route assignment) of the $(k, 1)_{\min}$ -TSP.
- To compute the energy of any solution, all parties securely compute the scalar product using Algorithm 12, and each party holds a share of the result.
- To determine whether move or not, all parties implement FairplayMP [12] to securely compare the functions with their input shares. Finally, only the client E_C knows the comparison result and whether move or not.
- For top and bottom-level simulated annealing, we use different sensitivity parameters: initial temperature T_1 and T_2 , cooling coefficient ρ_1, ρ_2 (note that the temperature will be lowered after several iterations). We also setup different maximum number of iterations for top and bottom-level simulated annealing respectively max_iter_1 and max_iter_2 .
- In order to reduce computational complexity, we define an indicator $route_assigned(x) \in \{0, 1\}$ to avoid running the route assignment for every route x twice in the protocol – as a neighboring solution and the current solution respectively. If the optimal route assignment for x has been found in previous iterations, we let $route_assigned(x) = 1$; Otherwise,

$route_assigned(x) = 0$. As shown in Step 6, we examine the status of $route_assigned(x)$ before going to the bottom-level simulated annealing. Then, the efficiency can be improved.

- Note that *best* in the protocol does not mean the current solution is the best solution so far since simulated annealing allows moving to a worse solution with a reasonable probability for evading getting stuck in local optimum.
- In Step 22, E_C and k shipping companies can securely sum the shares of near-optimal cost and let E_C know the result $f(best\{x\})$, which is the minimum cost in the $(k, 1)_{\min}$ -TSP. We only need to simply implement the secure sum with Paillier's Homomorphic cryptosystem [101], such that E_C can only learn the sum of the shares. It is straightforward, so we do not discuss this in this section.

6.3.3 Security and Cost Analysis

Security Analysis

For analyzing the security of the protocol, we must build a polynomial machine to simulate the view of every party with SMC theory.

Theorem 6.5 *Privacy-preserving simulated annealing protocol (Algorithm 13) reveals only the near-optimal traveling route $best\{x\}$, the route assignment $best\{x_1, \dots, x_k\}$ and the minimum traveling cost $f(best\{x\})$ to the client E_C .*

Proof. We first look at the steps that do not need communication between different parties in the protocol. Notice that all the neighboring solutions are proposed by E_C in the top-down simulated annealing, then most of the steps are

locally implemented by E_C , e.g., finding the neighboring solution (either the traveling route or the route assignment), updating the current solution based on the comparison, and reducing the temperature in simulated annealing. These steps can be simulated by simply executing those steps.

In addition, we must simulate each party's view (all the received messages in the protocol) that requires communication in polynomial time. More specifically, the client E_C and k shipping companies P_1, \dots, P_k iteratively communicate with each other in Secure Scalar Product (Algorithm 12) and Secure Comparison. We now examine the messages received by each party.

- *Client E_C 's view:* First, while calling the secure scalar product computation every time, E_C receives k encrypted random shares. k random shares are the actual messages received by E_C in those steps. W.l.o.g., E_C gets $s_i = x_i \cdot y_i + r_i$ from shipping company P_i . All the random shares generated in all iterations can be simulated by generating a random from the *uniform probability distribution over \mathcal{F}* , assuming that s_i is scaled to fixed precision over a closed field, enabling such a selection. Thus, $\text{Prob}[s_i = t] = \text{Prob}[r_i = t - s_i] = \frac{1}{\mathcal{F}}$, and all the shares can be simulated in polynomial time.

Second, E_C receives a series of comparison results from FairplayMP. To simulate the sequence of comparison results (" $>$ " or " \leq "), the inverse step of the simulated annealing can be utilized. Specifically, E_C starts from the near-optimal traveling route, and then finds the given neighboring solutions in sequence by running the top-down simulated annealing inversely (note

that temperature increase can be imposed to tune the sensitivity of the moving probability in the inverse optimization). While comparing $f(x'_1, \dots, x'_k) - f(x_1, \dots, x_k)$ and $T_2 \log \eta$ in the bottom-level simulated annealing, if the result is “>”, the simulator outputs an “1”, otherwise “0”. Now we discuss how to simulate them in polynomial time.

Recall that all the searched solutions are known to E_C in sequence, but the energy of any state (which is the sum of the local random shares) is unknown to E_C since E_C does not know the cost vectors from each shipping company. Fortunately, since E_C knows its final traveling route $best\{x\}$ and the minimum traveling cost $f(best\{x\})$, we can use the same simulator in [125] to simulate a (minimum) cost function in polynomial time. Then, the energy of two compared states (solutions) can be polynomially simulated as well simply because both solutions are regarded as input for E_C . Consequently, we can simulate “1” or “0” for two reasons: 1) the probability of generating each of them is deterministic with Equation 6.5, and 2) another parameter η is uniformly distributed in $[0, 1)$. Therefore, a sequence of such comparison results in E_C ’s view can be simulated in polynomial time. Similarly, the sequence of comparison results for top-level simulated annealing can be simulated with the same polynomial machine.

In summary, client E_C learns only the near-optimal traveling route $best\{x\}$, the route assignment $best\{x_1, \dots, x_k\}$ and the minimum traveling cost $f(best\{x\})$ from the protocol.

- *Shipping Company $\forall i \in [1, k], P_i$ ’s View:* In the protocol, every shipping company only receives the random shares in secure scalar product compu-

tation and E_C 's public key pk . As analyzed above, the random shares can be simulated in polynomial time using the same machine as E_C 's random share. Hence, every shipping company only learns the public key pk in the protocol.

This completes the proof.

Besides the protocol security guaranteed by the SMC theory, it would be value-added if we can also resolve the privacy leakage resulting from inference of the messages. Recall that we have to impose synchronous move to resolve the inference attack occurred in the graph coloring with tabu search, and synchronous move slightly compromises efficiency. However, simulated annealing provides excellent mechanism to naturally mitigate the inference attack in every move. Specifically, unlike tabu search, simulated annealing allows moving from the current solution to its *worse neighboring solution*, and the overall cost is unrevealed in any solution, thus it becomes difficult for E_C to learn which solution outperforms its neighboring solution at this time. Note that if we build the protocol based on tabu search, E_C can claim that the neighboring solution is better than the current solution in any move.

Precisely speaking, simulated annealing cannot absolutely eliminate the inference attack. E_C can still obtain some information from other parties in the protocol. For example, from the near-optimal solution, E_C can guess that which company may provide the cheapest freight service for a particular segment of the traveling route (since this is supposed to be optimized in the route assignment). Since such information is trivial and do not hurt any other party, the security of

the protocol is not impacted.

Cost Analysis

Given a $(k, 1)_{\min}$ -TSP with n cities, top-down simulated annealing is implemented in the secure communication protocol. In top and bottom level simulated annealing, the maximum number of iterations is given as max_iter_1 and max_iter_2 respectively. For simplicity of notation, we denote them as $O(m)$. We now discuss the communication and computation cost required in the protocol.

Communication Cost. In the protocol, only secure scalar product computation and secure comparison require communication amongst $k + 1$ parties. First, while calling the secure scalar product computation, it needs one round communication between E_C and each shipping company. Then, the communication cost of total secure scalar product computation is $O(2m^2 * k * n(n - 1)/2) = O(m^2 n^2 k)$ messages of bit communication. Second, the number of communication messages in every secure comparison is equal to the number of computing parties [62]. Then, the communication cost of total secure comparison is $O(m(2m + 1)k) = O(m^2 k)$ messages of bit communication. Moreover, public key pk is delivered from E_C to the remaining k parties (every party can use the same public key in all iterations, thus pk can be considered as offline cost). Therefore, the communication complexity of the protocol is $O(m^2 n^2 k)$.

Computation Cost. First, E_C locally finds the neighboring solution, moves the solution and updates the temperature in both top and bottom-level simulated annealing. The computation cost of the above process is ignorable compared to encryption/decryption work. Second, if we estimate the runtime for a single se-

cure scalar product computation and a single secure comparison as t_s and t_c respectively, the total computation cost based on cryptography can be written as

$$(2m^2 * k) * t_s + m(2m + 1) * k * t_c \approx 2m^2k(t_s + t_c).$$

CHAPTER 7

CONCLUSION AND FUTURE WORK

This dissertation studies the security and privacy issues in some fundamental optimization problems which are jointly formulated and solved by different parties in real-world. First, we address various practical problems existing in different variants of the collaborative linear programming problems. More specifically, the prior work on protecting the privacy in horizontally partitioned LP [75, 90] have been detected to be vulnerable to inference attack. We then resolve such attack with our inference approach. For arbitrarily partitioned LP problem, we initially work on the privacy issue in the semi-honest adversarial model, and then look at the cheating and collusion that would potentially occur among all the participants, namely malicious adversarial model. Finally, secure and incentive compatible protocols are proposed to guarantee that the arbitrarily partitioned LP can be solved with limited information disclosure and significantly mitigated self-interested behavior. Second, we extend the research to some other well-known optimization models beyond LP problems, including non-linear programming, graph coloring and traveling salesman problem. We particularly discuss the potential privacy concern in at least one practical cooperative scenario involving different organizations. In order that they can be solved with limited information leakage, we propose three efficient communication protocols by securing some state of art

techniques such as linear approximation, tabu search and simulated annealing.

Note that we present formal analysis on security/privacy and the communication/computation cost, as well as extensive experimental results in this dissertation to justify the applicability, accessibility and scalability of our proposed approaches. Being widely known, it is extremely difficult to give a generic privacy notion for all collaborative optimization problems due to their computational hardness. Instead, we define several privacy measures for those problems we studied, such as rigorously quantified information leakage with SMC theory, measured trivial information disclosure in transformation, and probability of inferring information. Also, when necessary, we compare our approaches with the prior work in terms of efficiency, though the data partition scenario may vary.

The aim of this dissertation is to open the window for this broad and promising research area. There are a lot of research directions remaining:

First, the security and privacy concern in many other well-known optimization problems have not been investigated yet. In general, non-linear optimization provides fundamental insights into mathematical analysis and is widely used in a variety of fields such as engineering design, regression analysis, inventory control, geophysical exploration, and economics. As a special case of non-linear programming, quadratic programming comprises an area of optimization whose broad range of applicability is second only to linear programs. Moreover, constraint satisfaction is one of the most important problems in AI and industrial decision problems. These open even more opportunities for us to further explore this area.

Second, different collaborative scenarios of the same optimization problem should have thoroughly distinct privacy concern and may need different solvers. Formally stating, every optimization problem can be partitioned to many formulations since all the ingredients of the problem might be held by more than one party. Take the knapsack problem for example, the problem can be horizontally partitioned by assuming that every party holds a sack to bound some items' capacities in one dimension. Contrarily, the problem can be vertically partitioned by assuming that every party holds a set of items and all the parties jointly hold the sack or multiple sacks. Therefore, it is also worth exploring different methods for various partitioned variants of the above mentioned optimization problems, particularly the non-linear optimization problems.

Third, since many optimization problems are utilized to allocate shared resources, the participants in many collaborative optimization problems have potential selfish intent to gain more economic benefits from each other by breaking the communication protocol. For instance, like the LP problem we have studied, every shipping companies in our $(k, 1)_{\min}$ -TSP model would like to solicit the client to use its delivery service only, then they may cheat by faking their cost vector or collude with another company to monopolize the delivery service. Moreover, we have mitigated the cheating and collusion in for the arbitrarily partitioned LP problem in Chapter 5, however the incentive compatible protocol still suffers the input cheating attack. For such LP and $(k, 1)_{\min}$ -TSP model, we intend to tackle the input cheating issue with advanced game theoretical techniques and/or audit mechanisms in the near future. Also, for every new collaborative optimization model, we will address the advanced adversarial model and guarantee the security

against attackers with effective and efficient solvers.

Fourth, apart from working on the security enforcement, we will try to find more efficient approaches for solving each collaborative optimization problem model without compromising security. A typical method of doing that is to securely outsource the computation to an external party or the cloud. Two challenging points exist in such approach: 1) finding mathematical transformation methods to generate the outsourced problem, and 2) building sufficiently secure protocol to transform and outsource the computation. Ideally, each party only obtains its share of the optimal solution from the protocol. In essence, iteratively securing each step of the optimization model brings considerable communication overheads and expensive computation cost to the protocol. If a correct secure transformation can be found, only one round communication is required and most of the computation cost will be considered as offline cost. Then, the efficiency can be significantly improved. This is also a potential research direction in privacy-preserving collaborative optimization. Meanwhile, the security, privacy, accuracy and efficiency might have different notion and tradeoff for the collaborative optimization underlying the cloud computing environment. We will explore them in the future.

Finally, since optimization problems are fundamental, almost all the research areas need all kinds of optimization models to solve real-world problems. By extending the research in this dissertation, we can improve the performance of the optimization-based approach in other research areas with value-added benefits. For instance, if a new optimization model is proposed in query log anonymization [50, 60] to maximize the output utility, we can develop a novel secure opti-

mization technique for such model to enable collaborative anonymization, which improves the output utility by better utilizing the distributed resources from different organizations without violating their privacy concern [58]. This applies to many research areas – to the best of my knowledge, some of my other research areas such as data mining [56, 83–86, 93] and health informatics [51, 52, 82, 87] still have considerable space to improve the performance without sacrificing the privacy requirement. Therefore, we also intend to extend the methodologies in this dissertation to those research areas in the future.

REFERENCES

- [1] http://www.proofwiki.org/wiki/traveling_salesman_problem_is_np-complete.
- [2] AGRAWAL, D., AND AGGARWAL, C. C. On the design and quantification of privacy preserving data mining algorithms. In *Proceedings of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (Santa Barbara, California, USA, May 21-23 2001), ACM, pp. 247–255.
- [3] AGRAWAL, R., AND SRIKANT, R. Privacy-preserving data mining. In *Proceedings of the 2000 ACM SIGMOD Conference on Management of Data* (Dallas, TX, May 14-19 2000), ACM, ACM, pp. 439–450.
- [4] AHUJA, R. K., AND ORLIN, J. B. Inverse optimization. *Operations Research* 49, 5 (Sept.–Oct. 2001), 771–783.
- [5] ATALLAH, M. J., BLANTON, M., FRIKKEN, K. B., AND LI, J. Efficient correlated action selection. In *Proceedings of Financial Cryptography* (2006).
- [6] ATALLAH, M. J., DESHPANDE, V., ELMONGUI, H. G., AND SCHWARZ, L. B. Secure supply-chain protocols. In *Proceedings of the 2003 IEEE*

International Conference on E-Commerce (Newport Beach, California, June 24–27 2003).

- [7] AWERBUCH, B., AND AZAR, Y. Local optimization of global objectives: Competitive distributed deadlock resolution and resource allocation. In *IEEE Symposium on Foundations of Computer Science* (1994), pp. 240–249.
- [8] BÁRÁNY, I. Fair distribution protocols or how the players replace fortune. *Math. Oper. Res.* 17, 2 (1992), 327–340.
- [9] BARTAL, Y., BYERS, J. W., AND RAZ, D. Fast, distributed approximation algorithms for positive linear programming with applications to flow control. *SIAM Journal on Computing* 33, 6 (2004), 1261–1279.
- [10] BEDNARZ, A. *Methods for Two-party Privacy-preserving Linear Programming*. PhD thesis, The University of Adelaide, Adelaide, Australia, 2012.
- [11] BEDNARZ, A., BEAN, N., AND ROUGHAN, M. Hiccups on the road to privacy-preserving linear programming. In *Proceedings of the 8th ACM workshop on Privacy in the electronic society* (New York, NY, USA, 2009), WPES '09, ACM, pp. 117–120.
- [12] BEN-DAVID, A., NISAN, N., AND PINKAS, B. Fairplaymp: a system for secure multi-party computation. In *Proceedings of the 15th ACM conference on Computer and communications security* (New York, NY, USA, 2008), CCS '08, ACM, pp. 257–266.

- [13] BEN-OR, M., GOLDWASSER, S., AND WIGDERSON, A. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing* (1998), pp. 1–10.
- [14] BEN-PORATH, E. Correlation without mediation: Expanding the set of equilibrium outcomes by “cheap” pre-play procedures. *Journal of Economic Theory* 80, 1 (1998), 108 – 122.
- [15] BENALOH, J., AND TUINSTRA, D. Receipt-free secret-ballot elections (extended abstract). In *STOC '94: Proceedings of the twenty-sixth annual ACM symposium on Theory of computing* (New York, NY, USA, 1994), ACM Press, pp. 544–553.
- [16] BERTSIMAS, D., AND TSITSIKLIS, J. Simulated annealing. *Statistical Science* 8, 1 (1993), 10–15.
- [17] BRANDT, F. *Fundamental Aspects of Privacy and Deception in Electronic Auctions*. Doctoral Thesis, Department for Computer Science, Technical University of Munich, 2003.
- [18] BRANDT, F., AND SANDHOLM, T. Efficient privacy-preserving protocols for multi-unit auctions. In *Proceedings of 9th International Conference on Financial Cryptography and Data Security* (Aug. 2005), vol. 3570 of *Lecture Notes in Computer Science*, pp. 298 – 312.
- [19] BUSS, D. Case study: Land o’lakes and collaborative logistics. *Online, World Wide Web*, <http://www.cioinsight.com/c/a/Case-Studies/Case-Study->

Land-OLakes-and-Collaborative-Logistics/.

- [20] CANETTI, R. Asynchronous secure computation. Tech. Rep. 755, CS Department, Technion, 1992.
- [21] CANETTI, R., AND GENNARO, R. Incoercible multiparty computation. In *FOCS '96: Proceedings of the 37th Annual Symposium on Foundations of Computer Science* (Washington, DC, USA, 1996), IEEE Computer Society, p. 504.
- [22] CANETTI, R., KUSHILEVITZ, E., AND LINDELL, Y. On the limitations of universally composable two-party computation without set-up assumptions. *Journal of Cryptology* 19, 2 (2006), 135–167. An extended abstract appeared in Eurocrypt 2003, Springer-Verlag (LNCS 2656), pages 68-86, 2003.
- [23] CANETTI, R., LINDELL, Y., OSTROVSKY, R., AND SAHAI, A. Universally composable two-party and multi-party secure computation. In *STOC '02: Proceedings of the thirty-fourth annual ACM symposium on Theory of computing* (New York, NY, USA, 2002), ACM Press, pp. 494–503.
- [24] CATRINA, O., AND DE HOOGH, S. Secure multiparty linear programming using fixed-point arithmetic. In *ESORICS* (2010), pp. 134–150.
- [25] CHAUM, D., CRÉPEAU, C., AND DAMGARD, I. Multiparty unconditionally secure protocols. In *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing* (New York, NY, USA, 1988), ACM Press, pp. 11–19.

- [26] CLIFTON, C., IYER, A., CHO, R., JIANG, W., KANTARCIOGLU, M., AND VAIDYA, J. An approach to identifying beneficial collaboration securely in decentralized logistics systems. *Management & Service Operations Management* (submitted).
- [27] CLIFTON, C., KANTARCIOGLU, M., LIN, X., VAIDYA, J., AND ZHU, M. Tools for privacy preserving distributed data mining. *SIGKDD Explorations* 4, 2 (Jan. 2003), 28–34.
- [28] DANTZIG, G. *Linear Programming and Extensions*. Princeton University Press, 1963.
- [29] DENG, X., AND PAPADIMITRIOU, C. H. Distributed decision-making with incomplete information. In *Proceedings of the 12th IFIP Congress* (1992).
- [30] DODIS, Y., HALEVI, S., AND RABIN, T. A cryptographic solution to a game theoretic problem. In *CRYPTO '00: Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology* (London, UK, 2000), Springer-Verlag, pp. 112–130.
- [31] DREIER, J., AND KERSCHBAUM, F. Practical privacy-preserving multiparty linear programming based on problem transformation. In *Social-Com/PASSAT* (2011), pp. 916–924.
- [32] DU, W. *A Study of Several Specific Secure Two-party Computation Problems*. PhD thesis, Purdue University, West Lafayette, Indiana, 2001.

- [33] DU, W., AND ATALLAH, M. J. Privacy-preserving cooperative scientific computations. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop* (2001), pp. 273–282.
- [34] DU, W., AND ZHAN, Z. Building decision tree classifier on private data. In *IEEE International Conference on Data Mining Workshop on Privacy, Security, and Data Mining* (Maebashi City, Japan, Dec. 9 2002), C. Clifton and V. Estivill-Castro, Eds., vol. 14, Australian Computer Society, pp. 1–8.
- [35] EVFIMIEVSKI, A., SRIKANT, R., AGRAWAL, R., AND GEHRKE, J. Privacy preserving mining of association rules. In *The Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (new York, NY, July 23-26 2002), ACM, pp. 217–228.
- [36] FENG, X., AND ZHANG, Z. The rank of a random matrix. *Applied Mathematics and Computation* 185, 1 (2007), 689–694.
- [37] FISCHER, M., AND WRIGHT, R. An application of game theoretic techniques to cryptography, 1993.
- [38] FRANKLIN, J. N. *Matrix Theory*. Dover Publications, 2000.
- [39] GERARDI, D. Unmediated communication in games with complete and incomplete information. *Journal of Economic Theory* 114, 1 (2004), 104 – 131.
- [40] GINTIS, H. *Game Theory Evolving: A Problem-Centered Introduction to Modeling Strategic Interaction*. Princeton University Press, 2009.

- [41] GLOVER, F. Tabu search - part i. *ORSA Journal on Computing* 1, 3 (1989), 190–206.
- [42] GLOVER, F. Tabu search - part ii. *ORSA Journal on Computing* 2, 1 (1990), 4–32.
- [43] GLOVER, F. Tabu search: A tutorial. *Interfaces* 20, 4 (1990), 74–94.
- [44] GLOVER, F., MCMILLAN, C., AND NOVICK, B. Interactive decision software and computer graphics for architectural and space planning. *Annals of Operations Research* 5, 3 (1985), 557–573.
- [45] GOLDREICH, O. *The Foundations of Cryptography*, vol. 2. Cambridge University Press, 2004, ch. Encryption Schemes.
- [46] GOLDREICH, O., MICALI, S., AND WIGDERSON, A. 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* (New York, NY, 1987), ACM, pp. 218–229.
- [47] GOLDWASSER, S., AND LEVIN, L. A. Fair computation of general functions in presence of immoral majority. In *CRYPTO '90: Proceedings of the 10th Annual International Cryptology Conference on Advances in Cryptology* (London, UK, 1991), Springer-Verlag, pp. 77–93.
- [48] HAN, W. D. Y. S., AND CHEN, S. Privacy-preserving multivariate statistical analysis: Linear regression and classification. In *Proceedings of the 2004 SIAM International Conference on Data Mining* (2004).

- [49] HERTZ, A., AND DE WERRA, D. D. Using tabu search techniques for graph coloring. *Computing* 39, 4 (1987).
- [50] HONG, Y., HE, X., VAIDYA, J., ADAM, N. R., AND ATLURI, V. Effective anonymization of query logs. In *CIKM* (2009), pp. 1465–1468.
- [51] HONG, Y., LU, S., LIU, Q., WANG, L., AND DSSOULI, R. A hierarchical approach to the specification of privacy preferences. In *Innovations in Information Technology* (2007), IEEE, pp. 660–664.
- [52] HONG, Y., LU, S., LIU, Q., WANG, L., AND DSSOULI, R. Preserving privacy in e-health systems using hippocratic databases. In *COMPSAC* (2008), IEEE, pp. 692–697.
- [53] HONG, Y., AND VAIDYA, J. An inference-proof approach to privacy-preserving horizontally partitioned linear programs. *Optimization Letters* (2013), to appear.
- [54] HONG, Y., AND VAIDYA, J. Secure transformation for multiparty linear programming. *Rutgers Technical Report* (2013).
- [55] HONG, Y., VAIDYA, J., AND LU, H. Efficient distributed linear programming with limited disclosure. In *DBSec* (2011), pp. 170–185.
- [56] HONG, Y., VAIDYA, J., AND LU, H. Search engine query clustering using top-k search results. In *Web Intelligence* (2011), pp. 112–119.
- [57] HONG, Y., VAIDYA, J., AND LU, H. Secure and efficient distributed linear programming. *Journal of Computer Security* 20, 5 (2012), 583–634.

- [58] HONG, Y., VAIDYA, J., LU, H., AND KARRAS, P. Search log sanitization with guaranteed differential privacy and optimal utility. *Rutgers Technical Report* (2013).
- [59] HONG, Y., VAIDYA, J., LU, H., AND SHAFIQ, B. Privacy-preserving tabu search for distributed graph coloring. In *SocialCom/PASSAT* (2011), pp. 951–958.
- [60] HONG, Y., VAIDYA, J., LU, H., AND WU, M. Differentially private search log sanitization with optimal output utility. In *EDBT* (2012), pp. 50–61.
- [61] HUANG, Z., DU, W., AND CHEN, B. Deriving private information from randomized data. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data* (Baltimore, MD, June 13-16 2005), ACM.
- [62] IOANNIDIS, I., AND GRAMA, A. An efficient protocol for yao’s millionaires’ problem. In *Hawaii International Conference on System Sciences (HICSS-36)* (Waikoloa Village, Hawaii, Jan. 6-9 2003), pp. 205–210.
- [63] JAGANNATHAN, G., AND WRIGHT, R. N. Privacy-preserving distributed k -means clustering over arbitrarily partitioned data. In *Proceedings of the 2005 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Chicago, IL, Aug. 21-24 2005), ACM, pp. 593–599.
- [64] JOHNSON, D., ARAGON, C., MCGEOCH, L., AND SCHEVON, C. Optimization by simulated annealing: An experimental evaluation, part iii: The

traveling salesman problem. *Unpublished Manuscript*.

- [65] JOHNSON, D., ARAGON, C., MCGEOCH, L., AND SCHEVON, C. Optimization by simulated annealing: An experimental evaluation, part i: Graph partitioning. *Operations Research* 37 (1990), 865–892.
- [66] JOHNSON, D., ARAGON, C., MCGEOCH, L., AND SCHEVON, C. Optimization by simulated annealing: An experimental evaluation, part ii: Graph coloring and number partitioning. *Operations Research* 39 (1991), 378–406.
- [67] KANTARCIOGLU, M., AND VAIDYA, J. An architecture for privacy-preserving mining of client information. In *IEEE International Conference on Data Mining Workshop on Privacy, Security, and Data Mining* (Mae-bashi City, Japan, Dec. 9 2002), C. Clifton and V. Estivill-Castro, Eds., vol. 14, Australian Computer Society, pp. 37–42.
- [68] KANTARCIOĞLU, M., AND CLIFTON, C. Privacy-preserving distributed mining of association rules on horizontally partitioned data. *IEEE Transactions on Knowledge and Data Engineering* 16, 9 (Sept. 2004), 1026–1037.
- [69] KARGUPTA, H., DATTA, S., WANG, Q., AND SIVAKUMAR, K. On the privacy preserving properties of random data perturbation techniques. In *Proceedings of the Third IEEE International Conference on Data Mining (ICDM'03)* (Los Alamitos, CA, Nov. 19-22 2003), IEEE Computer Society.
- [70] KARR, A. F., LIN, X., SANIL, A. P., AND REITER, J. P. Secure regressions on distributed databases. *Journal of Computational and Graphical*

- Statistics 14* (2005), 263 – 279.
- [71] KIRKPATRICK, S., GELATT, C. D., AND JR., M. P. V. Optimization by simulated annealing. *Science* 220, 4598 (1983), 671–680.
 - [72] KIRKPATRICK, S., GELATT, C. D., AND VECCHI, M. P. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983* 220, 4598 (1983), 671–680.
 - [73] KUSHILEVITZ, E., LINDELL, Y., AND RABIN, T. Information-theoretically secure protocols and security under composition. In *STOC '06: Proceedings of the thirty-eighth annual ACM symposium on Theory of computing* (New York, NY, USA, 2006), ACM Press, pp. 109–118.
 - [74] LI, J., AND ATALLAH, M. J. Secure and private collaborative linear programming. In *Collaborative Computing: Networking, Applications and Worksharing, 2006. CollaborateCom 2006. International Conference on* (2006), pp. 1 –8.
 - [75] LI, W., LI, H., AND DENG, C. Privacy-preserving horizontally partitioned linear programs with inequality constraints. *Optimization Letters* 7, 1 (2013), 137–144.
 - [76] LIN, X., CLIFTON, C., AND ZHU, M. Privacy preserving clustering with distributed EM mixture modeling. *Knowledge and Information Systems* 8, 1 (July 2005), 68–81.
 - [77] LINDELL, Y. *Composition of Secure Multi-Party Protocols – A Comprehensive Study*, vol. 2815 of *Lecture Notes in Computer Science*. Springer-

Verlag, 2003.

- [78] LINDELL, Y. General composition and universal composability in secure multi-party computation. *focs 00* (2003), 394.
- [79] LINDELL, Y., LYSYANSKAYA, A., AND RABIN, T. On the composition of authenticated byzantine agreement. In *STOC '02: Proceedings of the thirty-fourth annual ACM symposium on Theory of computing* (New York, NY, USA, 2002), ACM Press, pp. 514–523.
- [80] LINDELL, Y., AND PINKAS, B. Privacy preserving data mining. In *Advances in Cryptology – CRYPTO 2000* (New York, NY, Aug. 20-24 2000), Springer-Verlag, pp. 36–54.
- [81] LINDELL, Y., AND PINKAS, B. Privacy preserving data mining. *Journal of Cryptology* 15, 3 (2002), 177–206.
- [82] LIU, Q., LU, S., HONG, Y., WANG, L., AND DSSOULI, R. Securing telehealth applications in a web-based e-health portal. In *ARES 08* (2008), IEEE, pp. 3–9.
- [83] LU, H., HONG, Y., STREET, W. N., WANG, F., AND TONG, H. Overlapping clustering with sparseness constraints. In *ICDM Workshops* (2012), pp. 486–494.
- [84] LU, H., HONG, Y., YANG, Y., DUAN, L., AND BADAR, N. Towards user-oriented rbac model. In *DBSec* (2013), p. to appear.
- [85] LU, H., VAIDYA, J., ATLURI, V., AND HONG, Y. Extended boolean matrix decomposition. In *ICDM* (2009), pp. 317–326.

- [86] LU, H., VAIDYA, J., ATLURI, V., AND HONG, Y. Constraint-aware role mining via extended boolean matrix decomposition. *IEEE Trans. Dependable Sec. Comput.* 9, 5 (2012), 655–669.
- [87] LU, S., HONG, Y., LIU, Q., WANG, L., AND DSSOULI, R. Access control in e-health portal systems. In *Innovations in Information Technology* (2007), IEEE, pp. 88–92.
- [88] MAILLER, R., AND LESSER, V. Solving distributed constraint optimization problems using cooperative mediation. *aamas 01* (2004), 438–445.
- [89] MANGASARIAN, O. L. Privacy-preserving linear programming. *Optimization Letters* 5, 1 (2011), 165–172.
- [90] MANGASARIAN, O. L. Privacy-preserving horizontally partitioned linear programs. *Optimization Letters* 6, 3 (2012), 431–436.
- [91] MARX, D., AND MARX, D. A. Graph coloring problems and their applications in scheduling. In *in Proc. John von Neumann PhD Students Conference* (2004), pp. 1–2.
- [92] MATSUURA, K. Information security and economics in computer networks: An interdisciplinary survey and a proposal of integrated optimization of investment. *Computing in Economics and Finance* 2003 48, Society for Computational Economics, Aug. 2003. available at <http://ideas.repec.org/p/sce/scecf3/48.html>.
- [93] MEHMOOD, D., SHAFIQ, B., VAIDYA, J., HONG, Y., ADAM, N. R., AND ATLURI, V. Privacy-preserving subgraph discovery. In *DBSec* (2012),

pp. 161–176.

- [94] METROPOLIS, N., ROSENBLUTH, A. W., ROSENBLUTH, M. N., TELLER, A. H., AND TELLER, E. Equation of state calculations by fast computing machines. *The journal of chemical physics* 21 (1953), 1087.
- [95] MIELIKAINEN, T. Privacy problems with anonymized transaction databases. In *Discovery Science: 7th International Conference Proceedings* (Jan. 2004), vol. 3245 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 219 – 229.
- [96] MODI, P. J., SHEN, W.-M., TAMBE, M., AND YOKOO, M. An asynchronous complete method for distributed constraint optimization. In *AA-MAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems* (New York, NY, USA, 2003), ACM Press, pp. 161–168.
- [97] NEMHAUSER, G. L., AND WOLSEY, L. A. *Integer and combinatorial optimization*. Wiley-Interscience, New York, NY, USA, 1988.
- [98] OMOTE, K. *A study on Electronic Auctions*. Doctoral Thesis, Japan Advanced Institute of Science and Technology, 2002.
- [99] OSTROVSKY, R., AND YUNG, M. How to withstand mobile virus attacks (extended abstract). In *PODC '91: Proceedings of the tenth annual ACM symposium on Principles of distributed computing* (New York, NY, USA, 1991), ACM Press, pp. 51–59.

- [100] ÖZENER, O. Ö., AND ERGUN, Ö. Allocating costs in a collaborative transportation procurement network. *Transportation Science* 42, 2 (2008), 146–165.
- [101] PAILLIER, P. Public key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology - Eurocrypt '99 Proceedings, LNCS 1592* (1999), pp. 223–238.
- [102] PAPADIMITRIOU, C. H., AND STEIGLITZ, K. *Combinatorial optimization: algorithms and complexity*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1982.
- [103] PAPADIMITRIOU, C. H., AND STEIGLITZ, K. *Combinatorial optimization: algorithms and complexity*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1982.
- [104] PAPADIMITRIOU, C. H., AND YANNAKAKIS, M. On the value of information in distributed decision-making (extended abstract). In *PODC '91: Proceedings of the tenth annual ACM symposium on Principles of distributed computing* (New York, NY, USA, 1991), ACM Press, pp. 61–64.
- [105] PAPADIMITRIOU, C. H., AND YANNAKAKIS, M. Linear programming without the matrix. In *STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing* (New York, NY, USA, 1993), ACM Press, pp. 121–129.
- [106] PETCU, A., AND FALTINGS, B. Approximations in distributed optimization. In *CP05 - workshop on Distributed and Speculative Constraint Pro-*

cessing(DSCP) (Oct. 2005).

- [107] PETCU, A., AND FALTINGS, B. An efficient constraint optimization method for large multiagent systems. In *AAMAS05 - LSMAS workshop* (July 2005).
- [108] PETCU, A., AND FALTINGS, B. Incentive compatible multiagent constraint optimization. In *WINE'05: Workshop on Internet and Network Economics* (Dec. 2005).
- [109] PETCU, A., AND FALTINGS, B. Optimal solution stability in continuous time optimization. In *IJCAI05 - Distributed Constraint Reasoning workshop(DCR05)* (Aug. 2005).
- [110] PETCU, A., AND FALTINGS, B. A scalable method for multiagent constraint optimization. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05)* (Aug. 2005).
- [111] PETCU, A., AND FALTINGS, B. Superstabilizing, fault-containing multiagent combinatorial optimization. In *Proceedings of the National Conference on Artificial Intelligence(AAAI-05)* (July 2005).
- [112] PETCU, A., AND FALTINGS, B. Distributed generator maintenance scheduling. In *Proceedings of the First International ICSC Symposium on artificial intelligence in energy systems and power (AIESP'06)* (Feb. 2006).
- [113] RIZVI, S. J., AND HARITSA, J. R. Maintaining data privacy in association rule mining. In *Proceedings of 28th International Conference on Very*

- Large Data Bases* (Hong Kong, Aug. 20-23 2002), VLDB, VLDB Endowment, pp. 682–693.
- [114] SAKUMA, J., AND KOBAYASHI, S. A genetic algorithm for privacy preserving combinatorial optimization. In *GECCO* (2007), pp. 1372–1379.
 - [115] SANIL, A. P., KARR, A. F., LIN, X., AND REITER, J. P. Privacy preserving regression modelling via distributed computation. In *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining* (New York, NY, USA, 2004), ACM Press, pp. 677–682.
 - [116] SILAGHI, M.-C., FALTINGS, B., AND PETCU, A. Secure combinatorial optimization simulating dfs tree-based variable elimination. In *9th Symposium on Artificial Intelligence and Mathematics* (Ft. Lauderdale, Florida, USA, Jan 2006). <http://www2.cs.fit.edu/~msilaghi/papers/>.
 - [117] SILAGHI, M. C., AND MITRA, D. Distributed constraint satisfaction and optimization with privacy enforcement. *iat 00* (2004), 531–535.
 - [118] SILAGHI, M.-C., AND RAJESHIRKE, V. The effect of policies for selecting the solution of a discsp on privacy loss. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems* (2004), pp. 1396–1397.
 - [119] SUZUKI, K., AND YOKOO, M. Secure generalized vickrey auction using homomorphic encryption. *Lecture Notes in Computer Science* 2742.
 - [120] TAHA, H. *Operations Research: An Introduction*. Prentice Hall, 2010.

- [121] TEAGUE, V. Selecting correlated random actions. In *Proceedings of Financial Cryptography* (2004).
- [122] TEBBOTH, J. R. *A Computational Study of Dantzig-Wolfe Decomposition*. PhD thesis, University of Buckingham, 2001.
- [123] TURBAN, E., RAINER, R. K., AND POTTER, R. E. Introduction to information technology, chapter 2nd, information technologies: Concepts and management. *John Wiley and Sons, 3rd edition*.
- [124] VAIDYA, J. Privacy-preserving linear programming. In *SAC* (2009), pp. 2002–2007.
- [125] VAIDYA, J. A secure revised simplex algorithm for privacy-preserving linear programming. In *AINA '09: Proceedings of the 23rd IEEE International Conference on Advanced Information Networking and Applications* (2009).
- [126] VAIDYA, J., AND CLIFTON, C. Privacy preserving association rule mining in vertically partitioned data. In *The Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Edmonton, Alberta, Canada, July 23-26 2002), ACM, pp. 639–644.
- [127] VAIDYA, J., AND CLIFTON, C. Privacy-preserving k -means clustering over vertically partitioned data. In *The Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Washington, DC, Aug. 24-27 2003), ACM, pp. 206–215.

- [128] VAIDYA, J., AND CLIFTON, C. Privacy preserving naïve bayes classifier for vertically partitioned data. In *2004 SIAM International Conference on Data Mining* (Philadelphia, PA, Apr. 22-24 2004), SIAM, pp. 522–526.
- [129] VAIDYA, J., AND CLIFTON, C. Privacy-preserving outlier detection. In *Proceedings of the Fourth IEEE International Conference on Data Mining (ICDM'04)* (Los Alamitos, CA, Nov.1 - 4 2004), IEEE Computer Society Press, pp. 233–240.
- [130] VAIDYA, J., AND CLIFTON, C. Privacy-preserving decision trees over vertically partitioned data. In *The 19th Annual IFIP WG 11.3 Working Conference on Data and Applications Security* (Storrs, Connecticut, Aug. 7-10 2005), Springer.
- [131] WRIGHT, R., AND YANG, Z. Privacy-preserving bayesian network structure computation on distributed heterogeneous data. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Seattle, WA, Aug.22-25 2004), ACM.
- [132] YAO, A. C. How to generate and exchange secrets. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science* (Los Alamitos, CA, USA, 1986), IEEE, IEEE Computer Society, pp. 162–167.
- [133] YOKOO, M., DURFEE, E. H., ISHIDA, T., AND KUWABAR, K. Distributed constraint satisfaction for formalizing distributed problem solving. In *Proceedings of International Conference on Distributed Computing Systems* (1992), pp. 614–621.

- [134] YOKOO, M., AND HIRAYAMA, K. Distributed breakout algorithm for solving distributed constraint satisfaction problems. In *Proceedings of the First International Conference on Multi-Agent Systems* (1995), V. Lesser, Ed., MIT Press.
- [135] YOKOO, M., SUZUKI, K., AND HIRAYAMA, K. Secure distributed constraint satisfaction: Reaching agreement without revealing private information. In *Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming (CP-2002)* (2002).
- [136] YU, H., JIANG, X., AND VAIDYA, J. Privacy-preserving svm using non-linear kernels on horizontally partitioned data. In *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing* (New York, NY, USA, 2006), ACM Press, pp. 603–610.
- [137] YU, H., VAIDYA, J., AND JIANG, X. Privacy-preserving svm classification on vertically partitioned data. In *Proceedings of PAKDD '06* (Jan. 2006), vol. 3918 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 647 – 656.
- [138] ZHANG, N., WANG, S., AND ZHAO, W. A new scheme on privacy-preserving association rule mining. In *The 8th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD 2004)* (Pisa, Italy, Sept. 20-24 2004).
- [139] ZHANG, W., AND WITTENBURG, L. Distributed breakout algorithm for distributed constraint optimization problems - dbarelay. In *Proceedings of*

the International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS-03) (2003).

- [140] ZHU, Y., AND LIU, L. Optimal randomization for privacy preserving data mining. In *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining* (New York, NY, USA, 2004), ACM Press, pp. 761–766.

VITA

Yuan Hong

1983.7	Born in Jingmen, Hubei Province, China.
2000	Graduated from Jingshan High School, Jingmen, Hubei Province; Third Prize, National Olympiad Contest in Physics, China.
2000-2004	B.S., Management Info Systems, Beijing Institute of Technology, China.
2004-2005	Software Engineer, China.
2005-2008	M.S., Computer Science, Concordia University, Canada.
2008-2012	Graduate Assistantship, Rutgers Business School.
2009, 2010, 2012	Rutgers Business School Dean's Research Award - Ph.D. Competition.
2012-2013	Dissertation Fellowship, Rutgers Business School.
2013	Ph.D. in Management (IT Major), Rutgers University.

Publications

2007	Y. Hong, S. Lu, Q. Liu, L. Wang, and R. Dssouli. A Hierarchical Approach to the Specification of Privacy Preferences. IIT 2007. S. Lu, Y. Hong, Q. Liu, L. Wang, and R. Dssouli. Access Control in e-Health Portal Systems. IIT 2007.
2008	Y. Hong, S. Lu, Q. Liu, L. Wang, and R. Dssouli. Preserving Privacy in e-Health Systems using Hippocratic Databases. COMPSAC 2008. Q. Liu, S. Lu, Y. Hong, L. Wang, and R. Dssouli. Securing Tele-health Applications in a Web-based e-Health Portal. ARES 2008.
2009	Y. Hong, X. He, J. Vaidya, N. Adam, and V. Atluri. Effective Anonymization of Query Logs. CIKM 2009. H. Lu, J. Vaidya, V. Atluri, and Y. Hong. Extended Boolean Matrix Decomposition. ICDM 2009.
2011	Y. Hong, J. Vaidya, and H. Lu. Efficient Distributed Linear Programming with Limited Disclosure. DBSEC 2011. Y. Hong, J. Vaidya, and H. Lu. Search Engine Query Clustering using Top-k Search Results. WI 2011. Y. Hong, J. Vaidya, H. Lu, and B. Shafiq. Privacy-preserving Tabu Search for Distributed Graph Coloring. PASSAT 2011.
2012	Y. Hong, J. Vaidya, and H. Lu. Secure and Efficient Distributed Linear Programming. Journal of Computer Security, Vol. 20(5), pp. 583-634, 2012 H. Lu, J. Vaidya, V. Atluri, and Y. Hong. Constraint-Aware Role Mining Via Extended Boolean Matrix Decomposition. IEEE Transactions on Dependable and Secure Computing, Vol. 9(5), pp. 655-669, 2012. Y. Hong, J. Vaidya, H. Lu, and M. Wu. Differentially Private Search Log Sanitization with Optimal Output Utility. EDBT 2012. D. Mehmood, B. Shafiq, J. Vaidya, Y. Hong, N. Adam, and V. Atluri, Privacy Preserving Subgraph Discovery. DBSEC 2012. H. Lu, Y. Hong, N. Street, F. Wang, and H. Tong, Overlapping Clustering with Sparseness Constraints, ICDM Workshops 2012.
2013	Y. Hong and J. Vaidya. An Inference-proof Approach to Privacy-preserving Horizontally Partitioned Linear Programs. Optimization Letters. H. Lu, Y. Hong, Y. Yang, L. Duan, and N. Badar, Towards User-Oriented RBAC Model, DBSEC 2013.