

Practical Privacy-Preserving Multiparty Linear Programming Based on Problem Transformation

Jannik Dreier
 Université Grenoble 1, CNRS, Verimag
 Grenoble, France
 jannik.dreier@imag.fr

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

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.

I. INTRODUCTION

Linear Programming (LP) can be used to solve many practical optimization problems, e.g. supply chain master planning [21]. 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].

Secure multi-party computation (SMC) [4, 10, 18, 25] 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, 22].

Nevertheless these solutions suffer from a prohibitively bad computational performance. We estimate that our prototypical implementation of Toft’s protocol [22] 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 [23] Vaidya presents a different approach to the problem. Instead of implementing a LP solver using SMC the problem is

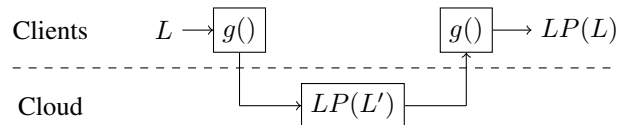


Fig. 1. Privacy-Preserving Linear Programming based on Problem Transformation

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, 22] 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 using problem transformation.

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 computing 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 review related work. In Section III we present our main result, the privacy-preserving transformation. We prove the correctness of the transformation in Section IV and analyze its performance in Section V. In Section VI we summarize our results from leakage quantification of this transformation. We show the protocol for securely computing the transformation in Section VII. The results from our use case example are described in Section VIII. We conclude the paper in Section IX.

II. RELATED WORK

Our work is related to secure multi-party LP solvers [7, 14, 22], secure outsourcing [1, 2, 3, 9, 15, 16, 23] and leakage quantification [5, 20, 24].

A. Secure Multi-Party LP Solvers

As mentioned above, different distributed LP solvers based on secure multi-party computation have been developed. Toft [22] used a distributed secure Simplex algorithm based on secret sharing. This algorithm is usable for any number of participants and provides information-theoretic security. Yet it 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 [22] 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 small inputs as it suffers from a higher asymptotic and worst-case complexity than our transformation.

B. Secure Outsourcing

Vaidya [23] 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].

C. 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 [20] 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-party attacks as it gives a worst-case evaluation. To assess complex operations we make use of some theorems by Wibmer et al. [24] on the leakage of combined channels.

III. 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):

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

We use a positive monomial matrix¹ Q to hide c (as proposed by [23, 3]):

$$\begin{aligned} \min \quad & c^T Q Q^{-1} x \\ & M_1 Q Q^{-1} x = b_1 \\ & M_2 Q Q^{-1} x \leq b_2 \\ & Q^{-1} x \geq 0 \end{aligned}$$

and a positive vector r to hide x

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

For $z = Q^{-1} x + r$ and a strictly positive diagonal matrix S^2 we have

¹A monomial matrix contains exactly one non-zero entry per row and column.

²A diagonal matrix where the entries on the main diagonal $A_{i,i}$ are strictly positive.

$$\begin{aligned}
\min c^T Qz \\
M_1 Qz &= b_1 + M_1 Qr \\
M_2 Qz &\leq b_2 + M_2 Qr \\
Sz &\geq Sr
\end{aligned}$$

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

$$M' = \begin{pmatrix} M_1 Q & 0 \\ M_2 Q & A \\ -S & \end{pmatrix}, b' = \begin{pmatrix} b_1 + M_1 Qr \\ b_2 + M_2 Qr \\ -Sr \end{pmatrix}$$

where A is a permutation matrix³ representing slack-variables⁴. This allows us to rewrite the program as follows:

$$\begin{aligned}
\min c'_s{}^T z_s \\
\text{s.t. } M' z_s &= b' \\
z_s &\geq 0
\end{aligned}$$

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 P and with $M'' = P * M'$ and $b'' = P' * b'$ we have

$$\begin{aligned}
\min c''_s{}^T z \\
\text{s.t. } M'' z_s &= b'' \\
z_s &\geq 0
\end{aligned}$$

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

IV. 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 show (for more details see the technical report [8]) 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.

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. ■

³A permutation matrix is a monomial matrix where the non-zero entries are “1”.

⁴A 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 $s_1 \geq 0$.

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.*

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

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

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

V. PERFORMANCE

Performance is a key aspect of our approach: 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, 22] 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 discuss practical performance in our use case example in Section VIII.

VI. SECURITY

We analyze the security of our transformation in the following setting: An attacker wants to obtain the input data, i.e. 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.

Ideally we would like to give a classical security proof. However, as our transformation is based on disguising using random noise, our transformation will probably leak some information. This makes the classical cryptological security definitions unsuitable. Nevertheless leaking some information can be acceptable for many applications as long as the leakage is small and bounded. In return, we get dramatic performance improvements which can be more important than perfect security. Nevertheless, we want to formally evaluate this leakage, and to show that it is “small and bounded”. This is possible using Leakage Quantification methods based on information theory.

Due to space limitations, we are not able to discuss the entire analysis here (for details see our Technical Report [8]). We limit ourselves to defining the metric “Multiplicative Advantage” and presenting the results of our analysis, i.e. the bounds on the leakage of different parts of the transformation. In the last part of this section we analyze possible attacks.

A. Multiplicative Advantage

The disguising transformation is modeled as a communication channel as in information theory, and we try to measure by how much the knowledge of the output increases the chances of guessing the input.

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

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

In this model 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 [20]. For a random variable X characterizing the input distribution let

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

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

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

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

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

the factor by which the knowledge of the output of the channel increases the chances of a right guess for a given X^5 .

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

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

$L(\mathcal{C})$ 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, 24] which yields

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

This result is very useful in practice as $p(y_j|x_i)$ are the entries of the channel matrix. Furthermore Wibmer et al. [24] showed that multiplicative leakage can be used to easily bound

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

the leakage of compositions of independent channels which simplifies the analysis of complex protocols. In our technical report [8] we calculate bounds on the leakage of several basic operations (permutations, scalar and matrix multiplication) which we now use to analyze the overall leakage of our transformation.

B. Establishing Bounds on the Leakage

Using the bounds on the leakage of the basic operations, we establish bounds on the leakage of the three main parts of the transformation: The cost function, the variables and the constraints.

1) *Leakage of the cost function c :* In our transformation c is hidden by a monomial matrix Q , i.e. a permutation followed by a multiplication of each entry with a scalar value. Wibmer et al. [24] showed that the leakage of two independent channels arranged one after the other is bounded by the minimum of both leakages. Hence we obtain

$$L(c_{n,k}^l) \leq \min \left(\left(\frac{n+l}{l+1} + 1 \right)^k, \frac{(A(n,l)+k)!}{k! * (A(n,l))!} \right)$$

where $c_{n,k}^l$ denotes the channel for a k -dimensional input vector c hidden by a multiplication by a monomial matrix and $A(n,l)$ is the number of different values of $f = \prod_{i=1}^l f_i$ with $f_i \in \{1, \dots, n\}$. The first value is the leakage of a scalar multiplication, the second value bounds the leakage of the permutation (cf. [8]). Note that this is an evaluation 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). When evaluating all parts together, this results in partly dependent channels. However, this simplification is realistic as the other occurrences are well-hidden, as we discuss later on.

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

$$\log(x_m + 2) \leq L(y = rx + r') \leq 2 \log(x_m + 2)$$

for $r_m \rightarrow +\infty$. 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

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

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 (see cryptanalysis below).

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. This further complicates exploiting correlation by common factors.

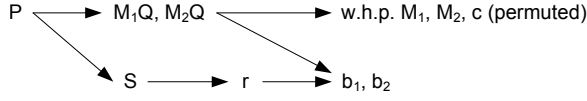


Fig. 2. Overview of a possible attack

3) *Leakage of the Constraints (M_1 , M_2 , b_1 and b_2):* The leakage of the constraints is difficult to estimate due to the number and complexity of related transformation steps. We can however argue that their security is essentially based on P , as the following analysis shows.

On the one hand, breaking the hiding with P almost completely reveals all other hiding operations (see Figure 2). If the attacker knows P , 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 the cryptanalysis 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. Hence 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 .

We 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

$$P * B = P * \begin{pmatrix} B' & 0 \\ & A \end{pmatrix}$$

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_1Q , M_2Q 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 is reasonable.

This model allows us to use another theorem by Wibmer et al. [24] on parallel channels. We split the channel \mathcal{C}_P into a first channel $\mathcal{C}_{P,1}$ which calculates $P * B'$, and a second channel $\mathcal{C}_{P,2}$ which calculates

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

Then the theorem gives that

$$\max\{L(\mathcal{C}_{P,1}), L(\mathcal{C}_{P,2})\} \leq L(\mathcal{C}_P) \leq L(\mathcal{C}_{P,1})L(\mathcal{C}_{P,2})$$

$\mathcal{C}_{P,1}$ corresponds to a matrix multiplication channel $\mathcal{M}'_{k,n,l}$ [8]. The leakage of $\mathcal{C}_{P,2}$ can be bounded as follows

$$L(\mathcal{C}_{P,2}) \leq \frac{\prod_{i=1}^{m_2+k} ((n+1)^{m_1+m_2+l-i})}{(m_2+k)!}$$

when P contains values between 0 and n . This is formally shown in the technical report [8].

C. Cryptanalysis

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

1) *Attacks on P :* The most promising approaches to attack P are based on the structure of the matrices, in particular the slack-variables as the following example illustrates. Consider these constraints:

$$\begin{aligned} M * x &\leq b \\ x &\geq 0 \end{aligned}$$

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

$$\begin{aligned} (M \quad I) * x' &= b \\ x' &\geq 0 \end{aligned}$$

If we multiply by P the result is

$$\begin{aligned} (P * M \quad P) * x' &= P * b \\ x' &\geq 0 \end{aligned}$$

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

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

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].

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. 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], 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 now analyze all occurrences of Q inside the transformation.

- The cost function

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

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

- $M_{\{1,2\}} Q$ 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. III), $M'' = P * M'$ has a different structure. For example

$$\begin{aligned} M''_{1,1} &= \sum_{i=1}^{m_1} P_{1,i} * (M_1 Q)_{i,1} \\ &+ \sum_{i=1}^{m_2} P_{1,i+m_1} * (M_2 Q)_{i,1} - P_{1,m_1+m_2+1} * S_1 \end{aligned}$$

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

- $M_{\{1,2\}} Q r$ is secure against factorization attacks since

$$(M_1 Q r)_1 = \sum_{i=1}^n (M_1 Q)_{1,i} * r_i$$

where each $(M_1 Q)_{1,i}$ contains a different D_j .

- $z = Q^{-1}x + r$ 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 .

VII. GOING MULTI-PARTY

To be able use the transformation in a multi-party scenario, we propose to use secure computations based on Shamir-shared values as developed by Ben-Or et al. [4]. In short, this technique 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.

As in our scenario the input values ($M_{\{1,2\}}, b_{\{1,2\}}, c$) are distributed among the parties, we have to assemble the data and 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.

When choosing the random matrices and vectors, we have to ensure the randomness and the correct form of the result, i.e. guarantee that Q is a monomial matrix, A a permutation matrix and so on. 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. This leads to following 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 .

Algorithm VII.1 Matrix Multiplication $C = A * B$, naive

```
1: for  $x = 0$  to  $m - 1$  do
2:   for  $z = 0$  to  $n - 1$  do
3:      $sum \leftarrow 0$ 
4:     for  $y = 0$  to  $k - 1$  do
5:        $temp \leftarrow A_{x,y} * B_{y,z}$ 
6:       execute degree-reducing step on  $temp$ 
7:        $sum \leftarrow sum + temp$ 
8:     end for
9:      $C_{x,z} \leftarrow sum$ 
10:  end for
11: end for
```

- 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:

$$\text{Calculation of } M'' = P * \begin{pmatrix} M_1Q & 0 \\ M_2Q & A \\ -S & \end{pmatrix},$$

$$b'' = P * \begin{pmatrix} b_1 + M_1Qr \\ b_2 + M_2Qr \\ -Sr \end{pmatrix} \text{ and}$$

$$c_s'^T = (c^T * Q \quad 0 \quad \dots \quad 0).$$

- 4) $c_s'^T$, M'' and b'' are reconstructed from the shares and passed into the cloud.
5) The Cloud solves the Linear Program

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

- 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.

A. Optimization: Fast Matrix Multiplication

Matrix multiplication is the key operation of our transformation. When using secure computations on Shamir-shared values, multiplications are expensive due to the degree-reducing step on the shares. We want to reduce this overhead as much as possible.

A naive implementation (see Algorithm VII.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 $\mathcal{O}(mkn p^2)$ as each degree-reducing step requires one round and $\mathcal{O}(p^2)$ messages [4]. This can be reduced to one round and $\mathcal{O}(mnp^2)$ messages by postponing 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 $2k$. Yet, 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

Algorithm VII.2 Matrix Multiplication $C = A * B$, optimized communication

```
1: for  $x = 0$  to  $m - 1$  do
2:   for  $z = 0$  to  $n - 1$  do
3:      $sum \leftarrow 0$ 
4:     for  $y = 0$  to  $k - 1$  do
5:        $sum \leftarrow sum + A_{x,y} * B_{y,z}$ 
6:     end for
7:      $C_{x,z} \leftarrow sum$ 
8:   end for
9: end for
10: execute degree-reducing step on all  $m * n$  entries of  $C$  in parallel
```

reduce round complexity to 1 and communication complexity to $\mathcal{O}(mnp^2)$ (see Algorithm VII.2⁶).

VIII. USE CASE: SUPPLY CHAIN OPTIMIZATION

Supply Chain Optimization problems can be expressed as linear programs [21]. 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 , 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.

A. Experimental Results

To analyze the practical performance of our approach, we implemented a prototype. The test data 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 and back-transforming the solution using secure computation again (for more details see the technical report [8]).

The results (see Figure VIII-A) are promising: The standard test case (six periods: 180 constraints, 282 variables) is solved

⁶This algorithm is somewhat similar to the inner product algorithm of Catrina and de Hoogh [7], but generalized to a complete matrix multiplication.

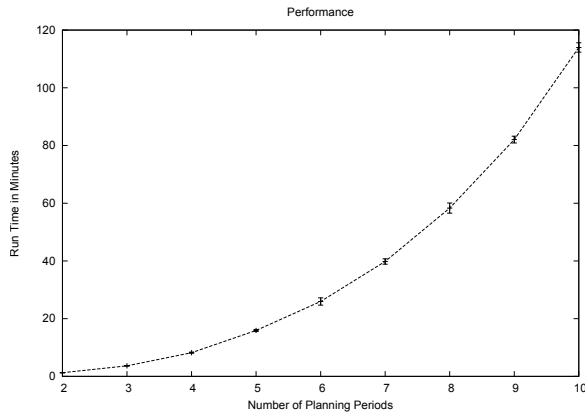


Fig. 3. Performance in Distributed Supply Chain Management Case

in approximately 25 minutes, 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.

Based on the bounds established above, we calculated the numerical values for the a-posteriori probabilities of a right guess in the six-period case for a uniform input distribution:

- 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 $2.34 \cdot 10^{-746370}$.

IX. 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 establishing bounds on the leakage of the secret data. 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.

Moreover, the basic ideas of the transformation can easily be extended and generalized to fit other problems such as systems of linear equations. This highlights the power of disguising approaches.

A. Future Work

We can imagine several extensions to the transformation to enhance security, for example 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.

The security analysis could 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 interesting 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 the case of distributed supply chain management, it would be particularly interesting to examine if parties are able to obtain a financial advantage by manipulating the outcome using specially prepared input data, and if this would be noticeable to the other parties.

REFERENCES

- [1] M. J. Atallah, K. B. Frikken. Securely outsourcing linear algebra computations. *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, 2010
- [2] M. J. Atallah, K. N. Pantazopoulos, J. R. Rice, E. H. Spafford. Secure outsourcing of scientific computations. *Advances in Computers*, 2002.
- [3] A. Bednarsz, N. Bean, 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, 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, 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, S. de Hoogh. Secure multiparty linear programming using fixed-point arithmetic. *Proceedings of the 15th European Symposium on Research in Computer Security*, 2010.
- [8] J. Dreier, F. Kerschbaum. Practical Secure and Efficient Multiparty Linear Programming Based on Problem Transformation. *Cryptology ePrint Archive: Report 2011/108*, Available at <http://eprint.iacr.org/2011/108>, 2011.

- [9] W. Du, M. J. Atallah. Privacy-preserving cooperative scientific computations. *Proceedings of the 14th IEEE workshop on Computer Security Foundations*, 2001.
- [10] O. Goldreich, S. Micali, 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] E. Kiltz, G. Leander, J. Malone-Lee. Secure computation of the mean and related statistics. *Proceedings of the 2nd Theory of Cryptography Conference*, 2005.
- [13] V. Klee, G. J. Minty. How good is the simplex algorithm? *Inequalities, Vol. III*, 1972.
- [14] J. Li, 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>, 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>, 2010.
- [17] R. Pibernik, 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. On the Foundations of Quantitative Information Flow. *Proceedings of the 12th International Conference on Foundations of Software Science and Computational Structures*, 2009.
- [21] H. Stadler, C. Klinger. Supply chain management and advanced planning. *Springer*, 2002.
- [22] T. Toft. Solving linear programs using multiparty computation. *Proceedings of the 13th International Conference on Financial Cryptography and Data Security*, 2009.
- [23] J. Vaidya. Privacy-preserving linear programming. *Proceedings of the 24th ACM Symposium on Applied Computing*, 2009.
- [24] M. Wibmer, D. Biswas, F. Kerschbaum. Leakage Quantification of Cryptographic Protocols. *The 5th International Symposium on Information Security*, 2010.
- [25] A. Yao. Protocols for Secure Computations. *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science*, 1982.