University of Tartu
Faculty of Mathematics and Computer Science
Institute of Computer Science

Alisa Pankova

Insecurity of Transformation-Based Privacy-Preserving Linear Programming

Master's thesis (30ECTS)

Supervisor: Peeter Laud
Co-supervisor: Margus Niitsoo

Author: ..... "....." May 2013
Supervisor: ..... "....." May 2013
Co-supervisor: ..... "....." May 2013

Allowed to defend
Professor: ..... "....." May 2013

Tartu 2013

Contents

Abstract

Applied mathematics is used in many real-world problems. Solving some of these problems may involve sensitive data. In this case, cryptographic techniques become necessary. Although it has been proven that any function can be computed securely, it is still a question how to do it efficiently. While it may be difficult to solve optimization tasks securely and efficiently in general, there may still be solutions for some particular classes of tasks, such as linear programming. This thesis gives an overview of the transformation-based privacy-preserving linear programming. The thesis introduces some problems of this approach that have been present in the previous works and demonstrates its insecurity. It presents concrete attacks against published methods following this approach. Possible methods of protection against these attacks are proposed. It has been proven that there are issues that cannot be resolved at all using the particular known class of efficient transformations that has been used before.

1 Introduction

This thesis is devoted to protection of sensitive data in the field of applied mathematics. In general, this work considers cryptography-like methods that are more related to linear algebra than to classical cryptography.

1.1 Problem Statement

Applied mathematics is a field that deals with real-world problems. One large class of problems that it solves are optimization tasks. A company would like to know how it has to share its resources in order to maximize its profit, or how to distribute the work in such a way that the expenses would be minimized. There exist many standard mathematical ways how it could be done.

Different companies often have a strong collaboration with each other. In order to maximize their profit in the best way, the optimization task may be needed to be solved based on the data of several different companies. A good example that is used in many works (like [3]) is related to transportation companies that may significantly reduce the total distance their trucks have to cover if they adjust their routes according to each other. However, it may happen that the companies do not want to reveal their data, or it may be just prohibited by law. They want to be sure that their data remains private and will not be revealed in any way. This is the point where using cryptographic methods becomes essential.

It has been proven that it is in fact possible to compute securely any function [27]. The question is how to do it efficiently. If the companies need to wait several months until the function will be computed, it may happen that the situation changes, and the same function should be computed based on a completely different input.

Narrowing the set of mathematical tasks to a particular class of problems, it is possible that there exists an easier and a more efficient way of solving them. In this work, we consider *linear programming* optimization tasks.

1.2 Outline

First of all, we explain what is linear programming and why the privacy of its input data may be important. We show some different protection methods that have already been proposed in related works. We give a brief overview of the existing security definitions, explain why they are too weak, and then state another security definition that seems standard and reasonable. We find out which requirements have to be satisfied in order to achieve this security definition. In the end, we conclude that it is impossible to achieve it using a known class of efficient techniques of privacy-preserving linear programming.

1.3 Contribution

The aim of this thesis is to give an overview of existing approaches and their problems together with some new attacks. After that, it introduces new protection methods that nevertheless still do not solve all the problems. The thesis proposes a reasonably standard and practical security definition. This security definition is proven to be unachievable for any affine transformation, and this proof is the main contribution of this thesis.

1.4 Notation

Throughout this work, we use the following notation:

- Matrices are denoted with upright capital letters: A, B.
- Sets are denoted with italic capital letters: V, E.
- Vectors are denoted with bold font, while their elements remain non-bold. A vector of length n is denoted as $\mathbf{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$. A zero vector is denoted as $\mathbf{0} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}$.
- The *transpose* of a vector denotes just writing the vector not as a column, but as a row: $\mathbf{x}^{\mathrm{T}} = (x_1, \ldots, x_n)$.
- The binary relational operations $\leq, =, \geq, <, >$ are applied componentwise to the corresponding vectors: $\mathbf{x} \leq \mathbf{y} \iff (x_1 \leq y_1) \wedge (x_2 \leq y_2) \wedge \ldots \wedge (x_n \leq y_n)$.
- The binary arithmetic operations $+, -$ are applied componentwise to the corresponding vectors: $\mathbf{x} + \mathbf{y} = \begin{pmatrix} x_1 + y_1 \\ \vdots \\ x_n + y_n \end{pmatrix}$.

  The exception is the $\cdot$ operation which denotes the scalar product of two vectors: $\mathbf{x}^{\mathrm{T}} \cdot \mathbf{y} = x_1 y_1 + \ldots + x_n y_n$.
- No operator sign is used when a matrix is being multiplied by a vector or another matrix: A$\mathbf{x}$, AB.
- Sometimes several matrices are augmented to each other. If an $m \times n$ matrix A is augmented from the right by a $m \times p$ matrix B, this is denoted as $\begin{pmatrix} \mathrm{A} & \mathrm{B} \end{pmatrix}$. If a $k \times (n + p)$ matrix C is in turn augmented to this construction from below, the obtained matrix is denoted as $\begin{pmatrix} \mathrm{A} & \mathrm{B} \\ & \mathrm{C} \end{pmatrix}$.
- An identity matrix $\begin{pmatrix} 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 \end{pmatrix}$ is denoted by I.

2 Linear Programming and its Applications

This section gives a brief overview of linear programming, its properties, and different forms.

Let us first look at an example from [5]. Suppose that a small brewery produces ale and beer. The ingredients for these products are corn, hops, and malt, and different products use them in different proportions. These ingredients are not infinite, and their usage is bounded. Given how much resources each product consumes, bounds on the resources, and how much profit a barrel of each product gives, the question is how the brewery can maximize its profit.

Let a single barrel of beer bring a larger profit than a barrel of ale. Does it mean that the best plan is to produce only beer? This idea may be wrong since beer may consume more resources per barrel than ale. Moreover, it may happen that brewing only beer uses up all the corn much faster than the other resources. The optimal solution may be not so obvious, and that's why we need linear programming.

The term "programming" is used in the sense of "planning", not computer programming, and "linear" is related to the way how it is defined mathematically. In this work, we refer to a linear programming task as a "linear program", although it cannot be treated as an ordinary computer program.

Linear programming technique is widely used in the field of economics. More examples and motivation can be found in [5].

2.1 Formal Definition of Linear Programming

Suppose that we have the following settings (an example from [5]). One barrel of ale requires 5 units of corn, 4 units of hops, and 35 units of malt, whereas one barrel of beer requires 15 units of corn, 4 units of hops, and 20 units of malt. There are 480 units of corn, 160 units of hops, and 1190 units of malt available in total. The profit obtained from a barrel of ale is 13 currency units, and the profit obtained from a barrel of beer is 23 currency units.

Let x_1 denote the number of barrels of beer produced, and x_2 the number of barrels of ale produced. We may formulate the profit maximization task in the following way:

$$\begin{aligned} \textbf{maximize} \quad & 13x_1 + 23x_2 & & \\ \textbf{subject to} \quad & 5x_1 + 15x_2 & \leq & \ 480 \\ & 4x_1 + 4x_2 & \leq & \ 160 \\ & 35x_1 + 20x_2 & \leq & \ 1190 \\ & x_1 & \geq & \ 0 \\ & x_2 & \geq & \ 0 \end{aligned}$$

The constraints $x_1 \geq 0$ and $x_2 \geq 0$ ensure that the number of produced barrels is not negative.

If we write down the inequalities in a matrix form, the profit maximization problem looks like

$$\text{maximize } \begin{pmatrix} 13 \\ 23 \end{pmatrix}^{\mathrm{T}} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \text{ subject to } \begin{pmatrix} 5 & 15 \\ 4 & 4 \\ 35 & 20 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \leq \begin{pmatrix} 480 \\ 160 \\ 1190 \end{pmatrix}, \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \geq \begin{pmatrix} 0 \\ 0 \end{pmatrix} .$$

Represented in such a way, the profit maximization task is an instance of linear programming. Formally, linear programming is a technique used in optimization of a linear function, subject to linear equality and linear inequality constraints.

Definition 1. *The canonical form of a linear program is*

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

where the system $\mathrm{A}\mathbf{x} \leq \mathbf{b}$ *represents the constraints,* $\mathbf{c}^{\mathrm{T}}$ *the cost function,* $\mathbf{x}$ *the vector of variables.*

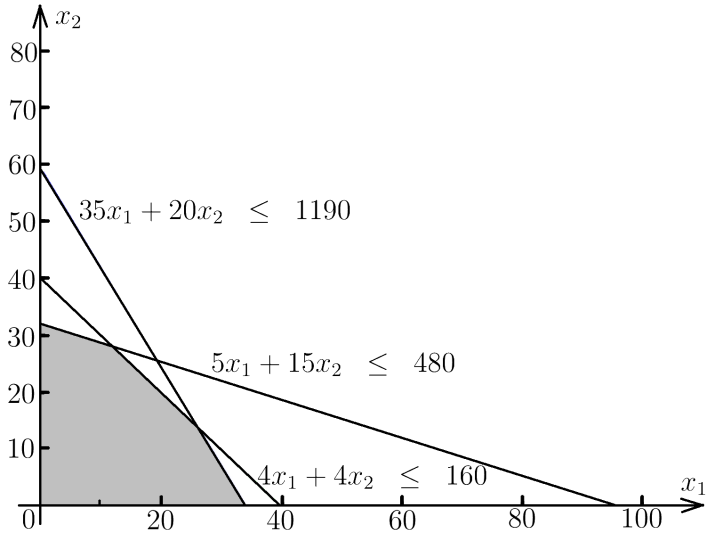
The aim of linear programming is to find a solution to the linear inequality system $\mathrm{A}\mathbf{x} \leq \mathbf{b}$ such that $\mathbf{c}^{\mathrm{T}} \cdot \mathbf{x}$ is maximal.

There exist efficient polynomial-time algorithms that solve linear programming tasks on the assumption that the values of the variables are real numbers. In the given example, it means that the products and the resources are divisible, i.e it must be allowed to produce a half of a barrel. Adding the constraints that some variables must be integers turns a linear programming task to a so-called *integer programming* task, which has been proven to be NP-complete. In this work, we concentrate just on real number linear programming.

One thing that deserves mentioning is that the constraint $\mathbf{x} \geq \mathbf{0}$ is needed just for correctness, and it does not affect the efficiency of the solver. Any variable may be set to *free*, what means that it is allowed to take negative values. This should be taken into account when we talk about security, since we cannot restrict the adversary from allowing the variables to take negative values.

2.2 Geometrical Representation of a Linear Program

In the brewery example, we have just two variables, and therefore it is easy to represent the set of possible solutions of the corresponding linear program geometrically in a two-dimensional space. It turns out to be a polygon.



We will see that the same intuition generalizes to multidimensional space. Let a linear program be given in its canonical form (1).

Definition 2. *A feasible solution of a linear program is any vector* $\mathbf{v}$ *that satisfies its constraints:* $\mathrm{A}\mathbf{v} \leq \mathbf{b}$, $\mathbf{v} \geq 0$.

Definition 3. *An optimal solution of a linear program is any feasible solution* $\mathbf{v}$ *that minimizes the value of its cost function* $\mathbf{c}^{\mathrm{T}} \cdot \mathbf{v}$.

Definition 4. *The feasible region of a linear program is the set of all its feasible solutions.*

Let us look at the feasible region of a linear program in its geometric sense. Its properties may become useful when we talk about privacy since a linear programming task is in fact defined by its feasible region and the optimization direction.

Although the following definitions can be generalized, in this work we are dealing with Euclidean spaces only. More information about the properties of polyhedrons can be found in [12].

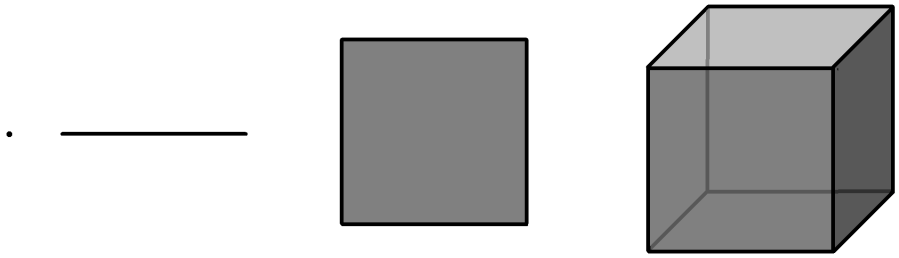
Definition 5. *A set* $S \subseteq \mathbb{R}^n$ *is called convex if for any* $x \in S$ *and* $y \in S$, *the straight line segment between* x *and* y *also belongs to* S *(formally,* $\lambda x + (1 - \lambda)y \in S$ *for any* $\lambda \in [0, 1]$*).*

Definition 6. *A hyperplane is a set of points* $S \subseteq \mathbb{R}^n$ *defined by a linear equation* $a_1 x_1 + \ldots + a_n x_n = b$, *where* $b \in \mathbb{R}$, $a_i \in \mathbb{R}$ *for any* $i \in \{1, \ldots, n\}$, *and* $a_j \neq 0$ *for some* j.

Definition 7. *A half-space is a set of points* $S \subseteq \mathbb{R}^n$ *defined by a linear inequality* $a_1 x_1 + \ldots + a_n x_n \leq b$, *where* $b \in \mathbb{R}$, $a_i \in \mathbb{R}$ *for any* $i \in \{1, \ldots, n\}$, *and* $a_j \neq 0$ *for some* j.

Definition 8. *A set* $K \subseteq \mathbb{R}^n$ *is called a polyhedron if it is an intersection of a finite number of half-spaces of* $\mathbb{R}^n$.

Example: a point, a closed line segment, a closed square, and a closed cube are bounded polyhedrons in the spaces of dimensions $0, 1, 2, 3$.



Proposition 1. *The intersection of convex sets is also a convex set.*

Proof: Let $\{S_i\}_{i \in I}$ for some set of indices I (possibly infinite) be a collection of convex sets. Let $S = \bigcap_{i \in I} S_i$. Let $x, y \in S$. Then $x, y \in S_i$ for any $i \in I$. Let $z = \lambda x + (1 - \lambda y)$ for some $\lambda \in [0, 1]$. By convexity of S_i, $z \in S_i$ for any S_i. Thus $z \in \bigcap_{i \in I} S_i$, and S is indeed a convex set. $\square$

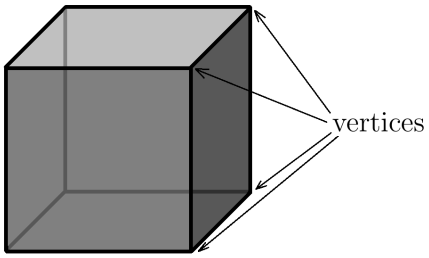
Proposition 2. *A polyhedron is a convex set.*

Proof: A polyhedron is by definition an intersection of a finite number of half-spaces. Any half-space is a convex set. The intersection of a finite number of convex sets is also a convex set. □

Why do we need all these definitions? The solution set of the system $\mathrm{A}\mathbf{x} \leq \mathbf{b}$ is a polyhedron by definition, since each inequality in fact defines a half-space of $\mathbb{R}^n$. Some known properties of polyhedrons may help to analyze the feasible region of a linear program.

Definition 9. *A point v of a polyhedron P is called a vertex if it cannot be expressed as a convex combination of points in $P \setminus \{v\}$.*

Example: in the space of dimension 3, a vertex is just a corner point.



Definition 10. *A basic solution to the linear program is the solution that is located in a vertex of the corresponding polyhedron.*

It is possible to find in polynomial time not just an arbitrary optimal solution, but more precisely a basic solution (if it exists). The reason is that the linear programming solvers are based on finding a local maximum of the feasible region (that is actually the global maximum because of convexity), and the local maximum is necessarily located in a vertex.

2.3 Forms of Linear Programming

We define some representation forms of linear programming that will be used in further sections.

It is possible to define any polyhedron in such a way that the only inequalities are of the form $\mathbf{x} \geq \mathbf{0}$. Any inequality may be transformed to an equality by introducing a *slack variable* and thus increasing the dimension. An inequality of the form $a_1x_1+\ldots+a_nx_n \leq b$ may be rewritten as $a_1x_1 + \ldots + a_nx_n + x_s = b$, where x_s is called a slack variable. This equation is equivalent to the initial inequality on the condition $x_s \geq 0$. Each inequality in general requires its own slack variable to ensure that no solutions will be lost.

Definition 11. *The standard form of a linear programming task is*

$$\textit{maximize } \begin{pmatrix}\mathbf{c}\\ \mathbf{0}\end{pmatrix}^{\mathrm{T}} \cdot \begin{pmatrix}\mathbf{x}\\ \mathbf{x_s}\end{pmatrix}, \textit{ subject to } \begin{pmatrix}\mathrm{A} & \mathrm{I}\end{pmatrix}\begin{pmatrix}\mathbf{x}\\ \mathbf{x_s}\end{pmatrix} = \mathbf{b}, \begin{pmatrix}\mathbf{x}\\ \mathbf{x_s}\end{pmatrix} \geq \mathbf{0} \ , \tag{2}$$

where inequalities have been transformed to equalities by introducing slack variables $\mathbf{x_s}$.

On the other hand, any linear equation system $\mathrm{A}\mathbf{x} = \mathbf{b}$ may be transformed into a system of inequalities by replacing each equality $\mathbf{a}_i^{\mathrm{T}}\mathbf{x} = b_i$ with two inequalities $\mathbf{a}_i^{\mathrm{T}}\mathbf{x} \leq b_i$ and $-\mathbf{a}_i^{\mathrm{T}}\mathbf{x} \leq -b_i$. This means that there is no difference whether the constraints of the linear program are represented by equalities or inequalities. These two representations are equivalent.

Another thing that we may do is to include the cost function into the constraint matrix. This is what is actually being done in the preprocessing phase of some linear programming solvers.

Definition 12. *The augmented form of a linear programming task is*

$$\textit{maximize } w, \textit{ subject to } \begin{pmatrix} 1 & -\mathbf{c}^{\mathrm{T}} & 0 \\ 0 & \mathrm{A} & \mathrm{I} \end{pmatrix} \begin{pmatrix} w \\ \mathbf{x} \\ \mathbf{x_s} \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \mathbf{b} \end{pmatrix}, \begin{pmatrix} w \\ \mathbf{x} \\ \mathbf{x_s} \end{pmatrix} \geq \mathbf{0} \ . \tag{3}$$

where w is the variable to be maximized that represents the cost of the solution.

From the geometrical interpretation, it can be seen that we may maximize in any direction. An analogous minimization problem can be easily transformed into a maximization problem just by reversing the direction of the objective function (flipping the signs of the elements of $\mathbf{c}$). Hence we may further consider both maximization and minimization problems.

2.4 Example

The brewery example was quite specific, and it was related to a particular profit maximization task. Linear programming can be actually used in solving more general standard problems. For example, some graph-related tasks such as finding the maximum flow, the minimal spanning-tree, and shortest path, can be formalized as linear programs.

Let us look at the shortest path example. Given a directed graph (V, E) (V is the set of nodes, and E is the set of arcs) with source node s, target node t, and cost w_{ij} for each arc $(i, j) \in E$, the task is to find the path from s to t that has the minimal cost. This task may be expressed by the following linear program with variables x_{ij}:

$$\text{minimize} \sum_{(i,j)\in E} w_{ij}x_{ij}$$

$$\text{subject to } \mathbf{x} \geq \mathbf{0} \text{ and } \forall i \sum_{j\in V\setminus\{i\}} x_{ij} - \sum_{j\in V\setminus\{i\}} x_{ji} = \begin{cases} 1 \text{ if } i = s \\ -1 \text{ if } i = t \\ 0 \text{ otherwise} \end{cases} .$$

Why is it defined like that? Assume that $x_{ij} = k$ iff the arc (i, j) is taken k times (if it is not taken, then $x_{ij} = 0$). In this way, assigning integer values to the variables x_{ij} represents a multiset of arcs. The constraints ensure that this multiset indeed forms a path. Note that $\sum_{j\in V\setminus\{i\}} x_{ji}$ is the number of arcs entering the node i, and $\sum_{j\in V\setminus\{i\}} x_{ij}$ is the number of arcs leaving the node i. The number of entering arcs should be equal to the number of exiting arcs iff i is neither the source nor the target node. If it is the source node, then it should contain one extra exiting arc, and if it is the target node, it

should contain one extra entering arc. The value to be minimized $\sum_{(i,j)\in E} w_{ij}x_{ij}$ is the sum of the weights of the arcs that are included into the path.

According to such definition, the path is allowed to pass through the same node several times, and the same arc may be taken multiple times, but that is not a problem since the shortest path will be a simple path (assuming that the length of the path cannot be reduced to $-\infty$ by assigning negative weights to the arcs).

Although it seems that, in order to get a correct solution, we should restrict the answers to integers, this is actually not necessary. In the case of the shortest path problem, finding an arbitrary optimal solution is no harder than finding an integral solution. The reason is that all its basic solutions are integral, including the optimal one. The proof is not so trivial, and it can be found in [17].

Representing the shortest path problem as a linear program is not intended to be the most efficient algorithm for solving this task. However, it is a good example that shows how linear programming can be used in solving more general problems. This can be interesting from cryptographic point of view since having a secure privacy-preserving linear programming solver would allow secure computation of some graph-related problems.

3 Privacy of Linear Programming

In this section, we give some examples of tasks that require privacy-preserving linear programming. We give an overview of the existing works, and then state a stronger security definition that seems to be reasonable for real world problems.

Suppose that there are several companies that produce malt beverages. It may happen that each company is responsible for its own products, and the recipes are held in secret. Each company has its storage of corn, hops, and malt, whose amount they also do not want to reveal. However, all the companies belong to a larger organization, and they are interested that they all get their maximum profit. In this case, they would like to solve the linear program that corresponds to the data of all the companies, but without revealing any private data.

Another potential use is related to the shortest path problem. Suppose that some organization has hidden dangerous, possibly explosive devices underground in a particular area. It is however still possible to move through this area knowing the locations of the devices. The landscape may be represented by a grid that forms a graph where the dangerous spots have no edges. Now suppose that the organization has some friends who would like to cross this area from point s to point t. The problem is that the organization does not trust their friends with the complete plans, and it does not want to reveal anything except the shortest path, since otherwise these friends may be moving around too much, or even try to dig out some of the devices. On the other hand, the friends also do not want to reveal their movement path. This is the case of a shared linear program where one party holds the constraints (where the points s and t denote the source and the target vertices), and the other party the cost vector (that represents the graph). Some parts of the constraints are public since they correspond to a standard graph representation.

There have been attempts to implement an effective privacy-preserving version of linear programming. In general, there are two main approaches to this problem.

1. One approach is the straightforward cryptographic implementation of a privacy-preserving version of some LP solving algorithm [19, 23]. The main problem of this approach is efficiency since the entire optimization process must be performed online. The users are also restricted to particular implemented algorithms, and each new modification of an LP solving algorithm should be each time translated to its privacy-preserving version.

2. Another approach is to transform the program in such a way that it could be given to a solver for offline computation without fearing that the solver gets some useful information out of it. This has turned out to be challenging since the optimal solution to the initial program has to be recoverable from the optimal solution to the transformed program. This approach can be further divided into algebraic transformation-based methods and potential decomposition based approaches, which are surveyed and generalized in [26] (that relates not just to linear programming, but to optimization tasks in general).

In this work, we focus on the algebraic transformation-based approach. It seems more tempting since it would potentially allow much more efficient solutions. We try to formalize the security requirements and see if any of them can be actually achieved.

3.1 Security Definition

In the transformation-based approach, the input data of the users is being disguised using some cryptographic operations (that should be less complex than solving the linear program itself) and afterwards sent to a server (or a computing cloud) for solving. The security requirement is that the server should not extract any information about the initial input data. The brewery owners do not want to reveal their recipes not only to each other, but also to the server, since he may be not trustworthy. But what does it actually mean that the server should not learn anything? For example, the number of the variables in the input program will barely be hidden since the only way to do it would be to introduce exponential number of variables, and in practice it would be too inefficient. The security requirements have to be formalized more precisely.

The security definition that has been used in the previous works related to the transformation-based approach is the *acceptable security*. This notion was first used in [10].

Definition 13. *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.*

It turns out that such a security definition is too weak. For example, as mentioned in [3], there are infinitely many ways of multiplying a linear equation (or an inequality) by a constant, but actually each of them represents the same constraint. For the brewery owners, this would mean that they just give out almost in plaintext the bounds of their resources. When we talk about practical problems, using such a definition may be just illegal since it violates the requirement that no single bit of information should be revealed about private data.

Some works have tried more formal approaches to security analysis. For example, [3] proposes some ideas based on computing the probability that the adversary derives some secret value. However, such analysis is difficult since for example, in order to discover some values from a permutation, the adversary may use more specific algorithms than just guessing and checking.

The transformations of [8] are provided with some proofs of the leakage bounds using channel-based definitions. However, such a security analysis seems not quite correct for the given problem since the analysis works well if the channel selects a new randomness each time a value passes through it. In practice, the adversary may use the channel with the same randomness several times with different inputs by obtaining several distinct, probably non-optimal solutions from the same linear programming task.

There is one problem that narrows the possible usage of privacy-preserving linear programming. Namely, the security of the existing constructions is too dependent on the initial sharing of the problem. There exist constructions that work relatively well for horizontally-partitioned linear programs (each constraint entirely belongs to a single

user), but not vertically-partitioned (where the same constraint is distributed amongst several users). It would be nice to have a more general security definition that would work well for any partitioning. In this work, we are going to state a pretty standard, indistinguishability-based confidentiality definition, since it seems a quite reasonable security requirement. The idea of indistinguishability-based security is that the adversary is unable to distinguish two different linear programs after their transformation.

In the context of linear programming, it is quite possible that the adversary has some significant partial information about the initial linear program. We do not know exactly how the linear program is partitioned, and the server who solves the problem may collaborate with some users. Our security definition states that even if the adversary has some information about the initial linear program, he is unable to get any information provided by the other users. Such information is completely unavoidable for example in the shortest path problem, where the structure of the constraint matrix is almost entirely public. Additionally, the indistinguishability-based definition is quite standard in cryptography, and the techniques of integrating the transformation into a more complex protocol are well-known.

In our security definition, we have the following settings. The adversary is allowed to use any polynomial-time algorithm. First, the adversary chooses the dimensions m,n and two instances of linear programming tasks $\mathrm{A}_0, \mathbf{b}_0, \mathbf{c}_0$ and $\mathrm{A}_1, \mathbf{b}_1, \mathbf{c}_1$ that share a common optimal solution $\mathbf{x}_{\text{opt}}$. The reason why we let the adversary to choose m and n is that it gives a much stronger security definition since some choices of m and n might be weaker than the others, and we want to ensure that the linear program is secure also for its worst-case parameters. After choosing everything, the adversary sends $m, n, \mathrm{A}_0, \mathrm{A}_1, \mathbf{b}_0, \mathbf{b}_1, \mathbf{c}_0, \mathbf{c}_1, \mathbf{x}_{\text{opt}}$ to the environment. The environment checks that $\mathbf{x}_{\text{opt}}$ is an optimal solution to both linear programming tasks. Then it chooses a bit b, performs a transformation $\mathcal{T}$ on the task $\mathrm{A}_b, \mathbf{b}_b, \mathbf{c}_b$ and sends the transformed program back to the adversary. The adversary attempts to guess b. The transformation $\mathcal{T}$ is secure if whatever polynomial-time algorithm $\mathcal{A}$ the adversary uses, he is unable to guess the value of b much better than simply guessing it randomly.

Definition 14. *The transformation algorithm $\mathcal{T}$ is secure if, for any polynomial-time algorithm $\mathcal{A}$, the probability that the following program code returns* true *(on the condition that it does not return $\perp$) is $\frac{1}{2} + \alpha(t)$:*

$\mathcal{G}(t)$

$m, n, \mathrm{A}_0, \mathbf{b}_0, \mathbf{c}_0, \mathrm{A}_1, \mathbf{b}_1, \mathbf{c}_1, \mathbf{x}_{\text{opt}} \leftarrow \mathcal{A}(t)$

if $\mathcal{S}(m, n, \mathrm{A}_0, \mathbf{b}_0, \mathbf{c}_0, \mathbf{x}_{\text{opt}}) = \text{false}$ or $\mathcal{S}(m, n, \mathrm{A}_1, \mathbf{b}_1, \mathbf{c}_1, \mathbf{x}_{\text{opt}}) = \text{false}$ then ***return*** $\perp$

$b \xleftarrow{\$} \{0, 1\}$

$T \leftarrow \mathcal{T}(\mathrm{A}_b, \mathbf{b}_b, \mathbf{c}_b, t)$

$b' \leftarrow \mathcal{A}(T)$

return $b = b'$

where:

- *the parameters m and n define the size of the initial linear program;*

- $\mathcal{S}(m, n, \mathrm{A}, \mathbf{b}, \mathbf{c}, \mathbf{x}_{\mathrm{opt}})$ *returns* true *iff indeed* A *is a* $m \times n$ *matrix,* $\mathbf{b}$ *is a vector of length* m*,* $\mathbf{c}$ *is a vector of length* n*, and* $\mathbf{x}_{\mathrm{opt}}$ *is a valid optimal solution to a linear program stated as "maximize* $\mathbf{c}^{\mathrm{T}} \cdot \mathbf{x}$*, subject to* $\mathrm{A}\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq \mathbf{0}$*";*

- t *is the security parameter (the parameter that sets bounds on the time that the adversary may spend on solving the problem, and on the probability that he guesses the secret correctly).*

- α *is a negligible function (a function that approaches* 0 *faster than any inverse polynomial; a function* $f : \mathbb{N} \to \mathbb{R}^+$ *such that* $\forall c \in \mathbb{N} : \lim_{t\to\infty} t^c \cdot f(t) = 0$*).*

Why is the probability that the adversary guesses b correctly is defined as $\frac{1}{2} + \alpha(t)$ for negligible α? It may seem that the less the probability is, the more secure is the transformation. However, the adversary may always guess b with probability $\frac{1}{2}$ just by guessing randomly. On the other hand, if he is so unlucky that he guesses the right answer with probability $\varepsilon < \frac{1}{2}$, he may just flip his "yes" and "no" answers, changing his winning probability to $1 - \varepsilon > \frac{1}{2}$. Hence $\frac{1}{2}$ is indeed the worst case for the adversary, and if we want a secure construction, we want the probability of a correct guess to be as close to $\frac{1}{2}$ as possible.

An indistinguishability-based approach would allow to make the security independent from the initial sharing, since whatever evil information is inserted into the transformed program, the adversary cannot identify his own transformed program afterwards. Unfortunately, even such a definition is still not enough in the case if revealing the optimal solution is dangerous. Namely, as mentioned in [3], no matter how secure the scheme is, even in the cryptography-based approach, if one of the parties knows the constraints and the other one the objective function, then knowing the optimal solution and the constraints may already reveal some information about the possible directions of the objective function. This fact may be harmful for example for the shortest path problem, where the graph is actually represented by the objective function. This case is not covered by our security definition since the adversary is obliged to choose two programs that have the same optimal solution.

3.2 Existing Approaches and their Problems

Several transformations of linear programs have been proposed beforehand [4, 8, 9, 15, 20, 21, 24, 25]. Unfortunately, these transformations either lack statements of security at all (or the statements are informal), or are difficult to combine with usual cryptographic security definitions in the construction of larger applications.

In this section, we give an overview of some existing approaches that are not sufficient for the indistinguishability-based security. We describe in more details the proposal by the authors of [8] since it generalizes to some extent the other works. We find out the particular problems of [8] and suggest some attacks against them.

3.2.1 Basic Operations from Related Works

Let us first give an overview of the basic hiding operations proposed in the previous works. More or less all of them are used in [8]. Let a linear programming task be given in its standard form (2).

Multiplying from the left. The idea of multiplying A and $\mathbf{b}$ by a random invertible matrix P from the left was first introduced in [9]. Since P is invertible, all the solutions to the system, including the optimal solution, remain the same. This means that the transformation is not harmful for the correctness. However, it conceals only the outer appearance of A and $\mathbf{b}$, but the inner essence of the linear program remains the same (the feasible region does not change). This is not enough for our security definition, since only the matrices that have exactly the same feasible region would be indistinguishable.

Multiplying from the right. The idea of multiplying A and $\mathbf{b}$ by a random invertible matrix Q from the right was also proposed in [9]. This operation hides also the cost vector $\mathbf{c}$. This would make the transformation perfectly secure, but unfortunately it changes the optimal solution if some external constraints of the form $\mathrm{B}\mathbf{x} \geq \mathbf{b}'$ are present. 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. Since in practice linear programs do require such constraints (in general of the form $\mathbf{x} \geq \mathbf{0}$), this solution is not enough.

Scaling and Permutation. Using only Q without P was proposed in [24]. It improved correctness a bit, but the problem was still in Q, not in P. This problem has been discovered afterwards in [4]. The main result is that in order to preserve the inequality $\mathbf{x} \geq \mathbf{0}$, the most general type of matrix by which we may multiply from the right is a positive monomial matrix (see Definition 15). Multiplying by a positive monomial matrix results in scaling and permuting the variables. This is again not enough since it just scales the corresponding polyhedron and permutes its coordinates. We show in the section 4.4 that there exist attacks based on finding projections of polyhedrons.

Definition 15. *A matrix is called monomial if it contains exactly one non-zero entry per row and column (its other name is generalized permutation matrix).*

Some interesting hiding provided with security analysis has been proposed in [8]. The initial variable vector $\mathbf{x}$ is not only scaled, but also shifted. As we see later, the shifting actually reduces to scaling. Let us look at this work in more details.

3.2.2 Example of a Transformation

The authors of [8] propose to hide the variables by shifting. The initial linear program is given in its canonical form (1), the equalities intentionally separated from the inequalities.

$$\begin{aligned} \min \mathbf{c}^{\mathrm{T}}\mathbf{x} & \\ \mathrm{A}_1\mathbf{x} &= \mathbf{b}_1 \\ \mathrm{A}_2\mathbf{x} &\leq \mathbf{b}_2 \\ \mathbf{x} &\geq 0 \end{aligned}$$

By assumption, the initial constraints are split into two parts: $\mathrm{A}_1\mathbf{x} = \mathbf{b}_1$ and $\mathrm{A}_2\mathbf{x} \leq \mathbf{b}_2$. When a linear program solver gets a linear program as an input, it translates inequalities into equalities by introducing slack variables. The idea of this work is that only the $\mathbf{x}$ variables have to be hidden, and the slack variables may remain public.

A positive monomial matrix Q is used to hide $\mathbf{c}$. We have already seen from the previous works that it has been proven to be the most general type of matrix that is allowed for multiplication from the right.

$$\begin{aligned} \min \mathbf{c}^{\mathrm{T}}\mathrm{Q}(\mathrm{Q}^{-1}\mathbf{x}) & \\ \mathrm{A}_1\mathrm{Q}(\mathrm{Q}^{-1}\mathbf{x}) &= \mathbf{b}_1 \\ \mathrm{A}_2\mathrm{Q}(\mathrm{Q}^{-1}\mathbf{x}) &\leq \mathbf{b}_2 \\ (\mathrm{Q}^{-1}\mathbf{x}) &\geq 0 \end{aligned}$$

Besides scaling, the variables $\mathbf{x}$ are also shifted. A positive vector $\mathbf{r}$ is used to hide $\mathbf{x}$.

$$\begin{aligned} \min \mathbf{c}^{\mathrm{T}}\mathrm{Q}(\mathrm{Q}^{-1}\mathbf{x}+\mathbf{r}) & \\ \mathrm{A}_1\mathrm{Q}(\mathrm{Q}^{-1}\mathbf{x}+\mathbf{r}) &= \mathbf{b}_1+\mathrm{A}_1\mathrm{Q}\mathbf{r} \\ \mathrm{A}_2\mathrm{Q}(\mathrm{Q}^{-1}\mathbf{x}+\mathbf{r}) &\leq \mathbf{b}_2+\mathrm{A}_2\mathrm{Q}\mathbf{r} \\ (\mathrm{Q}^{-1}\mathbf{x}+\mathbf{r}) &\geq \mathbf{r} \end{aligned}$$

Since the vector $\mathbf{r}$ is supposed to be secret, it cannot be exposed in plaintext. The last inequality is therefore multiplied by a positive diagonal matrix S and is rewritten as $\mathrm{S}(\mathrm{Q}^{-1}\mathbf{x}+\mathbf{r}) \geq S\mathbf{r}$.

$$\begin{aligned} \min \mathbf{c}^{\mathrm{T}}\mathrm{Q}(\mathrm{Q}^{-1}\mathbf{x}+\mathbf{r}) & \\ \mathrm{A}_1\mathrm{Q}(\mathrm{Q}^{-1}\mathbf{x}+\mathbf{r}) &= \mathbf{b}_1+\mathrm{A}_1\mathrm{Q}\mathbf{r} \\ \mathrm{A}_2\mathrm{Q}(\mathrm{Q}^{-1}\mathbf{x}+\mathbf{r}) &\leq \mathbf{b}_2+\mathrm{A}_2\mathrm{Q}\mathbf{r} \\ \mathrm{S}(\mathrm{Q}^{-1}\mathbf{x}+\mathbf{r}) &\geq \mathrm{S}\mathbf{r} \end{aligned}$$

The inequalities are transformed to equalities by introducing slack variables $\mathbf{s}$. Let $\mathbf{y} = (\mathrm{Q}^{-1}\mathbf{x}+\mathbf{r})$. An inequality of the form $\mathbf{a}_i^{\mathrm{T}}\mathbf{y} \leq b_i$ can be transformed into an equality by rewriting it as $\mathbf{a}_i^{\mathrm{T}}\mathbf{y} + s_i = b_i$. The conditions $s_i \geq 0$ ensure that the value of $\mathbf{a}_i^{\mathrm{T}}\mathbf{y}$ is not greater than b_i. For additional hiding, the equations that correspond to $y_i \geq r_i$ are rewritten into the form $-S_i y_i + s_i = -S_i r_i$, where S_i is the non-zero entry of the diagonal matrix S that is associated with the variable y_i. The slack variables s_i are randomly permuted amongst the slack variables of the $\mathrm{A}_2\mathbf{x} \leq \mathbf{b}_2$ inequality by introducing a permutation matrix T instead of just an identity matrix.

$$\mathrm{A}' = \begin{pmatrix} \mathrm{A}_1\mathrm{Q} & 0 \\ \mathrm{A}_2\mathrm{Q} & \\ -\mathrm{S} & \mathrm{T} \end{pmatrix}$$

$$\mathbf{b}' = \begin{pmatrix} \mathbf{b_1}+\mathrm{A}_1\mathrm{Q}\mathbf{r} \\ \mathbf{b_2}+\mathrm{A}_2\mathrm{Q}\mathbf{r} \\ -\mathrm{S}\mathbf{r} \end{pmatrix}$$

The quantities A' and $\mathbf{b}'$ are multiplied from the left by a random invertible matrix P to conceal the appearance of A' and $\mathbf{b}'$. The published linear program is

$$\begin{aligned} \min \mathbf{c'}^{\mathrm{T}}\mathbf{z} & \\ \mathrm{PA}'\mathbf{z} &= \mathrm{P}\mathbf{b}' \\ \mathbf{z} &\geq 0 \end{aligned}$$

where $\mathbf{c'}^{\mathrm{T}} = \mathbf{c}^{\mathrm{T}}\mathrm{Q}$ extended with zeroes for the slack variable costs, and $\mathbf{z} = \begin{pmatrix}\mathbf{y}\\ \mathbf{s}\end{pmatrix}$.

In general, the only difference between the equality and the inequality constraints is that the values of the slack variables in the inequality constraints are not supposed to be hidden. However, it is essential to treat them separately since the security analysis of the transformation is based on the assumption that the adversary has no information about A_1, A_2, $\mathbf{b_1}$, $\mathbf{b_2}$. If the columns of slack variables would be included into the equialities $\mathrm{A}_1\mathbf{x} = \mathbf{b_1}$, then they would most probably be represented by at most a positive monomial matrix, and this would be some kind of information about A_1.

3.2.3 Some New Attacks

We describe some new attacks that may be used to disable some previously proposed protection methods.

Reducing Shifting back to Scaling The transformation maps each variable x_i to a new variable $y_i = q_i \cdot x_i + r_i$ where q_i and r_i are random positive real numbers. The problem is that the scaled values of x_i leak through the slack variables. The slack variables s_i are used in equations of the form $-S_i \cdot (q_i \cdot x_i + r_i) + s_i = -S_i \cdot r_i$ where S_i is the non-zero entry of the matrix S that corresponds to the variable x_i. From these equations, we get $s_i = S_i \cdot q_i \cdot x_i$, what means that s_i actually obtain the scaled values of x_i, and hence shifting does not give more security than just scaling (multiplying by a positive monomial matrix from the right). The same problem is present in [25], another work that is based on shifting the variables.

For the given construction, this shifting is better than nothing since otherwise both $\mathbf{c}$ and $\mathbf{x}$ would be scaled by the same monomial matrix Q (actually, while $\mathbf{c}$ is scaled by the entries of Q, the vector $\mathbf{x}$ is scaled by their inverses). However, in any case $\mathbf{s}$ still contains the same factors, although its entries are additionally multiplied by entries of S. As proposed in [8], selecting the matrix T as a positive monomial matrix may also contribute to scaling of $\mathbf{s}$, but then $s_i = T_i^{-1} \cdot S_i \cdot q_i \cdot x_i$, and it is still just scaling of $\mathbf{x}$.

Removing the Permutation Another type of hiding used in [8] is permutation. After the variables $\mathbf{x}$ are shifted, the new slack variables that come from the shifting are permuted amongst the slack variables of the inequalities $\mathrm{A}_2\mathbf{x} \leq \mathbf{b}_2$. It seems that it gives additional security since the adversary does not know which variables correspond to the scaled $\mathbf{x}$. However, the permutation may still be easily traced down.

Each shifted initial variable from $\mathbf{y}$ is associated with exactly one slack variable from $\mathbf{s}$, and the adversary may therefore look through all the pairs of columns and recover the relations. He even does not need to take into account all the possible pairs of columns in PA' since he can distinguish the initial variables from the slack 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 $\alpha_1 x_{i1} + \ldots + \alpha_t x_{it} = \beta$ 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 columns of size t using Gaussian elimination.

1. Fix a set of t columns indexed $i_1, \ldots, i_t$ from the matrix A.

2. Take any other non-fixed column (that has not been deleted yet). Let it be with index j.

3. Take an equation where x_j has non-zero coefficient. Express the variable x_j in terms of the other variables. Substitute it into all the other equations. Remove the equation and the j-th column. If a column contains only zeroes, then remove just the column without removing any rows.

4. Continue removing until only the initially fixed t columns are left.

5. Note that all the previous operations do not change the solution set of the system. Therefore, if there are any rows left, then there exist $\alpha_i, \beta \in \mathbb{R}$ (not all $\alpha_i = 0$) such that $\alpha_1 x_{i1} + \ldots + \alpha_t x_{it} = \beta$.

Therefore if the constraints contain initial-slack variable pairs, the adversary may easily recover which initial variable is in the relation with which slack variable. He could potentially get such related pairs also purely from A, but in this case the initial variables should be indistinguishable from the slack variables, which is not the case of [8] transformation.

What if the slack variables $\mathbf{s}$ were permuted also amongst the $\mathbf{y}$ variables? It would not help since $\mathbf{y}$ and $\mathbf{s}$ are clearly distinguishable because of their distinct lower bounds (0 vs some element of $\mathbf{r}$). Finding the lower bounds of variables is easy, it is just a linear programming task where the value of a particular variable is being minimized.

The Problems with Inequality Constraints We introduce an attack that allows to remove the scaling and the permutation of variables. This is possible in the setting where all the constraints are represented by inequalities, one of the parties knows at least two inequality constraints, and the locations of the slack variables can be traced down. Some similar steps can be found in the attacks introduced in [3], but these attacks are actually different and are based on looking through an exponential number of combinations. Our attack is polynomial-time. Let us state this problem in general.

Suppose that the initial linear program is given in its canonical form (1). In the standard approach proposed by the previous works, the inequalities are transformed to equalities by bringing the linear program to its standard form (2).

$$\text{maximize } \begin{pmatrix}\mathbf{c}\\ \mathbf{0}\end{pmatrix}^{\mathrm{T}} \cdot \begin{pmatrix}\mathbf{x}\\ \mathbf{x_s}\end{pmatrix}, \text{ subject to } \begin{pmatrix}\mathrm{A} & \mathrm{I}\end{pmatrix} \begin{pmatrix}\mathbf{x}\\ \mathbf{x_s}\end{pmatrix} = \mathbf{b}, \begin{pmatrix}\mathbf{x}\\ \mathbf{x_s}\end{pmatrix} \geq \mathbf{0} \ .$$

The columns of the matrix are first being scaled and permuted multiplying it by a monomial matrix Q from the right. Then it is multiplied by a random invertible matrix P from the left.

If the vector $\mathbf{c}$ is treated separately from the constraints as it was done in the previous works, then the slack variables are clearly visible since they do not participate in the cost vector. This allows to use row operations to reduce the constraints back to the standard form $\begin{pmatrix}\mathrm{A}' & \mathrm{I}\end{pmatrix} \begin{pmatrix}\mathbf{x}\\ \mathbf{x_s}\end{pmatrix} = \mathbf{b}'$. Here A' and $\mathbf{b}'$ may be different from A and $\mathbf{b}$. However, since we know that the feasible region is just scaled, the inequalities $\mathbf{A}'\mathbf{x} \leq \mathbf{b}'$ define the

scaled polyhedron of the initial program. We may use the knowledge about the initial inequalities to cancel out the scaling as follows.

Suppose that one of the parties knows at least two inequalities from $\mathrm{A}\mathbf{x} \leq \mathbf{b}$. Take any two of them, let these inequalities be $a_{k1}x_1 + \ldots + a_{kn}x_n = b_k$ and $a_{\ell 1}x_1 + \ldots + a_{\ell n}x_n = b_\ell$ for some k, ℓ. We want to find the corresponding scaled inequalities from the transformed polyhedron $\mathrm{A}'\mathbf{x} \leq \mathbf{b}'$. Let us take any two inequalities $a'_{k'1}x_1 + \ldots + a'_{k'n}x_n \leq b'_{k'}$ and $a'_{\ell'1}x_1 + \ldots + a'_{\ell'n}x_n \leq b'_{\ell'}$. We may verify if they are indeed the same as follows.

With inequality replaced by an equality, each constraint corresponds to a hyperplane that bounds the polyhedron. The idea is to find the points where the hyperplanes cut a particular axis. For the variable x_i, the axis is being cut in the points $(0, \ldots, 0, \frac{b_k}{a_{ki}}, 0, \ldots, 0)$ and $(0, \ldots, 0, \frac{b_\ell}{a_{\ell i}}, 0, \ldots, 0)$ in the constraints coming from the initial program. Let i' denote the index that corresponds to the location of the variable x_i in the transformed program. In the constraints coming from the transformed program, the axis is being cut in the points $(0, \ldots, 0, \frac{b'_{k'}}{a'_{k'i'}}, 0, \ldots, 0)$ and $(0, \ldots, 0, \frac{b'_{\ell'}}{a'_{\ell'i'}}, 0, \ldots, 0)$. Since the transformed polyhedron is actually a scaled initial polyhedron, we have $\frac{b_k/a_{ki}}{b_\ell/a_{\ell i}} = \frac{b'_{k'}/a'_{k'i'}}{b'_{\ell'}/a'_{\ell'i'}}$. Although the variables are permuted, these ratios can be sorted. The bounds from the transformed polyhedron correspond to the bounds in the initial polyhedron only if the sorted ratios are exactly the same. Since some ratios may actually repeat, knowing more initial inequalities allows better testing. The scaling for x_i can be computed as $q_i = \frac{b_k a'_{k'i'}}{b'_{k'} a_{ki}}$, and it will be uniquely determined at least for those variables whose ratios in the obtained sequence are unique.

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 already reduce the number of secure linear programming tasks significantly. While it would not work well with the shortest path problem (there are equalities instead of inequalities), it can be very well applied to the situation similar to the brewery example where each party yields several inequalities that correspond to different products. It is very likely that the same product (the same variable) is also produced by some other party, and revealing the scaling of this particular variable could be interesting.

Therefore, hiding the cost vector $\mathbf{c}$ just by scaling is not sufficient since it reveals the locations of the slack variables. We propose to solve this issue by using the augmented form of linear programming (3) that includes the cost vector into A. There may nevertheless be other means of locating the slack variables. We consider them in the section 4.6.

3.2.4 Implementation-Related Problems

In the real world, the computation is not as nice as in ideal mathematics. The theory of privacy-preserving linear programming is based on the assumption that we are dealing with true real numbers. However, this situation is not achievable in practice. Real numbers are actually represented by fixed/floating point numbers, and their precision is finite.

The authors of [8] take into account the possibility of factoring attacks. This may

indeed be a problem since two random numbers are coprime with a relatively high probability. The described attacks are related to the common factors of column entries that can be read directly from the transformed matrix. For example, if an arbitrary integer matrix is being multiplied by an integer monomial matrix from the right, then any column in the resulting matrix has each entry containing the same factor. The same problem generalizes to rational numbers.

Since the variables are just scaled, the search for common factors may be possible also in the solution vectors, depending on the implementation details. Modifying the cost vector allows to obtain several distinct solutions that all contain the same scaling factor. Extracting these factors allows to eliminate the scaling from any solution. In theory, there would be no factoring problems if we were dealing with true real numbers. On the other hand, in practice the factoring attack also seems not to work well due to imprecision of solving algorithms, and thus the problem can be solved by giving the server a linear program that is a bit rounded. Hopefully, the real computer arithmetic will not be too harmful for the correctness.

4 Bounds of Transformation-Based Linear Programming

In this section, we are trying to define a construction that would be secure according to our indistinguishability-based definition. First of all, we show that achieving perfect secrecy is impossible. Then we focus on computational assumptions. We show that linear programming does not work correctly in finite fields, and thus the assumptions need to be defined in $\mathbb{R}$. In the end, we prove that, whatever affine transformation we use, there is an attack against which there is no protection with respect to the indistinguishability-based definition.

4.1 The Problems of Perfect Secrecy

Theoretically, it is possible to transform the linear program in such a way that it is perfectly secure. A suitable transformed program would be any random polyhedron that has the same optimal solution. The question is how to do this transformation efficiently if the optimal solution is initially unknown. We show that if there is a way of constructing a random linear program with the same optimal solution, then the optimal solution can be approximated efficiently (up to an arbitrary small error ε) with a workload that is not significantly higher than the workload of the transformation.

Definition 16. *The ℓ_2-norm of a vector* $\mathbf{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$ *is the quantity* $\|\mathbf{x}\| = \sqrt{x_1^2 + \ldots + x_n^2}$.

Definition 17. *The distance of a point x from a hyperplane $\mathcal{H}$ is the quantity*

$$\min_{y \in \mathcal{H}} \|x - y\| \ .$$

Assume that we have achieved perfect secrecy for our security definition. This means that we have a black-box algorithm $\mathcal{T}$ that, given an objective vector $\mathbf{c}$ and a set of bounds $\mathrm{A}\mathbf{x} \leq \mathbf{b}$ (let the $\mathbf{x} \geq \mathbf{0}$ bounds be for simplicity included in $\mathrm{A}\mathbf{x} \leq \mathbf{b}$) as an input, outputs another linear program that has exactly the same optimal solution, but otherwise:

- The objective function is independent from A,$\mathbf{b}$,$\mathbf{c}$ (however, it may depend on the values m,n).
- The bounding hyperplanes are independent from A,$\mathbf{b}$,$\mathbf{c}$ (probably dependent on the values m,n).

Now, given a linear program in its canonical form (1), the error bound ε, and the algorithm $\mathcal{T}$, the goal is to find a solution vector $\mathbf{x}$ such that $\|\mathbf{x} - \mathbf{x}_{opt}\| < \varepsilon$.

Let the bounding hyperplanes be located at the distance at most d from the optimal solution (the value of d may either depend or not depend on m and n, but it should be independent from A, $\mathbf{b}$, $\mathbf{c}$).

First, we will bound the error with $2d$. Then we will see if it is possible to reduce the error by scaling the initial polyhedron.

1. Treating the hyperplanes as equalities, find a point $\mathbf{x}$ that minimizes the sum of the distances from all the hyperplanes. Since each hyperplane by assumption contains a solution that is at most d apart from $\mathbf{x}_{opt}$, the better solution will be not worse than the optimal one. By triangle inequality, we get $\|\mathbf{x} - \mathbf{x}_{opt}\| \leq 2d$.

   The question is whether the point $\mathbf{x}$ can be found efficiently. This can be done for example using least squares method that is just one iteration and may therefore be computed efficiently using ordinary cryptographic operations. According to [14], this can be done by computing

   $$\beta = (\mathrm{A}^{\mathrm{T}}\mathrm{A})^{-1}\mathrm{A}^{\mathrm{T}}\mathbf{b} \ ,$$

   then the optimal answer is $\mathbf{x} = \mathrm{A}\beta$. There is just one matrix multiplication (the transposes do not matter), three multiplications of a matrix by a vector from the right, and one inverse. According to the method of computing inverse functions proposed in [2] and afterwards applied particularly to matrix inverses in [13], computing the inverse needs just one random matrix generation and two matrix multiplications.

2. The next problem is that we do not have any assumptions about d. It may be selected by $\mathcal{T}$ independently from the input, or it may depend on some parameters of the input program.

   (a) Let d be independent from the input. If the distance of the hyperplanes from the optimal solution is chosen randomly, there still exists some upper bound on that value, so $d < \infty$. We may multiply each column of A by $\frac{\varepsilon}{2d}$ so that the elements in $\mathbf{x}_{opt}$ will increase $\frac{2d}{\varepsilon}$ times. Let this new scaled solution be denoted $\mathbf{y}_{opt} := \mathbf{x}_{opt} \cdot \frac{2d}{\varepsilon}$, and its approximated solution $\mathbf{y}$. Then still $\|\mathbf{y} - \mathbf{y}_{opt}\| \leq 2d$, since the value of d does not depend on the input. Then:

   $$\begin{aligned} \|\mathbf{y} - \mathbf{y}_{opt}\| &\leq 2d \\ \|\mathbf{y} - \frac{2d}{\varepsilon} \cdot \mathbf{x}_{opt}\| &\leq 2d \\ \frac{2d}{\varepsilon} \cdot \|\frac{\varepsilon}{2d} \cdot \mathbf{y} - \mathbf{x}_{opt}\| &\leq 2d \\ \|\frac{\varepsilon}{2d} \cdot \mathbf{y} - \mathbf{x}_{opt}\| &\leq \varepsilon \end{aligned}$$

   The quantity $\frac{\varepsilon}{2d} \cdot \mathbf{y}$ is the required approximation for $\mathbf{x}_{opt}$.

   (b) What if d actually does depend on the input? Let us split this situation into two mutually exclusive events.

   - If scaling of the initial polyhedron affects the value of d, then it reveals information about the initial program, and the transformation is therefore not perfectly secure. The adversary may break the security assumption by selecting the initial programs whose polyhedrons have different sizes.
   - If scaling does not change the value of d, then we may again scale it in the same way we do it in the case where d does not depend on the input.

The conclusion is that if an efficient transformation that provides perfect secrecy was possible, its perfect secrecy would be unnecessary since there would be an easier way of

finding the optimal solution. In other words, having such a transformation algorithm implies having the optimal solution, and it is quite possible that knowing the optimal solution is essential for constructing a perfectly secret transformation. Therefore, this work will be based on the computational security.

4.2 Computational Security Assumptions

From the previous works, it seems that there are just two main operations that might be useful:

- Multiplying by a random invertible matrix from the left hides the outer appearance of A and $\mathbf{b}$, but does not modify the feasible region at all.
- Multiplying by a random positive monomial matrix from the right scales and permutes the variables, and thus stretches the polyhedron.

As we have seen before, this is not enough for indistinguishability-based security. Therefore we need something new. Could we use the permutation of columns more efficiently than in the previous works?

Although the previous solutions propose hiding of the cost vector $\mathbf{c}$ by scaling, we will use the augmented form of linear programming (3) that includes the cost vector into A. As we have shown in the section 3.2.3, if the cost vector $\mathbf{c}$ is not included into A, the linear program becomes insecure since the zero entries of the cost vector reveal the locations of the slack variables. This also makes the security analysis simpler since we no longer need to think on hiding the cost vector separately. However, we must assume that the location of the column that corresponds to the cost variable must be public. Otherwise the solver has to solve the same task multiple times, optimizing each variable separately. We assume that revealing one column does not affect the security too much since the adversary could guess it on his own.

Using the augmented form allows us to use some approach based on difficulty of distinguishing the variables. In this way, using permutation of columns for hiding becomes reasonable since we may mix the variables that do participate and that do not participate in the cost vector.

We are going to try the approach based on the computational difficulty of recovering t columns from the matrix whose columns are randomly permuted. This is tightly related to the *Strong Secret Hiding Assumption* (SSHA) [1] defined over finite fields. The aim of SSHA is to introduce two distributions that are indistinguishable from each other. While one of them contains actual secret data, the other one is completely random, and if the real data is indistinguishable from random data, then the scheme is secure. The definition may look a bit specific since it was designed for a particular secret sharing system based on polynomial interpolation.

The SSHA is defined as follows. There are two distributions, denoted $U(t, m, e, p)$ and $R(t, m, e, p)$, over the set of $(2t + 2e + 2) \times m$ matrices of values in $\mathbb{Z}_p$. The distribution $U(t, m, e, p)$ chooses all values uniformly from $\mathbb{Z}_p$. The distribution $R(t, m, e, p)$ results from the following process:

1. Choose values $k_1, \ldots, k_{t+1}$ uniformly from $\mathbb{Z}_p^*$. Choose mt values uniformly from $\mathbb{Z}_p$, denote them by a_{ij} ($i \in [1, m]$ and $j \in [1, t]$). For a particular i, the values a_{ij} will

represent coefficients of a polynomial of degree t (its free term is 0), and the values $k_1, \ldots, k_{t+1}$ will be the points on which each of these polynomials will be evaluated.

2. For $r = 1$ to $t+e+1$ compute the r-th row of the matrix as $[\sum_{j=1}^{t} a_{1j}k_r^j, \ldots, \sum_{j=1}^{t} a_{mj}k_r^j]$. These rows are referred as "special".

3. For $r = t + e + 2$ to $2t + 2e + 2$ choose the r-th row by choosing each component from $\mathbb{Z}_p$ uniformly. These rows are referred as "non-special".

4. Choose a random permutation π of the set $\{1, \ldots, 2t + 2e + 2\}$ and set the i-th row of the final matrix to be the $\pi(i)$-th row in the matrix defined above.

The SSHA states that the distribution $R(t, m, e, p)$ is computationally indistinguishable from $U(t, m, e, p)$. The idea behind it is that the only thing that makes one distribution different from the other one is the existence of the "special" rows. Note that the i-th entry of the r-th "special" row $\sum_{j=1}^{t} a_{ij}k_r^j$ equals to the value of the polynomial $a_i(x) = a_{i1}x + a_{i2}x^2 + \ldots + a_{it}x^t$ of degree t at the point k_r. Given at least $t + 1$ points $a(k_{i1}), \ldots, a(k_{i(t+1)})$ for distinct $k_{i1}, \ldots, k_{i(t+1)}$, there is a known non-trivial linear combination $\alpha_1, \ldots, \alpha_{t+1}$ such that $\alpha_1 a(k_{i1}) + \ldots + \alpha_{t+1} a(k_{i(t+1)}) = 0$. This means that if the adversary somehow guesses the "special" rows correctly, he may verify his guess by checking if the linear combination works. However, it is known that less than $k + 1$ points do not reveal any information about the corresponding polynomial of degree k. The SSHA assumes that there is no efficient algorithm other than just guessing and verifying all the sets of rows of size $t + 1$ by exhaustive search, and therefore the problem is hard.

We could try to use something similar by adding randomly generated columns to our matrix and trying to hide the initial program amongst them. If we would like to use this assumption directly, we would have to think on how to solve a linear program over $\mathbb{Z}_p$. Another approach would be to think of translating this security assumption to $\mathbb{R}$. Let us see which of them seems better.

4.2.1 Linear Programming over $\mathbb{Z}_p$

If it was possible to solve a linear programming task in $\mathbb{Z}_p$, there could be even more hard problems other than SSHA that could potentially be used (like the minimum-distance problem in the linear codes, or security of some McEliece-type cryptosystem [22]). This is the reason why linear programming over $\mathbb{Z}_p$ could be interesting.

Suppose that we have some transformation $\mathcal{T}$, the security of which is based on some computational assumption related to $\mathbb{Z}_p$. Given the initial constraints A and $\mathbf{b}$ (the cost vector is contained in A), we have $\mathcal{T}(\mathrm{A}, \mathbf{b}) = (\mathrm{A}', \mathbf{b}')$. Since $\mathcal{T}$ works in $\mathbb{Z}_p$, the elements of A' and $\mathbf{b}'$ belong to $\mathbb{Z}_p$, and the linear program to be solved is

$$\text{maximize } w, \text{ subject to } \mathrm{A}'\mathbf{x} = \mathbf{b}' \pmod{p}, \ \mathbf{x} \geq \mathbf{0} \ ,$$

where $w \in \mathbf{x}$ is the variable that represents the cost. This is equivalent to introducing additional variables k_i such that each equation of the form $\mathbf{a}_i^{\mathrm{T}}\mathbf{x} = b_i$ becomes $\mathbf{a}_i^{\mathrm{T}}\mathbf{x} - p \cdot k_i = b_i$. However, in this case a linear program becomes an integer program (since we are working in $\mathbb{Z}_p$), and solving it may become very slow. We could use different approximation algorithms that might work relatively well in practice (like the lattice

basis reduction algorithm LLL [18]). Nevertheless, since integer programming is known to be NP-complete, solving the program itself might actually mean breaking the security assumption, and therefore it may be difficult to use computational assumptions correctly.

However, the main problem of this approach is that it is not correct. There may be additional solutions that are optimal for the transformed program, but infeasible for the initial program. Suppose that we have the following 3-variable linear programming task:

$$\begin{aligned} &\text{minimize } w, \\ &\text{subject to} \\ x - z &= 2 \\ 2x - w &= 0 \\ x, z, w &\geq 0 \end{aligned}$$

The constraint $x - z = 2$ ensures that $x \geq 2$, and the optimal solution in $\mathbb{R}$ is therefore $x = 2, z = 0, w = 4$. If we solve the same program in $\mathbb{Z}_p$, then for example $x = (p+1)/2, z = x - 2, w = 2x = p + 1 \equiv 1 \pmod{p}$ is an optimal solution for any odd p. For $p = 2$ we would clearly have problems with the programs that use numbers greater than 1, for example all the even numbers would be mapped to 0. We could try to use rings instead of fields, but a ring would have the same problems. There would be even more possibilities of getting false zeroes (if the matrix entries turn out to be zero divisors). For example, for any even n, the following linear program would have a wrong solution in $\mathbb{Z}_n$:

$$\begin{aligned} &\text{minimize } w, \\ &\text{subject to} \\ x - z &= 1 \\ 2x - w &= 0 \\ x, z, w &\geq 0 \end{aligned}$$

The optimal solution in $\mathbb{R}$ is $x = 1, z = 0, w = 2$, while the optimal solution in $\mathbb{Z}_n$ is $x = n/2, z = x - 1, w = 2x = n \equiv 0 \pmod{n}$.

Since linear programming does not work correctly in $\mathbb{Z}_p$, the preferable field is either $\mathbb{Q}$ or $\mathbb{R}$. We may try to use something similar to the security assumptions over finite fields, but we will need to define new assumptions that hold over $\mathbb{Q}$ or $\mathbb{R}$. Since $\mathbb{Q}$ may be vulnerable to number-theoretical attacks, we will choose $\mathbb{R}$.

4.2.2 Security Assumptions over $\mathbb{R}$

Working with $\mathbb{R}$, we would most likely have to define some completely new assumption, because cryptography over reals has not received much attention so far (although there have been some related works, for example [7]). The assumptions that work well in $\mathbb{Z}_p$ may be not so easily extendable to $\mathbb{R}$. If we try to use a transformation similar to SSHA in some other fields, we may have problems. For example, if we just take the same description over $\mathbb{Z}$, we get that all the entries of the "special" rows are divisible by the same number (the powers of k_r that appear in each term of the sum), and since two random numbers are coprime with a high probability, these rows may be easily discovered.

This problem may be solved by going from $\mathbb{Z}$ to $\mathbb{R}$, then the number-theoretical problems will be avoided. Still, another problem is that the entries in the special rows are sums of numbers with large exponents, and they will be just visually distinguishable unless we start selecting the entries of the "non-special" rows from some distribution other than just from the span $[0, p)$ as in the initial SSHA definition. Even if we fix these issues, we still cannot be sure that there are no other problems.

For a linear program, the difference in the sizes of the columns may be harmful. Even though the variables are permuted, if one column is sufficiently larger than the others, it may already reveal some information. In the brewery example, it may show that there is a product that consumes significantly more resources per unit than the others, and the adversary may use this fact to identify to what actual product that variable might correspond. This may be partially avoided by scaling the columns by a number that comes from a sufficiently big range (like multiplying the initial matrix by a positive monomial matrix Q in [8]). The problem is that if we use scaling to make the columns more or less of the same size, then the same scaling causes the corresponding variables to differ in sizes and leak the same information.

To our knowledge, there have been no solutions for this problem. We propose a solution that brings the columns to similar size.

Let us introduce some definitions and propositions that will be needed for convincing why this approach might work.

Definition 18. *The normal distribution with mean μ and standard deviation σ, denoted $\mathcal{N}(\mu, \sigma^2)$, is defined by its probability density function $f_{\mathcal{N}(\mu,\sigma^2)}(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp(-\frac{1}{2}(\frac{x-\mu}{\sigma})^2)$.*

Proposition 3. *If $X_1, \ldots, X_n$ are independent random variables sampled from the distribution $\mathcal{N}(0, 1)$, then the variable $Z = \sum_{i=1}^{n} a_i X_i$ (for fixed a_i) is distributed according to $\mathcal{N}(0, \sum_{i=1}^{n} a_i^2)$.*

Proof: This property can be derived from the definition of the probability density function of normal distribution [11]. □

The sizes of the columns of the matrix A will be normalized in the following way:

1. Let L be the maximal ℓ_2-norm of the columns of A.
2. The goal is to bring the ℓ_2-norms of each column to the value L (or some larger value, which gives better results). There are several ways to do it:
   - Scale the columns. This may unfortunately produce some other security leakage since it at the same time scales the variables.
   - Add a row of positive entries where the value of each entry is equal to the value $\sqrt{L^2 - L_i^2}$, where L_i is the ℓ_2-norm of the corresponding column. For the correctness of the linear program, it requires introduction of an additional variable that compensates the entire sum (the entry in the corresponding column of the added row should be a negative value).
   - Multiply A from the left not just by any invertible matrix, but more precisely, by a matrix P whose entries are sampled from $\mathcal{N}(0, 1)$.

Treating the entries of A as arbitrary constants, each entry in the matrix obtained after the multiplication will be distributed according to $\mathcal{N}(0, L)$ due to the linearity of $\mathcal{N}(0, 1)$ distribution. There however are still some problems with this approach:

- The entries of A are not quite constants, they are used several times in the matrix multiplication. This construction may produce slightly different distributions in practice.
- Actually, the indistinguishability of the columns should not depend on the distribution of P since the adversary may always bring the matrix to its reduced row echelon form, and it will be the same regardless of the choice of P. A better security analysis should be based on comparing the sizes of columns in the reduced row echelon form.
- If a new variable is introduced, it is difficult to make it indistinguishable. Namely, although its ℓ_2-norm can be set to L by scaling, this scaling makes the size of the variable itself too different. Most probably, we should assume that this variable is distinguishable from the others, and it may reveal some information.

In practice, the columns become very similar when L is sufficiently larger that the initial ℓ_2-norms.

4.3 Possible Ways of Hiding

Since we have proven that obtaining a transformed polyhedron with perfect secrecy is impossible, the next goal is to make the transformed polyhedrons at least computationally indistinguishable from each other. We will consider the approach that is based on making the variables indistinguishable by permuting the columns. We will also make use of the existing methods, such as multiplying the matrix from the left and scaling the columns.

Since just permuting the initial columns amongst each other will barely be useful, we need to add more columns and hide the real ones amongst them. Each column introduces a new variable. The new variables may in general be added in two different ways:

- included into additional row constraints;
- augmented to the columns of the initial matrix.

We must be careful that adding these variables still make it possible to recover the optimal solution of the initial program.

We will further assume that the linear program is given in its augmented form (3) as minimize w, subject to $\mathrm{A}\mathbf{x} = \mathbf{b}$, where $w \in \mathbf{x}$. The constraint $\mathbf{x} \geq \mathbf{0}$ is being added by the solver for correctness. The system may already be multiplied by an invertible matrix from the left, and its columns scaled and permuted. Since we have shown that these operations do not provide enough privacy, we assume that revealing information about the solution space of $\mathrm{A}\mathbf{x} = \mathbf{b}$ violates the security requirements. In our analysis, we will not treat the column that corresponds to the cost variable w separately. Revealing one column does not affect the security too much since the adversary could guess it on his own.

4.3.1 Adding Constraints

In general, additional constraints can be added to the system $\mathrm{A}\mathbf{x} = \mathbf{b}$ in the following way:

$$\begin{pmatrix} \mathrm{A} & 0 \\ \mathrm{D} & \mathrm{E} \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} = \begin{pmatrix} \mathbf{b} \\ \mathbf{b}' \end{pmatrix} \ ,$$

where D and E are arbitrary matrices, and $\mathbf{b}'$ is an arbitrary vector. Of course, this system in turn needs to be multiplied by an invertible matrix P from the left, otherwise the structure of A would remain clearly visible. The system is also multiplied by a positive monomial matrix Q from the right to scale and permute the variables.

The new constraints reduce the solution space in the initial dimension, and some more dimensions are added due to the new variables $\mathbf{y}$. The matrices D and E and the vector $\mathbf{b}'$ have to be chosen in such a way that at least one optimal solution of the system $\mathrm{A}\mathbf{x} = \mathbf{b}$ is still feasible. Since we assume that the users do not have any information about the optimal solution initially, we require that all the solutions are still feasible.

An example that does not affect the initial solution space is where $\mathrm{E} = -\mathrm{I}$, $\mathbf{b}' = \mathbf{0}$, and D is a matrix with non-negative entries. In this case, the variables $\mathbf{y}$ represent positive linear combinations of the variables $\mathbf{x}$.

Definition 19. *Linear equation systems are called equivalent if their solution spaces are equal.*

An important requirement is that the variables $\mathbf{y}$ should be indistinguishable from the variables $\mathbf{x}$. Otherwise the adversary may get rid of them using Gaussian elimination, by expressing the variables $\mathbf{y}$ as linear combinations of the variables $\mathbf{x}$.

- If the rank of E is equal to the number of its rows, then after eliminating all the $\mathbf{y}$ variables the adversary is left with the system that is equivalent to initial system $\mathrm{A}\mathbf{x} = \mathbf{b}$.
- Otherwise it is possible that after eliminating the constraints the adversary is left with a system equivalent to a system of the form $\begin{pmatrix} \mathrm{A} \\ \mathrm{D}' \end{pmatrix} \mathbf{x} = \begin{pmatrix} \mathbf{b} \\ \mathbf{b}'' \end{pmatrix}$ for some matrix D' and a vector $\mathbf{b}''$. However, this means that before the Gaussian elimination the system already sets some constraints on the variables $\mathbf{x}$. Setting these constraints properly (without affecting the optimal solution $\mathbf{x}_{opt}$) would require some knowledge about $\mathbf{x}_{opt}$, and we assume that the users who want to compute the linear program securely do not have any information about it. Therefore this case is impossible.

The conclusion is that the indistinguishability of $\mathbf{x}$ and $\mathbf{y}$ is essential.

4.3.2 Augmenting the Columns

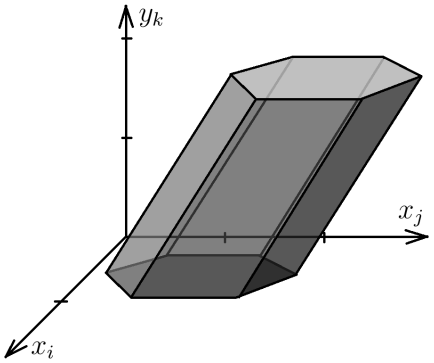
We may get additional solutions for the initial variables by adding columns to A

$$\begin{pmatrix} \mathrm{A} & \mathrm{C} \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} = \mathbf{b} \ ,$$

followed by multiplying the system by a random invertible matrix P from the left and a positive monomial matrix Q from the right.

The optimal solution has to be still easily recoverable. An example that does not change the optimal solution is C = AV, where V is a matrix whose rows have positive entries except the one that corresponds to the variable w that is being optimized. If w is being minimized, then this row is negative, otherwise it is also positive.

This adds more solutions by extending the polyhedron to new dimensions. The optimal solution remains in the region where all the new variables have the value 0.



Why can the optimal solution be still correctly recovered? Let $\mathrm{B} = \begin{pmatrix}\mathrm{A} & -\mathbf{b}\end{pmatrix}$. Let us add one empty row to V, so that AV = BV. Now consider the matrix $\begin{pmatrix}\mathrm{B} & \mathrm{BV}\end{pmatrix} = \begin{pmatrix}\mathrm{A} & -\mathbf{b} & \mathrm{AV}\end{pmatrix}$. This matrix has one extra variable z that corresponds to the column $-\mathbf{b}$. If we fix the value of z to 1, then the solution set to the system

$$\begin{pmatrix}\mathrm{A} & -\mathbf{b} & \mathrm{AV}\end{pmatrix}\begin{pmatrix}\mathbf{x}\\ z\\ \mathbf{y}\end{pmatrix} = 0$$

is equivalent to the solution set of the system

$$\begin{pmatrix}\mathrm{A} & \mathrm{AV}\end{pmatrix}\begin{pmatrix}\mathbf{x}\\ \mathbf{y}\end{pmatrix} = \mathbf{b}\ .$$

Definition 20. *The null space of a matrix* A *is the solution space of the system* $\mathrm{A}\mathbf{x} = \mathbf{0}$.

Definition 21. *Let* A *be a* $m \times n$ *full-rank matrix. A kernel matrix* $\mathrm{A}^{\perp}$ *is a* $(n-m)\times n$ *matrix whose rows generate the null space of* A.

One important property of the kernel matrix is that $\mathrm{A}(\mathrm{A}^{\perp})^{\mathrm{T}} = \mathrm{A}^{\perp}\mathrm{A}^{\mathrm{T}} = 0$.

Consider the null space of $\begin{pmatrix}\mathrm{B} & \mathrm{BV}\end{pmatrix}$. It is generated by the rows of the matrix

$$\mathrm{K} = \begin{pmatrix}\mathrm{B}^{\perp} & 0\\ -\mathrm{V}^{\mathrm{T}} & \mathrm{I}\end{pmatrix}\ ,$$

where $\mathrm{B}^{\perp}$ is a kernel matrix of B. By definition, $\mathrm{B}^{\perp}$ is a matrix whose rows generate the null space of B, and one if its important properties is that $\mathrm{B}\mathrm{B}^{\perp} = 0$.

The definition of K is correct since

$$\begin{aligned}\begin{pmatrix}\mathrm{B} & \mathrm{BV}\end{pmatrix}\cdot\begin{pmatrix}\mathrm{B}^{\perp} & 0\\ -\mathrm{V}^{\mathrm{T}} & \mathrm{I}\end{pmatrix}^{\mathrm{T}} &= \\ \begin{pmatrix}\mathrm{B} & \mathrm{BV}\end{pmatrix}\cdot\begin{pmatrix}(\mathrm{B}^{\perp})^{\mathrm{T}} & -\mathrm{V}\\ 0 & \mathrm{I}\end{pmatrix} &= \\ \begin{pmatrix}(\mathrm{B}(\mathrm{B}^{\perp})^{\mathrm{T}} + \mathrm{BV}\cdot 0) & (\mathrm{B}\cdot(-\mathrm{V}) + \mathrm{BV})\end{pmatrix} &= 0\end{aligned}$$

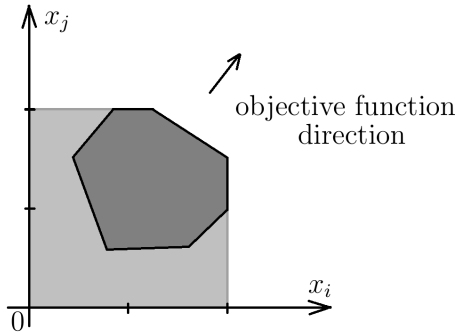
The solution space for $\begin{pmatrix} \mathrm{B} & \mathrm{BV} \end{pmatrix} \mathbf{x} = \mathbf{0}$ is represented by linear combinations of the rows of K. A linear combination of the rows of K is a feasible solution to the initial linear program iff the entry that corresponds to z equals 1, and all the other entries are non-negative (due to the $\begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} \geq \mathbf{0}$ requirement). This implies that the lower rows of K that are associated with I must have non-negative coefficients in such a feasible linear combination. On the other hand, decreasing the coefficients of these lower rows will improve the value of the objective function w since the corresponding column of $-\mathrm{V}^{\mathrm{T}}$ contains only numbers with the sign opposite to the optimization direction of w. Hence the best value for w is achieved when $\mathbf{y} = \mathbf{0}$, and this is definitely not harmful for the correctness since it returns just the initial equation system $\mathrm{A}\mathbf{x} = \mathbf{b}$.

Also, no solution to the initial linear program becomes infeasible by the same reasoning that any solution for the initial linear program is available when $\mathbf{y} = 0$.

This approach is described more precisely in [6].

Another possible method is to replace each initial variable with a positive linear combination of t new variables. This is equivalent to replacing each column of A with several scaled instances of the same column. Recall that we want to obtain a system of the form $\begin{pmatrix} \mathrm{A} & \mathrm{C} \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} = \mathbf{b}$. Without loss of generality, assume that at least one coefficient in each of these linear combinations is 1. In this case, the other coefficients introduce scaled copies of the columns of A as the columns of C. If we do not require that one of the coefficients should be 1, then the columns of A will also be scaled, but the scaled A still leaks the same information as the non-scaled A (since we initially assumed that A is already scaled).

This transformation produces additional solutions since the optimal solution may be obtained not only from the variables $\mathbf{x}$, but also from the variables $\mathbf{y}$, and each variable may now take value down to 0.



It is again essential that the elements of $\mathbf{x}$ would be indistinguishable from the elements of $\mathbf{y}$. Otherwise, setting these new variables to 0 will return back the initial program.

This issue can be solved by generalizing the transformation to

$$\begin{pmatrix} \mathrm{A} & \mathrm{C} \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} = \mathbf{b}'' \ ,$$

where $\mathbf{b}''$ is some new vector different from $\mathbf{b}$. Setting $\mathbf{y} = \mathbf{0}$ yields the system $\mathrm{A}\mathbf{x} = \mathbf{b}''$ instead of $\mathrm{A}\mathbf{x} = \mathbf{b}$. However, the adversary may easily set the right hand side of the equation to $\mathbf{0}$, obtaining the equation system $\mathrm{A}\mathbf{x} = \mathbf{0}$. This violates our security definition since the two initial equation systems $\mathrm{A}_1\mathbf{x} = \mathbf{b}_1$ and $\mathrm{A}_2\mathbf{x} = \mathbf{b}_2$ that the adversary has to

distinguish may have different solution sets for $A_1\mathbf{x} = \mathbf{0}$ and $A_2\mathbf{x} = \mathbf{0}$. Another question is how to introduce $\mathbf{b}''$ so that all the solutions would still be feasible.

4.3.3 Combining Different Types of Variables

Suppose that we have added variables that at once extend the columns of A and participate in the constraints

$$\begin{pmatrix} A & C \\ D & E \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} = \begin{pmatrix} \mathbf{b}'' \\ \mathbf{b}' \end{pmatrix} \ ,$$

and multiplied the system by a random invertible matrix P form the left.

In this case, the adversary may again use Gaussian elimination to get rid of $\mathbf{y}$. There are now two possible outcomes of this operation.

1. If he is left with a system that has more equations than the number of rows in A, then we have the same problem as we had in the case of just adding constraints. Namely, the system sets some additional constraints on the variables $\mathbf{x}$ which requires knowledge about the optimal solution. Hence such an outcome is impossible.

2. If he is left with a system that has less equations than the number of rows in A, then he may already lose some important information about A. However, if in the process of Gaussian elimination he stops at the moment where the number of rows in the matrix is the same as it should be in the rows of A, then knowing the locations of the $\mathbf{y}$ variables allows to set them to 0. Due to correctness of the transformation, the solutions to the initial system $A\mathbf{x} = \mathbf{b}$ should be still recoverable, otherwise it would be the case where the initial program contained additional constraints set on $\mathbf{x}$. Hence the adversary gets the initial linear program.

Again, we get that indistinguishability of $\mathbf{x}$ and $\mathbf{y}$ is essential.

For example, in [15], the constraints are applied only to the additional variables. These variables indeed participate in the equations, but their effect is already included in $\mathbf{b}$. In this particular case, the additional variables are used to hide $\mathbf{b}$. The partially transformed program (without taking into account scaling etc) looks like

$$\begin{pmatrix} A & T \\ 0 & I \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} = \begin{pmatrix} \mathbf{b} + T\mathbf{r} \\ \mathbf{r} \end{pmatrix} \ ,$$

where T is a diagonal matrix. Since the variables are clearly distinguishable ($\mathbf{y}$ is fixed), the adversary may get a system equivalent to the initial system $A\mathbf{x} = \mathbf{b}$ just by using Gaussian elimination. However, this transformation does not hide the feasible region of $A\mathbf{x} = \mathbf{b}$ anyway, its goal is just to hide the initial outer appearance of $\mathbf{b}$.

4.4 Indistinguishability of the Variables

The conclusion from the previous is that if we use additional variables in the transformation, then their effect on the feasible region may be easily removed unless these variables are indistinguishable from each other. We will analyze the requirements that must be fulfilled for the indistinguishability of the variables. Then we propose a construction in which we are trying to achieve these properties.

4.4.1 Security Requirements

Suppose that after the transformation we obtain a matrix with k columns meaning that the transformed system has k variables. Let t of them be the "special" variables, knowing whose location may allow the adversary to perform an attack (such as applying Gaussian elimination to them). A necessary requirement is that the values k and t must be large enough, so that the adversary would require a big computational effort to look through all the $\binom{k}{t}$ sets of variables.

But can we be sure that he definitely needs to guess correctly all the t "special" variables at once in order to perform his attack? It may happen that the initial variables take values from different spans, and thus the adversary may discover the type of a variable just by minimizing and maximizing its value (using the same linear programming solver algorithm). To avoid this, we may scale the variables and make all their upper and lower bounds similar. This is possible if the lower bound of all the variables is 0, otherwise we cannot use scaling to adjust both the upper and the lower bound. But even if each single variable comes from the same span, what about the linear combinations of pairs of variables, or their triples?

We will state this problem more generally. The adversary may try to project the solution space of the system to some space with small dimension (for example, to three dimensions, then he even has a visual picture). There exists an efficient algorithm for that according to [16], and therefore it is a real attack. The algorithm works in polynomial time with respect to the number of halfspaces that define the projection. Since each halfspace that defines a polyhedron that corresponds to a linear program is represented by an inequality, the algorithm works in polynomial time with respect to the number of rows in the constraint matrix. There is no way to increase this number exponentially since it affects the efficiency of solving the linear program. Even a small projection may already reveal too much. For example minimizing and maximizing each variable would be equivalent to looking for the one-dimensional projections.

Let us consider the case where the initial polyhedron has been objected to some kind of affine transformation (scaling and shifting). The adversary may extract small projections of the initial polyhedron whose structures may seem familiar even after such a transformation. This may allow to distinguish the two initial linear programs by choosing them in such a way that they have some interesting special small projections of their feasible regions. For example, if one of them contains tetrahedrons and pyramids as projections, and the other one just cubes, then even if similar projections will occasionally be produced by the additional variables $\mathbf{y}$, the probability of getting sufficient number of exactly the same projections is very small.

Suppose that we are going to hide $\mathbf{x}$ amongst additional variables $\mathbf{y}$. We have shown before that indistinguishability of $\mathbf{x}$ and $\mathbf{y}$ is essential. Although all the variables may have the same upper and lower bounds, they may be still distinguished by their higher-dimensional projections. Even if the adversary finds a way of distinguishing whether all t variables come from $\mathbf{x}$ only, or they involve also some variables of $\mathbf{y}$, he would get much additional power. Therefore we require that not only all projections to the initial variables $\mathbf{x}$ should look the same, but also all projections onto any t variables from the entire $\begin{pmatrix}\mathbf{x}\\ \mathbf{y}\end{pmatrix}$ should look the same.

We are going to define a transformation where we are trying to compose a secure

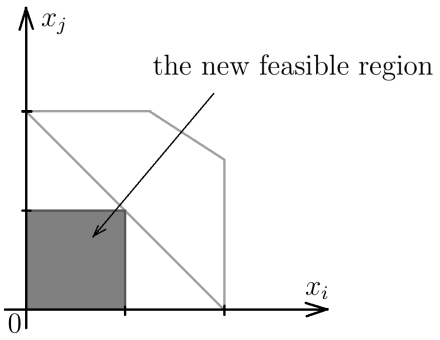
structure step by step. Then we analyze the security requirements of our construction and see if they can be achieved at all.

4.4.2 Defining a Construction

Suppose that we have transformed $A\mathbf{x} = \mathbf{b}$ into $\begin{pmatrix} A & C \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} = \mathbf{b}$ by replacing each column of A with a positive linear combination of t columns. Of course, everything is multiplied by a random invertible matrix P from the left to hide the outer appearance of the system.

As we have shown before, each variable may now take value down to 0. We may hope that if t is large enough, then it is difficult for the adversary to recover even a single variable. However, the problem is that the faces of the polyhedron that are located in the direction of the objective function will not be changed. The projections to smaller dimension will be not exactly the same as before, but there will still be something special. For each projection to t coordinates, we need to cut off the part of the polyhedron that has not changed. Therefore we have to introduce inequalities that add bounds also on the initial variables. The question is how to do it without losing the optimal solution.

Since we have replaced each initial variable with a positive linear combination of t new variables, we may bound each single variable in such a way that the optimal solution is still achievable. This is possible if positive linear combinations of the copies of a single initial variable still achieve the necessary bounds for that variable. Taking into account the convexity property of a polyhedron, we may define an artificial bound for each variable so that the shape of the polyhedron is no longer seen at least for the sets of t initial variables.



We may try different formations inside the smaller region, possibly some hypercube (like on the picture) since making the variables too dependent on each other may affect the correctness. Let us make it step by step.

- If we start introducing upper bounds on some linear combinations of the variables, then each bound requires a unique slack variable (although we can probably reduce the number of slack variables somehow by using them multiple times). Since small sets of related variables can be discovered using Gaussian elimination (as we have shown in the attack on [8]), we need to involve at least t variables in each equation. We may either group the initial variables, or introduce more slack variables.

- Now the problem is that any bound can be removed just by allowing the corresponding slack variable to take negative values. In order to remove a single bound,

it is sufficient to free at least one variable from the corresponding equation, and therefore adding more slack variables to the same equation does not help. We need at least t instances of each bound as t separate inequalities.

- But can we be sure that the adversary indeed has to try all the sets of t columns? If he finds a way of distinguishing the slack variables from the non-slack variables, he may easily remove all the bounds and recover the initial linear program without affecting its correctness. Therefore, the slack variables should be indistinguishable from the other variables, so that the bounds could not be removed. This means that any projection to t variables (including the slacks) should look exactly the same to the adversary. We will see in which case it is possible.

Definition 22. *An affine space is a set of points that can be represented as a solution set of some linear equation system.*

Since we are now interested in the indistinguishability of both the initial and the slack variables, let us consider the standard form of linear programming (2). Let us for simplicity denote the matrix of the linear equation system by A:

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

Let the length of $\mathbf{x}$ be n. Then the solution set of the system $\mathrm{A}\mathbf{x} = \mathbf{b}$ defines an affine space V in $\mathbb{R}^n$. The polyhedron that corresponds to that linear program can thus be seen as the intersection $V \cap \mathbb{R}^n_+$. In further analysis, we will treat linear programs as affine spaces.

4.5 Proofs of Insecurity

We will show that, whatever affine transformation we use, it is impossible to protect the linear program against projection-based attacks. First, we formally define what the transformation of a linear program means mathematically. Then we find the possible classes of affine spaces that achieve the desired security properties (make all projections to any t variables look the same). It turns out that the corresponding class of linear programs would be very small and not interesting in practice.

4.5.1 Formal Definition of a Transformation

Let $P_0 \subseteq \mathbb{R}^n$ be the affine space that corresponds to the initial linear program. By the principle of transformation-based linear programming, P_0 should be transformed to some other affine space $P \subseteq \mathbb{R}^k$. It is possible that $n \neq k$ since we may introduce additional variables, or probably even reduce their number in some way (although it is not clear how we preserve the correctness in this case).

An important requirement is that it must be possible to extract at least one optimal solution $\mathbf{x}_{opt}$ of the initial program from any optimal solution $\mathbf{y}_{opt}$ of the transformed program. Therefore, there must exist a function $f : \mathbb{R}^k \to \mathbb{R}^n$ such that $f(\mathbf{y}_{opt}) = \mathbf{x}_{opt}$. Initially, there is no information about $\mathbf{x}_{opt}$ or $\mathbf{y}_{opt}$, so let $f(P) = P_0$. Since a linear program is being transformed into another linear program, we may assume that the function f is an affine transformation. Then $P = f^{-1}(P_0) \cap \mathbb{R}^k_+ \cap V$, where $V \subseteq \mathbb{R}^k$

is an affine space that represents the additional constraints that may be added to the transformed program. We must be sure that these constraints do not affect the feasible region of the initial program (that still $f(P) = P_0$).

In order to protect the initial program against projection-based attacks, we require that all projections of P to any t coordinates must look the same. Formally, if $I \subseteq \{1, \ldots, k\}$, define $W_I := \pi_I(P)$. We require that, for any rearrangement of the coordinates of two sets I and I' of size t, the sets W_I and $W_{I'}$ must be exactly the same.

The question is whether there are non-trivial possible classes of P that could be useful in practice.

4.5.2 Possible Classes of Affine Spaces

We will show that even if we take $t = 2$, the class of suitable affine spaces is very small. Namely, we show that P must be either a single point $(0, \ldots, 0)$, the entire $\mathbb{R}^k_+$, or a set of the form $\{(x_1, \ldots, x_n) \mid x_1 + \ldots + x_n \leq c\}$ for some $c > 0$.

Let the transformed affine space P be defined by a system $\mathrm{A}'\mathbf{x} = \mathbf{b}'$, where A' is an $m \times n$ matrix for $m \leq n - 1$. If $m = n - 1$, then we would unfortunately have to consider more different cases in our proof. Anyway, $m = n - 1$ would be a very special case of one-dimensional linear programming, and that's why we decided not to adjust the proofs to it.

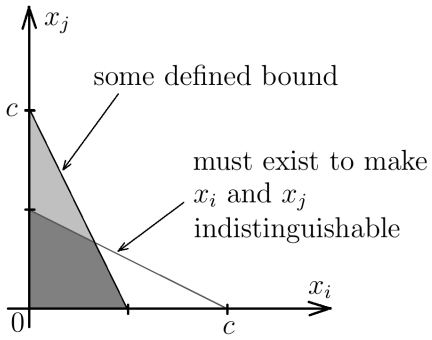
If $m \leq n - 2$, then the linear equation system $\mathrm{A}'\mathbf{x} = \mathbf{b}'$ definitely contains solutions where at least two variables are 0 since A' contains a $m \times m$ invertible submatrix, and setting the two remaining columns to 0 still yields a solvable linear equation system.

Let us now construct an arbitrary two-dimensional projection of P in dimensions x_1 and x_2 that contains the point $(0, 0)$. Since P is a polyhedron, this projection must be a convex polygon. Suppose that one of the bounds defining this polygon is $\alpha_1 x_1 + \alpha_2 x_2 \leq c$ for some $c \in \mathbb{R}$, $\alpha_1 \in \mathbb{R}$, $\alpha_2 \in \mathbb{R}$. More precisely, $c \geq 0$ since otherwise the point $(0, 0)$ would not belong to the projection.

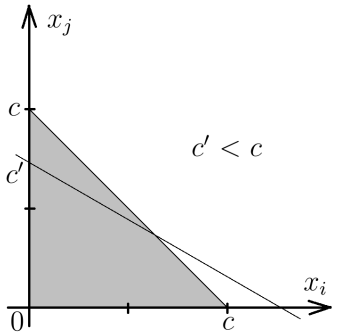
- If both $\alpha_1 \leq 0$ and $\alpha_2 \leq 0$, then the inequality has no meaning. Multiplying both sides by (-1), we see that it actually states that non-negative combinations of non-negative values should be non-negative.
- Otherwise, we may divide everything by $\max\{\alpha_1, \alpha_2\}$, and we get an inequality of the form $\alpha x_1 + x_2 \leq c$ for some $-\infty < \alpha \leq 1$ (the variables x_1 and x_2 may switch their places, depending on whether $\alpha_1 > \alpha_2$).

Hence without loss of generality we may assume that any bound may be given in the form $\alpha x_1 + x_2 \leq c$ for $c \geq 0$, $-\infty < \alpha \leq 1$.

Since all projections onto any pair of variables should look exactly the same, we get a system of $2 \cdot \binom{n}{2}$ such inequalities for a single bound in the projection. Here $\binom{n}{2}$ is the number of possible pair choices, and this number is multiplied by 2 since the inequality $\alpha x_1 + x_2 \leq c$ implies the existence of the inequality $x_1 + \alpha x_2 \leq c$, otherwise the projection would not be symmetric and would make the two coordinates distinguishable.



Let our projection be defined by an arbitrary number of bounds. Let us take the bound with minimal value of c. This means that each variable may take any value in the span $[0, c]$, otherwise it would mean that there exists a bound with a lower value $c' < c$. Due to convexity of the projection, we may say more precisely that any valuation (v_i, v_j) of x_i and x_j such that $v_i + v_j \leq c$ must be possible.

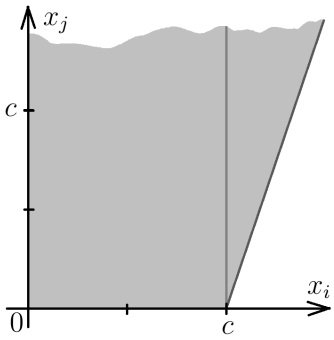


Given n variables, any inequality $\alpha x_1 + x_2 \leq c$ comes from some equation of the form $\alpha x_1 + x_2 + a_3 x_3 + \ldots + a_n x_n = c$ defined by P, where $a_i \geq 0$, and $a_j > 0$ for some j (the variable x_j acts as a slack variable for the inequality). In total, we get $2 \cdot \binom{n}{2}$ equations that all follow from the equations defined by P:

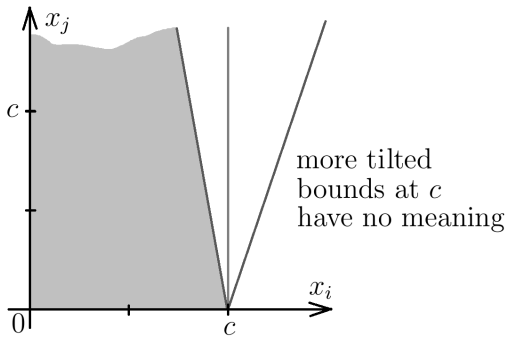
$$\begin{cases} \alpha x_1 + x_2 + a_{123} x_3 + \ldots + a_{12(n-1)} x_{n-1} + a_{12n} x_n = c \\ x_1 + \alpha x_2 + a_{213} x_3 + \ldots + a_{21(n-1)} x_{n-1} + a_{21n} x_n = c \\ a_{231} x_1 + \alpha x_2 + x_3 + \ldots + a_{23(n-1)} x_{n-1} + a_{23n} x_n = c \\ a_{321} x_1 + x_2 + \alpha x_3 + \ldots + a_{32(n-1)} x_{n-1} + a_{32n} x_n = c \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n(n-1)1} x_1 + a_{n(n-1)2} x_2 + a_{n(n-1)3} x_3 + \ldots + \alpha x_{n-1} + x_n = c \\ a_{(n-1)n1} x_1 + a_{(n-1)n2} x_2 + a_{(n-1)n3} x_3 + \ldots + x_{n-1} + \alpha x_n = c \end{cases} \tag{4}$$

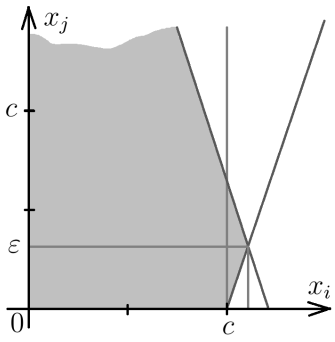
where $a_{ijk} \geq 0$ for all $i, j, k \in \{1, \ldots, n\}$, and each equation contains at least one positive a_{ijk}. We will show that these equations imply $x_1 + \ldots + x_n = c$.

1. **The case** $\alpha \leq 0$**.** The lines of the bound are either parallel to the x_j axis, or are tilted in the direction opposite from the x_j axis.



Consider a point $v = (v_1, \ldots, v_n) \in P$ where $(v_i, v_j) = (c - \alpha\varepsilon, \varepsilon)$ for some $\varepsilon > 0$. Although there may be other bounds, such a point (v_i, v_j) definitely exists on the projection to (x_i, x_j) since otherwise the bound would not add anything new to the projection.





Consider the equation $a_{ij1}x_1 + \ldots + x_i + \ldots + \alpha x_j + \ldots + a_{ijn}x_n = c$. Since each equation represents an inequality, there exists some $a_k > 0$ that corresponds to some variable x_k, $k \neq i$, $k \neq j$ (x_k acts as a slack variable).

(a) **The case** $\alpha = 0$. Since $\alpha = 0$, there is nothing that would compensate too large positive values. At the same time, due to the symmetry of the projections, there must exist a point $v' = (v'_1, \ldots, v'_n) \in P$ such that $(v'_i, v'_k) = (c - \alpha\varepsilon, \varepsilon) = (c, \varepsilon)$. Then $v'_i + a_k v'_k = c + a_k\varepsilon > c$. Contradiction!

(b) **The case** $\alpha < 0$. We have $v_i + \alpha v_j = c - \alpha\varepsilon + \alpha\varepsilon = c$. Hence for the valuation v we already have at least c in the left-hand-side of the equation. Since the coefficient of x_k in this equation is $a_k > 0$, it must be $v_k = 0$, otherwise we get something larger than c. Now we have $(v_i, v_k) = (c - \alpha\varepsilon, 0)$. Consider the equation $a_{ik1}x_1 + \ldots + x_i + \ldots + \alpha x_k + \ldots + a_{ikn}x_n = c$. We get $v_i + \alpha v_k = c - \alpha\varepsilon + 0 > c$. Contradiction!

Thus the only possible case is $\alpha > 0$.

2. **The case** $\alpha > 0$. If $c = 0$, then the only possible solution for the system (4) would be $x_1 = \ldots = x_n = 0$, since there are no negative entries at all. Consider the case $c > 0$.

   Take the equation $a_{ij1}x_1 + \ldots + x_i + \ldots + \alpha x_j + \ldots + a_{ijn}x_n = c$. Let $v = (v_1, \ldots, v_n) \in P$ be such that $v_i = c$. Then we already have c in the left-hand-side of the equation.

Since $\alpha > 0$, the value of v_j must be 0 since otherwise it would only increase the value of the left-hand-side.

Analogically, consider one by one all the equations $a_{ik1}x_1 + \ldots + x_i + \ldots + \alpha x_k + \ldots + a_{ikn}x_n = c$ for $k \neq i$. Similarly, we get $v_k = 0$ for all $k \neq i$. The entire valuation v must be therefore $(v_1, \ldots, v_{i-1}, v_i, v_{i+1}, \ldots, v_n) = (0, \ldots, 0, c, 0, \ldots, 0)$.

If the system (4) contains any equation where the coefficient of x_i is $a \neq 1$, we get a contradiction since $av_i \neq v_i = c$, and the other variables cannot affect this value since they are all 0. We get that the coefficient of x_i should be 1 in each equation.

In the same way, we get that the coefficients of all the variables in all equations should be 1. This means that the only equation that remains is $x_1 + \ldots + x_n = c$.

Another thing that we would like to show is that if the equation system defined by P contains a constraint $x_1 + \ldots + x_n = c$ for some $c > 0$, then it is not allowed to have any other constraints. Suppose that the system contains the following two equations:

$$\begin{aligned} x_1 + \ldots + x_n &= c \\ a_1x_1 + \ldots + a_nx_n &= b \end{aligned}$$

where $a_i, b \in \mathbb{R}$. We will show that the second equation can be at most a multiple of the first equation, representing the same constraint.

Without loss of generality, let $a_1 = \min_i a_i$, $a_2 = \max_i a_i$. Then $a_2 > 0$ since otherwise all the a_i would be non-positive, and the only possible solution would be $x_i = 0$ in the case $b = 0$. For $b > 0$, there would be no solution at all. We may also assume that $a_2 > a_1$ since if it was the case $a_2 = a_1$, then all the a_i would be equal, and the only possible way to avoid contradiction with the first equation would be to assign $b = a_ic$, making the second equation a multiple of the first one.

Multiplying the first equation by a_1, we get the following:

$$\begin{aligned} a_1x_1 + \ldots + a_1x_n &= a_1c \\ a_1x_1 + \ldots + a_nx_n &= b \end{aligned}$$

Subtracting the first equation from the second one, we get

$$(a_2 - a_1)x_2 + \ldots + (a_n - a_1)x_n = b - a_1c$$

Since $a_2 > a_1$, we may express the variable x_2 in terms of other variables:

$$x_2 = \frac{1}{a_2 - a_1}\left(b - a_1c - \sum_{i=3}^{n}(a_i - a_1)x_i\right)$$

We know that the only allowed valuations of x_i are positive.

- Since $x_2 \geq 0$, the right-hand-side must also be positive.

$$b - a_1c \geq \sum_{i=3}^{n}(a_i - a_1)x_i$$

- From the first equation, we may express $x_1 = c - \sum_{i=2}^{n} x_i$. We get

$$\begin{aligned}
0 \leq x_1 &= c - \frac{1}{a_2 - a_1}\left(b - a_1 c - \sum_{i=3}^{n}(a_i - a_1)x_i\right) - \sum_{i=3}^{n} x_i \\
&= \frac{1}{a_2 - a_1}\left((a_2 - a_1)c - (b - a_1 c - \sum_{i=3}^{n}(a_i - a_1)x_i - (a_2 - a_1)\sum_{i=3}^{n} x_i\right) \\
&= \frac{1}{a_2 - a_1}\left(a_2 c - a_1 c - b + a_1 c + \sum_{i=3}^{n}(a_i - a_1)x_i - \sum_{i=3}^{n}(a_2 - a_1)x_i\right) \\
&= \frac{1}{a_2 - a_1}\left(a_2 c - b - \sum_{i=3}^{n}(a_2 - a_i)x_i\right)
\end{aligned}$$

  This means that

$$a_2 c - b \geq \sum_{i=3}^{n}(a_2 - a_i)x_i$$

We obtain two inequalities, $b - a_1 c \geq \sum_{i=3}^{n}(a_i - a_1)x_i$ and $a_2 c - b \geq \sum_{i=3}^{n}(a_2 - a_i)x_i$. Recall that any variable must be allowed to take any value in the span $[0, c]$. Consider the evaluation where $x_i = c$ for some i. From the first equation, we get that $\forall j \neq i : \ x_j = 0$. We get the following system:

$$\begin{cases} a_i c - a_1 c \leq b - a_1 c \\ a_2 c - a_i c \leq a_2 c - b \end{cases} \implies \begin{cases} a_i c \leq b \\ -a_i c \leq -b \end{cases} \implies a_i = \frac{b}{c} \ .$$

Similarly, we can show that for any $i \neq 1, 2$ we have $a_i = \frac{b}{c}$. What about a_1 and a_2? From the first equation, we get that $x_3 + \ldots + x_n = c - x_1 - x_2$.

$$\begin{aligned}
a_1 x_1 + \ldots + a_n x_n &= b \\
a_1 x_1 + a_2 x_2 + \frac{b}{c}x_3 + \ldots + \frac{b}{c}x_n &= b \\
a_1 x_1 + a_2 x_2 + \frac{b}{c}(x_3 + \ldots + x_n) &= b \\
a_1 x_1 + a_2 x_2 + \frac{b}{c}(c - x_1 - x_2) &= b \\
a_1 x_1 + a_2 x_2 + b - \frac{b}{c}x_1 - \frac{b}{c}x_2 &= b \\
\left(a_1 - \frac{b}{c}\right)x_1 + \left(a_2 - \frac{b}{c}\right)x_2 &= 0
\end{aligned}$$

If either $\left(a_1 - \frac{b}{c}\right) \neq 0$ or $\left(a_1 - \frac{b}{c}\right) \neq 0$, then one variable may be expressed in terms of the other one, what means that the projection to (x_1, x_2) can be only a line. This would mean that the entire P is actually represented by a line, and as we have told before, this is a very particular case of linear programming. Therefore we need $a_1 = a_2 = \frac{b}{c}$.

We have obtained an equation $\frac{b}{c}x_1 + \ldots + \frac{b}{c}x_n = b$, and multiplying both sides by $\frac{c}{b}$ we get the same equation $x_1 + \ldots + x_n = c$.

4.5.3 Conclusions for the Indistinguishability-Based Security

If we want to protect the transformed linear program against distinguishing projections up to t coordinates for some security parameter t, then we definitely have to protect it against projections to 2 coordinates. The set of such linear programs has turned out to be very limited. We have shown that the only possible class of a transformed program is the equation $x_1 + \ldots + x_n = c$ for some $c \geq 0$ ($c = 0$ defines a single point $(0, \ldots, 0)$).

The transformation may still possible if it is not affine. In this case, the requirement that the projections to t coordinates must be the same may be unnecessary. However, it is not clear how a polyhedron could be efficiently transformed to a polyhedron with some other kind of transformation while preserving the correctness. For example, theoretically it could be a function that solves the linear program by itself and does not use the optimal solution of the transformed linear program in any way.

4.6 Weaker Security Requirements

It may seem that the indistinguishability-based security definition is too strong. It would be nice to state more precisely what kind of information about the initial program is sufficient for the adversary to conduct his evil deeds.

As we have shown in 3.2.3, revealing the locations of the slack variables may be fatal if the adversary holds at least two inequality constraints. The proposed solution was to use the augmented form of linear programming that hides the cost vector into the constraint matrix. However, the adversary may trace down the locations of the slack variables by some other means.

Our initial idea was that performing optimization in random direction may give situations where the slack variables take values 0 much more often than the non-slack variables. We have tried to verify it experimentally, and it has turned out that sometimes, this is indeed true. In order to describe our experiments in details, we have to first define the following probability distribution:

Definition 23. *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 a folded normal distribution and is denoted $\mathcal{N}_{\mathrm{f}}(\mu, \sigma^2)$.*

The outline of our experiments is the following:

1. Fix the parameters m, n that denote the size of the linear program, $p \in [0, 1) \subseteq \mathbb{R}$ that denotes the fraction of zero entries of A, and $e \in \{0, \ldots, m-1\}$ that denotes the number of equations that will be removed from the transformed system before commencing with the optimization task.

2. Generate a random point $v = (v_1, \ldots, v_n)$ where v_i is chosen uniformly from $(0, 100] \subseteq \mathbb{R}$. The idea is that in order to ensure that the polyhedron is non-empty, we generate the bounding hyperplanes in such a way that the polyhedron contains at least the point v.

3. Generate a random $m \times n$ matrix A whose entries are assigned in the following way:

   - The value 0 is taken with the probability p.

- A random value is taken uniformly from $[-100, 100] \subseteq \mathbb{R}$ with probability $1 - p$.

We have experimented with $p \in \{0, 0.25, 0.5, 0.75\}$. We have also tried the cases where all the entries are non-negative since that may correspond to a wide class of real-world linear programs, such as profit maximization task.

4. Generate the entries of 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 $b_i = a_{i1}v_1 + \ldots + a_{in}x_n + r$, where r can be any positive random value. We have chosen r uniformly from $[1000, 2000] \subseteq \mathbb{R}$ since in this case the projections of the polyhedron to 2-dimensional spaces gave nice pictures in the area $[0, 100] \times [0, 100]$.

5. Augment the $m \times m$ identity matrix to the right of A. Get the system of equations $\begin{pmatrix} \mathrm{A} & \mathrm{I} \end{pmatrix} \mathbf{x} = \mathbf{b}$.

6. Multiply $\begin{pmatrix} \mathrm{A} & \mathrm{I} \end{pmatrix}$ and $\mathbf{b}$ from the left by a random invertible $(m + n) \times (m + n)$ matrix P whose entries are sampled uniformly from $[-100, 100] \subseteq \mathbb{R}$. It does not affect the feasible region in any way, but it was interesting for our experiment since we started removing equations from the transformed program at some point.

7. Scale and permute the columns of $\mathrm{P}\begin{pmatrix} \mathrm{A} & \mathrm{I} \end{pmatrix}$. Remove the first e equations from the obtained system. Let the transformed system be denoted $\mathrm{A}'\mathbf{x} = \mathbf{b}'$.

8. Generate a cost vector $\mathbf{c}$ sampling each entry from the distribution $\mathcal{N}_{\mathrm{f}}(0, 1)$. Our experiments have shown that sampling from $\mathcal{N}_{\mathrm{f}}(0, 1)$ provides at least as good distinguishability of slack and non-slack variables as $\mathcal{N}(0, 1)$.

9. Construct a linear program "minimize $\mathbf{c}^{\mathrm{T}} \cdot \mathbf{x}$, subject to $\mathrm{A}'\mathbf{x} = \mathbf{b}', \mathbf{x} \geq \mathbf{0}$". Feed the program to a linear programming solver. Use the one that outputs basic solutions.

10. Repeat the steps 8 and 9 k times. For each variable count the frequency, in how many solutions it has been 0. We have tried it with $k = 100$. Larger value of k does not seem to give any significant difference.

We have performed our experiments with different settings. Each row of each subtable of Table 1 represents the settings for which 20 experiments have been conducted. In each experiment, we made a guess that the slack variables are exactly those that have taken the value 0 with the highest frequencies. The success rate denotes the number of outcomes for which the guess was perfectly correct.

It can be seen that 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 may be 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 less constraints than before, and the slack variables again have higher probabilities of becoming 0.

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

m	n	$A \geq 0$	e	Success Rate
5	25	True	0	10
5	25	False	0	5
15	15	True	0	10
15	15	False	0	5
25	5	True	20	14
25	5	True	15	15
25	5	True	10	11
25	5	True	5	1
25	5	True	0	0
25	5	False	20	7
25	5	False	15	14
25	5	False	10	10
25	5	False	5	1
25	5	False	0	0

$p = 0$

m	n	$A \geq 0$	e	Success Rate
5	25	True	0	9
5	25	False	0	2
15	15	True	0	10
15	15	False	0	2
25	5	True	20	12
25	5	True	15	18
25	5	True	10	13
25	5	True	5	0
25	5	True	0	0
25	5	False	20	11
25	5	False	15	11
25	5	False	10	4
25	5	False	5	5
25	5	False	0	0

$p = 0.25$

m	n	$A \geq 0$	e	Success Rate
5	25	True	0	3
5	25	False	0	0
15	15	True	0	7
15	15	False	0	2
25	5	True	20	17
25	5	True	15	14
25	5	True	10	10
25	5	True	5	0
25	5	True	0	0
25	5	False	20	11
25	5	False	15	10
25	5	False	10	9
25	5	False	5	4
25	5	False	0	0

$p = 0.5$

m	n	$A \geq 0$	e	Success Rate
5	25	True	0	0
5	25	False	0	0
15	15	True	0	2
15	15	False	0	0
25	5	True	20	14
25	5	True	15	12
25	5	True	10	4
25	5	True	5	0
25	5	True	0	0
25	5	False	20	4
25	5	False	15	3
25	5	False	10	1
25	5	False	5	0
25	5	False	0	0

$p = 0.75$

Table 1: Results of the experiments

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.

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.

The results of these experiments still do not mean that we may apply this attack to any transformed linear program. We have discovered that locating the slack variables allow to perform an attack if all the constraints are inequalities, but it is still not clear how to apply it if any equalities are present. Nevertheless, the class of secure linear programs still gets reduced. The attack is applicable at least to the brewery example and many similar profit maximization problems.

Although it may seem that there are not so many direct attacks that can be conducted nicely and clearly without additional thinking, it is still possible that there are some insecure settings that have just not been noticed yet. Using indistinguishability-based security definition would protect us against such unexpected problems. Unfortunately, it has been proven to be impossible.

5 Conclusion

In this thesis, we have given an overview of existing techniques of transformation-based privacy-preserving linear programming. Since the existing security definitions in this field are too weak, we have tried to establish a more reasonable indistinguishability-based security definition that would make the privacy independent on the way the initial data is shared, and at the same time would be pretty standard to be integrated into more complex protocols.

First of all, we have shown that achieving perfect secrecy with respect to this definition is impossible for any transformation. Hence we have concentrated on computational security. We have found the requirements that a transformation has to satisfy in order to achieve this security property. It has turned out that it is impossible for any affine transformation. It may still happen that this kind of security can be achieved by using some other kind of transformation, but there are no other known entirely correct and efficient methods yet.

Privaatsust säilitav lineaarne planeerimine

Magistritöö(30 EAP)

Alisa Pankova

Lühikokkuvõte

Rakendusmatemaatika on matemaatika osa, mis tegeleb teistes teadusharudes rakendatavate matemaatiliste mudelite ja nende uurimiseks määratud meetodite loomisega. Seega on see tihedalt seotud reaalse maailma probleemidega. Üks suur ülesannete klass, mida kasutatakse päris paljudes valdkondades, on optimeerimisülesanded. Rakendusmatemaatika pakub erinevaid meetodeid nende probleemide lahendamiseks.

Optimeerimisülesandeid on sageli tarvis lahendada mitme sõltumatu asutuse andmete põhjal. On aga täiesti võimalik, et need asutused ei taha avaldada oma isiklikke andmeid või see lihtsalt ei ole lubatud seaduse järgi. Siis on vaja, et optimeerimise käigus ei lekiks mitte ühtegi bitti tundlikkest andmetest. See on koht, kus tasub mõelda krüptograafiliste meetodite kasutamise peale.

On tõestatud, et suvalist funktsiooni saab arvutada turvaliselt, ilma sisendandmete lekitamiseta. Küsimus on selles, kuidas seda teha efektiivselt. Kitsendades matemaatiliste ülesannete hulka väiksematele klassidele on võimalik leida lihtsamaid ja efektiivsemaid meetodeid, mis sobivad hästi just nende probleemide jaoks. Käesolevas töös vaadeldakse lineaarse planeerimise optimeerimisülesandeid.

Selles valdkonnas on juba tehtud palju tööd. Viimastes töödes käsitletakse muuhulgas ka transformatsioonipõhise privaatsust säilitava lineaarse planeerimise meetodeid. Olemasolevad turvadefinitsioonid on aga väga nõrgad. Selles töös on toodud uus eristamatusel põhinev definitsioon, mis tundub piisavalt standardne, et sellele vastava transformatsiooni saaks pärast keerulisema protokolli sisse integereerida. Lisaks on toodud mõned konkreetsed uued ründed olemasolevate skeemide vastu, mis näitavad, et kuigi käesolevas töös toodud turvadefinitsioon võib tunduda liiga tugev, on see tegelikult väga mõistlik.

Käesolevas töös on kõigepealt näidatud, et täielik turvalisus on sellise definitsiooniga võimatu. Edaspidi on uuritud, mis tingimustel on võimalik saavutada arvutuslikku turvalisust. On avastatud mõned tingimused, mida transformatsioon peab sellisel juhul rahuldama. Siis on tõestatud, et affinse transformationi korral on nende tingimuste rahuldamine võimatu. On võimalik, et sellist turvadefinitsiooni on endiselt võimalik saavutada kasutades mingit muud transformatsiooni, kuid muud efektiivset ja korrektset transformatsiooni hetkel teada ei ole.

References

[1] Mikhail J. Atallah and Keith B. Frikken. Securely outsourcing linear algebra computations. In Dengguo Feng, David A. Basin, and Peng Liu, editors, *ASIACCS*, pages 48–59. ACM, 2010.

[2] Donald Beaver, Silvio Micali, and Phillip Rogaway. The Round Complexity of Secure Protocols (Extended Abstract). In Harriet Ortiz, editor, *STOC*, pages 503–513. ACM, 1990.

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

[4] 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.

[5] Robert G. Bland. The allocation of resources by linear programming. *Scientific American*, 244(6):108–119, June 1981.

[6] Dan Bogdanov, Roberto Guanciale, Liina Kamm, Peeter Laud, Riivo Talviste, and Jan Willemson. Advances in SMC techniques, January 2013. UaESMC Deliverable 2.2.1.

[7] Benny Chor and Eyal Kushilevitz. Secret sharing over infinite domains. *J. of Cryptology*, 6:87–96, 1989.

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

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

[10] 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.

[11] Charles M. Grinstead and J. Laurie Snell. *Introduction to probability*, chapter 5.2 "Important Densities". 2nd ed. American Mathematical Society, 1997.

[12] Branko Grünbaum. *Convex polytopes*. Graduate texts in mathematics. Springer, New York, Berlin, London, 2003.

[13] Shuguo Han and Wee Keong Ng. Privacy-preserving linear fisher discriminant analysis. In *Proceedings of the 12th Pacific-Asia conference on Advances in knowledge discovery and data mining*, PAKDD'08, pages 136–147, Berlin, Heidelberg, 2008. Springer-Verlag.

[14] Fumio Hayashi. *Econometrics*, chapter 1.2 " The Algebra of Least Squares". Princeton University Press, 2011.

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

[16] Colin N. Jones, Eric C. Kerrigan, and Jan M. Maciejowski. Equality set projection: A new algorithm for the projection of polytopes in halfspace representation. Technical Report CUED/F-INFENG/TR.463, Department of Engineering, University of Cambridge, 2004.

[17] William Kocay and Donald L. Kreher. *Graphs, Algorithms and Optimization*, chapter 15.5 "The dual of the shortest-path problem". Chapman & Hall/CRC, 2004.

[18] Arjen Lenstra, Hendrik Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515–534, 1982.

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

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

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

[22] Robert J. McEliece. A public-key cryptosystem based on algebraic coding theory. Technical report, Jet Propulsion Lab Deep Space Network Progress report, 1978.

[23] 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.

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

[25] 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.

[26] Pradeep Chathuranga Weeraddana, George Athanasiou, Martin Jakobsson, Carlo Fischione, and John S. Barras. Per-se privacy preserving distributed optimization. *CoRR*, abs/1210.3283, 2012.

[27] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.

Non-exclusive licence to reproduce thesis and make thesis public

I, Alisa Pankova (date of birth: 20.10.1989),

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

   1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

   1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

   "Insecurity of Transformation-Based Privacy-Preserving Linear Programming",

   supervised by Peeter Laud and Margus Niitsoo,

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, **20.05.2013**