# Hiccups on the Road to Privacy-Preserving Linear Programming

Alice Bednarz Nigel Bean Matthew Roughan

School of Mathematical Sciences, University of Adelaide, SA, 5005, Australia
{alice.bednarz, nigel.bean, matthew.roughan}@adelaide.edu.au

## ABSTRACT

Linear programming is one of maths' greatest contributions to industry. There are many places where linear programming could be beneficially applied across more than one company, but there is a roadblock: companies have secrets. The data needed for joint optimization may need to be kept private because of concerns about leaking competitively sensitive data, or due to privacy legislation.

Recent research has tackled the problem of privacy-preserving linear programming. One appealing group of approaches uses a 'disguising' transformation to allow one party to perform the joint optimization without seeing the secret data of the other parties. These approaches are very appealing from the point of view of simplicity, efficiency, and flexibility, but we show here that all of the existing transformations have a critical flaw.

## Categories and Subject Descriptors

G.1.6 [**Numerical Analysis**]: Optimization—*Linear programming*; K.4.4 [**Computers and Society**]: Electronic Commerce—*Distributed commercial transactions*

## General Terms

Algorithms, Security

## 1. INTRODUCTION

Optimization techniques are valuable tools. They can help businesses improve efficiency and minimise costs, but often companies have private data they wish to protect. This may prevent them from taking part in joint-optimization ventures with partners, even though such ventures could be mutually beneficial. For example, delay of traffic delivered through a shared network could be reduced if the network operators cooperated. However, they may be unwilling to do so for fear of (i) revealing company secrets, (ii) breaching privacy legislation, or (iii) possible embarrassment.

Often, optimization problems can be formulated as linear programs. In particular, we consider jointly-defined linear programs where two or more parties each contribute various components to form a global optimization problem. The critical information to be kept hidden is dependent on the situation but could include the constraint set (which may contain private information about budgets, financial health, production capacity, or network layout) and the objective function (which may express costs or company aims). The goal is to solve the joint-program and yet keep the private information hidden as much as possible (acknowledging that the optimum solution must unavoidably leak some information about the parameters). We shall refer to this type of problem as 'privacy-preserving linear programming'.

More formally, suppose that we have two parties, Alice and Bob, who each hold some private data. They wish to combine their data into a single linear program, of the general form

$$\begin{array}{lrcl} \max & \mathbf{c}^T\mathbf{x} & & \\ \text{s.t.} & M\mathbf{x} & \leq & \mathbf{b} \\ & \mathbf{x} & \geq & \mathbf{0}, \end{array} \tag{1}$$

where the various components come from the two parties. For instance $M = M_1 + M_2$, $\mathbf{b} = \mathbf{b_1} + \mathbf{b_2}$, and $\mathbf{c} = \mathbf{c_1} + \mathbf{c_2}$, where $M_1$, $\mathbf{b_1}$ and $\mathbf{c_1}$ are contributed by Alice (and must be kept secret from Bob), and $M_2$, $\mathbf{b_2}$ and $\mathbf{c_2}$ are likewise held by Bob (and must be kept secret from Alice). The meaning of these components will be discussed later, but note that the privacy requirement prevents us from using standard linear programming techniques such as the Simplex algorithm.

To the best of our knowledge, two different approaches to privacy-preserving linear programming have been developed. The first is *transformation-based*. Methods of this type are essentially heuristic and disguise the linear program via random matrices. One party can then solve the problem in the disguised domain. Two such methods have been proposed by Du [4] and Vaidya [13]. However, we present a flaw in these techniques. The problem is so severe that the answer returned by them may actually be *infeasible*.

The second approach invokes *formal cryptographic techniques* [8, 12, 14] to implement a privacy-preserving version of the popular Simplex method. These methods hide the private data by using cryptographic techniques at each step in the algorithm. The solution in [14] is the most efficient, but it currently only applies to the case where one party owns the objective function, and the other owns the constraints.

Though the formal cryptographic approaches provide very good security, there are some appealing aspects of the transformation-based approaches. They avoid much of the conceptual complexity of the cryptographic techniques, because the information hiding protocols are only needed once at the start of the algorithm, not at each step. This also drastically reduces the computational complexity and communications overhead of the cryptographic approach. Moreover, the user is no longer bound to use Simplex, but can take advantage of the last half-century of improvements to linear-programming algorithms. Finally, we can use the full power of floating point arithmetic.

In this paper we first provide a precise explanation of the flaws in the existing transformation-based methods. We then show that Vaidya's transformation is valid under additional restrictions, but unfortunately this comes at the expense of security unless an untrusted third party is available. Developing other transformation methods with improved security is the subject of ongoing research.

## 2. BACKGROUND AND RELATED WORK

### 2.1 Secure Multiparty Computation

The idea of Secure Multiparty Computation (SMC) was introduced by Yao [17, 18]. He considered two parties, one with private data $x$, and the other with private data $y$, who wish to jointly compute $f(x, y)$, such that only the output is public, while the private data remains hidden from the other person. Yao presented general results, for the two-party case, showing that *any* function $f(x, y)$ that could be represented as a Boolean circuit could be computed in a secure fashion [18]. Yao's work was soon generalized to the $n$-party case by Goldreich, Micali and Wigderson [5]. Together, these papers proved the remarkable result that *any* polynomial-time function could be computed in a secure fashion.

An important issue is how to model possible cheating by the participants. The most common model is to assume all participants are "honest-but-curious", which means that they will follow the protocol correctly, but will conduct extra calculations 'on the side' in an attempt to gain information about others' private data [5]. The goal in designing secure protocols is to ensure such additional calculations are fruitless. One can also allow "malicious" adversaries, who may deviate from the protocol [5]. In this paper we adopt the "honest-but-curious" model, since we assume that the solution to the joint-LP is of mutual benefit to all parties, so no-one has an incentive to deliberately sabotage the protocol.

At the time of its inception, SMC was of largely theoretical interest. However, the development of more efficient algorithms and increased computing power has led to a number of real-world applications. These include privacy-preserving data mining [9, 15], secure voting [2], and secure auctions [10].

Last year saw the first large-scale commercial application of SMC, namely an online double-auction of sugar-beet licenses in Denmark [1]. The auction was successful, involving 1200 bidders and 25,000 tonnes of production rights. SMC provided a low-cost electronic solution and circumvented the need for security policies.

### 2.2 Privacy-Preserving Linear Programming

A Linear Program (LP) is an optimization problem of the form shown in (**??**), which has a linear objective function and linear constraints [16]. Many years of effort have gone into deriving fast, computationally efficient methods for solving LPs. For instance, the Simplex method [3] and Interior Point Methods [7]. However, these all assume that $\mathbf{c}, \mathbf{b}$ and $M$ are completely known.

Privacy-preserving linear programming involves LPs with private data. We can express the two-party problem as

$$\begin{array}{lrcl} \max & (\mathbf{c_1}+\mathbf{c_2})^T\mathbf{x} & & \\ \text{s.t.} & (M_1+M_2)\mathbf{x} & \leq & (\mathbf{b_1}+\mathbf{b_2}) \\ & \mathbf{x} & \geq & \mathbf{0}, \end{array} \tag{2}$$

where $\mathbf{x}$ is an $n$-dimensional vector of variables, $\mathbf{c_1}+\mathbf{c_2}$ is an $n$-dimensional vector of objective function coefficients, $M_1 + M_2$ is an $m \times n$ matrix of constraint coefficients (where there are $m$ constraint inequalities) and $\mathbf{b_1}+\mathbf{b_2}$ is an $m$-dimensional vector of constraint values. The components $\{\mathbf{c_1}, M_1, \mathbf{b_1}\}$ contain Alice's data, while $\{\mathbf{c_2}, M_2, \mathbf{b_2}\}$ contain Bob's data. Precisely which of these components are private depends on the situation. We now outline some possible scenarios.

The constraint matrix $(M_1 + M_2)$ can be either:

1. **Public**: known in full by both parties;
2. **Private**: known in full by only *one* of the parties (which means either $M_1 = 0$ or $M_2 = 0$); or
3. **Shared**: known in *part* by each of the parties, such that each party knows only a subset of the total constraint coefficients. In this case the constraint matrix can either be:
   - **Row-Partitioned:** each party contributes entire *rows* of the constraint matrix. Thus $M_1 + M_2$ is row partitioned, i.e. $M_1 + M_2 = \begin{bmatrix} M_1' \\ \hdashline 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \hdashline M_2' \end{bmatrix}$. This case corresponds to having a set of shared variables, but constraints that express private informaiton.
   - **Column-Partitioned:** each party contributes entire *columns* of the constraint coefficient matrix. Thus $M_1 + M_2$ is column-partitioned, i.e. $M_1+M_2 = \left[\, M_1' \,\vdots\, 0 \,\right] + \left[\, 0 \,\vdots\, M_2' \,\right]$. Here each party controls a subset of private variables, with collaborative constraints.
   - **Mixed:** each party contributes a *mix* of rows and columns.

Similarly, the objective function coefficients $\mathbf{c_1}+\mathbf{c_2}$ can be either (i) public, (ii) private, or (iii) shared. In the case of (iii), the only options are column-partitioned or mixed, since we only have a single row vector.

Privacy-preserving linear programming is a relatively new research area. Li and Atallah [8] and Toft [11, 12] each implement a privacy-preserving version of the popular Simplex method, where Toft [12] identified several problems with Li and Atallah's technique. Although different in their execution, both methods utilise cryptographic techniques such as homomorphic encryption, blind and permute procedures, and secret sharing at each step of the algorithm. By considering a specific class of problems, Vaidya [14] devised a more efficient approach based on revised Simplex.

Du [4] presented another, much simpler, method, based on disguising the problem using random matrices, but it is flawed. Part of the flaw has been recognised by Vaidya [13] but his alternative method still has the potential to produce infeasible output. We first describe Du's method in its original form, and what goes wrong during its execution, before discussing Vaidya's version.

## 3. DU'S TRANSFORMATION METHOD

The main idea behind Du's method is to convert the original LP into an equivalent system in which the constraints have been *disguised* by multiplication with random matrices. One party solves the disguised problem, and then the solution is converted back to the original domain. Du's full protocol can be found in [4]. Central to Du's method is the assertion that the transformed LP is equivalent to the original LP. Here we consider the critical elements and see why this equivalence fails.

The original LP — which we will refer to as Program A — is shown below.

*Program A*

$$\begin{array}{lrcl} \max & f(\mathbf{x}) = & \mathbf{c}^T\mathbf{x} & \\ \text{s.t.} & M\mathbf{x} & \leq & \mathbf{b} \\ & \mathbf{x} & \geq & \mathbf{0}, \end{array} \tag{3}$$

Program A is disguised via 2 random matrices: an $m \times m$ matrix $P$ and an $n \times n$ matrix $Q$. These matrices are populated with random elements, with the restrictions that the matrices are *invertible*, and the elements of $P$ and $Q^{-1}$ are all *strictly positive*. Bob generates the random matrices, and then Alice and Bob jointly engage in secure multiplication protocols such that *only Alice* receives the coefficients of the disguised program, Program B, given as:

*Program B*
$$\begin{array}{llrcl} \max & \hat{f}(\hat{\mathbf{x}}) = & \hat{\mathbf{c}}^T\hat{\mathbf{x}} & & \\ \text{s.t.} & & \hat{M}\hat{\mathbf{x}} & \leq & \hat{\mathbf{b}} \\ & & \hat{\mathbf{x}} & \geq & \mathbf{0}, \end{array} \tag{4}$$

where $\hat{M} = PMQ$, $\hat{\mathbf{b}} = P\mathbf{b}$, $\hat{\mathbf{c}}^T = \mathbf{c}^TQ$, and $\hat{\mathbf{x}} = Q^{-1}\mathbf{x}$.

Alice must not know $P$ or $Q$, and Bob is not permitted to know Program B. Otherwise, they could derive the other's private data.

Alice solves Program B and obtains the solution $\hat{\mathbf{x}}^*$. She sends this to Bob, who transforms it back to the original variables by premultiplying by $Q$ to obtain: $\mathbf{x}^* = Q\hat{\mathbf{x}}^*$, which Du claims is the optimal solution to the original program.

On the surface, the transformation process from Program A to Program B appears to be valid — after all, $QQ^{-1}$ is simply the identity matrix. Indeed, for all $\mathbf{x}$,

$$f(\mathbf{x}) = \mathbf{c}^T\mathbf{x} = \mathbf{c}^TQQ^{-1}\mathbf{x} = \hat{\mathbf{c}}^T\hat{\mathbf{x}} = \hat{f}(\hat{\mathbf{x}}). \tag{5}$$

Also, since $P$ and $Q^{-1}$ contain strictly positive elements, they should not disturb the inequality constraints or the non-negativity constraints (see [4, Theorem 4.3.1]).

## 3.1 Why Programs A & B are not equivalent

For Du's method to hold, Programs A and B need only have the same feasible region (if we plot them both in terms of $\mathbf{x}$) since we have shown in (5) that the objective functions are equivalent.

The problem with Du's method is that by choosing $P$ and $Q^{-1}$ to be positive, we force $P^{-1}$ and $Q$ to contain some negative elements. These play havoc with inequality constraints: for a simple example consider what happens to the constraint $x \geq 0$ when we multiply by $-1$; it becomes $x \leq 0$. The combination of negative matrix elements and the inequality constraints mean that the constraints forming the two feasible regions are not equivalent. More precisely, if the constraint $\hat{\mathbf{x}} \geq \mathbf{0}$ is satisfied in Program B, this does *not* guarantee that the corresponding constraint, $\mathbf{x} \geq \mathbf{0}$ will be satisfied in Program A, and likewise $\hat{M}\hat{\mathbf{x}} \leq \hat{\mathbf{b}}$ does *not* guarantee that $M\mathbf{x} \leq \mathbf{b}$.

Let us consider a brief example to bring home the point. We consider the following optimization problem (Program A):

$$\begin{array}{llrcl} \max & f(\mathbf{x}) = & 5x_1 + \ x_2 & & \\ \text{s.t.} & & 2x_1 + 3x_2 & \leq & 12 \\ & & 2x_1 + \ x_2 & \leq & 8 \\ & & x_1, x_2 & \geq & 0, \end{array} \tag{6}$$

and transform it using matrices $P = \begin{bmatrix} 1 & 3 \\ 5 & 1 \end{bmatrix}$ and $Q^{-1} = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$. Obviously, these are not random, but the problem is generic and will occur with almost all choices of $P$ and $Q^{-1}$. The transformed problem (Program B) is then

$$\begin{array}{llrcl} \max & \hat{f}(\hat{\mathbf{x}}) = & -\hat{x_1} + \ 3\hat{x_2} & & \\ \text{s.t.} & & \frac{4}{3}\hat{x_1} + \frac{10}{3}\hat{x_2} & \leq & 36 \\ & & \frac{20}{3}\hat{x_1} + \ \frac{8}{3}\hat{x_2} & \leq & 68 \\ & & \hat{x_1}, \hat{x_2} & \geq & 0. \end{array} \tag{7}$$

The optimal solution for the original LP (6) is $\mathbf{x}^* = (4, 0)$, with objective function $f_{\max} = 20$. The solution to the transformed problem (7) is $\hat{\mathbf{x}}^* = (0, 10.8)$ with $\hat{f}_{\max} = 32.4$. When we transform back into the original coordinates this solution corresponds to $\mathbf{x}^* = (7.2, -3.6)$ which is infeasible since $x_2^*$ is negative.

How can the transformed problem result in an infeasible solution? Consider the associated feasible regions shown in Figure 1. Program A's (6) feasible region is shown in black, while the feasible region of Program B (7), represented in the original coordinates, is shown in grey. Notice that it is larger than the original feasible region, and it even extends outside the non-negative quadrant (where

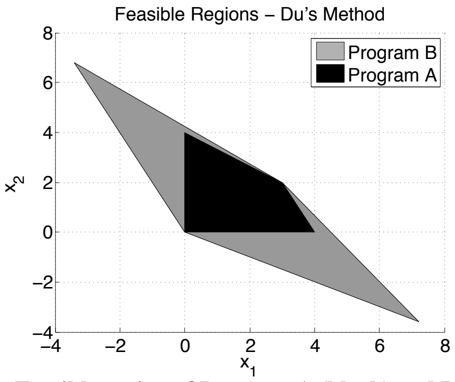

**Figure 1: Feasible region of Program A (black) and Program B (grey) after applying Du's method.**

feasible solutions cannot exist). Hence, it is possible to find a solution to (7) that is infeasible for (6).

By choosing $P$ and $Q^{-1}$ to be strictly positive, we ensure that any feasible solution to Program A also satisfies the constraints of Program B, and hence the feasible set of A is a subset of that of B. The problem with Du's transformation is that the converse is not true, since the inverses of $P$ and $Q^{-1}$ contain negative elements. Hence we cannot correct the problem by instead requiring $P^{-1}$ and $Q$ to be positive, as this will break the equivalence in the other direction.

## 4. VAIDYA'S METHOD

Vaidya [13] recognised that Du's method fails in some cases, without providing detail. Vaidya developed a new method for a specific scenario in which one party holds the objective function, and the other party holds all the constraints. For this case, he proposed a transformation approach based on Du's method, with the difference being that matrix $P$ is not used, so the disguise is imparted only by matrix $Q$. Vaidya avoids the troubles caused by $P$'s impact on the inequality constraints, but his approach does not resolve the problems with the non-negativity constraints. For instance, let's reuse the example program (6), and this time apply only the associated $Q^{-1}$ to arrive at a new Program B:

$$\begin{array}{llrcl} \max & \hat{f}(\hat{\mathbf{x}}) = & -\hat{x_1} + 3\hat{x_2} & & \\ \text{s.t.} & & \frac{4}{3}\hat{x_1} + \frac{1}{3}\hat{x_2} & \leq & 12 \\ & & \hat{x_2} & \leq & 8 \\ & & \hat{x_1}, \hat{x_2} & \geq & 0, \end{array} \tag{8}$$

where $\hat{\mathbf{x}} = Q^{-1}\mathbf{x}$, $\hat{M} = MQ$, $\hat{\mathbf{c}}^T = \mathbf{c}^TQ$.

The transformed problem (8) has solution $\hat{\mathbf{x}} = (0, 8)$ with $\hat{f}_{\max} = 24$. When we transform back into the original coordinates, we obtain $\mathbf{x}^* = (5\frac{1}{3}, -2\frac{2}{3})$, which is once again infeasible since $x_2$ is negative.

Comparing the feasible regions of Program A and Program B in Figure 2, we see that the feasible region of Program B is still larger and extends outside the non-negative quadrant. However, within the non-negative quadrant the feasible regions now agree, as a result of removing $P$ from the transform.

## 4.1 Validity versus Security

The oversight in both methods is that we cannot choose random matrices $A$ such that both $A$ and $A^{-1}$ are strictly positive. If we relax strict positivity to allow non-negative elements (we allow zeros) then there *are* some matrices that satisfy this requirement. These are generalized permutation matrices (products of non-singular diagonal and permutation matrices) [6, Theorem 1.1].

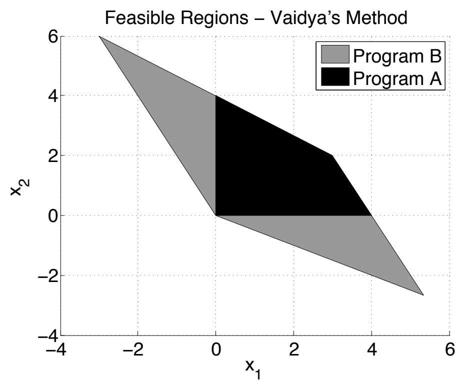

**Figure 2: Feasible region of Program A (black) and Program B (grey) after applying Vaidya's method.**

So Vaidya's transformation will be valid if we require $Q$ to be a generalized permutation matrix. We can then deconstruct $Q$ into a positive diagonal *scaling* matrix $D$ and a permutation matrix $L$.

The drawback is that generalized permutation matrices provide a much lower degree of disguise, and can severely compromise security. In fact, if Alice knew the permutation $L$, then she could use her knowledge of $\mathbf{c}$ and $\hat{\mathbf{c}}$ to deduce $D$ precisely. There are only finitely many possible permutations, and so Alice can consider each in turn. For instance, Alice might start by guessing the permutation to be $L_i$, and from this determine $D_i$ and hence $Q_i$ such that

$$\hat{\mathbf{c}}^T = \mathbf{c}^T Q_i. \tag{9}$$

In the current form of the problem Alice also learns both the transformed solution $\hat{\mathbf{x}}^*$, and the untransformed solution $\mathbf{x}^*$, and so can then test whether $\mathbf{x}^* = Q_i\hat{\mathbf{x}}^*$. If it is not, then she can rule out that particular permutation. She will in fact be able to rule out most permutations in this fashion. The only permutations she cannot eliminate occur when $\mathbf{x}^*$ contains repeated values: in these cases she cannot eliminate permutations that swap such repeated values.

Consider our example (6) again and choose $Q = DL$ to be the generalized permutation matrix

$$Q = \begin{bmatrix} 2 & 0 \\ 0 & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 2 \\ \frac{1}{2} & 0 \end{bmatrix}.$$

Alice knows $\mathbf{c}^T = \begin{bmatrix} 5 & 1 \end{bmatrix}$, and from Bob she receives $\hat{\mathbf{c}}^T = \mathbf{c}^T Q = \begin{bmatrix} \frac{1}{2} & 10 \end{bmatrix}$. Since Alice knows that $Q$ only applies a generalized permutation, she considers the two possible cases:

*Case 1:* $L_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ and hence $Q_1 = \begin{bmatrix} \frac{1}{10} & 0 \\ 0 & 10 \end{bmatrix}$

*Case 2:* $L_2 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ and hence $Q_2 = \begin{bmatrix} 0 & 2 \\ \frac{1}{2} & 0 \end{bmatrix}$.

Alice knows the transformed solution $\hat{\mathbf{x}}^* = (0, 2)$ and the untransformed solution $\mathbf{x}^* = (4, 0)$, and just tests which $Q_i$ matrix is consistent.

$$i = 1: \quad Q_1\hat{\mathbf{x}}^* = \begin{bmatrix} \frac{1}{10} & 0 \\ 0 & 10 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 0 \\ 20 \end{bmatrix} \neq \mathbf{x}^*$$

$$i = 2: \quad Q_2\hat{\mathbf{x}}^* = \begin{bmatrix} 0 & 2 \\ \frac{1}{2} & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 4 \\ 0 \end{bmatrix} = \mathbf{x}^*$$

Therefore Alice knows that $Q = Q_2$, and can immediately deduce Bob's private data $M$. Thus this approach does not have satisfactory security.

However, the security breach described can be avoided if we assume the existence of an untrusted third party, Eve. We can let Eve, instead of Alice, solve the disguised program, thus eliminating the attack arising from any party knowing both $\hat{\mathbf{c}}$ and $\mathbf{c}$.

## 5. CONCLUSIONS

The privacy-preserving linear programming method proposed by Du [4] is incorrect due to problems with the disguise transformation. Specifically, the feasible region of the transformed LP is not equivalent to the feasible region of the original LP, leading to infeasible solutions being returned by the method. Vaidya [13] went some way to improving Du's transformation, but his method can still give infeasible results.

The door is open for new ways of implementing 'disguising' transformations, or for new methods altogether that manage to achieve some of the desirable qualities of Du's and Vaidya's methods, such as conceptual simplicity, computational simplicity, and the freedom to use any commercial LP solver. This is the subject of ongoing research.

## 6. ACKNOWLEDGMENTS

This work was partially supported by ARC Grant DP0985063.

## 7. REFERENCES

[1] P. Bogetoft, D. L. Christensen, I. Damgard, M. Geisler, T. Jakobsen, M. Krøigaard, J. D. Nielsen, J. B. Nielsen, K. Nielsen, J. Pager, M. Schwartzbach, and T. Toft. Multiparty computation goes live. Available at http://www.sikkerhed.alexandra.dk/uk/projects/simap/MPCannouncement.pdf, 2008.

[2] R. Cramer, R. Gennaro, and B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. *European Transactions on Telecommunications*, 8(5):481–490, 1997.

[3] G. Dantzig. *Linear programming and extensions*. Princeton Univ Pr, 1963.

[4] W. Du. *A study of several specific secure two-party computation problems*. PhD thesis, Purdue University, Indiana, 2001.

[5] O. Goldreich, S. Micali, and A. Wigderson. How to play ANY mental game. *Proceedings of the 19th annual ACM conference on Theory of Computing*, pages 218–229, 1987.

[6] T. Kaczorek. *Positive 1D and 2D systems*. Springer, 2002.

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

[8] J. Li and M. Atallah. Secure and private collaborative linear programming. In *Proceedings of International Conference on Collaborative Computing*, pages 1–8, 2006.

[9] Y. Lindell and B. Pinkas. Privacy preserving data mining. *Journal of Cryptology*, 15(3):177–206, March 2002.

[10] M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM Conference on Electronic Commerce*, pages 129–139. ACM New York, 1999.

[11] T. Toft. *Primitives and Applications for Multi-party Computation*. PhD thesis, University of Aarhus, Denmark, 2007.

[12] T. Toft. Solving linear programs using multiparty computation. In *Financial Cryptography and Data Security '09*, Barbados, 2009.

[13] J. Vaidya. Privacy-preserving linear programming. In *Proceedings of the 2009 ACM symposium on Applied Computing*, pages 2002–2007. ACM New York, NY, USA, 2009.

[14] J. Vaidya. A secure revised simplex algorithm for privacy-preserving linear programming. In *International Conference on Advanced Information Networking and Applications*, pages 347–354, 2009.

[15] J. Vaidya, C. Clifton, and M. Zu. *Privacy Preserving Data Mining*. Springer Science+Business Media, Inc., 2006.

[16] R. J. Vanderbei. *Linear Programming: Foundations and Extensions*. Kluwer Academic Publishers, 1996.

[17] A. Yao. Protocols for secure computations. *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science*, pages 160–164, 1982.

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