New Attacks against Transformation-Based Privacy-Preserving Linear Programming*

Peeter Laud[†] Alisa Pankova[†,‡,§]

[†]Cybernetica AS
[‡]Software Technology and Applications Competence Centre (STACC)
[§]University of Tartu, Institute of Computer Science
{peeter.laud|alisa.pankova}@cyber.ee

June 3rd, 2013

Abstract

In this paper we demonstrate a number of attacks against proposed protocols for privacy-preserving linear programming, based on publishing and solving a transformed version of the problem instance. Our attacks exploit the geometric structure of the problem, which has mostly been overlooked in the previous analyses and is largely preserved by the proposed transformations. The attacks are efficient in practice and cast serious doubt to the viability of transformation-based approaches in general.
Keywords: Cryptanalysis, Secure multiparty computation, Linear programming

1 Introduction

Linear programming (LP) is one of the most versatile polynomial-time solvable optimization problems. It is usually straightforward to express various production planning and transportation problems as linear programs. There exist LP solving algorithms that are efficient both in theory and in practice. If the instances of these problems are built from data belonging to several mutually distrustful parties, the solving procedure must preserve the privacy of the parties. Thus it would be very useful to have an efficient privacy-preserving protocol that the data owners (and possibly also some other parties that help with computation) could execute for computing the optimal solution to a linear program that is obtained by combining the data of different owners. It is likely that such protocol would directly give us efficient privacy-preserving protocols for many other optimization tasks.

Several such protocols have indeed been proposed, following one of two main approaches. In the *secure multiparty computation (SMC) approach*, composable protocols for privacy-preserving arithmetic and relational operations are used to build a privacy-preserving implementation of some LP solving algorithm, typically the simplex algorithm. In the *transformation-based approach*, the algebraic structure of systems of linear inequalities and equations is used to apply a linear transformation to the description of the original problem, thus disguising it and allowing it to be solved publicly.

*The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 284731 (Usable and Efficient Secure Multiparty Computation, UaESMC), and from the European Regional Development Fund through the Estonian Center of Excellence in Computer Science, EXCS.

The security properties of the protocols of SMC approach can be derived from the properties of the protocols for primitive arithmetic and relational operations through composability. The privacy guarantees these protocols offer are thus pretty well understood. The transformation-based methods have so far lacked the understanding of their privacy properties at a comparable level. The current paper demonstrates that such unavailability of security definitions is dangerous.

2 Privacy-Preserving Linear Programming

The *canonical form* for a linear programming task is the following:

$$\text{minimize } \mathbf{c}^{\mathrm{T}} \cdot \mathbf{x}, \text{ subject to } \mathrm{A}\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq \mathbf{0} \ . \tag{1}$$

Here A is an $m \times n$ matrix, $\mathbf{b}$ is a vector of length m and $\mathbf{c}$ is a vector of length n. There are n variables in the vector $\mathbf{x}$. The inequality of vectors is defined pointwise.

The LP solving algorithms, as well as protocols for privacy-preserving solution commonly expect the task to be in the *standard form*:

$$\text{minimize } \mathbf{c}^{\mathrm{T}} \cdot \mathbf{x}, \text{ subject to } \mathrm{A}\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0} \ . \tag{2}$$

The inequality constraints of the canonical form can be transformed to equality constraints by introducing *slack variables*. The system of constraints $\mathrm{A}\mathbf{x} \leq \mathbf{b}$, $\mathbf{x} \geq \mathbf{0}$ is equivalent to the system $\mathrm{A}\mathbf{x} + \mathrm{I}\mathbf{x}_{\mathrm{s}} = \mathbf{b}$, $\mathbf{x}, \mathbf{x}_{\mathrm{s}} \geq 0$, where I is $m \times m$ identity matrix and $\mathbf{x}_{\mathrm{s}}$ is a vector of m new variables.

In the privacy-preserving setting, the elements of the matrix A and the vectors $\mathbf{b}, \mathbf{c}$ are somehow contributed by several different parties. The cost vector $\mathbf{c}$ may be either held entirely by some party, or its entries may belong to different parties. Two standard ways of partitioning the constraints $\mathrm{A}\mathbf{x} \leq \mathbf{b}$ are the horizontal partitioning (each party contributes some of the constraints) and the vertical partitioning (each party knows certain columns of the matrix A). More general ways of data partitioning are possible, but these are not considered by the transformation methods that we are attacking.

In general, there are two main approaches to privacy-preserving linear programming. One approach is the straightforward cryptographic implementation of a privacy-preserving version of some LP solving algorithm [13, 9]. Its main problem is efficiency since the entire optimization process must be performed in a manner that protects all intermediate values and comparison results. Another approach is transforming the program such a way that it could be given to a solver for offline computation. The optimal solution to the initial program has to be recoverable from the optimal solution to the transformed program.

In this work we present new attacks against some of the existing transformation methods. Without lessening the generality, we assume the number of parties to be 2, called Alice and Bob.

2.1 Transformation methods

Transformation-based methods have been proposed in [4, 3, 14, 11, 12, 15, 8, 2, 10, 7]. A set of "standard" transformations, applicable to the initial program, have been proposed over the years. Depending on the partitioning of constraints and the objective function, the application of a transformation may require cryptographic protocols of varying complexity. Each of the methods proposed in the literature typically uses several of these standard transformations.

Multiplying from the left. The idea of multiplying A and $\mathbf{b}$ in (2) by a random $m \times m$ invertible matrix P from the left was first introduced by Du [4]. This transformation conceals the outer appearance of A and $\mathbf{b}$, but the feasible region remains unchanged.

Multiplying from the right. The idea of multiplying A and $\mathbf{b}$ in (2) by a random invertible matrix Q from the right was also proposed by Du [4]. This hides also the cost vector $\mathbf{c}$. Unfortunately, it changes the optimal solution if some external constraints (e.g. the non-negativity constraints) of the form $\mathrm{B}\mathbf{x} \geq \mathbf{b}'$ are present, as it has been shown in [2]. In this case, the vector $\mathbf{b}'$ should also be modified according to the transformation, but that in fact reveals all the information about Q.

Scaling and Permutation. Bednarz et al. [2] have shown that, in order to preserve the inequality $\mathbf{x} \geq \mathbf{0}$, the most general type of Q is a positive generalized permutation matrix (a square matrix where each row and each column contains exactly one non-zero element). This results in scaling and permuting the columns of A. This transformation may also be applied to a problem in the canonical form (1).

Shifting. The shifting of variables has first been proposed in [3], and it has been also used in [15]. This transformation is achieved by replacing the constraints $\mathrm{A}\mathbf{x} \leq \mathbf{b}$ with $\mathrm{A}\mathbf{y} \leq \mathbf{b} + \mathrm{A}\mathbf{r}$, where $\mathbf{r}$ is a random non-negative vector of length n and $\mathbf{y}$ are new variables, related to the variables $\mathbf{x}$ through the equality $\mathbf{y} = \mathbf{x} + \mathbf{r}$. To preserve the set of feasible solutions, the inequalities $\mathbf{y} \geq \mathbf{r}$ have to be added to the system. A different transformation must then be used to hide $\mathbf{r}$.

2.2 Security Definition

There are no formal security definitions used in the transformation-based approach. The definition that has been used in the previous works is the *acceptable security*. This notion was first used in [5].

Definition 1. A protocol achieves acceptable security if the only thing that the adversary can do is to reduce all the possible values of the secret data to some domain with the following properties:

1. The number of values in this domain is infinite, or the number of values in this domain is so large that a brute-force attack is computationally infeasible.
2. The range of the domain (the difference between the upper and lower bounds) is acceptable for the application.

Although acceptable security could make the analysis simpler, it is not very well applicable in practice. Attacks on schemes that are secure by this definition have been found [2, 1]. The security of different transformation methods is very dependent on the initial settings of the problem — the partitioning of initial data, as well as on the type of used constraints (inequalities or equations).

2.3 Classification of Initial Settings

For each of the proposed transformation methods, the applicability and security strongly depend on the initial settings of the problem. For that reason, Bednarz [1] has introduced a classification of initial settings, provided with corresponding notation. She proposes to consider the following parameters:

Objective Function Partitioning How is the vector $\mathbf{c}$ initially shared? Is it known to Alice, to Bob, or to both of them? Are some entries known to Alice and others to Bob? Or does $\mathbf{c} = \mathbf{c}_{\text{Alice}} + \mathbf{c}_{\text{Bob}}$ hold, where $\mathbf{c}_{\text{Alice}}$ is "completely" unknown to Bob and vice versa?

Constraint Partitioning How is the matrix A initially shared? Is it public, known to one party, partitioned horizontally or vertically, or additively shared?

RHS Vector Partitioning How is the vector $\mathbf{b}$ initially shared?

Allowable Constraint Types Does the method admit only equality constraints, only inequalities, or both of them? Note that admitting only equality constraints means that the "natural" representation of the optimization problem is in terms of equalities. The use of slack variables to turn inequalities to equalities is not allowed.

Allowable Variable Types May the variables be assumed non-negative? Or may they be assumed free? Or can both types be handled?

Additionally, the classification considers which party or parties learn the optimal solution. This aspect does not play a role for our attacks.

The attacks described in this paper mostly target the transformation methods for LP tasks where the constraints are in the form of inequalities (1), and the set of constraints has been horizontally partitioned between Alice and Bob. The optimization direction $\mathbf{c}$ and its sharing does not play a big role in the main attacks, although some proposed transformation methods leave into it information that makes the attacks simpler. In our treatment, we assume all variables to be non-negative.

2.4 Overview of proposed methods

For exactly the setting described in the previous paragraph, Bednarz [1, Chap. 6] has proposed the following transformation. The set of constraints in (1) is transformed to

$$\hat{\mathrm{A}}\mathbf{y} = \hat{\mathbf{b}}, \mathbf{y} \geq \mathbf{0}, \tag{3}$$

where $\hat{\mathrm{A}} = \mathrm{P}\begin{pmatrix}\mathrm{A} & \mathrm{I}\end{pmatrix}\mathrm{Q}$, $\hat{\mathbf{b}} = \mathrm{P}\mathbf{b}$, I is the $m \times m$ identity matrix, P is a random invertible $m \times m$ matrix and Q is a random positive $(m+n) \times (m+n)$ generalized permutation matrix. New variables $\mathbf{y}$ are related to the original variables $\mathbf{x}$ and the slack variables $\mathbf{x}_{\mathrm{s}}$ by the equation $\binom{\mathbf{x}}{\mathbf{x}_{\mathrm{s}}} = \mathrm{Q}\mathbf{y}$. The objective function is disguised as $\hat{\mathbf{c}}^{\mathrm{T}} = (\mathbf{c}^{\mathrm{T}}\ \mathbf{0}^{\mathrm{T}})Q$, where $\mathbf{0}$ is a vector of m zeroes.

Other proposed transformations for horizontally partitioned constraints can be easily compared with Bednarz's. Du [4] applied the multiplication with both P and Q (where Q was more general) directly to the system of inequalities (1). Unfortunately, this transformation did not preserve the feasible region (and possibly the optimal solution) as shown by Bednarz et al. [2]. Vaidya [14] uses only the matrix Q, with similar correctness problems. Mangasarian [12] uses only the multiplication with P for a system with only equality constraints (2). Hong et al. [8] propose a complex set of protocols for a certain kind of distributed linear programming problems. Regarding the security, they prove that these protocols leak no more than what is made public by Bednarz's transformation. Li et al. [10] propose a transformation very similar to Bednarz's, only the matrix Q is selected from a more restricted set. This transformation is analyzed by Hong and Vaidya [7] and shown to provide no security (their attack has slight similarities with the one we present in Sec. 3.2). They propose a number of methods to make the transformation more secure and to also hide the number of inequalities in (1), including the addition of superfluous constraints and the use of more than one slack variable per inequality to

turn them to equalities. We will further discuss the use of more slack variables in Sec. 3.1. The transformation by Dreier and Kerschbaum [3], when applied to (1), basically shifts the variables (Sec. 2.1), followed by Bednarz's transformation. We discuss the details and attacks specific to this transformation in Sec. 3.3.

3 Attacks

The system of constraints (1) consists of m inequalities of the form $\sum_{i=1}^{n} a_{ji}x_i \leq b_j$ for $j \in \{1, \ldots, m\}$, in addition to the non-negativity constraints. We assume that Alice knows the first r of these inequalities.

When Alice attempts to recover (1) from the result of Bednarz's transformation (3), she will first try to locate the slack variables, as described in Sec. 3.1. When she has located the slack variables, she can remove these, turning the equalities back to inequalities of the form $\mathrm{A}'\mathbf{x}' \leq \mathbf{b}'$. These constraints are related to (1) by $\mathrm{A}' = \mathrm{P}'\mathrm{A}\mathrm{Q}'$, $\mathbf{b}' = \mathrm{P}'\mathbf{b}$, where both P' and Q' are generalized permutation matrices (of size $m \times m$ and $n \times n$, respectively; Q' is also positive). Multiplication with P' from the left does not actually change the constraints, so the goal of Alice is to find Q'. The correspondence of the variables in $\mathbf{x}$ and $\mathbf{x}'$ can be found by looking at scale-invariant quantities related to constraints. Once the correspondence is found, the scaling factors can be easily recovered. All this is described in Sec. 3.2.

3.1 Identifying the Slack variables

3.1.1 Looking at the objective function

When we add the slack variables to the system of inequalities in order to turn them to equations, then the coefficients of these slack variables in the cost vector $\mathbf{c}$ will be 0. In the existing transformation methods, the cost vector $\mathbf{c}$ is hidden by also multiplying it with a monomial matrix Q from the right. In this way, the zero entries in $\mathbf{c}$ are not changed. If all original variables had non-zero coefficients in the objective function, then the location of zeroes in the transformed vector $\mathbf{c}$ tells us the location of slack variables.

This issue can be solved by applying the transformation to the *augmented form* of linear program that includes the cost vector into the constraint matrix, and the cost value is expressed by a single variable:

$$\text{minimize } w, \text{ subject to } \begin{pmatrix} 1 & -\mathbf{c}^{\mathrm{T}} & 0 \\ 0 & \mathrm{A} & \mathrm{I} \end{pmatrix} \begin{pmatrix} w \\ \mathbf{x} \\ \mathbf{x_s} \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \mathbf{b} \end{pmatrix}, \begin{pmatrix} w \\ \mathbf{x} \\ \mathbf{x_s} \end{pmatrix} \geq \mathbf{0} \ . \tag{4}$$

The slack variables may be now hidden amongst the real variables by permutation. The location of the variable w should be known to the solver, although he may also solve all the n instances of linear programming tasks: for each variable in the task, try to minimize it.

There may be possibly other means of hiding $\mathbf{c}$. Hence we introduce more attacks that are not related to $\mathbf{c}$.

3.1.2 Looking at sizes of entries

If the positions of slack variables have been hidden in the cost vector, they may be located by exploiting the structure of A. Namely, after the slack variables are introduced, they form an identity matrix that is attached to A from the right. Thus each slack column contains exactly one non-zero entry. The columns of A are very unlikely to contain just one non-zero entry. We have found that the columns of $\mathrm{P}\begin{pmatrix} \mathrm{A} & \mathrm{I} \end{pmatrix}$ can be distinguished by performing statistical analysis

on the sizes of their entries. Even if using both positive and negative entries in A makes the mean more or less the same, the variance is smaller for the slack variables. The following scaling of the columns with the entries of Q does not provide any more protection.

We have discovered this problem occasionally, just because the columns appeared too different after applying the existing transformation methods. The previous works do not state precisely the distribution from which the entries of P (and Q) should be sampled. We have made experiments where we have sampled these entries independently of each other, according to the uniform distribution, or the normal distribution (the parameters of the distribution are currently unimportant, they only affect the scale of the resulting matrix, as well as the variance of its entries relative to each other). It turns out that selecting the entries of P randomly according to either one of these distributions keeps the variables distinguishable.

We performed a series of experiments, described below in detail. First, let us define the following probability distribution:

Definition 2. If a random variable X is distributed according to the normal distribution $\mathcal{N}(\mu, \sigma^2)$, then the distribution of the absolute value $|X|$ is called the *folded normal distribution* and is denoted $\mathcal{N}_{\mathrm{f}}(\mu, \sigma^2)$.

Our experiments were parametrized by the following quantities:

- the number of variables n and the number of inequality constraints m in (1);
- the fraction $p \in [0, 1]$ of zero entries in A;
- the fraction $a \in [0, 1]$ of constraints with non-negative coefficients;
- the fraction $q \in [0, 1]$ of zero entries in P;

We performed two sets of experiments. In one of them we sampled the entries of P, Q from a uniform distribution, and in the other one from a normal distribution.

An experiment proceeded as follows.

1. Generate a random point $v = (v_1, \ldots, v_n) \in \mathbb{R}^n$ where v_i is chosen uniformly from $(0, 100]$. This point will be contained in the polyhedron defined by the constraints in (1), thereby ensuring its non-emptiness.
2. Generate a random $m \times n$ matrix $\mathrm{A} = (a_{ij})_{i,j=1,1}^{m,n}$ whose entries are assigned in the following way:
   - The value 0 is taken with the probability p.
   - A random value is sampled uniformly from $[-100, 100] \subseteq \mathbb{R}$ (or from a normal distribution $\mathcal{N}(0, 100)$) with probability $1 - p$.
   - After a row of A is generated, with probability a all entries in this row are replaced with their absolute values.
3. Generate the entries of the vector $\mathbf{b}$ of length m in such a way that the polyhedron defined by $\mathrm{A}\mathbf{x} \leq \mathbf{b}$ definitely contains the point v. That is, for each $i \in \{1, \ldots, m\}$, compute $b_i = a_{i1}v_1 + \ldots + a_{in}v_n + s$, where s is a random positive number. In our experiments, s was chosen uniformly from $[1000, 2000]$.
4. Let P be a $m \times m$ random matrix, the entries of which are assigned in the following way:
   - The value 0 is taken with the probability q (except the main diagonal, which stays non-zero in any case).

- A random value is sampled uniformly from $[-100, 100]$ (or from a normal distribution $\mathcal{N}(0, 100)$) with probability $1 - q$.

Note that P is invertible with probability 1.

5. Let Q be a $(m + n) \times (m + n)$ random positive generalized permutation matrix. The permutation defined by Q was picked uniformly from S_{m+n} and the non-zero entries of Q were uniformly sampled from $[1, 100]$ (or sampled from a folded normal distribution $\mathcal{N}_f(0, 100)$).

6. Construct $\hat{\mathrm{A}}$ and $\hat{\mathbf{b}}$ according to Bednarz's transformation.

7. For each column of $\hat{\mathrm{A}}$ compute the mean and the variance of its entries. Find the sets of m columns where (a) the means are the largest, (b) the means are the smallest, (c) the variances are the largest, or (d) the variances are the smallest.

8. The experiment was considered *successful* if one of the four sets of m columns found in the previous step exactly corresponded to the slack variables in $\mathbf{y}$ introduced by Bednarz's transformation.

When sampling the entries of P, Q from the uniform distribution, we ran 5 experiments for all possible values of the parameters, where $m+n \in \{100, 250, 500\}$, $m/(m+n) \in \{25\%, 50\%, 75\%\}$, $p, q \in \{0\%, 25\%, 50\%, 75\%, 90\%\}$, and $a \in \{0\%, 25\%, 50\%, 100\%\}$. For almost all settings, there was at least one experiment that was successful. The experiments were less successful only if m was small and p was large. When sampling the entries of P, Q from the normal distribution, we ran the same number of experiments with the same parameters, except that the case $m+n = 500$ was not covered. Again, for most settings, at least one of the experiments was successful. Again, we had less success if many entries in A were 0 (i.e. p was large) and there were less constraints than variables (i.e. $m/(m + n)$ was small).

This problem can be potentially resolved by scaling the columns by a value that comes from a sufficiently large distribution to hide these differences. Although this makes the columns approximately the same size, it makes the values of the slack variables in the optimal solution to the transformed LP task much smaller than the values of the original variables, still keeping them distinguishable. Also, this modification does not affect the variances of the variables.

Another way is to add extra constraints whose entries that are large enough to provide noise for all the variables. The problem is that introducing more constraints requires introducing more slack variables for correctness. These slack variables cannot be protected by the same method. Once they have been revealed, they may be removed from the system by Gaussian elimination.

We would also like to note that the adversary may always bring the transformed matrix to its reduced row echelon form. This means that this transformation provides the best possible hiding, and the security analysis should be performed on this form. Unfortunately, it cannot be used for hiding instead of P since it is expensive to compute it while preserving the privacy.

3.1.3 Sampling the vertices of the polyhedron

If the previous attack does not work well because the random values used during the transformation have been sampled so, that the entries of the resulting matrix have similar distributions, then there are still more ways of locating the slack variables. Consider (3), where each of the new variables $y_i \in \mathbf{x}$ is either a scaled copy of some original variable $x_{i'} \in \mathbf{x}$ or a (scaled) slack variable. The constraints (3) define an n-dimensional polyhedron in the space $\mathbb{R}^{m+n}$ (due to its construction, the matrix $\hat{\mathrm{A}}$ has full rank). In each vertex of this polyhedron, at least n of the

variables in $\mathbf{y}$ are equal to zero. We have hypothesized that for at least a significant fraction of linear programs, it is possible to sample the vertices of this polyhedron in such manner, that slack variables will be 0 more often than the original variables.

To verify our hypothesis, we performed a series of experiments, described below in detail. Our experiments were parametrized by the quantities m, n, p, a described at the previous experiment. Additionally, the number $k \in \mathbb{N}$ determines the number of vertex samples done in an experiment, and the fraction $e \in [0, 1]$ affects the polyhedron that we use to look for variables that most often take the value 0 in vertices.

An experiment proceeded as follows.

1–6. Generate A, $\mathbf{b}$, $\hat{\mathrm{A}}$, $\hat{\mathbf{b}}$ as in the previous experiment, using the current values of m, n, p, a, and taking $q = 0$. The entries of all matrices are sampled from the uniform distribution.

7. Modify $\hat{\mathrm{A}}$ [resp. $\hat{\mathbf{b}}$] by removing their first $e \cdot m$ rows [resp. elements]. This corresponds to discarding a fraction of e equations from the system $\hat{\mathrm{A}}\mathbf{y} = \hat{\mathbf{b}}$. We have found that such removal increases the success rate of the experiments for certain parameters.

8. Initialize the counters $z_1, \ldots, z_{m+n}$ to 0.

9. Repeat the following k times.

   (a) Generate the optimization direction $\mathbf{c} \in \mathbb{R}^{m+n}$ sampling each entry from the distribution $\mathcal{N}_{\mathrm{f}}(0, 1)$.

   (b) Find an optimal *basic* solution (a solution located in a vertex of the polyhedron) to the linear program

   $$\text{minimize } \mathbf{c}^{\mathrm{T}} \cdot \mathbf{y}, \text{ subject to } \hat{\mathrm{A}}\mathbf{y} = \hat{\mathbf{b}}, \mathbf{y} \geq \mathbf{0} \ .$$

   (c) If the optimal solution $\mathbf{y}_{\mathrm{opt}}$ exists, then increase by one each z_i where the i-th element of $\mathbf{y}_{\mathrm{opt}}$ equals 0.

10. The experiment was considered *successful* if the counters with n largest values exactly corresponded to the slack variables in $\mathbf{y}$ introduced by Bednarz's transformation.

We have performed our experiments with different settings. In all experiments, k was fixed to 100 (larger values did not seem to give any significant difference). For each set of values for the parameters (m, n, p, a, e), we performed 20 experiments. The results for all sets of experiments are reported in Table 1. For given (m, n, p, a), the symbol $*$ in the corresponding cell of the table indicates that none of 20 experiments performed for all values of e we considered were successful. If at least one experiment was successful for some value of e, given the parameters (m, n, p, a), then this value of e is given in the corresponding cell of the table.

We also performed some initial experiments where the entries of the optimization direction $\mathbf{c}$ were sampled from $\mathcal{N}(0, 1)$. This choice did not perform better (and sometimes performed much worse) than the sampling from $\mathcal{N}_{\mathrm{f}}(0, 1)$.

We see that the worst case for our algorithm is when m is much smaller than n and the fraction of zero entries in A is large. The problem is that there are too few inequalities already in the beginning, and the zeroes make the initial matrix A even sparser and less constraining. The initial variables thus do not differ too much from the slack variables.

For $m > n$ it may happen that even the slack variables will not be allowed to take the value 0 at all because of too tight bounds. In this case, some equations have been just eliminated from the transformed program. This is not equivalent to removing bounds from the initial polyhedron, and it is not quite clear what exactly happens to it. However, there are definitely

m	n	p	a			
			0.0	0.25	0.5	1.0
25	75	0	*	0	0	0
		0.25	*	*	0	0
		0.5	0	0	0	0
		0.75	0	0	0	0
		0.9	*	*	*	*
50	50	0	0	0	0	0
		0.25	0	0	0	0
		0.5	0	0	0	0
		0.75	*	0	0	0
		0.9	*	*	*	*
62	188	0	*	*	*	*
		0.25	*	*	*	*
		0.5	*	*	*	0
		0.75	*	*	*	0
		0.9	*	*	*	*
75	25	0	0.75	0.5	0.75	0.5
		0.25	0.75	0.5	0.75	0.5
		0.5	0.75	0.5	0.75	0.5
		0.75	0.75	0.5	0.75	0.5
		0.9	*	*	0.75	0.5
125	125	0	0	0	0	0
		0.25	0	*	0	0
		0.5	0	0	0	0
		0.75	*	0	0	0
		0.9	*	*	*	*

m	n	p	a			
			0.0	0.25	0.5	1.0
125	375	0	*	*	*	*
		0.25	*	*	*	*
		0.5	*	*	*	*
		0.75	*	*	*	*
		0.9	*	*	*	*
187	63	0	0.75	0.75	0.75	0.5
		0.25	0.75	0.5	0.75	0.5
		0.5	0.75	0.75	0.75	*
		0.75	*	0.75	0.75	0.5
		0.9	*	*	*	0.9
250	250	0	*	0	*	0
		0.25	*	0	*	0
		0.5	*	*	*	0
		0.75	*	*	*	0
		0.9	*	*	*	*
375	125	0	*	0.75	0.75	0.5
		0.25	0.75	0.75	0.75	0.75
		0.5	0.75	*	*	0.5
		0.75	*	0.75	*	0.5
		0.9	*	0.75	*	0.75
475	25	0	0.9	0.9	0.9	0.9
		0.25	0.9	0.9	0.9	0.9
		0.5	0.9	0.9	0.9	0.9
		0.75	0.9	0.9	0.9	0.9
		0.9	0.9	0.9	0.9	0.9

Table 1: Results of the vertex-sampling experiments

less constraints than before, and the slack variables again have higher probabilities of becoming 0.

The results also show something interesting about the effect of the structure of A on the outcome of the attack. It can be seen than the attack performs better when all the entries of A are non-negative. The success rate is in general higher for smaller fraction of zero elements in A, especially for the smaller number of constraints.

Our experimental results show that for many linear programs in canonical form (1), it is possible to identify the slack variables after Bednarz's transformation. The validity of our hypothesis has been verified.

3.1.4 Several slack variables per inequality

The authors of [7] proposed introducing multiple slack variables for the same inequality. We have tried experimentally that in this case there is even higher probability that the slack variables are those that most often take the value 0 in a vertex sampled as described previously; this can also be explained in theory. Also, in this case, the columns in $\hat{\text{A}}$, corresponding to slack variables added to the same inequality, are multiples of each other. This makes them easily locatable.

3.1.5 Removing the slack variables

Once we have located the slack variables, we will reorder the variables in the constraints $\hat{\mathrm{A}}\mathbf{y} = \hat{\mathbf{b}}$ so, that the non-slack variables are the first n variables and the slack variables are the last m variables in $\mathbf{y}$. This corresponds to the first n columns of $\hat{\mathrm{A}}$ containing the coefficients of non-slack variables in the system of equations, and the last m columns containing the coefficients of slack variables. We will now use row operations to bring the system to the form $\begin{pmatrix}\mathrm{A}' & \mathrm{I}\end{pmatrix}\mathbf{y} = \mathbf{b}'$, where I is $m \times m$ identity matrix. This system, together with the non-negativity constraints, is equivalent to the system of inequalities $\mathrm{A}'\mathbf{x}' \leq \mathbf{b}'$, where $\mathbf{x}'$ are the first n elements of $\mathbf{y}$.

3.2 Finding the permutation of variables

We will now describe the attack that allows to remove the scaling and the permutation of variables. Alice knows r inequalities $\sum_{i=1}^{n} a_{ji}x_i \leq b_j$ (where $j \in \{1, \ldots, r\}$) of the original system of constraints, from a total of m. We assume that r is at least 2. Alice also knows all scaled and permuted constraints $\sum_{i=1}^{n} a'_{ji}x'_i \leq b'_j$ (where $j \in \{1, \ldots, m\}$). If we could undo the scaling and permuting, then this set of m inequalities would contain all original r inequalities known by Alice. As next we show how Alice can recover the permutation of the variables. Once this has been recovered, the scaling is trivial to undo.

Alice picks two of the original inequalities she knows (e.g. k-th and l-th, where $1 \leq k, l \leq r$) and two inequalities from the scaled and permuted system (e.g. k'-th and l'-th, where $1 \leq k', l' \leq m$). She makes the guess that k-th [resp. l-th] original inequality is the k'-th [resp. l'-th] scaled and permuted inequality. This guess can be verified as follows. If the guess turns out to be correct, then the verification procedure also reveals the permutation (or at least parts of it).

For the inequality $\sum_{i=1}^{n} a_{ji}x_i \leq b_j$ in the original system let H_j be the corresponding hyperplane where "$\leq$" has been replaced by "$=$". Similarly, let H'_j be the hyperplane corresponding to the j-th inequality in the scaled and permuted system. The hyperplane H_j intersects with the i-th coordinate axis in the point $(0, \ldots, 0, z_{ji}, 0, \ldots, 0)$, where $z_{ji} = b_j/a_{ji}$ (here z_{ji} is the i-th component in the tuple). Also, let $(0, \ldots, 0, z'_{ji}, 0, \ldots, 0)$ be the point where H'_j and the i-th coordinate axis intersect.

Note that scaling the (initial) polyhedron s times along the i-th axis would increase z_{ji} by s times, too, for all j. Scaling it along other axes would not change z_{ji}. Hence the quantities z_{ki}/z_{li} (for $i \in \{1, \ldots, n\}$) are scale-invariant.

To verify her guess, Alice computes the (multi)sets $\{z_{ki}/z_{li} \mid 1 \leq i \leq n\}$ and $\{z'_{k'i}/z'_{l'i} \mid 1 \leq i \leq n\}$. If her guess was correct, then these multisets are equal. Also, if they are equal, then the i-th coordinate in the original system can only correspond to the i'-th coordinate in the scaled and permuted system if $z_{ki}/z_{li} = z'_{k'i'}/z'_{l'i'}$. This allows her to recover the permutation. If there are repeating values in the multisets, or if division by 0 occurs somewhere, then she cannot recover the complete permutation. In this case she repeats with other k, l, k', l'. But note that the presence of zeroes in the coefficients also gives information about the permutation.

This attack does not allow to discover precise permutations if the known inequalities are symmetric with respect to some variables, and the scaling cannot be derived for the variables whose coefficients in all the known inequalities are 0. It is also impossible if the right sides of all the known inequalities are 0. However, it would reduce the number of secure linear programming tasks significantly. Also, if two variables in the system look the same for Alice (they participate in the same way in all inequalities she knows) then it should not matter to her how they end up in the recovered permutation.

We have followed up our experiments reported in the previous section, and verified that the attack works in practice.

3.3 Attacks specific to [3]

Dreier and Kerschbaum [3] propose a transformation that is applicable to LP tasks containing both equality and inequality constraints. In this paper, we only consider its application to tasks with inequality constraints only (although the operations presented in this section are also applicable to equations). In their transformation, the variables are first shifted by a positive vector (as described in Sec. 2.1), and then Bednarz's transformation is applied to the resulting system. In [3], the construction is described somewhat differently and the resulting positive generalized permutation matrix Q used to scale and permute the columns of the constraint system is not the most general matrix possible. The attacks described below work for any possible Q.

3.3.1 Shifting back

The shifting of variables that has been used in [3] (and also in the transformation presented by Wang et al. [15], which only applies to LP tasks with equality constraints, and is thus outside the scope of this paper) reduces to scaling. The inequalities $\mathbf{y} \geq \mathbf{r}$ for the variables $\mathbf{y}$ are transformed to equalities by the introduction of new slack variables $\mathbf{s}$. For the variable $y_i \in \mathbf{y}$, related to the original variable x_i through the equality $y_i = x_i + r_i$, we have the equality $y_i - s_i = r_i$, where s_i is a new slack variable. After applying Bednarz's transformation, the variables are scaled and this equality becomes $q_i\hat{y}_i - q'_i\hat{s}_i = r_i$. The new variables $\hat{y}_i$ and $\hat{s}_i$ are related to the previous ones by $y_i = q_i\hat{y}_i$ and $s_i = q'_i\hat{s}_i$, where q_i and q'_i are certain non-zero entries in the matrix Q. Thus $\hat{s}_i = (q_i\hat{y}_i - r_i)/q'_i = (y_i - r_i)/q'_i = x_i/q'_i$. I.e. the slack variable $\hat{s}_i$ is a scaled copy of the original variable x_i.

We could now eliminate the variables $\mathbf{y}$ (the shifted versions of the original variables $\mathbf{x}$) from the system of constraints and the objective function. We will then be left with the system that involves only the slack variables $\mathbf{s}$ from the inequalities $\mathbf{y} \geq \mathbf{r}$ and the slack variables $\mathbf{x}_\mathrm{s}$ from the inequalities in the original system. The resulting LP task could have been obtained from the original task through Bednarz's transformation and the attacks described above can be applied to it.

To eliminate the variables $\mathbf{y}$, we need to know their location. Dreier's and Kerschbaum's transformation [3] does not actually hide these variables, due to their choice of Q. But even if the permutation encoded in Q were more general, we could still recover the locations of the variables $\mathbf{y}$ as described below. The procedure described below also recovers the pairs $(\hat{y}_i, \hat{s}_i)$ of variables and corresponding slack variables, the difficulty of which is postulated in the cryptanalysis performed in [3].

3.3.2 Affine relationships in small sets of variables

Each variable from $\mathbf{y} = \mathbf{x} + \mathbf{r}$ is associated with exactly one slack variable from $\mathbf{s}$. To find the pairs $(\hat{y}_i, \hat{s}_i)$, the adversary can just pick pairs of variables and then verify that they correspond to each other. The correspondence that the adversary can verify is the affine relationship $q_i\hat{y}_i - q'_i\hat{s}_i = r_i$ between these variables.

This problem can be stated more generally. Suppose that we have a linear equation system $\mathrm{A}\mathbf{x} = \mathbf{b}$. Consider the solution space of this system. If the space contains small sets of t variables that are in affine relationship $\alpha_1 x_{i_1} + \ldots + \alpha_t x_{i_t} = \beta$ for some $\alpha_i, \beta \in \mathbb{R}$ (that may be not obvious from the outer appearance A), then these equations may be recovered by looking through all the sets of variables of size t. To expose the affine relationship between $x_{i_1}, \ldots, x_{i_t}$, we will just use Gaussian elimination to get rid of all other variables. The procedure can be described as follows:

1. Repeat the following, until only variables $x_{i_1}, \ldots, x_{i_t}$ remain in the system.
   (a) Pick any other variable x_j that has not been removed yet.
   (b) Take an equation where x_j has non-zero coefficient. Through this equation, express the variable x_j in terms of the other variables. Substitute it into all the other equations. Remove the equation and the variable x_j. If there are no equations where x_j has non-zero coefficient, then remove only x_j, without touching any remaining equations.
2. The previous operations do not change the solution set of the system (for the remaining variables). Therefore, if there are any equations left, then there exist $\alpha_i, \beta \in \mathbb{R}$ (not all $\alpha_i = 0$) such that $\alpha_1 x_{i_1} + \ldots + \alpha_t x_{i_t} = \beta$.

In this manner, the adversary is able to find all unordered pairs $\{\hat{y}_i, \hat{s}_i\}$ related to each other through $q_i\hat{y}_i + q'_i\hat{s}_i = r_i$. The signs of q_i, q'_i, r_i in this relationship determine, which one is the original variable ($q_i r_i > 0$), and which one the slack variable ($q'_i r_i < 0$).

4 Conclusions

We have presented attacks against transformation-based methods for solving LP tasks in privacy-preserving manner. The attacks are not merely theoretical constructions, but work with reasonable likelihood on problems of practical size.

We have presented our attacks against methods that handle LP tasks where the constraints are specified as inequalities. May the methods for differently-represented LP tasks, e.g. as systems of equations [12, 15], still be considered secure? Our attacks are not directly applicable against this setting because the set of equations representing the subspace of feasible solutions is not unique and the hyperplanes in the original and transformed systems of constraints cannot be directly matched against each other like in Sec. 3.2. In our opinion, one still has to be careful because there is no sharp line delineating systems of constraints represented as equations, and systems of constraints represented as inequalities. The canonical form (1) and the standard form (2) can be transformed to each other and the actual nature of the constraints may be hidden in the specified LP task.

The lack of precise definitions of confidentiality for transformation-based methods makes it harder to argue about the (in)security of a particular method. Further advances in this field would benefit from an indistinguishability-based definition of security, similar to [6]. In such a definition, the adversary would be allowed to pick two LP tasks, one of which would then be transformed by the environment. The adversary's goal is to find out, which of the two tasks was transformed. In this definition, it would also be possible to precisely state which parts of the task the transformation will not attempt to protect: the environment would check that these parts are equal for the two tasks selected by the adversary.

References

[1] Alice Bednarz. *Methods for two-party privacy-preserving linear programming.* PhD thesis, University of Adelaide, 2012.

[2] Alice Bednarz, Nigel Bean, and Matthew Roughan. Hiccups on the road to privacy-preserving linear programming. In *Proceedings of the 8th ACM workshop on Privacy in the electronic society*, WPES '09, pages 117–120, New York, NY, USA, 2009. ACM.

[3] Jannik Dreier and Florian Kerschbaum. Practical privacy-preserving multiparty linear programming based on problem transformation. In *SocialCom/PASSAT*, pages 916–924. IEEE, 2011.

[4] Wenliang Du. *A Study Of Several Specific Secure Two-Party Computation Problems*. PhD thesis, Purdue University, 2001.

[5] Wenliang Du and Zhijun Zhan. A practical approach to solve secure multi-party computation problems. In *New Security Paradigms Workshop*, pages 127–135. ACM Press, 2002.

[6] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[7] Yuan Hong and Jaideep Vaidya. An inference-proof approach to privacy-preserving horizontally partitioned linear programs. *Optimization Letters*, 2013. To appear. Published online 05 October 2012.

[8] Yuan Hong, Jaideep Vaidya, and Haibing Lu. Secure and efficient distributed linear programming. *Journal of Computer Security*, 20(5):583–634, 2012.

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

[10] Wei Li, Haohao Li, and Chongyang Deng. Privacy-preserving horizontally partitioned linear programs with inequality constraints. *Optimization Letters*, 7(1):137–144, 2013.

[11] Olvi L. Mangasarian. Privacy-preserving linear programming. *Optimization Letters*, 5(1):165–172, 2011.

[12] Olvi L. Mangasarian. Privacy-preserving horizontally partitioned linear programs. *Optimization Letters*, 6(3):431–436, 2012.

[13] Tomas Toft. Solving linear programs using multiparty computation. In Roger Dingledine and Philippe Golle, editors, *Financial Cryptography and Data Security*, pages 90–107, Berlin, Heidelberg, 2009. Springer-Verlag.

[14] Jaideep Vaidya. Privacy-preserving linear programming. In Sung Y. Shin and Sascha Ossowski, editors, *SAC*, pages 2002–2007. ACM, 2009.

[15] Cong Wang, Kui Ren, and Jia Wang. Secure and practical outsourcing of linear programming in cloud computing. In *INFOCOM, 2011 Proceedings IEEE*, pages 820–828, 2011.