CERIAS Tech Report 2006-64
Secure and Private Collaborative Linear Programming
by Mikhail J. Atallah
Center for Education and Research
Information Assurance and Security
Purdue University, West Lafayette, IN 47907-2086

SECURE AND PRIVATE COLLABORATIVE LINEAR PROGRAMMING

Jiangtao Li
Intel Corporation
2111 NE 25th Avenue, Hillsboro, OR 97124
Email: jiangtao.li@intel.com

Mikhail J. Atallah
Department of Computer Science, Purdue University
350 N. University Street, West Lafayette, IN 47907
Email: mja@cs.purdue.edu

Abstract— **The growth of the Internet has created tremendous opportunities for online collaborations. These often involve collaborative optimizations where the two parties are, for example, jointly minimizing costs without violating their own particular constraints (e.g., one party may have too much inventory, another too little inventory but too much production capacity, etc). Many of these optimizations can be formulated as linear programming problems, or, rather, as** collaborative linear programming, **in which two parties need to jointly optimize based on their own private inputs. It is often important to have online collaboration techniques and protocols that carry this out without either party revealing to the other anything about their own private inputs to the optimization (other than, unavoidably, what can be deduced from the collaboratively computed optimal solution). For example, two organizations who jointly invest in a project may want to minimize some linear objective function while satisfying both organizations' private and confidential constraints. Constraints are usually private when they reveal too much about the organizations' financial health, its future business strategy, etc. Linear programming problems have been widely studied in the literature. However, the existing solutions (e.g., the simplex method) do not extend to the above-mentioned framework in which the linear constraints are shared by the two parties, who do not want to disclose their own to the other party. In this paper, we give an efficient protocol for solving linear programming problems in the honest-but-curious model, such that neither party reveals anything about their private input to the other party (other than what can be inferred from the result). The amount of communication and computation done by our protocol is proportional to the time complexity of the simplex method, a widely used linear programming algorithm. We also provide a practical solution that prevents certain malicious behavior of the participants. The use of the known general circuit-simulation solutions to secure function evaluation is unacceptable for the simplex method, as it implies an exponential size circuit.**

I. Introduction

The potential benefits of collaboration (e.g., sharing of information, trading of resources, etc.) are widely documented by economists – if Alice has a very low production cost for oranges and Bob has a very low production costs for apples, they both benefit by trading apples for oranges. Replace, in the above, apples and oranges by bandwidth, computational power, storage capacity, inventory, production capacity, electronic components, etc, and the relevance of exploiting such "win-win" situations in an online environment becomes apparent. The advent of the Internet and of grid computing [12], [13] makes this trading easier and more beneficial by decreasing the "friction" in the system (costs of carrying out the negotiations, of actually exchanging the resources, etc.). Although there are so many situations where collaboration is mutually advantageous, it often does not occur and its potential goes unexploited, that is, the participants often do not engage in the trade even though its outcome would be mutually beneficial to both of them. This occurs when the online negotiation is "too revealing" of the participants' private or proprietary data: That a company has a massive excess of bandwidth, inventory, production capacity, etc, can be damaging to the company's future negotiating position (or even to its stock price and the survival of its management). The formulation of such online collaboration often leads to a linear-programming formulation, and the task is then to solve this problem so that both parties achieve their benefits yet without revealing their private/proprietary data. Consider the following example where there are n resources that are shared by Alice and Bob according to some initial partition, each of the two parties has a private valuation function for each resource, and the goal is to re-partition the resources so as to achieve the optimal win-win outcome.

Example 1: Alice and Bob share n resources. We use p_i to denote Alice's fraction of resource i, hence $1 - p_i$ is Bob's initial fraction. Let a_i be Alice's private valuation of resource i, b_i be Bob's; the a_i's and b_i's are normalized so that $\sum_i a_i = \sum_i b_i = K$ where K is fixed (so a party cannot increase its valuation of a resource without decreasing its valuation of another resource). Alice and Bob want to re-allocate their resource, so that the re-allocation is fair, i.e., equally beneficial to both parties. We use q_i to denote the fraction of resource i that would end up with Alice, hence $1 - q_i$ is Bob's final fraction. This problem can be formulated as a linear program as follows:

$$\begin{array}{lll} \text{maximize} & \sum_{i=1}^{n} a_i(q_i - p_i) & \\ \text{subject to} & \sum_{i=1}^{n} a_i(q_i - p_i) & = \sum_{i=1}^{n} b_i(p_i - q_i) \\ & \sum_{i=1}^{n} a_i & = \sum_{i=1}^{n} b_i \\ & \sum_{i=1}^{n} p_i & = \sum_{i=1}^{n} q_i \end{array}$$

In the above equations, the first constraint ensures that the benefits Alice gains from this re-allocation is the same as the benefits Bob gains. The second constraint ensures that the total valuations for Alice is equal to the total valuations for Bob. The third constraint ensures that the total fractions Alice owns in pre-allocation is same as in after-allocation.

1-4244-0429-0/06/$20.00 ©2006 IEEE

Linear programming problems [10], [25] have proved valuable for modeling many types of problems in planning, routing, scheduling, optimization, and assignment. In the linear programming problems, there are n decision variables, m linear constraints, and a linear objective function. The goal is then to find a solution that minimizes the objective function while satisfying the m constraints[1].

Although linear programming problems have been well studied in the literature ([10], [25], [18], [1], to list a few), these solutions do not readily extend to the framework considered here. For example, Alice has α private constraints on n decision variables, and Bob has β private constraints on the same decision variables. Alice and Bob want to jointly find a non-negative solution that minimizes some linear objective function and also satisfies the combined $\alpha + \beta$ linear constraints. Without loss of generality, we assume A, b, and c are additively shared between Alice and Bob. That is, $A = A' + A''$ where A' is known only to Alice and A'' is known only to Bob. Vectors b and c are shared by Alice and Bob in an analogous way. It is easy to see that the previous example is a special case of this general sharing. We now define the *secure collaborative linear programming problem* as follows.

Problem 1: Alice has a matrix A' and two vectors b' and c', and Bob has a matrix A'' and two vectors b'' and c'', where A', A'' are $m \times n$ matrices, b', b'' are m-dimensional vectors, and c', c'' are n-dimensional vectors. Let $A = A' + A''$, $b = b' + b''$, and $c = c' + c''$. Without disclosing their private inputs to the other party, Alice and Bob want to find a solution x that

$$\begin{aligned} &\text{minimizes} && c^T \cdot x, \\ &\text{subject to} && A \cdot x = b, \;\; x \geq 0. \end{aligned}$$

In this paper, we propose a secure linear programming protocol that solves problem 1 efficiently. The computational cost and amount of communication done by our protocol are proportional to the time complexity of the simplex method [10], [25] for solving the linear programming problem; they are $O(k(mn + \ell n) + \ell mn)$, where k is number of steps used in the simplex method, m is number of constraints, n is number of decision variables, and ℓ is number of bits of precision per variable. Although k is reasonable in practice and simplex usually performs quite well, there are pathological inputs (rare in practice) for which k is exponential, which is why we cannot use the general solutions to secure function evaluation (SFE) [26], [17], [16] for the simplex algorithm, as the size of the circuit would have to be exponential.

Note that given the output of the protocol x, Alice (or Bob) may infer some information about the other party's input. For example, Alice can learn that $A''x - b'' = b' - A'x$ from the protocol, in other words, she learns m equations about Bob's private input A'' and b''. This is unavoidable by definition. In many practical situations the leakage of so little information is acceptable. Consider the resource re-allocation problem in Example 1, Alice may learn some information about Bob's private valuations from the output of the linear programming, but not the exact values.

[1]If the original goal is to find a solution that maximizes the objective function, we can easily modify the objective function to reverse the goal.

The rest of paper is organized as follows. We begin with a brief introduction of previous work in Section II, then review the simplex method in Section III. We review a few cryptographic tools that we use in the solution in Section IV, and present some building blocks in Section V. In Section VI, we describe our protocol for solving collaborative linear programming problems. We present a practical solution that prevents certain dishonest behaviors in Section VII. Finally, we conclude the paper and discuss the future work in Section VIII.

II. Related Work

One of the fundamental techniques for solving linear programming problems is the simplex method developed by Dantzig [10]. The simplex method is an efficient procedure for solving large practical linear programs on the computer. But the simplex method is not a polynomial-time algorithm: Klee and Minty [19] created a worst-case example in which the simplex algorithm would require an exponential number of pivots, however, such cases seem never to be countered in real world problems. Karmarkar [18] developed a polynomial-time algorithm for solving linear programmes. Although polynomial, the cost of his algorithm is much higher than the cost of the simplex method when solving practical problems. In this paper, we focus on the simplex method, and propose a secure linear programming protocol based on it.

Secure Function Evaluation (SFE) [26], [17], [16] is a powerful and general cryptographic primitive. It allows two or more parties to jointly compute some function while hiding their inputs to each other. Secure collaborative linear programming problem is a special case of SFE. However general solutions to SFE cannot solve our problem efficiently, because the simplex method is an exponential-time algorithm (in the worst case), the circuit construction to compute the simplex method has exponential size. Even if we break the simplex method into k pivot steps, each of which is implemented using circuit evaluation; the total cost is at least $O(k\ell mn)$, whereas the time complexity of our protocol is $O(k(mn+\ell n)+\ell mn)$.

Cramer and Damgård [7] developed efficient protocols that solve linear algebra problems in secure two-party computation setting. Their solution works only for finite fields rather than floating point arithmetic. For example, in finite field GF(7), 3 dividing by 2 becomes 5, instead of 1.5. In the collaborative linear programming, all the numeric operations are performed under floating point arithmetic, therefore, their solution cannot directly apply to our problem. Moreover, their protocols aim at solving linear equations, it is not clear how their scheme can be used for linear programming problems where the goal is to find an optimal solution.

Du [11] proposed a heuristic solution to the secure collaborative linear programming problem. In his solution, one party disguises the constraint matrix A by multiplying a random matrix B, then the other party conducts linear programming on the disguised constraint matrix. In the end, the first party recovers the result of the linear programming using the knowledge of B. Du's solution is heuristic – the security property of his solution is not defined. Compare to his solution, our

protocol is provably secure in the honest-but-curious model (in which both parties follow the protocol), and can be enhanced to handle many dishonest behaviors.

III. Review of Simplex Method

In this section, we briefly review the simplex method [10], [25], a widely used method to solve linear programming problems in practice. We do so in order to be able to later refer to specific steps of the method, and also to introduce our own notation (of course the readers who are already familiar with the simplex method can go through this section very quickly).

In linear programming problems, there are n non-negative decision variables, denoted as $x_1, x_2, \ldots, x_n$. The objective is to minimize some linear function of these variables $z = c_1x_1 + c_2x_2 + \cdots + c_nx_n$. In addition to the object function, there are m constraints. Each constraint is a linear equation of the decision variables. The definition of a linear program in *standard form* is to find values of $x_1 \geq 0, x_2 \geq 0, \ldots, x_n \geq 0$ and minimum z satisfying

$$\begin{array}{rcl} c^T \cdot x & = & z \\ A \cdot x & = & b \end{array} \quad (1)$$

Where A is an $m \times n$ matrix, b is an m-dimensional vector, c is an n-dimensional vector, x is an n-dimensional vector, and the superscript T stands for transpose.

A. Simplex Algorithm

A system of m equations with n variables is said to be in *canonical form* with respect to an ordered set of variables $(x_{j_1}, x_{j_2}, \ldots, x_{j_m})$ if and only if x_{j_i} has a unit coefficient in ith equation and a zero coefficient in all other equations. We refer $x_{j_1}, x_{j_2}, \ldots, x_{j_m}$ as *basic variables*, and other decision variables as *non-basic variables*. The simplex algorithm always starts with a system of equations in canonical form. For example, suppose we have the following canonical system with the basic variables $-z, x_{j_1}, x_{j_2}, \ldots, x_{j_m}$.

$$\begin{array}{rcl} (-z) \quad + \quad c^T \cdot x & = & -z_o \\ A \cdot x & = & b \end{array} \quad (2)$$

Let $x_B = (x_{j_1}, x_{j_2}, \ldots, x_{j_m})^T$ and $x_N = (x_{j_{m+1}}, x_{j_{m+2}}, \ldots, x_{j_n})^T$ be vectors of the basic variables and non-basic variables, respectively. Then the *basic solution* for this canonical form is $z = z_0$, $x_B = b$, $x_N = 0$. In the simplex algorithm, it is required that the initial basic solution is *feasible*, that is, $x_B = b \geq 0$ (recall that the decision variables have to be non-negative).

Algorithm 1 (Simplex Algorithm): Assume that a linear program in standard form (1) has been converted to a feasible canonical form (2). Throughout the algorithm, we use $(x_{j_1}, x_{j_2}, \ldots, x_{j_m})$ to denote the ordered set of the basic variables. The algorithm steps are as follows:

1) *Find incoming variable*. Let s, $1 \leq s \leq n$, be the index where c_s is a minimum in c, that is $c_s = \min c$. If $c_s \geq 0$, then report the basic feasible solution as optimal and **stop**. If $c_s < 0$, then s is the index of the incoming basic variable.
2) *Test for unbounded z*. If $a_{is} \leq 0$ for each $0 \leq i \leq m$, then the linear program has no bounded solution. Report $z \rightarrow -\infty$ and **stop**.
3) *Find outgoing variable*. Let r, $1 \leq r \leq m$, be the index where
$$\frac{b_r}{a_{rs}} = \min_{\{i|a_{is}>0\}} \frac{b_i}{a_{is}} \geq 0.$$
In case of ties, choose r at random. The index of the outgoing basic variable is j_r.
4) *Pivot*. Pivot on a_{rs} to find a new basic feasible solution, set $j_r = s$ and return to step 1.

If there exists an optimal basic feasible solution, then the solution to the linear program is $x_B = b$, $x_N = 0$, and $z = z_0$.

B. Simplex Method

The simplex method has two phases [10]. In phase 1, it tries to determine whether there exists a starting basic feasible solution, then if such a solution exists, phase 2 finds an optimal basic feasible solution or reports unbounded z. Due to page constraints, we omit the review of the simplex method.

IV. Review of Cryptographic Tools

In this section, we briefly review two cryptographic tools that shall be used in our protocol.

a) Homomorphic Encryption: A homomorphic encryption scheme [21], [22], [8], [9] is an encryption scheme with the following property: $E(x) \cdot E(y) = E(x + y)$. A homomorphic encryption scheme is *semantically secure* if $E(x)$ reveals no information about x. Hence $x = y$ does not imply $E(x) = E(y)$. Damgård and Jurik [9] proposed a homomorphic encryption scheme in which all users can use the same RSA modulus N when generating key pairs.

In the rest of this paper, we assume that Alice and Bob use the Damgård-Jurik homomorphic encryption scheme. Alice and Bob each generate their own key pairs with a shared RSA modulus N, of which the factorization is known to neither of them. The generation of N can be done either by a trusted third party or using two-party protocols from [15], [4]. Throughout the paper, we use $E_A(\cdot)$ to denote encryption with Alice's public key, and $D_A(\cdot)$ to denote decryption with Alice's private key. Analogously, $E_B(\cdot)$ and $D_B(\cdot)$ denote encryption and decryption using Bob's key.

b) Scrambled Circuit Evaluation: The scrambled circuit evaluation protocol was developed by Yao [26]. This protocol involves a generator and an evaluator, in which the evaluator has private input x and the generator has private input y, and they jointly compute $f(x, y)$ without revealing their private input to the other party. Recently, the scrambled circuit protocol has been implemented in [20].

The scrambled circuit protocol is secure against honest-but-curious adversaries. When the size of the circuit that computes f is linear to $|x|$, the size of the input, the complexity of the scrambled circuit protocol is $O(|x|)$ modular exponentiations. In this paper, we use the scramble circuit protocol to compute some simple functions, such as, finding the minimum value from an array that is shared between Alice and Bob.

V. Notation and Building Blocks

This section aims at making the later presentation of the protocol much crisper by presenting some of the ideas and building blocks for it ahead of time. In the rest of this paper we assume, without loss of generality, that all inputs are integers; the reason this is not a limitation is the linearity of the optimization problem we are solving, which allows us to "scale up" to integer values the inputs to our protocol and later on scale the answers back down.

A. Our Security Model

At this point, we consider only *honest-but-curious* adversaries who follow the prescribed protocol and attempt to learn more information than allowed from the execution. Adversaries that try to influence the result of the protocol shall be considered in Section VII.

Our security model follows directly from [16]. Let x be Alice's private input and y be Bob's private input. Let f be a function $f : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^*$ such that $f(x,y)$ is the result of the collaborative linear programming. Suppose there is a two-party protocol Π in which Alice inputs x and Bob inputs y, in the end, both Alice and Bob learn $f(x,y)$. The security of the protocol Π is analyzed by comparing what an adversary can do in Π to what she can do in an ideal scenario that is secure by definition. In ideal scenario, there is a trust third party who receives x from Alice and y from Bob, computes $f(x,y)$, and returns $f(x,y)$ to both party. A protocol Π is *secure* if any adversary interacting in the real protocol Π can do no more harm than in the ideal scenario.

B. Additively Split Data

In the rest of this paper, we use following notations: any items superscripted with $'$ are known to Alice but not to Bob, those superscripted with $''$ are known to Bob but not to Alice. In what follows, we often *additively split* an item x between Alice and Bob for the purpose of hiding it from either party. *An item x is said to be additively split between Alice and Bob if Alice has x' and Bob has x'' such that $x = x' + x'' \pmod N$, but the value of x is known to either party.* The modulus N is the RSA modulus used in Damgård-Jurik homomorphic encryption. We use the same N for additively sharing data and for homomorphic encryption, because suppose x is additively split into two numbers x' and x'', then we are sure that $E(x') \cdot E(x'') = E(x' + x'' \mod N) = E(x)$ always holds. Assume all arithmetic in this paper is modulo N unless specified.

Suppose Alice and Bob additively share x and y, they can easily compute $x + y$ and $x - y$ in additively split fashion. Alice and Bob can run a secure multiplication protocol [14] to compute $x \cdot y$ in split fashion, and run a secure division protocol [2] to compute x/y in split fashion. The computational costs of these protocols are $O(1)$.

C. Additively Sharing the Matrix

Let us define an $(m+1) \times (n+1)$ matrix

$$D = \begin{pmatrix} c^T & -z_0 \\ A & b \end{pmatrix}, \tag{3}$$

where A, b, c, and z_0 are specified in the canonical form (2) of a linear program. The simplex algorithm is essentially a sequence of pivot steps on the matrix D. In the rest of this paper, we refer the linear system corresponding to D to the following linear program

$$\begin{aligned} c^T \cdot x &= -z_o \\ A \cdot x &= b \end{aligned}$$

Our secure linear programming protocol computes the same matrix D as the simplex method. A crucial difference is that the matrix is additively split by Alice and Bob: Alice and Bob each hold matrix D' and D'', respectively, the sum of which is the matrix D; i.e., $D = D' + D''$. The protocol will keep this as an invariant through all its steps.

D. Pivoting the Matrix

We now describe how Alice and Bob perform pivoting on the additively split matrix D. It is shown in [2] that the split division protocols are much expensive than the split multiplication protocol. We need to avoid divisions in the matrix pivot if possible. Thus we modify the matrix pivot as follows.

Definition 1 (Modified Matrix Pivot): To pivot on position (i,j) of matrix D, for each $k = 1, \ldots, m+1$ except $k = i$, we replace the kth equation by the sum of the kth equation multiplied by d_{ij} and the ith equation multiplied by $-d_{kj}$.

The resulting system of the modified matrix pivot is equivalent to the original system, as the pivot does not alter the solution set. However, the resulting system may no longer be in the canonical form. Observe that for the basic variables $x_{j_1}, x_{j_2}, \ldots, x_{j_m}$ of the system, x_{j_i} has a *non-zero* (instead of a unit) coefficient in ith equation and a zero coefficient in all other equations. We call such system to be in a *semi-canonical* form. Suppose we have a semi-canonical system with the basic variables $-z, x_{j_1}, x_{j_2}, \ldots, x_{j_m}$, then the solution for this semi-canonical form is $z = z_0$, $x_{j_1} = b_1/a_{j_1 1}, \ldots, x_{j_m} = b_m/a_{j_m m}$. We next give the matrix pivoting protocol that performs the modified matrix pivot.

Protocol 1: Matrix Pivoting Protocol

Input Alice and Bob share an $(m+1) \times (n+1)$ matrix D in additively split form, i.e., Alice has D' and Bob has D'' such that $D = D' + D''$. Let (i,j) be an index where $2 \le i \le m+1$ and $1 \le j \le n$.

Output Alice and Bob additively share $\bar{D}$, the result of pivoting on the position (i,j) of D.

The protocol steps are as follows. For each $k = 1, \ldots, m+1$ except $k = i$, and $\ell = 1, \ldots, n+1$:

1) We use d_{ij} and $\bar{d}_{ij}$ to denote the (i,j) entries of D and $\bar{D}$, respectively. By Definition 1, $\bar{d}_{k\ell}$ is computed as $d_{k\ell} d_{ij} - d_{i\ell} d_{kj}$.
2) Alice and Bob run a split multiplication protocol twice, once to compute $d_{k\ell} d_{ij}$ and once to compute $d_{i\ell} d_{kj}$.
3) Alice and Bob each subtract her/his shares of $d_{k\ell} d_{ij}$ and $d_{i\ell} d_{kj}$ locally. Now the value of $\bar{d}_{k\ell}$ is additively split between Alice and Bob.

Analysis Alice and Bob need to conduct $2m(n+1)$ secure split multiplications. As each split multiplication requires $O(1)$

modular exponentiations, the matrix pivoting protocol takes $O(mn)$ modular exponentiations.

E. Blind-and-Permute

Recall that in the matrix pivoting protocol, Alice and Bob have to know which position in the matrix to pivot. However, knowing the pivot position may leak information about the original matrix unnecessarily. To avoid leaking private information about the matrix, let us first define the matrix permutation as follows:

Definition 2 (Matrix Permutation): Let D be an $(m+1) \times (n+1)$ matrix, let π_C be a permutation on $\{1, \ldots, n\}$ and π_R be a permutation on $\{1, \ldots, m\}$. The matrix permutation on D is the matrix D permuted by the first n columns based on π_C and then permuted by the last m rows of D based on π_R. We use $\pi_R(\pi_C(D))$ to denote the permuted matrix.

Note that the matrix permutation preserves the solution set of the linear system corresponding to the matrix. Consider the canonical form 2, π_C is used to permute x variables, π_R is used to permute the m constraints. The linear system of the permuted matrix is equivalent to the linear system of the original D.

Our solution to avoid leaking the pivot position is as follows: Suppose Alice and Bob additively share the matrix D, Alice and Bob jointly permute D in a way that the permutation is known to neither of them, and in the end the permuted matrix $\bar{D}$ is additively split between them using different randomness. Now it no longer matters if Alice and Bob know where to pivot in $\bar{D}$ because they have no way of relating the pivot position in $\bar{D}$ to the corresponding position in the original matrix D, and therefore using matrix pivoting protocol becomes acceptable. We next describe a protocol that achieves this "blinded permutation".

Protocol 2: Matrix Blinding-and-Permuting Protocol

Input Alice and Bob additively share an $(m+1) \times (n+1)$ matrix D. Alice has a random permutation π_C^A on $\{1, \ldots, n\}$ and a random permutation π_R^A on $\{1, \ldots, m\}$. Similarly, Bob has a random permutation π_C^B, and a random permutation π_R^B.

Output Let $\pi_C = \pi_C^A \cdot \pi_C^B$ and $\pi_R = \pi_R^A \cdot \pi_R^B$. Alice and Bob additively share $\bar{D} = \pi_R(\pi_C(D))$ with different randomness.

The protocol steps are:

1) Since D is additively split between Alice and Bob, as usual, we assume Alice has a matrix D' and Bob has a matrix D'' such that $D' + D'' = D$. Alice encrypts each entry d'_{ij} of D' using homomorphic encryption and send to Bob $E_A(d'_{ij})$ in order. We use $E_A(D')$ to denote the encrypted matrix.
2) Bob generates an $(m+1) \times (n+1)$ random matrix R'' where each entry r_{ij} is chosen randomly from $\mathbb{Z}_N$. Bob then computes $E_A(D' + R'')$, by computing $E_A(d'_{ij} + r_{ij})$ for each $i \in [1..m+1]$ and $j \in [1..n+1]$.
3) Bob permutes matrix $E_A(D'+R'')$ according to π_C^B and π_R^B. Bob sends $\pi_R^B(\pi_C^B(\ E_A(D' + R'')))$, the permuted version of $E_A(D' + R'')$, to Alice. Alice decrypts each entry and obtains $F' = \pi_R^B(\pi_C^B(D' + R''))$.
4) Bob sets each entry of D'' as $d''_{ij} = d''_{ij} - r_{ij}$ and obtains $D'' - R''$. Bob then permutes $D'' - R''$ according to π_C^B and π_R^B, and gets $F'' = \pi_R^B(\pi_C^B(D'' - R''))$. Note that $F' + F'' = \pi_R^B(\pi_C^B(D))$.
5) Alice and Bob repeat the above four steps with the roles of Alice and Bob exchanged. That is, Bob sends the encrypted matrix $E_B(F'')$ to Alice. Then Alice chooses a random matrix R', adds R' to F'', subtracts R' from F', and performs permutation on both matrices based on her π_C^A and π_R^A. At the end, Alice obtains $\bar{D}' = \pi_R^A(\pi_C^A(F' - R'))$ and Bob obtains $\bar{D}'' = \pi_R^A(\pi_C^A(F'' + R'))$, such that their sum $\bar{D}$ is equal to $\pi_R^A(\pi_C^A(F' + F'')) = \pi_R^A(\pi_C^A(\pi_R^B(\pi_C^B(D)))) = \pi_R(\pi_C(D))$.

Analysis At the end of the protocol, Alice and Bob additively share $\pi_R(\pi_C(D))$ where the permutations π_R and π_C are known to neither of them. Alice and Bob each perform $2(m+1) \times (n+1)$ homomorphic encryptions. Therefore the computational cost of the matrix blinding-and-permuting protocol is $O(mn)$.

F. Recovering the Indexes of Basic Variables

As mentioned earlier, in order to prevent leaking information about the pivot position, Alice and Bob blind-and-permute the matrix D before each pivot step. Unfortunately, one side affect of the matrix permutation is that the indexes of the basic variables are also permuted. After a series of permutations, when we figure out the basic variables of the resulting matrix, we need to find out the corresponding variables in the initial matrix. In other words, we want to compute the corresponding column index of the original matrix given a column index of the final matrix D after a series of permutations. This is described next.

Protocol 3: Indexes Recovering Protocol

Input Alice has k private permutations on $\{1, 2, \ldots, n\}$, denoted as $\pi_A^1, \pi_A^2, \ldots, \pi_A^k$; Bob has k private permutations on $\{1, 2, \ldots, n\}$, denoted as $\pi_B^1, \pi_B^2, \ldots, \pi_B^k$.

Output Letting $\pi = \pi_B^k(\pi_A^k(\cdots(\pi_B^1(\pi_A^1(\cdot)))))$, both Alice and Bob obtain $\pi^{-1}(1), \pi^{-1}(2), \ldots, \pi^{-1}(n)$.

The protocol steps are:

1) Let $x = \langle 1, 2, \ldots, n \rangle$ be an array additively split between Alice and Bob initially, e.g., Alice generates an array $x' = \langle 1, 2, \ldots, n \rangle$ and Bob generates an array $x'' = \langle 0, 0, \ldots, 0 \rangle$.
2) For $j = 1, \ldots, k$, Alice and Bob run a vector blinding-and-permuting protocol [3] (a vector version of Protocol 2), which permutes x according to π_A^j and π_B^j, such that $\pi_A^j \cdot \pi_B^j$ is known to neither Alice nor Bob.
3) Alice publishes her array x' and Bob publishes his array x''. Let $x = x' + x'' = \langle x_1, \ldots, x_n \rangle$, x is equal to the original x permuted by π. Observe that position i in the original x corresponds to position $\pi(i)$ in the resulting x. Therefore for each $i = 1, \ldots, n$, $\pi^{-1}(i) = x_i$.

Analysis Alice and Bob run a vector blinding-and-permuting protocol k times. Each vector blinding-and-permuting protocol requires n homomorphic encryptions. Thus this protocol overall takes $O(kn)$ modular exponentiations.

G. Finding the Outgoing Variables

Recall in step 3 of the simplex algorithm, we need to find $r \in [1..m]$ where

$$\frac{b_r}{a_{rs}} = \min_{\{i|a_{is}>0\}} \frac{b_i}{a_{is}} \geq 0.$$

Assume the vector b and the matrix A are additively split between Alice and Bob, this subsection aims at finding r without leaking any other information. A solution using the scramble circuit protocol directly is inefficient, because it requires at least $2m$ multiplications within the circuit.

The main idea of our solution is to find r by comparing b_j/a_{js} with b_i/a_{is}, if both of them are positive, we choose the minimum one; if one of them is negative, we discard the negative one; if both of them are negative, we choose b_j/a_{js}. In the simplex algorithm, b is always positive. If a_{is}, b_i, a_{js}, b_j are all positive, the following holds

$$\frac{b_j}{a_{js}} \leq \frac{b_i}{a_{is}} \iff b_j a_{is} \leq b_i a_{js}.$$

We define a function $f(x, y)$ as

$$f(x,y) = \begin{cases} 1 & \text{if } (x>0) \wedge (y>0) \wedge (x \leq y); \\ 1 & \text{or if } x<0; \\ 0 & \text{otherwise.} \end{cases}$$

If $f(b_j a_{is}, b_i a_{js}) = 1$, then we can discard the consideration of i; otherwise, discard the consideration of j. f can be securely implemented by a scrambled circuit protocol. The scrambled circuit protocol for f is efficient, as the size of the circuit is linear to the size of the input. The scrambled circuit protocol takes $O(\ell)$ modular exponentiations, where ℓ is the number of bits of precision per variable[2]. Due to space limit, we omit the details.

VI. Secure Linear Programming Protocol

We now "put the pieces together" and give the overall protocol. Due to space limit, we only describe the protocol for solving simplex algorithm. The detail of the secure simplex method protocol will be given in the full version of this paper.

Protocol 4: Secure Simplex Algorithm Protocol

Input Alice and Bob share A, b, c, and z_0 in additively split fashion, where A, b, c, and z_0 are the parameters of a linear program that starts in a feasible semi-canonical form.

Output Alice and Bob output $x = (x_1, x_2, \ldots, x_n)$ such that x is the optimal solution to the linear program.

Let D be the $(m+1) \times (n+1)$ matrix defined in equation (3). D is additively split between Alice and Bob. The protocol steps are:

1) *Blind-and-permute*. Alice and Bob blind-and-permute the matrix D according to Protocol 2.
2) *Find incoming variable*. Alice and Bob run a scramble circuit protocol to find whether the vector $\langle d_{11}, d_{12}, \ldots, d_{1n}\rangle$ (which corresponds to c) is positive and to find the index of minimum entry. If the vector is positive, the linear system of D is a basic optimal feasible solution, jump to step 6 to output the result. Otherwise, go to next step. We use s to denote to the index of the minimum entry, i.e., $d_{1s} = \min\langle d_{11}, \ldots, d_{1n}\rangle$.
3) *Test for unbounded z*. Alice and Bob run a scramble circuit protocol to determine whether the vector $\langle d_{2s}, d_{3s}, \ldots, d_{m+1,s}\rangle$ (which corresponds to the sth column of A) is negative. If so, report $z \to -\infty$ (the linear program has no bounded solution) and stop.
4) *Find outgoing variable*. Alice and Bob find the minimum positive ratio on vectors $\langle d_{2s}, d_{3s}, \ldots, d_{m+1,s}\rangle$ (which corresponds to the sth column of A) and $\langle d_{2,n+1}, d_{3,n+1}, \ldots, d_{m+1,n+1}\rangle$ (which corresponds to b), i.e., find r such that
$$\frac{b_r}{a_{rs}} = \min_{\{i|a_{is}>0\}} \frac{b_i}{a_{is}}$$
In case of ties, we just choose the first r. Because D is already randomly permuted, choosing the first r on permuted D is in effect equal to choosing r randomly in the original matrix D.
5) *Pivot*. Alice and Bob pivot on position $(r+1, s)$ of the matrix D using Protocol 1, then return to step 1.
6) *Output the result*. Alice and Bob first run indexes recovering protocol (Protocol 3) to obtain the relation between the variables in the resulting system with the variables in the original system. That is, let π be the overall column permutation of D, both Alice and Bob learn $\pi^{-1}(i)$ for each $i = 1, \ldots, n$. Then for each column $i \in [1..n]$, let $j = \pi^{-1}(i)$, the following steps are performed:
   a) Let us first define a function g as
   $$g(a_1, \ldots, a_m) = \begin{cases} r & \text{if } (a_r \neq 0) \wedge \\ & \quad (a_t = 0 \text{ for } t \neq r); \\ 0 & \text{otherwise.} \end{cases}$$
   If $g(a_1, \ldots, a_m) = r \neq 0$, then array a has exactly one non-zero entry at a_r. g can be implemented by a scrambled circuit protocol with $O(\ell m)$ complexity, where ℓ is the maximum bit length of a_t for $t = 1, \ldots, m$. Alice and Bob determine whether the ith column of D has one non-zero value only, i.e., they run the scramble circuit protocol to compute $r = g(d_{2,i}, \ldots, d_{m+1,i})$.
   b) If $r = 0$. Alice and Bob output $x_j = 0$.
   c) If $r \neq 0$, Alice and Bob run a split division protocol on b_r and a_{ir}, and output $x_j = b_r/a_{ir}$.

Analysis In the above protocol, step 1 requires $O(mn)$ homomorphic encryptions. Let ℓ be number of bits of precision per variable, the costs of step 2 and step 3 are $O(\ell n)$ and $O(\ell m)$, respectively. Step 4 takes $O(\ell m)$ modular exponentiations. The cost of step 5 is about $O(mn)$ modular exponentiations. Let k be number of steps repeated. In step 6, the total cost of $O(kn)$ (the cost of the indexes recovering protocol) plus $O(\ell mn)$ (the cost of evaluating g for n times). The total cost of this protocol is about $O(kmn + k\ell n + k\ell m + \ell mn)$ modular

[2]Note that the techniques from [14] is used to reduce the computation from number of bits requires to represent the modulus N to the number of bits required to represent an unsplit value.

exponentiations. Because $n \geq m$, the total cost becomes $O(k(mn + \ell n) + \ell mn)$.

A. Security Proof

In proving the security of our protocol, we use the composition theorem of [5] extensively. This theorem states that if a function f is computed by invoking functions $f_1, \ldots, f_n$ and is proven secure if these functions are "perfectly" implemented in a secure manner (i.e., by using a trusted third party), then a protocol that computes f by invoking secure protocols for $f_1, \ldots, f_n$, each of which is proven secure, securely computes f. A consequence of this theorem is that to prove a protocol that sequentially invokes functions $f_1, \ldots, f_n$ is secure in the honest-but-curious model, all that needs to be shown is that: (1) the intermediate results of the protocol do not leak information and (2) the individual functions are secure. Since all of our protocols produce additively-split outputs, the first constraint is trivially true for all of our protocols. We formalize this notion in the following lemma:

Lemma 1: If a protocol Π invokes only functions that use additively-split input and produce additively-split output, and all of these functions are individually secure in the honest-but-curious model and independent (i.e., they do not share randomness), then Π is secure in the honest-but-curious model.

PROOF. Follows directly from the composition theorem in [5]. □

Theorem 2: The secure simplex algorithm protocol is secure in the honest-but-curious model.

PROOF SKETCH. Lemma 1 shows that the secure simplex algorithm protocol is secure in the honest-but-curious model if all the sub-protocols are secure. The sub-protocols we used are matrix blind-and-permute (step 1), matrix pivot (step 5), indexes recovering protocol (step 6), and scrambled circuit protocol (step 2, 3, 4, and 6). The scramble circuit protocol has been proved to be secure in the honest-but-curious model in [26]. Matrix blinding-and-permuting protocol is secure, as the matrix is permuted under the encrypted form. Matrix pivoting protocol is secure, as it simply invokes a secure multiplication protocol multiple times. Indexes recovering protocol is also secure, because it invokes a secure blind-and-permute protocol k times. Due to space limit, we omit the detailed proofs. □

VII. HANDLING DISHONEST BEHAVIOR

Our protocol assumed the honest-but-curious model. In real applications, however, a participant of the protocol may not necessary follow the protocol if her malicious behavior cannot be detected by the other party. To show such attack, we note that our protocol replies hugely on additively split data. That is, to share x, Alice keeps x' and Bob keeps x'' such that $x = x' + x'' \bmod N$. If a participant is malicious, she may manipulate x by increasing or decreasing her share of x.

We next present an efficient solution that prevents most of the dishonest behaviors. The idea of our solution is that Alice and Bob each commits her or his private inputs. After the secure linear programming protocol, each participant proves to the other party that the results of the linear programming satisfy all the constraints. Note that it is possible to use general solutions to SFE to make our protocol secure against malicious adversaries; such solutions, however, are expensive.

A. Building Blocks

As our solution needs to use cryptographic commitment schemes and zero-knowledge proof techniques, we here briefly review the Pedersen commitment scheme [23].

Definition 3 (The Pedersen Commitment Scheme): Let p and q be two large primes such that q divides $p - 1$. Let g be a generator of G_q, the unique order-q subgroup of $\mathbb{Z}_p^*$. Let h be a random value in G_q such that $\log_g h$ is unknown to anyone. To commit a value $a \in \mathbb{Z}_q$, the prover chooses $r \leftarrow \mathbb{Z}_q$ and computes the commitment $c = (g^a h^r \bmod p)$. To open a commitment c, the prover reveals a and r, and the verifier verifies whether $c = (g^a h^r \bmod p)$.

We now review a zero-knowledge proof protocol that will be used in our solution. Given n commitments, this protocol [6] proves that the committed values satisfy a linear equation. Let $c_1, \ldots, c_n$ be the commitments of $a_1, \ldots, a_n$, i.e., $c_i = g^{a_i} h^{r_i} \bmod p$ for $i = 1, \ldots, n$. Let $x_1, \ldots, x_n, b$ be $n + 1$ integers in $\mathbb{Z}_q$. In this protocol, the prover proves to the verifier that $a_1 x_1 + \cdots + a_n x_n = b$.

Protocol 5: Zero-knowledge Protocol Proving Linear Relationship

Input Let $c_1, \ldots, c_n$, $x_1, \ldots, x_n$, and b be the common input. The prover inputs $a_1, \ldots, a_n$, and $r_1, \ldots, r_n$.

The protocol steps are:

1) Let $c = c_1^{x_1} \cdots c_n^{x_n} \bmod p$. It is easy to see that $c = g^{a_1 x_1 + \ldots, a_n x_n} h^{r_1 x_1 + \ldots, r_n x_n} \bmod p$.
2) Both the prover and the verifier computes $\sigma = c/g^b$. Note that if $a_1 x_1 + \cdots + a_n x_n = b$, then $\sigma = h^{r_1 x_1 + \ldots, r_n x_n} \bmod p$.
3) The prover proves to the verifier the knowledge of $\log_h \sigma$. This can be done using the standard challenge-response technique [24].

Analysis The computational cost of this protocol is $O(n)$.

B. Our Solution

We now present an efficient protocol that can prevent most of the dishonest behaviors in the secure collaborative linear programming protocol. The idea is to make sure the results of the linear programming satisfy the constraints, i.e., $A \cdot x = b$.

Protocol 6: Constraints Verification Protocol

The protocol steps are:

1) Alice commits each entry of A' and b' using the Pedersen commitment scheme; analogously, Bob commits each entry of A'' and b''.
2) Alice and Bob run secure linear programming protocol and obtain x, the result of the linear programming.
3) Alice and Bob proves to each other that $A \cdot x = b$. More specifically, for $j = 1 \ldots, m$:
   a) Alice computes $e_a = a'_{1j} x_1 + \cdots a'_{nj} x_n - b'_j$ and Bob computes $e_b = a''_{1j} x_1 + \cdots a''_{nj} x_n - b''_j$. Both Alice and Bob reveals e_a and e_b. If $e_a \neq e_b$, the protocol outputs failure.
   b) Given the commitments of $a'_{1j}, \ldots, a'_{nj}$ and b'_j, Alice proves that $e_a = a'_{1j} x_1 + \cdots a'_{nj} x_n - b'_j$

holds using Protocol 5. Similarly, Bob proves that $e_b = a''_{1j}x_1 + \cdots a''_{nj}x_n - b''_j$.

c) Since $e_a = e_b$, it is clear that $a'_{1j}x_1 + \cdots a'_{nj}x_n - b'_j = a''_{1j}x_1 + \cdots a''_{nj}x_n - b''_j$. In other words, the constraint $a_{1j}x_1 + \cdots a_{nj}x_n = b_j$ holds.

Analysis The cost of this protocol (excluding step 2) is $O(mn)$, as it invokes the zero-knowledge protocol for proving linear relationship m times.

If a participant of the secure linear programming protocol tries to deviate from the protocol, it is very likely that the result x would not satisfy the constraints, thus such malicious behaviors will be detected by the constraints verification protocol. Note that the constraints verification protocol cannot guarantee that the resulting solution x is optimal, however, there is no incentive for a dishonest participant to deviate from the protocol in order to obtain a non-optimal solution. Also note that, in many business scenarios, if malicious behaviors during the protocol can be detected, a rational participant will follow the protocol, otherwise, she would lose her reputation.

VIII. Conclusion and Future Work

We gave an efficient protocol for solving collaborative linear programming problems, such that neither party reveals anything about their private input to the other party (other than what can be deduced from the result). The amount of computation done by our protocol is proportional to the time complexity of the simplex method. Such a protocol enables collaborative optimizations where the two parties can jointly minimize costs (or maximize profits) without violating their own private constraints. This work is only a first step towards a comprehensive toolkit for secure and private collaborative optimization. Natural extensions include:

- Nonlinear optimization, e.g., simulated annealing, Tabu search, etc. As stated earlier, assuming integer inputs is not a limitation on our techniques because of the linearity of the optimization problem we are solving (we can scale up everything to an integer value and scale the answers back down). This, however, cannot be done for non-linear optimization problems, the handling of which we leave for future work.
- Incentive compatibility through the introduction of "transfer payments", so that the participant gain nothing by lying about their inputs.
- Fully extend our work to the malicious model and the multi-party model.

Acknowledgment

Portions of this work were supported by Grants IIS-0325345, IIS-0219560, IIS-0312357, IIS-0242421, and CNS-0627488 from the National Science Foundation, and by sponsors of the Center for Education and Research in Information Assurance and Security. We thank the anonymous reviewers for their helpful comments.

References

[1] I. Adler and N. Megiddo. A simplex algorithm whose average number of steps is bounded between two quadratic functions of the smaller dimension. *Journal of the ACM*, 32(4):871–895, 1985.

[2] M. J. Atallah, M. Bykova, J. Li, K. B. Frikken, and M. Topkara. Private collaborative forecasting and benchmarking. In *Proceedings of the 3rd ACM Workshop on Privacy in the Electronic Society*, Oct. 2004.

[3] M. J. Atallah, F. Kerschbaum, and W. Du. Secure and private sequence comparisions. In *Proceedings of the 2nd ACM Workshop on Privacy in the Electronic Society*, Oct. 2003.

[4] D. Boneh and M. Franklin. Efficient generation of shared RSA keys. *Journal of the ACM*, 48(4):702–722, 2001.

[5] R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[6] R. Cramer and I. Damgård. Zero-knowledge proof for finite field arithmetic, or: Can zero-knowledge be for free? In *Advances in Cryptology: CRYPTO '98*, volume 1462 of *LNCS*, pages 424–441. Springer, 1998.

[7] R. Cramer and I. Damgård. Secure distributed linear algebra in a constant number of rounds. In *Advances in Cryptology: CRYPTO '01*, volume 2139 of *LNCS*, pages 119–136. Springer, 2001.

[8] I. Damgård and M. Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *Proceedings of the 4th International Workshop on Practice and Theory in Public Key Cryptography*, pages 119–136. Springer, 2001.

[9] I. Damgård and M. Jurik. A length-flexible threshold cryptosystem with applications. In *Proceedings of the 8th Australasian Conference on Information Security and Privacy*, volume 2727 of *LNCS*, pages 350–364. Springer, 2003.

[10] G. B. Dantzig and M. N. Thapa. *Linear Programming 1: Introduction*. Springer, Mar. 1997.

[11] W. Du. *A Study of Several Specific Secure Two-party Computation Problems*. PhD thesis, Purdue University, West Lafayette, Indiana, 2001.

[12] I. Foster and C. Kesselman, editors. *The grid: blueprint for a new computing infrastructure*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[13] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 15(3):200–222, 2001.

[14] K. B. Frikken and M. J. Atallah. Privacy preserving route planning. In *Proceedings of the 3rd ACM Workshop on Privacy in the Electronic Society*, pages 8–15, Oct. 2004.

[15] N. Gilboa. Two party RSA key generation. In *Advances in Cryptology: CRYPTO '99*, volume 1666 of *LNCS*, pages 116–129. Springer, 1999.

[16] O. Goldreich. *The Foundations of Cryptography — Volume 2*. Cambridge University Press, May 2004.

[17] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the 19th ACM Conference on Theory of Computing*, pages 218–229, May 1987.

[18] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

[19] V. Klee and G. J. Minty. How good is the simplex algorithm? In O. Shisha, editor, *Inequalitites III*, pages 159–175. Academic Press, 1972.

[20] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay – Secure two-party computation system. In *Proceedings of the 13th USENIX Security Symposium*, pages 287–302. USENIX, 2004.

[21] T. Okamoto, S. Uchiyama, and E. Fujisaki. Epoc: Efficient probabilistic public-key encryption. In *IEEE P1363: Protocols from other families of public-key algorithms*, Nov. 1998.

[22] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology: EUROCRYPT '99*, volume 1592 of *LNCS*, pages 223–238. Springer, 1999.

[23] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology: CRYPTO '91*, volume 576 of *LNCS*, pages 129–140. Springer, 1991.

[24] C. P. Schnorr. Efficient identification and signatures for smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[25] R. J. Vanderbei. *Linear Programming, Second Edition - Foundations and Extensions*. Kluwer Academic Publishers, May 2001.

[26] A. C. Yao. How to generate and exchange secrets. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, pages 162–167. IEEE Computer Society Press, 1986.