Practical Secure and Efficient Multiparty Linear Programming Based on Problem Transformation

Jannik Dreier
VERIMAG, Grenoble, France
jannik.dreier@imag.fr

Florian Kerschbaum
SAP Research, Karlsruhe, Germany
florian.kerschbaum@sap.com

March 4, 2011

Abstract

Cryptographic solutions to privacy-preserving multiparty linear programming are slow. This makes them unsuitable for many economically important applications, such as supply chain optimization, whose size exceeds their practically feasible input range. In this paper we present a privacy-preserving transformation that allows secure outsourcing of the linear program computation in an efficient manner. We evaluate security by quantifying the leakage about the input after the transformation and present implementation results. Using this transformation, we can mostly replace the costly cryptographic operations and securely solve problems several orders of magnitude larger.

1 Introduction

Linear Programming (LP) can be used to solve many practical optimization problems, e.g. supply chain master planning [22]. Many practical problems are distributed and require protection of the input data. For example, in supply chain master planning, the participating companies need to exchange information about production costs and capacities. This is very sensitive data, since it directly impacts the negotiation position. Consequently no master planning solution will be adopted in practice that reveals this data [17].

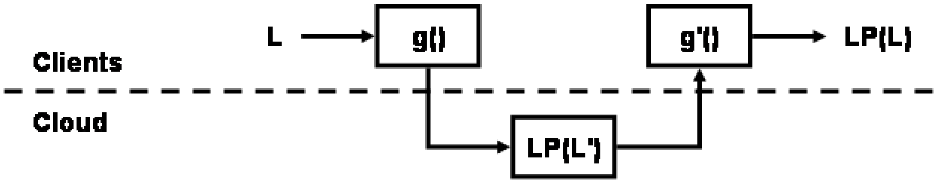

Figure 1: Privacy-Preserving Linear Programming in the Cloud

Secure multi-party computation (SMC) [4, 10, 18, 26] offers a cryptographic solution to the problem. Several parties can jointly compute a function, e.g. LP, without disclosing anything except what can be inferred by a party's input and output. In theory this offers an ideal solution to the conflict posed by these problems. There even exist a number of specialised protocols for secure LP [7, 14, 23].

Nevertheless these solutions suffer from a prohibitively bad computational performance. We estimate that our prototypical implementation of Toft's protocol [23] requires 7 years in order to solve our use case LP problem with 282 variables. Supply chain planning problems can easily reach 4 million variables [6] and these are only single company planning problems while we consider cross-organizational optimization. These size of problems are currently solved every day by non-secure LP solvers. Assuming an average computation complexity of $O(n^3)$ for LP and that Moore's Law continues to hold for the time being, it would still take roughly 80 years until these problems could be solved securely as fast as they can be solved non-securely today. Clearly, if we want to solve these problems today, we need a different approach.

In [24] Vaidya presents a different approach to the problem. Instead of implementing a LP solver using SMC the problem is randomly transformed and then the problem is solved using a non-secure LP solver. Unfortunately as Bednarz already points out in [3] Vaidya's transformation is incorrect and may lead to solutions which are not admitted in the original problem. In this paper we present a new, correct transformation. Furthermore we do not settle for an informal assessment of its security, but instead evaluate it in the framework of leakage quantification. Finally, we present performance results from our implementation.

Let L be an instance of a LP problem where parties X_i have input x_i, e.g. variables, constraints or costs. The approach of [7, 14, 23] is to implement a LP solver for L using SMC where x_i is the input of each party. Let g be our privacy-preserving transformation, then $L' = g(L)$ is another instance of a LP problem, but L' reveals little information about L. We can outsource the solution of L' to an untrusted service provider, e.g. the cloud. Either a single party computes g on its data, e.g. a single company instance of supply chain planning, or we can securely compute g using SMC for multiple parties. We show that computing g is much more efficient than implementing a LP solver and even present an optimized secure computation protocol for it. The solution computed by the cloud on L' then needs to be transformed by g' to the solution of L. Figure 1 depicts our approach to privacy-preserving linear programming in the cloud.

In summary this paper contributes

- a novel *privacy-preserving transformation* for linear programs regardless of their partitioning.

- a proof that differently from previous work the transformation is *correct*, i.e. an optimal solution to the transformed problem always corresponds to an optimal solution of the original problem.

- an analysis that the transformation is *secure* in the framework of leakage quantification. In our use case example the chances of guessing e.g. the optimal values x are less than $2.30 \cdot 10^{-1409}$.

- a *secure computation protocol* to efficiently compute the transformation in a multiparty case.

- a theoretical analysis and performance results of our implementation that show that the transformation is *efficient*. In our use case example the solution only required 25 minutes (compared to 7 years for Toft's protocol).

The remainder of this paper is structured as follows. In the next Section we will review related work. In Section 3 we present our main result, the privacy-preserving transformation. We will prove the correctness of the transformation in Section 4 and analyze its performance in Section 5. In Section 6 we summarize our results from leakage quantification of this transformation. We show the protocol for securely computing the transformation in Section 7. The results from our use case example are described in Section 8. We conclude the paper in Section 9.

2 Related Work

Our work is related to secure multi-party LP solvers [7, 14, 23], secure outsourcing [1, 2, 3, 9, 15, 16, 24] and leakage quantification [5, 21, 25].

2.1 Secure Multi-Party LP Solvers

As mentioned above, different distributed LP solvers based on secure multiparty computation have been developed. Toft [23] used a distributed secure Simplex algorithm based on secret sharing. This is usable for any number of participants and provides information-theoretic security, but turned out to be too slow in practice for realistically sized problems in many applications such as Supply Chain Management.

Li and Atallah [14] also used a distributed Simplex algorithm. In contrary to Toft [23] they concentrated on the case where the data is shared among only two parties. It is unclear if their algorithm can be efficiently extended to the multi-party case.

Catrina and de Hoogh [7] developed a more efficient version of the distributed Simplex algorithm using fixed-point arithmetics. However, their solution still is only suitable for relatively small inputs as it suffers from a higher asymptotic and worst-case complexity than our transformation.

2.2 Secure Outsourcing

Vaidya [24] used a transformation approach which is the basis for our transformation. He supposed a scenario where one party holds the objective function and the other party the constraints. As a consequence his transformation does not protect the entire Linear Program, which is not appropriate for most applications. Additionally, he did not provide a formal security analysis and his work suffers from correctness problems [3].

Mangasarian [15, 16] proposed similar transformations for special input distributions (horizontal or vertical partitioning). Supply chain optimization problems have a more complex distribution which combines both cases. Additionally, his approach for horizontal partitioning is limited to equality constraints. This is not sufficient for supply chain optimization either, as we have to deal with capacity constraints expressed as inequality constraints. Our transformation addresses both shortcomings and is suitable for more general data distributions. Moreover he does not provide a formal security analysis.

Furthermore there is work on related problems such as matrix multiplication and linear systems of equations [1, 2, 9].

2.3 Leakage Quantification

To give a formal security assessment of our transformation, we use methods of Leakage Quantification which measure how much information about the secret input is revealed to an attacker. In particular, we concentrate on the metric of “Multiplicative Advantage” which was first developed (but not called so) by Smith [21] based on entropy loss using a special type of min-entropy. Later on this was analyzed more closely by Braun et al. [5] who proposed another, simpler definition - the one we use in this paper. Compared to other metrics such as entropy loss or channel capacity using the standard Shannon-entropy, it is particularly expressive for one-try attacks as it gives a worst-case evaluation. To assess complex operations we make use of some theorems by Wibmer et al. [25] on the leakage of combined channels (see Appendix A).

3 The Transformation

Linear Programming is a standard tool in business optimization. A Linear Program (LP) consists of a set of unknown variables x, a linear target function $c(x)$ representing the costs which shall be minimized (or equivalently the gain which has to be maximized) and a set of constraints (linear equalities or inequalities). The standard form is

$$\begin{array}{rcc} \min & c^T x & \\ \text{s.t. } Mx & \leq & B \\ x & \geq & 0 \end{array} \tag{1}$$

As we will eventually transform all inequalities into equalities and as in our use case many of the input constraints are actually equalities, we will treat them separately. This reduces the size of the problem as we need less slack-variables[1] and thus increases performance as well as security. So let

[1] A slack-variable is used to express inequality constraints as equality constraints. The idea is to introduce an additional variable for each constraint which takes up the “remainder” or “slack”. For example, instead of $3x_1 \leq 10$ we can write $3x_1 + s_1 = 10$ for $s_1 \geq 0$.

the input problem - without loss of generality - be

$$\begin{array}{rcl} \min & c^T x & \\ \text{s.t. } M_1 x & = & b_1 \\ M_2 x & \leq & b_2 \\ x & \geq & 0 \end{array} \tag{2}$$

We use a positive monomial matrix[2] Q to hide c (as proposed by [24, 3]):

$$\begin{array}{rcl} \min c^T Q Q^{-1} x & & \\ M_1 Q Q^{-1} x & = & b_1 \\ M_2 Q Q^{-1} x & \leq & b_2 \\ Q^{-1} x & \geq & 0 \end{array} \tag{3}$$

and a positive vector r to hide x

$$\begin{array}{rcl} \min c^T Q(Q^{-1} x + r) & & \\ M_1 Q(Q^{-1} x + r) & = & b_1 + M_1 Q r \\ M_2 Q(Q^{-1} x + r) & \leq & b_2 + M_2 Q r \\ (Q^{-1} x + r) & \geq & r \end{array} \tag{4}$$

For $z = Q^{-1}x + r$ and a strictly positive diagonal matrix S[3] we have

$$\begin{array}{rcl} \min c^T Q z & & \\ M_1 Q z & = & b_1 + M_1 Q r \\ M_2 Q z & \leq & b_2 + M_2 Q r \\ S z & \geq & S r \end{array} \tag{5}$$

Then we define $c'^T = c^T Q$,

$$M' = \left(\begin{array}{cc} M_1 Q & 0 \\ M_2 Q & \\ -S & \text{A} \end{array} \right), b' = \left(\begin{array}{c} b_1 + M_1 Q r \\ b_2 + M_2 Q r \\ -S r \end{array} \right) \tag{6}$$

where A is a permutation matrix[4] representing slack-variables. This allows us to rewrite the program as follows:

$$\begin{array}{rcl} \min & c_s'^T z_s & \\ \text{s.t. } M' z_s & = & b' \\ z_s & \geq & 0 \end{array} \tag{7}$$

[2] A monomial matrix contains exactly one non-zero entry per row and column.

[3] A diagonal matrix where the entries on the main diagonal $A_{i,i}$ are strictly positive.

[4] A permutation matrix is a monomial matrix where the non-zero entries are "1".

where c'_s is c' with added zeros for the slack-variables and z_s is the variable vector (z with added slack-variables). To hide the contents of M' and b' we use a nonsingular matrix[5] P and with $M'' = P * M'$ and $b'' = P' * b'$ we have

$$\begin{array}{rcl} \min & c_s'^T z & \\ \text{s.t. } M'' z_s & = & b'' \\ z_s & \geq & 0 \end{array} \tag{8}$$

As $z = Q^{-1}x + r$, the resulting x can be obtained from z by calculating $x = Q(z - r)$.

4 Correctness

For the transformation to be useful in practice, we have to ensure that the transformed program can still be used to find a solution to the original problem. We will prove that any optimal (i.e. with minimal cost) and feasible (with respect to the constraints) solution to the transformed problem corresponds to an optimal and feasible solution of the original problem using the following two lemmas. More precisely, Lemma 1 gives the equivalence of the solution space of the original and the transformed problem, i.e. the fact that each feasible solution in the original problem corresponds to a feasible solution in the transformed problem and vice versa. Lemma 2 states that back-transforming an optimal solution of the transformed LP yields an optimal solution to the original LP.

Lemma 1. *A solution x is feasible in the original problem if and only if $z = Q^{-1}x + r$ is a feasible solution to the transformed problem.*

Proof. This is true by construction. The multiplication by P does not change the solution set as P is invertible. Q is monomial (which gives correctness for this part of the transformation as shown by Bednarz et al. [3]) and r is positive to not interfere with $z \geq 0$. As S is a strictly positive diagonal matrix, the multiplication by S is actually a multiplication of each row with a positive scalar which leaves the inequalities untouched. □

Lemma 2. *Let z be the solution that minimizes $c'^T z$ in the transformed LP. Then $x = Q(z - r)$ minimizes $c^T x$ in the original problem.*

[5] A $(n \times n)$-matrix A is called nonsingular if there exists another $(n \times n)$-matrix B so that $AB = BA = I$ where I is the identity matrix of dimension $(n \times n)$.

Proof. We will use a proof by contradiction. Suppose that there is a solution x' with $c^T x' < c^T x$. Then

$$\begin{array}{rrcl} & c^T x' & < & c^T Q(z-r) \\ \Leftrightarrow & c^T x' & < & c^T Qz - c^T Qr \\ \Leftrightarrow & c^T QQ^{-1}x' + c^T Qr & < & c^T Qz \\ \Leftrightarrow & c'^T (Q^{-1}x' + r) & < & c'^T z \end{array} \tag{9}$$

Apparently $z' = Q^{-1}x' + r$ is a valid solution for transformed LP with strictly lower cost than z. Hence z is not optimal. ↯ □

5 Performance

Performance is a key aspect: The transformation has to be significantly less complex than solving the problem itself, otherwise outsourcing would not represent an efficiency gain to the clients.

From a theoretical viewpoint this is easy to see, as the most expensive operation during the transformation is the multiplication by P. Let m_1 and m_2 denote the number of rows of M_1 and M_2 respectively, and n the number of variables. Then the multiplication using a naive algorithm is in $\mathcal{O}((m_1 + m_2 + n)^2 \times (2n + m_2))$. This is more efficient than the simplex algorithm (on which the secure LP solvers [7, 14, 23] are built on) with exponential worst case complexity [13]. Even in the average case or compared to interior point methods performance will be better due to smaller constants.

We will discuss practical performance in more detail our use case example in Section 8.

6 Security

We will now analyze the security of our transformation in the following setting: An attacker wants to obtain the input data (the original LP). He knows the transformed LP and some abstract facts about the input, for example that the input values are within a certain range (this is described more precisely later on).

Ideally we would like to give a classical security proof. However, as our transformation is based on disguising using random noise, the classical cryptological security definitions are unsuitable as our transformation will probably leak some information since the transformed LP is somewhat linked to the input and thus not entirely random. Nevertheless this can be acceptable for many applications as long as the leakage is small and bounded, in particular as we get dramatic performance improvements in return. To formally evaluate this leakage and to show that it is “small and bounded”, we use Leakage Quantification methods based on information theory.

We start by defining the metric “Multiplicative Advantage”. Then we analyze the leakage of several elementary operations such as permutations and scalar as well as matrix multiplication. By combining these results we establish bounds on the leakage of different parts of the transformation. In the last part we discuss possible attacks.

6.1 Multiplicative Advantage

The disguising transformation is modeled as a communication channel as known from information theory (for an introduction to information theory see e.g. Cover and Thomas [8]).

Definition 1 (Channel)**.** *A discrete, noisy and memoryless channel $\mathcal{C}$ is given by*

- *A finite set $\mathcal{X} = x_1, ..., x_n$ called the input alphabet,*
- *a finite set $\mathcal{Y} = y_1, ..., y_n$ called the output alphabet,*
- *for each $x \in \mathcal{X}$ a random variable $C|x$ that takes values in $\mathcal{Y}$.*

The input $x \in \mathcal{X}$ represents the data to hide, the output $y \in \mathcal{Y}$ is the ”encrypted” or ”disguised” data that is passed into the cloud. The definition of the input alphabet $\mathcal{X}$ is important as it reflects the attackers “abstract knowledge” about the input, for example that the input is a value within a certain range.

Braun et al. [5] defined the notion of “advantage” based on a similar notion (“vulnerability”) by Smith [21]. For a random variable X characterizing the input distribution let

$$PR_{priori}(X) = \max_i p(X = x_i) \tag{10}$$

denote the *a priori* probability of a right guess (i.e. the probability of an attacker correctly guessing the input if it has not yet seen the output of the channel). Similarly let

$$PR_{posteriori}(X) = \sum_j \max_i (p(y_j|x_i)p(X = x_i)) \tag{11}$$

denote the *a posteriori* probability of a right guess (i.e. the probability of an attacker correctly guessing the input after having seen the output of the channel). Then we define

$$L'(X) = \frac{PR_{posteriori}(X)}{PR_{priori}(X)} \tag{12}$$

the factor by which the knowledge of the output of the channel increases the chances of a right guess for a given X[6].

Definition 2 (Multiplicative Advantage)**.** *The* Multiplicative Advantage *or* Multiplicative Leakage *of a channel $\mathcal{C}$ is defined as*

$$L(\mathcal{C}) = \max_X L'(X) \tag{13}$$

$L(\mathcal{C})$ is independent of the input distribution, which is convenient if this distribution is not known. Additionally $L'(X)$ has the advantage of attaining its maximum for a uniform input distribution [5, 25] which yields

$$L(\mathcal{C}) = \sum_j \max_i p(y_j|x_i) \tag{14}$$

This result is very useful in practice as $p(y_j|x_i)$ are the entries of the channel matrix. Furthermore Wibmer et al. [25] showed that multiplicative leakage can be used to easily bound the leakage of compositions of independent channels which simplifies the analysis of complex protocols. A list of different channel compositions and their leakage can be found in Appendix A.

6.2 Building Blocks

We will now analyze the three main building blocks of our transformation: Permutations, scalar multiplications and matrix multiplications.

6.2.1 Permutation

To determine the leakage of a permutation, we model the "permutation channel" $\mathcal{P}_k^n$ as follows: The channel input and output are vectors with k elements from 0 to n each. More formally, let $\mathcal{X} = \mathcal{Y} = \{x|x = (x_1, ..., x_k)\}$ where $x_i \in 0, ..., n$.[7]

[6] Braun et al. [5] call this value the Multiplicative Leakage. The interesting point is the maximum of this value over all X, which we will call Multiplicative Leakage.

[7] Here we assume only positive values, but the result remains correct for negative values too. In fact, it works for any $n + 1$ distinguishable symbols.

Theorem 1 (Leakage of a Permutation)**.** *The multiplicative leakage of a uniformly distributed permutation of a vector with k elements ranging from 0 to n is given by*

$$L(\mathcal{P}_k^n) = \frac{(n+k)!}{k! * n!} \tag{15}$$

Proof. Idea: Partition all possible output vectors $\mathcal{Y}$ into classes of vectors $\mathcal{Y}_i$ that are permutations of each other. If each permutation is equally likely, the channel matrix contains - for each class - $|\mathcal{Y}_i|$ columns with a maximum of $\frac{1}{|\mathcal{Y}_i|}$ each. Hence the multiplicative leakage of a permutation is equal to the number of different classes. Each class can be characterized by the number of occurrences of each value in vector y. Counting these possibilities yields the result. □

An intuitive interpretation of this result is that a permutation leaks the "class" to which a vector belongs, but nothing more since each vector inside this class is equally likely.

6.2.2 Scalar Multiplication

We use the following model: The channel inputs are values $x \in \{0, ..., n\}$, the output is a value $y \in \{0, ..., n^{k+1}\}$ where k is the number of factors of f (i.e. $y = x * f = x * \prod_{i=1}^{k} f_i$ where $f_i \in \{1, ..., n\}$). Let f (not the f_i-s) be uniformly distributed among all the possible values[8] and $\mathcal{M}_n^k$ denote the channel for parameters n and k.

Theorem 2 (Leakage of a Multiplication)**.** *The leakage of a Multiplication channel $\mathcal{M}_n^k$ is bounded by*

$$L(\mathcal{M}_n^k) \leq \frac{n+k}{k+1} + 1 \tag{16}$$

Proof. Idea: Let $A(n,k)$ denote the number of different values of $f = \prod_{i=1}^{k} f_i$ with $f_i \in 1..n$. Bearing in mind that input value 0 is always completely leaked (which adds 1 to the leakage) we have

$$L(\mathcal{M}_n^k) = \frac{A(n,k+1)}{A(n,k)} + 1 \tag{17}$$

[8] This strangely looking choice reduces the leakage and allows us to give a simple closed-form bound. It can be difficult to implement in practice; a simpler alternative would be to choose the f_i-s uniformly which slightly increases the leakage.

$A(n,k)$ is bounded by the number of different combinations of the f_i when only distinguishing number of occurrences of each value in $1,..,n$ (and ignoring their order):

$$A(n,k) \leq \left(\begin{array}{c} n-1+k \\ k \end{array} \right) = \frac{(n+k-1)!}{k! * (n-1)!} \tag{18}$$

Although this is only an upper bound, the quotient

$$\frac{(n+(k+1)-1)!}{(k+1)! * (n-1)!} / \frac{(n+k-1)!}{k! * (n-1)!} = \frac{n+k}{k+1}$$

is still a valid bound for $\frac{A(n,k+1)}{A(n,k)}$ as the "errors" in the denominator cancel out with the "errors" in the nominator. □

6.2.3 Matrix Multiplication

We define a matrix multiplication channel $\mathscr{M}_{k,n,l}$ as follows. Channel inputs are $k \times l$-matrices ($k \leq l$) X with rank k and elements from 0 to n, the output $Y = P * X$ is the result of a multiplication by a randomly chosen invertible square matrix P of dimension $k \times k$ with elements from 0 to n.

Similarly we define another matrix multiplication channel $\mathscr{M}'_{k,n,l}$ as follows. Channel inputs are regular square $k \times k$-matrices X with elements from 0 to n, the output $Y = X * M$ is the result of a multiplication by a randomly chosen matrix M of dimension $k \times l$ ($k \leq l$) with rank k and elements from 0 to n.[9]

To minimize the leakage, we choose P (and M respectively) uniformly among all possible matrices of the correct size and values. Figures 2 and 3 show the resulting multiplicative leakage for different values of k, n and l.

The leakage grows with n and l for a fixed k. To obtain more meaningful values we calculated the resulting a-posteriori probability of a right guess for a uniform input distribution using the following equation

$$PR_{posteriori}(X) \leq L(\mathscr{M}_n^k) * PR_{priori}(X) \tag{19}$$

Note that these values coincide for $\mathscr{M}_{k,n,l}$ and $\mathscr{M}'_{k,n,l}$. Results are shown in Figure 4. The a-posteriori probability appears to converge to a constant value for each k but drops very quickly for growing k.

[9] This corresponds to $\mathscr{M}_{k,n,l}$ with inverted roles: P is now the input to hide and M the randomness. We use this channel in the security analysis of P later on.

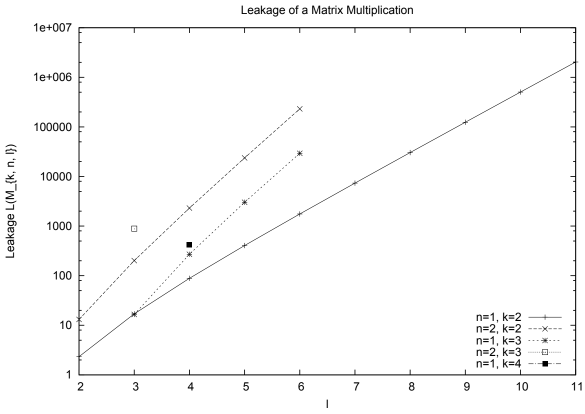

Figure 2: Leakage of a matrix multiplication channel $\mathscr{M}_{k,n,l}$

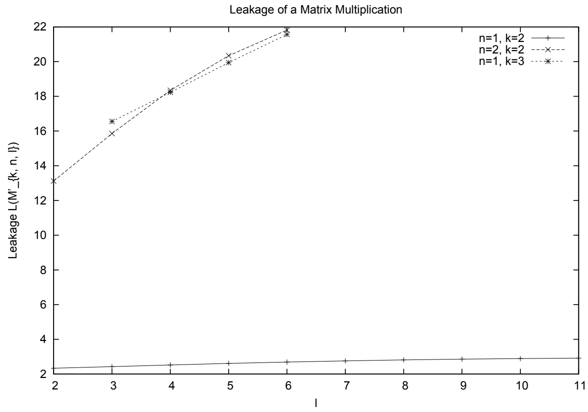

Figure 3: Leakage of a matrix multiplication channel $\mathscr{M}'_{k,n,l}$

6.3 The Complete Protocol

After having analyzed the leakage of the basic operations, we will now establish bounds on the leakage of the three main parts of the transformation.

6.3.1 Leakage of the cost function c

In our transformation c is hidden by a monomial matrix Q, i.e. a permutation and a multiplication of each entry with a scalar value. Supposing independent channels and applying Theorems 4 and 5 from Appendix A

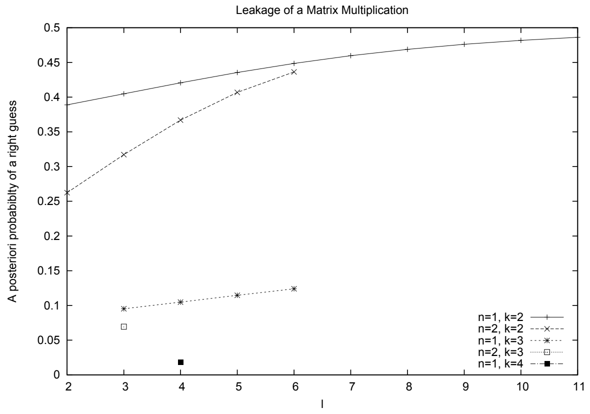

Figure 4: A posteriori probability of a right guess for a matrix multiplication channel

[25] we can conclude that

$$\begin{aligned} L(\mathfrak{c}^l_{n,k}) &\leq \min\left(\left(L(\mathcal{M}^l_n)\right)^k, L(\mathcal{P}^{A(n,l)}_k)\right) \\ &\leq \min\left(\left(\tfrac{n+l}{l+1}+1\right)^k, \tfrac{(A(n,l)+k)!}{k! * (A(n,l))!}\right) \end{aligned} \tag{20}$$

where $\mathfrak{c}^l_{n,k}$ denotes the channel for a k-dimensional input vector c hidden by a multiplication by a monomial matrix and $A(n,l)$ as defined above. Note that this is the leakage quantification of c only, although Q appears in other places in the transformation as well (in the constraint matrix, on the right hand side and in z). However, this simplification is realistic as the other occurrences are well-hidden (cf. Section 6.4.2).

6.3.2 Leakage of the variables x

The variable vector z_s consists of two parts: The non-slack variables z and the slack-variables. The non-slack variables contain values $z = Q^{-1}x + r$.

Wibmer et al. [25] showed that for a channel $y = rx + r'$ with $r' < r \leq r_m$ and $x \leq x_m$ the leakage is bounded between

$$\begin{aligned} \frac{(r_m - 1)\log(x_m+2)}{r_m+1} &\leq L(\text{"}y = rx + r'\text{"}) \\ &\leq \frac{2\left(x_m + 1 + (r_m-1)\log\left(\frac{r_m * (x_m+2) - 1}{r_m - 1}\right)\right)}{r_m+1} \end{aligned} \tag{21}$$

or for $r_m \to +\infty$

$$\log(x_m + 2) \leq L(\text{"}y = rx + r'\text{"}) \leq 2\log(x_m + 2) \tag{22}$$

The channel "$z = Q^{-1}x + r$" can be modeled as a permutation followed by the "$y = rx + r'$"-channel as Q resp. Q^{-1} is a monomial matrix and thus we have

$$L(\text{"}z = Q^{-1}x + r\text{"}) \leq \min\left(2\log(x_m + 2), \frac{(x_m + k)!}{k! * x_m!}\right) \tag{23}$$

As mentioned before, the matrix Q is also used in the hiding of c. This is not a problem, since the vector r prevents the search for common factors (cf. Section 6.4.2).

The slack variables contain potentially correlated values as well ($S * Q^{-1}x$), but since they are permuted it is hidden which variable belongs to which constraint (cf. Section 6.4.2). This further complicates exploiting correlation by common factors.

6.3.3 Leakage of the Constraints (M_1, M_2, b_1 and b_2)

The leakage of M_1, M_2, b_1 and b_2 is difficult to estimate due to the number and complexity of related transformation steps.

On the one hand, the following analysis shows that breaking the hiding with P almost completely reveals all other hiding operations: If P is known or can be guessed by the attacker, he can undo nearly the whole transformation (cf. Figure 5). First of all he can obtain S, which allows him to calculate r. This gives access to b_1 and b_2. As Q is a monomial matrix, each row of $M_{\{1,2\}}Q$ is multiplied with the same factor which he can guess with very high probability when searching for common factors (see Section 6.4.2 for details). This then leaks a permutation of M_1, M_2 and c, which may be a complete break if the attacker knows something about the structure of M_1 or M_2 which allows him to undo the permutation. This shows that the security of P is crucial for the security of the whole protocol. We will analyze attacks on P in more detail in Section 6.4.1. Thus we can say that the constraints are *at most* as safe as P.

On the other hand they are unlikely less safe than P, since conversely the knowledge of the constraints noticeably increases the chances of guessing P.

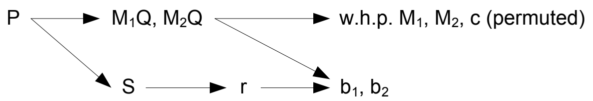

Figure 5: An overview of a possible attack

We will now give an estimation of the leakage of P. To simplify the analysis, we use the following channel model: The input alphabet $\mathcal{X}$ are invertible square matrices of dimension $(m_1 + m_2 + k) \times (m_1 + m_2 + k)$ (where m_1 and m_2 are the number of rows of M_1 and M_2; k is the number of variables). The channel output is

$$P * B = P * \left(\text{B'} \quad \begin{matrix} 0 \\ A \end{matrix} \right) \tag{24}$$

where A is a permutation matrix of dimension $(m_2 + k) \times (m_2 + k)$ and B' a random matrix of dimension $(m_1 + m_2 + k) \times (k + 1)$. B' represents the parts of M' which contain $M_1 Q$, $M_2 Q$ and $-S$ as well as b' (which can be seen as just another column of the matrix). The general idea is that A is responsible for most of the leaked information about P and that attackers have very limited knowledge about the other parts of the matrix, so that modeling them as random appears plausible.

This model allows us to benefit from Definition 6 and Theorem 6 form Appendix A. We split the channel $\mathcal{C}_P$ into a first channel $\mathcal{C}_{P,1}$ which calculates

$$P * B' \tag{25}$$

and a second channel $\mathcal{C}_{P,2}$ which calculates

$$P * \begin{pmatrix} 0 \\ A \end{pmatrix} \tag{26}$$

Then Theorem 6 from Appendix A (by Wibmer et al. [25]) gives that

$$\begin{aligned} \max\{L(C_{P,1}), L(C_{P,2})\} \leq L(C_P) \\ \leq L(C_{P,1}) L(C_{P,2}) \end{aligned} \tag{27}$$

$\mathcal{C}_{P,1}$ corresponds to the matrix multiplication channel $\mathscr{M}'_{k,n,l}$ analyzed above. The leakage of $\mathcal{C}_{P,2}$ can be bounded using the following theorem.

Theorem 3 (Leakage of Slack-Variables). *Let $\mathcal{A}_{n,k,l}$ denote the channel with invertible square matrices of dimension $(k + l) \times (k + l)$ with values between*

0 and n as input X and a matrix Y of dimensions $(k+l) \times k$ *as output so that*

$$Y = X * \begin{pmatrix} 0 \\ A \end{pmatrix} \tag{28}$$

where A is a permutation matrix of dimensions $k \times k$ *and 0 a* $l \times k$ *matrix containing zeros* $(0 \le l \le k)$*. Then*

$$\frac{\prod_{i=0}^{k-1}((n+1)^{k+l} - (n+1)^i)}{k!} \le L(\mathcal{A}_{n,k,l}) \le \frac{\prod_{i=1}^{k}((n+1)^{k+l} - i)}{k!} \tag{29}$$

Proof. Idea: The resulting matrix Y contains k vectors with $k+l$ entries each. By following the same reasoning as in Theorem 1 we can show that the leakage is equal to the number of permutation classes. In this case each permutation class corresponds to a set of k linearly independent vectors of dimension $k+l$ as X is invertible. Counting the number of possible sets yields the bounds. □

6.4 Cryptanalysis

In this Section we will discuss several attacks on P and Q, the most important parts of the transformation.

6.4.1 Attacks on P

Attacks on P will probably be based on the structure of the matrices as this is the most promising approach. The biggest issue are the slack-variables as the following example illustrates.

Consider these constraints:

$$\begin{array}{rcl} M * x & \le & b \\ x & \ge & 0 \end{array} \tag{30}$$

When adding slack-variables we obtain (I is the identity matrix)

$$\begin{array}{rcl} \begin{pmatrix} M & I \end{pmatrix} * x' & = & b \\ x' & \ge & 0 \end{array} \tag{31}$$

If we multiply by P the result is

$$\begin{array}{rcl} \begin{pmatrix} P * M & P \end{pmatrix} * x' & = & P * b \\ x' & \ge & 0 \end{array} \tag{32}$$

This means that the slack-variables completely reveal P. Unfortunately the most general type of matrix we can use instead of the identity matrix I is a monomial matrix A as A and A^{-1} have to be positive [11]. Yet this is only partly helpful as the knowledge of $P * A$ allows an attacker to calculate

$$\begin{aligned} M' &= (P * A)^{-1} * (P * M) \\ &= A^{-1} * P^{-1} * P * M \\ &= A^{-1} * M \end{aligned} \tag{33}$$

As A is a monomial matrix, $A^{-1} * M$ is a permutation of M where each column is multiplied with the same value - which can be obtained with very high probability when searching for common factors.

Not transforming the inequalities into equalities is not an option, as this would reduce our choice of P to monomial matrices for correctness reasons. This would turn $P * M$ into an easy target for factorization attacks and thus not provide enough security.

To solve the problem, we treat equality constraints separately. This reduces the size of A, as we need less slack-variables. If A is smaller than P, it only leaks a permutation of some columns of P and not the entire matrix. However, this requires that in the input problem some of the constraints are equations, which is not always the case. Yet in our target application Supply Chain Management most of the constraints are actually equations, which turns this into a very well-suited solution.

If the input problem contains only inequality constraints, we can still improve security by assigning non-zero costs to slack-variables and permuting the variables to hide them among the real variables. A possible way to do this without changing the optimal solution is by analyzing the dual problem [19].

6.4.2 Attacks on Q

Q is a positive monomial matrix, i.e. it can be written as $Q = D * E$ where D is a diagonal matrix and E a permutation matrix (i.e. a matrix with exactly one entry equal to 1 per row and column). Thus $Q^{-1} = (D * E)^{-1} = E^{-1} * D^{-1} = E^T * D^{-1}$ where

$$D = \begin{pmatrix} D_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & D_n \end{pmatrix}, \; D^{-1} = \begin{pmatrix} \frac{1}{D_1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{1}{D_n} \end{pmatrix}$$

Q appears in several places in the transformation. This can be an issue if the resulting products of Q and Q^{-1} with M_1, M_2, c etc. allow the search for common factors. Since two random numbers have an asymptotic probability $6/\pi^2 \approx 61\%$ of being coprime [12] and this rapidly increases for three or more numbers, this search is very likely to be successful if we have access to several numbers containing the same factor. Thus we have to ensure that there are no two or more values which contain the same factor.

We will now analyze all occurrences of Q inside the transformation.

- The cost function

  $$\begin{aligned} \boldsymbol{c'} &= c^T * Q = c^T * (D * E) \\ &= \begin{pmatrix} c_1 * D_1 & c_2 * D_2 & \cdots & c_n * D_n \end{pmatrix} * E \end{aligned}$$

  contains all the factors of D, but each factor appears only once inside one of the entries.

- $\boldsymbol{M_{\{1,2\}}Q}$ has the following structure

  $$\begin{aligned} & M_1 * Q = M_1 * (D * E) \\ &= \begin{pmatrix} M_{1,1} * D_1 & \cdots & M_{1,n} * D_n \\ \vdots & & \vdots \\ M_{m_1,1} * D_1 & \cdots & M_{m_1,n} * D_n \end{pmatrix} * E \end{aligned}$$

  This is open for attack as there are lots of values containing the same factor. However, since inside M' there is $-S$ below (cf. Eq. 6), $M'' = P * M'$ has a different structure. For example

  $$\begin{aligned} M''_{1,1} &= \sum_{i=1}^{m_1} P_{1,i} * (M_1Q)_{i,1} \\ &+ \sum_{i=1}^{m_2} P_{1,i+m_1} * (M_2Q)_{i,1} \\ &- P_{1,m_1+m_2+1} * S_1 \end{aligned}$$

  contains a common factor D_j in all summands except for the last. This is enough to make the values unusable for factorization attacks.

- $\boldsymbol{M_{\{1,2\}}Qr}$ is secure against factorization attacks since

  $$(M_1Qr)_1 = \sum_{i=1}^{n} (M_1Q)_{1,i} * r_i$$

  where each $(M_1Q)_{1,i}$ contains a different D_j.

- $\boldsymbol{z} = Q^{-1}x + r$ is protected against factorization attacks by r.
- The **slack-variables** could also leak the factors inside Q as the i-th constraint concerning S has the form $-S_i * z_i + Z_{s_i} = -S_i * r_i$ which gives

  $$\begin{aligned} Z_{s_i} &= S_i * (z_i - r_i) \\ &= S_i * ((Q^{-1}x)_i + r_i - r_i) \\ &= S_i * (Q^{-1}x)_i \end{aligned}$$

  This would allow the search for common factors. However, the slack-variables are permuted independently of Q so that it is unknown which variable belongs to which constraint. To find possible matches an attacker could search for pairs of variable whose difference is smaller than the maximum of r. However, this is made harder by S as $Z_{s_i} - z_i = S_i * z_i - S_i * r_i - z_i = (S_i - 1)z_i - S_i * r_i$ (and not $-r_i$ as it would be the case if $S_i = 1$ for all i).

Overall only in c' the factors inside Q are easily accessible. Thus the search for common factors is difficult, if possible at all, which is of great importance for the security of c.

7 Going Multi-Party

To be able use the transformation in a multi-party scenario we have to find a way to apply the transformation on the distributed data without exchanging it in clear. We propose to use secure computations based on Shamir-shared values as developed by Ben-Or et al. [4].

In short, this allow us to share values among the p parties in a way that the knowledge of less than $k < p/2$ of the p shares does not reveal any information about the secret value. Yet these shares can be used to make computations (additions, multiplications etc.) on the secret values. After finishing the computations, we can put the shares of the result together and reconstruct it - whereas all input and intermediate values remain secret.

Thus - once we have shares of all necessary data ($M_{\{1,2\}}$, $b_{\{1,2\}}$, c, P, Q, S, A, r) - we can use secure computations to perform the transformation. We still have to find a way to assembly the data ($M_{\{1,2\}}$, $b_{\{1,2\}}$, c) and a way to jointly choose the random values (P, Q, S, A, r).

The assembly of the data depends on the initial partitioning in the problem. However - as long as each value is clearly "owned" by one party (which is usually the case) - the assembly is not a problem: Each party shares its values and the matrices and vectors are filled with the shares[10].

The choice of the random matrices and vectors is more complicated. We have to guarantee the randomness and the correct form of the result, i.e. we have to make sure that Q is a monomial matrix, A a permutation matrix and that P is invertible. Moreover we need to ensure that no party knows the resulting values, because they would allow the parties to undo the transformation and access the secret data.

The basic idea is that each party randomly chooses its random values (i.e. party i chooses matrices P_i, Q_i, S_i, A_i and a vector r_i) and we combine them to have jointly and fairly chosen random values. Ideally, even the knowledge of $p-1$ of these p input values should not leak any information about the result.

To achieve this we rely on secure computations again. After each party has chosen its input, we calculate $P = \prod_{i=0}^{p-1} P_i$, $Q = \prod_{i=0}^{p-1} Q_i$, $S = \sum_{i=0}^{p-1} S_i$, $A = \prod_{i=0}^{p-1} A_i$ and $r = \sum_{i=0}^{p-1} r_i$. As these computations take place in a finite field, the result is random, and even knowing $p-1$ of the p input values does not reveal anything about the output.

An overview of the complete protocol:

1. Each party i chooses a random invertible matrix P_i, a random positive monomial matrix Q_i, a random positive diagonal Matrix S_i, a random permutation Matrix A_i and a random positive vector r_i.

2. Each party i securely shares its parts of M_1, M_2, b_1, b_2 and c as well as P_i, Q_i, S_i, A_i and r_i.

3. Secure computation of P, Q, S, A and r as described above. Assembly of M_1, M_2, b_1, b_2 and c.
   Transformation:
   Calculation of $M'' = P * \begin{pmatrix} M_1Q & 0 \\ M_2Q & \\ -S & \raisebox{1.5ex}{A} \end{pmatrix}$,
   $b'' = P * \begin{pmatrix} b_1 + M_1Qr \\ b_2 + M_2Qr \\ -Sr \end{pmatrix}$ and
   $c_s'^T = \begin{pmatrix} c^T * Q & 0 & \dots & 0 \end{pmatrix}$.

[10] Another option is that each party shares - e.g. for the cost vector c - a vector c_i where all values that are not owned by itself are set to zero. Then all parties calculate $c = \sum_{i=0}^{p-1} c_i$ using secure computations which merges the data.

4. $c_s'^T$, M'' and b'' are reconstructed from the shares and passed into the cloud.

5. The Cloud solves the Linear Program

$$\begin{array}{rcl} \min & c_s'^T z_s & \\ \text{s.t. } M'' z_s & = & b'' \\ z_s & \geq & 0 \end{array}$$

6. Secure sharing of the solutions z

7. Secure distributed computing of $x = Q(z - r)$

8. Output of values of x to their respective owner.

7.1 Optimization: Fast Matrix Multiplication

Matrix multiplication is the key operation of our transformation as it is used in many places. When using secure computations on Shamir-shared values multiplications are expensive due to the overhead caused by the degree-reducing step on the secret shares. Therefore we want to reduce this overhead as much as possible.

A naive implementation (see Algorithm 7.1) of the multiplication of a shared $(m \times k)$ matrix A and a $(k \times n)$ matrix B will result in a time complexity of $\mathcal{O}(mkn)$, a round complexity (i.e. the number of points in time when communication is necessary) of $\mathcal{O}(mkn)$ and a communications complexity (i.e. the number of messages exchanged) of $\mathcal{O}(mknp^2)$ as each degree-reducing step requires one round and $\mathcal{O}(p^2)$ messages [4]. We will now describe how to reduce this to one round and $\mathcal{O}(mnp^2)$ messages.

The idea is to postpone the degree-reducing step as much as possible. If $A_{x,y}$ and $B_{y,z}$ are shares of a polynomial with degree k, then *temp* is of degree $2k$, but, as adding two shares does not increase the degree of the polynomial, we can calculate $C_{x,z} = \sum_{y=0}^{k-1} A_{x,y} * B_{y,z}$ without exceeding a degree of $2k$. Thus we can postpone the degree-reducing step until the end of all calculations and execute it in parallel on the whole matrix to reduce round complexity to 1 and communication complexity to $\mathcal{O}(mnp^2)$ (see Algorithm 7.2[11]).

[11] This algorithm is somewhat similar to the inner product algorithm of Catrina and de Hoogh [7], but generalized to a complete matrix multiplication.

Algorithm 7.1 Matrix Multiplication $C = A * B$, naive

```
for $x = 0$ to $m - 1$ do
    for $z = 0$ to $n - 1$ do
        $sum \leftarrow 0$
        for $y = 0$ to $k - 1$ do
            $temp \leftarrow A_{x,y} * B_{y,z}$
            execute degree-reducing step on $temp$
            $sum \leftarrow sum + temp$
        end for
        $C_{x,z} \leftarrow sum$
    end for
end for
```

Algorithm 7.2 Matrix Multiplication $C = A*B$, optimized communication

```
for $x = 0$ to $m - 1$ do
    for $z = 0$ to $n - 1$ do
        $sum \leftarrow 0$
        for $y = 0$ to $k - 1$ do
            $sum \leftarrow sum + A_{x,y} * B_{y,z}$
        end for
        $C_{x,z} \leftarrow sum$
    end for
end for
execute degree-reducing step on all $m * n$ entries of $C$ in parallel
```

8 Use Case: Supply Chain Optimization

Supply Chain Optimization problems can easily be expressed as linear problems [22]. Each variable corresponds either to the number of units to produce at a certain factory at a certain time, to the inventory (per factory and time) or to transported units (between two factories). The constraints are flow constraints (e.g. "all produced units are either shipped to the next stage in the chain or remain in the inventory"), capacity constraints (e.g. "factory X can produce at most n units of product Y in period i") and demand constraints (e.g. "in period i exactly k units of product Z are sold to customers"). The cost function is composed of production costs (per produced unit), shipping costs (price of shipping one unit from factory X to factory Y) and inventory costs (the price of stocking products or intermediate components at a factory).

This means that variable is "owned" either by one company (for production an inventory variables) or by two companies (for transport variables). Hence we have a simple structure which allows us to easily set up the distributed linear program as discussed above. Furthermore, most of the constraints (the flow and demand constraints) are expressed as equations, which is convenient for our transformation.

8.1 Experimental Results

To analyze the practical performance of our approach in Supply Chain Management, we implemented a prototype. Each party is simulated by a separate Java thread which communicates using the network interface. The experiments were conducted on an 8 core AMD Opteron 1800 machine with 16 GB RAM running under Linux (SuSE Enterprise Server 11 64 bit). This setup is reasonably realistic as each thread runs on a separate core and has roughly as much computational power available as if it was running alone on a standard office PC.

The test data we used is a sample supply chain structure with five companies. To scale the size of the problem, this optimization was done for different numbers of planning periods which increases the number of variables and constraints. The measured timings include sharing of the values, performing the transformation, reconstructing the transformed LP, solving it using a standard LP solver[12] and back-transforming the solution using secure computation again.

[12] We used LP_Solve which is licensed under LGPL and available at http://lpsolve.sourceforge.net/

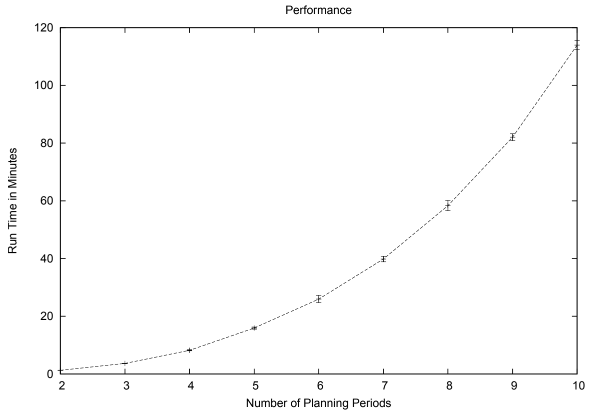

Figure 6: Performance in Distributed Supply Chain Management Case

The results (see Figure 8.1) are promising: The standard test case (six periods: 180 constraints, 282 variables) is solved in approximately 25 minutes (which are nearly entirely due to the secure computation for applying the transformation, solving the transformed LP in clear takes just a few seconds) and even the bigger test cases are completed within two hours. This is sufficiently fast for practical usage and much quicker than the distributed Simplex algorithms.

We also calculated the numerical values for the a-posteriori probabilities of a right guess in the six-period case for a uniform input distribution based on the bounds established in Section 6 and Eq. 19:

- cost function c: $PR_{posteriori} \leq 3.76 \cdot 10^{-220}$
- values x: $PR_{posteriori} \leq 2.30 \cdot 10^{-1409}$

As explained above, there is no easily calculable bound on the leakage of the constraints, but the chances of guessing P only based on A are less than $2.34 \cdot 10^{-746370}$.

9 Conclusion

In this paper we proposed a disguising transformation for linear programs. We proved correctness and analyzed security in the framework of leakage quantification by estimating the leakage of building blocks (permutation, scalar and matrix multiplication) and combining these results to establish bounds on the entire transformation. Subsequently we developed a secure multi-party protocol to make the transformation usable in a distributed case.

Based on this protocol we implemented a prototype for secure supply chain optimization. We conducted experiments which show significant performance gains compared to entirely distributed secure simplex algorithms.

Overall the proposed algorithm seems to be well-suited for the initial supply chain optimization problem as well as general outsourcing purposes. It is correct, fast and sufficiently secure for many applications.

Additionally, the basic ideas of the transformation can easily be extended and generalized to fit other problems as e.g. systems of linear equations. This underlines the power of disguising approaches.

9.1 Future Work

We have thought of several extensions to the transformations which can enhance security. Possibilities include splitting variables or creating fake variables in order to protect c. These measures have to be checked for their security as well as their impact on performance.

Furthermore the numerical stability of the transformation should be examined. During our experiments a standard LP-Solver was always able to find the correct solution, however rounding errors and performance deterioration could be observed from time to time. This also is of interest as numerical effects could lead to attacks.

We conducted the performance measurements under ideal network conditions and ignored common problems such as latencies, failures or packet loss. For a comprehensive performance analysis they should be re-run under more realistic networks conditions. Nevertheless we stress that the competing approaches based on secure LP solvers were benchmarked under the same ideal conditions. Furthermore, our transformation has a quasi-constant (only depending on the number of parties) round and small communication complexity compared to these approaches. Therefore it can be expected that our relative performance advantage will even further increase.

The security analysis should also be refined in some parts. In particular the interconnections between the different parts of the transformation should be examined more precisely as this is one of the most probable entrance point for attacks. Additionally it would be desirable to refine the bounds on the leakage of combined channels (i.e. to show that for example the combination of a multiplication and a permutation is strictly better than the best of them alone) and give - if possible - a closed-form bound on the leakage of a matrix multiplication.

As our security analysis currently supposes a *Honest-but-Curious*-Scenario, it could also be extended to analyze the possible influence of malicious participants with byzantine behavior (i.e. participants that give wrong input data or do not follow the protocol) on the results, in particular in the case of distributed supply chain management.

References

[1] M. J. Atallah, and K. B. Frikken. Securely outsourcing linear algebra computations. *ASIACCS '10: Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, 2010

[2] M. J. Atallah, K. N. Pantazopoulos, J. R. Rice, and E. H. Spafford. Secure outsourcing of scientific computations. *Advances in Computers*, 2002.

[3] A. Bednarz, N. Bean, and M. Roughan. Hiccups on the road to privacy-preserving linear programming. *Proceedings of the 8th ACM Workshop on Privacy in the Electronic Society*, 2009.

[4] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. *Proceedings of the 20th ACM Symposium on Theory of Computing*, 1988.

[5] C. Braun, K. Chatzikokoladis, and C. Palamidessi. Quantitative notions of leakage for one-try attacks. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 2009.

[6] H. Braun. Optimization with grid computing. *Presentation at the Workshop on Cyber Infrastructure in Chemical and Biological Systems.* Available at *http://oit.ucla.edu/nsfci/presentations/braun.ppt*, 2006.

[7] O. Catrina, and S. de Hoogh. Secure multiparty linear programming using fixed-point arithmetic. *Proceedings of the 15th European Symposium on Research in Computer Security*, 2010.

[8] T. M. Cover, and J. A. Thomas. Elements of information theory. *Wiley*, 1991.

[9] W. Du, and M. J. Atallah. Privacy-preserving cooperative scientific computations. *CSFW '01: Proceedings of the 14th IEEE workshop on Computer Security Foundations*, 2001.

[10] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. *Proceedings of the 19th ACM Symposium on Theory of Computing*, 1987.

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

[12] Eike Kiltz, Gregor Leander, and John Malone-Lee. Secure computation of the mean and related statistics. *TCC '05: Proceedings of the 2nd Theory of Cryptography Conference*, 2005.

[13] Victor Klee, and George J. Minty. How good is the simplex algorithm? *Inequalities, Vol. III*, 1972.

[14] J. Li, and M. Atallah. Secure and private collaborative linear programming. *Proceedings of the 2nd IEEE Conference on Collaborative Computing*, 2006.

[15] O. L. Mangasarian. Privacy-Preserving Horizontally-Partitioned Linear Programs. *Data Mining Institute Technical Report 10-02*, Available at *ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/10-02.pdf*, April 2010.

[16] O. L. Mangasarian. Privacy-Preserving Linear Programming. *Data Mining Institute Technical Report 10-01*, Available at *ftp://ftp.cs.wisc.edu/pub/dmi/tech-reports/10-01.pdf*, March 2010.

[17] R. Pibernik, and E. Sucky. An approach to inter-domain master planning in supply chains. *International Journal of Production Economics 108*, 2007.

[18] A. Shamir. How to share a secret. *Communications of the ACM*, 1979.

[19] G. Sierksma. Linear and Integer Programming: Theory and Practice. *Marcel Dekker, Inc.*, 1996.

[20] G. Smith. Adversaries and information leaks (tutorial). *TGC'07: Proceedings of the 3rd conference on Trustworthy global computing*, 2008.

[21] G. Smith. On the Foundations of Quantitative Information Flow. *FOSSACS '09: Proceedings of the 12th International Conference on Foundations of Software Science and Computational Structures*, 2009.

[22] H. Stadler, and C. Klinger. Supply chain management and advanced planning. *Springer*, 2002.

[23] T. Toft. Solving linear programs using multiparty computation. *Proceedings of the 13th International Conference on Financial Cryptography and Data Security*, 2009.

[24] J. Vaidya. Privacy-preserving linear programming. *Proceedings of the 24th ACM Symposium on Applied Computing*, 2009.

[25] M. Wibmer, D. Biswas, and F. Kerschbaum. Leakage Quantification of Cryptographic Protocols. *OTM IS'10: The 5th International Symposium on Information Security*, 2010.

[26] A. Yao. Protocols for Secure Computations. *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science*, 1982.

A Leakage of Combined Channels

We use the following definitions and theorems by Wibmer et al. [25]:

Definition 3 (Independent Channels). *Two channels $\mathcal{X} \xrightarrow{\mathcal{C}} \mathcal{Y}$ and $\mathcal{X}' \xrightarrow{\mathcal{C}'} \mathcal{Y}'$ are independent if $C|x$ is independent from $C'|x'$ for every $x \in X$ and $x' \in X'$.*

Definition 4 (Composition of Channels). *Let $\mathcal{X} \xrightarrow{\mathcal{C}_1} \mathcal{Y}$ and $\mathcal{Y} \xrightarrow{\mathcal{C}_2} \mathcal{Z}$ be two channels. Then we can define the composition channel $\mathcal{X} \xrightarrow{\mathcal{C}_2 \circ \mathcal{C}_1} \mathcal{Z}$ by*

$$(C_2 \circ C_1|x)(\omega) = (C_2|(C_1|x)(\omega))(\omega)$$

for every $x \in \mathcal{X}$ and $\omega \in \Omega$ (the event space).

Theorem 4 (Leakage of a Composition). *Let $\mathcal{X} \xrightarrow{\mathcal{C}_1} \mathcal{Y}$ and $\mathcal{Y} \xrightarrow{\mathcal{C}_2} \mathcal{Z}$ be two independent channels. Then*

$$L_\times(C_2 \circ C_1) \leq \min\{L_\times(C_1), L_\times(C_2)\}$$

Definition 5 (Direct Product of Channels). *Let $\mathcal{X} \xrightarrow{\mathcal{C}} \mathcal{Y}$ and $\mathcal{X}' \xrightarrow{\mathcal{C}'} \mathcal{Y}'$ be two channels. Then we can define the direct product channel $\mathcal{X} \times \mathcal{X}' \xrightarrow{\mathcal{C} \times \mathcal{C}'} \mathcal{Y} \times \mathcal{Y}'$ by*

$$(C \times C')(x, x') = (C|x, C'|x')$$

for $x \in \mathcal{X}$ and $x' \in \mathcal{X}'$.

Theorem 5 (Leakage of a Direct Product)**.** *Let* $\mathcal{X} \xrightarrow{\mathcal{C}} \mathcal{Y}$ *and* $\mathcal{X}' \xrightarrow{\mathcal{C}'} \mathcal{Y}'$ *be two independent channels. Then*

$$L_\times(C \times C') = L_\times(C) \cdot L_\times(C')$$

Definition 6 (Product of Channels)**.** *Let* $\mathcal{X} \xrightarrow{\mathcal{C}} \mathcal{Y}$ *and* $\mathcal{X} \xrightarrow{\mathcal{C}'} \mathcal{Y}'$ *be two channels. Then we can define the direct product channel* $\mathcal{X} \xrightarrow{\mathcal{C}\cdot\mathcal{C}'} \mathcal{Y} \times \mathcal{Y}'$ *by*

$$(C \cdot C')|x = (C|x, C'|x)$$

for $x \in \mathcal{X}$*.*

Theorem 6 (Leakage of a Product)**.** *Let* $\mathcal{X} \xrightarrow{\mathcal{C}} \mathcal{Y}$ *and* $\mathcal{X} \xrightarrow{\mathcal{C}'} \mathcal{Y}'$ *be two independent channels. Then*

$$\max\left\{L_\times(C), L_\times(C')\right\} \leq L_\times(C \cdot C') \leq L_\times(C)L_\times(C')$$