

Security against the Business Partner

Florian Kerschbaum
SAP Research
Karlsruhe, Germany

florian.kerschbaum@sap.com

Rafael J. Deitos
SAP Research
Karlsruhe, Germany

rafael.deitos@sap.com

ABSTRACT

Security research has long focused on protecting against outside attackers. This was augmented with protection against insider threats, but recently networked business is emerging. With it a new threat is emerging: security against the business partner.

A possible solution is secure multi-party computation (SMC) and we give examples of its usefulness. We show with the example of supply chain optimization that only SMC provides the necessary security guarantees.

A major challenge of SMC is its practical realization. We give a detailed study and analysis of multi-party permutation and show the relations of the different theoretical complexities in this case.

The paper concludes with a comparison of service-oriented architectures and SMC. We show several architectural differences that need to be overcome.

Categories and Subject Descriptors

D.4.6 [Operating Systems]: Security and Protection—*Cryptographic controls*; C.2.4 [Computer-Communication Networks]: Distributed Systems—*Distributed applications*

General Terms

Algorithms, Security

Keywords

Security against the Business Partner, Secure Multi-Party Computation, Secure Permutation

1. INTRODUCTION

Many years of research have been spent on securing against outside attackers and the successes are numerous, e.g. encryption, firewalls, etc. Security experts soon realized that there is a threat within each company albeit many security measures, so called insider threats. E.g. the \star -property in

the Bell-LaPadula model can be violated by an authorized insider. The research has been ongoing and we see several successes of new security mechanisms, e.g. separation of duty in role-based access control [27].

With the emergence of the Internet and the current move towards service-oriented architectures, we see a growing trend towards networked business. Companies are regularly engaging in electronic transactions vital to their business over the Internet. A whole industry is betting its future on this trend.

With this trend a new attacker is emerging: the business partner. While previously business partners were separated by physical boundaries and transactions were only bound to paper and goods, now the business partner may have access to internal systems. Examples include vendor-managed inventory or virtual organizations for business web services [10]. Studies show that already most security incidents involve business partners and that this a significantly growing trend over the last five years [1].

There are few established measures for security against business partners. For individuals there are privacy-enhancing technologies, e.g. privacy-preserving identity management [5], anonymous network communication [11], but companies have more assets to protect than their identification. If companies engage in networked business, this often involves vital business secrets. People sometimes speak of industrial privacy.

What can we do? There are reputation systems [19]. A reputation system collects the input of business partners of the past transactions and computes a score of your trustworthiness. The assumption is that the higher the score, the higher the less likely you are to cheat or attack, and the more business you should attract. The idea is to consult a reputation system before engaging in business.

Reputation systems have their inherent flaws. First, a reputation system does not give you any guarantee. It just gives its best guess. It might be just your transaction that gets abused or the parameters might have changed. Second, the reputation classification problem is hard. Seldom a clear attacker survives long in the business world. Rather there are excellent companies and very good companies and differentiating them is hard. Third, almost all reputation systems are attackable. A good score can be achieved by other means than good business. Few reputation systems have been analyzed for their attack resistance, even less systematically. Fourth, forensics is difficult and linking a security incident to a business partner is hard. Cleverly attacking companies might go a long way before being detected. In particular,

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SWS'08, October 31, 2008, Fairfax, Virginia, USA.

Copyright 2008 ACM 978-1-60558-292-4/08/10 ...\$5.00.

if they provide excellent business service, they might have excellent reputation scores while being your biggest threat.

There are also sticky policies. A sticky policy is transferred along with the data and then enforced on the remote system. The idea is to control your data on a remote system. The main drawback is again that there is no guarantee that your policy is being enforced, i.e. you do not get a security guarantee. Rather the system is built on trust and maybe contractual obligations.

Then there is secure multi-party computation (SMC) [2, 12, 29]. SMC is a cryptographic technique that allows a group of parties to compute a joint function without disclosing their inputs, i.e. their input remains private while everybody gets access to the result. SMC is realized as protocols that compute on shares of the data. In simple terms, each protocol takes as input the shares and produces as output shares of the output.

The great advantage of SMC is that it gives you a security guarantee. You get provable security that your input is not being leaked. The disadvantage is that its implementation is very complicated and slow.

2. MAIN SCENARIO

It is a well-known fact that information exchange within a supply chain reduces costs [6], but companies are very reluctant to share this data, since it reveals vital business secrets. Consider the case of supply chain optimization [26]. A number of companies want to collaborative plan production, warehousing and transportation. Such a centrally computed plan leads to a global optimum compared to the local optima achieved by individual planning and then forwarding orders (upstream planning). It therefore can significantly reduce overall costs and it also avoids negative side-effects of a lack of information exchange, e.g. the so called “bullwhip” effect [24]. Nevertheless the necessary data to be shared includes production costs and capacities, such that a revelation negatively impacts the negotiation position.

Reputation systems cannot help, since the decision to engage in business with the partners has already been made. All companies are part of one supply chain and there is a basic level of trust, but that does not suffice. Sticky policies cannot help, since the revelation of the data itself concerns the players. If somebody has read the data, he cannot prevent using it in the next negotiation, even if the policy would forbid that. Sticky policies do not apply to humans.

SMC can help. It prevents the disclosure of the input data and can compute the supply chain optimization. You get both, the optimal plan without disclosing any data and its associated risks.

2.1 Research Questions

Supply chain optimization is a difficult task and requires significant resources even when computed non-securely. The most important research challenge is to construct a solution that is practically feasible. This covers two aspects: First, the theoretic construction: the problem needs to be decomposed and analyzed. A special protocol can be constructed that optimizes the asymptotic complexity of the protocol over a general circuit construction.

There are two complexities that can be optimized: computation complexity and communication complexity. Computation complexity is the maximum effort any party needs to spent during the protocol in number of computation steps.

The computation complexity can be measured in number of modular exponentiations, which are the costly operations. Communication complexity is the overall number of units of information exchanged. Communication has another aspect which is round complexity, i.e. the number of rounds in a synchronous distributed system. Round complexity often dominates the absolute communication time.

Technically supply chain optimization can be modeled as linear programming (LP) [26]. It is therefore necessary to securely compute LP. LP mainly consists of an iteration on the main step of “pivoting”. An element of the matrix is selected according to some optimization rule and then involved in a computation with every other element of the matrix to form a new matrix.

There exist two solutions for secure LP [23, 28]. One main difference is the way the index of the pivot element is hidden. In [23] the matrix is permuted by every party, such that no party knows the entire permutation. Then the pivot element is selected openly, such that everyone knows the index, but the index is “meaningless”, because it is a random choice in the permuted matrix. In [28] the index is kept as shared variables, but it is difficult to index with shared variables. Every operation must be the size of the entire array (or matrix) and is therefore very communication intensive.

Permutation is very complex, too. Let’s examine the solution proposed in [23] where the *blind-and-permute* protocol is executed on every iteration. More detailed, before finding the pivot element index, the parties, one after another, permute the matrix in a way that the final permuted matrix is the result of the original matrix permuted by the combination of the parties’ private permutations, which is known to neither of them. In the end, the parties run the *index recovering* protocol once, to get back the indexes of the basic variables of the initial system.

In the two-party case its communication complexity is $\mathcal{O}(cmn)$, where c is the size of the cipher text space, m is the number of constraints and n is the number of variables of the LP, for each iteration. The *index recovering* protocol, executed once in the end, is bounded by $\mathcal{O}(kcn)$, where k is the number of private permutations (equal to the number of iterations). The round complexity in the two-party case, on the other hand, is constant. The computational complexity of the matrix blind-and-permute protocol is $\mathcal{O}(mn)$ while the index recovering protocol is bounded by $\mathcal{O}(kn)$ modular exponentiations.

3. MULTI-PARTY PERMUTATION

A supply-chain optimization usually involves more than two parties and therefore requires a multi-party solution. We therefore need a multi-party permutation in order to adapt the solution from [23] for the multi-party case.

With that in mind, we propose two solutions for the multi-party case permutation. The first is mainly based on the idea from [23], extended for the multi-party scenario. We propose a second solution with an improved round complexity as an attempt to overcome the effect of possible network delays.

We then give a detailed analysis based on theoretical evaluation and practical experimentation with implementations. While the second solution improves the round complexity, it is worse in communication and most importantly in computation complexity. Comparing the effects of these three com-

plexities with practical implementation, we can conclude that the second solution is only practical in rare cases of parameters. In most practical cases the first, straight-forward solution will perform better. We provide a first estimation of how these three complexities in combination impact practical performance.

In the next sections we will describe in details each proposal as well as their complexities. Later, some practical results and general considerations are presented.

3.1 Notation for Protocol Construction

Consider a multi-party scenario where p parties want to collaboratively solve a supply-chain master plan (SCMP) problem using a secure LP. Such problem is modeled as a linear system in the canonical form [9], represented by a $(m + 1) \times (n + 1)$ matrix (D) as shown in [23].

Each party i ($1 \leq i \leq p$) has one different private permutation for rows and one for columns for each iteration of the secure LP. We consider two possible ways of representing such private permutations: as an array or as a matrix [4].

A permutation array is the simplest way to represent such a function. Let x be an array of size η represented as $x = \{x_0, x_1, \dots, x_\eta\}$. The permutation π of η elements is a map of the elements of a set to other elements of the same set, i.e. $\pi : \{1, \dots, \eta\} \rightarrow \{1, \dots, \eta\}$. In other words, it describes the new positions of such elements in the resulting set (array). An example is $x = \{x_0, x_1, x_2, x_3\}$ and $\pi = \{3, 1, 0, 2\}$ so that $\pi(x) = \{x_3, x_1, x_0, x_2\}$. The notation for the private permutation arrays for rows and columns on party i is π_r^i and π_c^i , respectively.

A permutation matrix is a square (0,1)-matrix that has exactly one entry 1 in each row and each column and 0's elsewhere.

Given a permutation π (as defined above), its permutation matrix is the $\eta \times \eta$ matrix P_π whose entries are all 0 except that in row i , the entry $\pi(i)$ equals 1. The permutation matrix is denoted as follows:

$$P_\pi = \begin{bmatrix} \mathbf{e}_{\pi(1)} \\ \mathbf{e}_{\pi(2)} \\ \vdots \\ \mathbf{e}_{\pi(\eta)} \end{bmatrix}, \text{ e.g. } P_\pi = \begin{bmatrix} 010 \cdots 0 \\ 100 \cdots 0 \\ \vdots \\ 000 \cdots 1 \end{bmatrix}_{\eta \times \eta}$$

where \mathbf{e}_j is a row vector of length η with 1 in the j th position and 0 in every other position.

Since the permutations are bijective functions, the product of two permutations is thus the same as their composition as functions. Likewise, since bijections have inverses, so do permutations, and both $P \circ P^{-1}$ and $P^{-1} \circ P$ are the "identity permutation" that leaves all positions unchanged, which enables for index recovery.

The outcome of any permutation algorithm is defined as the resulting system matrix permuted according to the combination of the private permutations from all parties. Also consider a setup phase where every party receives the public key pk and a private secret share sk_i (for party i) from an homomorphic threshold cryptosystem. In this work we use the threshold version of Paillier cryptosystem [25] described in [8]. The threshold is denoted by t , so that at least $t + 1$ shares (parties) are needed to decrypt a cipher text [8]. In the semi-honest model the threshold $t = p - 1$ is possible.

Let $[D]_{pk}$ denote the random encryption of the matrix D using the public-key pk and $[d]_{pk}$ denote the random en-

ryption of an array d , while $[d_i]_{pk}$ denotes the random encryption of the i -th element of the array d , using the same key. For clarity of explanation we omit the reference to the public key and write $[D]$ and $[d]$, for example.

As permutation matrices are private, we introduce the notion of an Encrypted Permutation Matrix, which is an encrypted form of the already defined permutation matrix. This implies that each party must generate his own encrypted permutation matrix for rows and columns, i.e. the party first generates a permutation matrix in clear and then, using the public key pk distributed during the setup phase, encrypts all the elements from the matrix. From now on, for sake of clarity, the term permutation matrix is used to denote the encrypted permutation matrix.

The messages' size is defined in terms of the size of the cipher text of the cryptosystem being used and, of course, by the overhead inherent from the communication technology employed, i.e. RMI over TCP in our case. For this work, consider the cipher text space as $c \in \mathbb{Z}_{mod^s+1}$ where mod is the modulo and s is a secure parameter of the cryptosystem. For sake of simplicity, the theoretical results do not consider the communication technology overhead.

3.2 Linear Round Algorithm

We now introduce the *Sequential Permutation* algorithm (Algorithm 1) for the multi-party case.

Generally speaking, the algorithm works with the parties sequentially applying their private permutations (π_r^i and π_c^i) on the encrypted system matrix. In the end, the last party multicasts the final permuted system matrix, which is then used in the current iteration of the secure LP. Note that such final matrix is permuted by the combination of the private permutations from all parties, i.e. $\pi_c^n(\pi_r^n(\dots(\pi_c^1(\pi_r^1(D))))))$. Also observe that the order in which the permutations are applied is important and must be respected, e.g. $\pi_c^i \circ \pi_r^i(\cdot)$ which implicates on permuting first rows and then columns.

Figure 1 gives an overview on how the algorithm works.

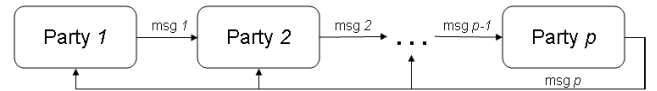


Figure 1: Message interaction for Algorithm 1

The first $(p - 1)$ messages are sent in a sequence, from party to party. Then, the last party (party p) multicasts a message with the final permuted matrices.

The MIXANDPERMUTE protocol (Algorithm 2) adds a random homomorphic encryption of zero to each value of the input matrix before the permutations are applied – the \oplus symbol stands for an homomorphic addition. By doing that, it is impossible to identify any applied permutation without actually knowing it.

Notice that for the execution of the instructions on lines 5-10 two matrix transpositions are implicitly computed. Their omission is for clarity of explanation and does not affect the overall complexity of the algorithm.

The permutation algorithm is applied on every round of the secure LP. Regarding matrix I , assume the first party receives such matrix as the *identity* matrix encrypted with the public key pk . The parties proceed in the same way mixing and permuting $[I]$, with the same private permutations used for the system matrix.

Algorithm 1 LINEARROUNDPERM

Require: The encrypted system matrix $[D]_{pk}$ and a matrix $[I]_{pk}$ with same size as D

Ensure: Both input matrices permuted on rows and columns according to this party's private permutations

- 1: $\pi_r^i \leftarrow$ generate new random permutation with size $m+1$;
 - 2: $\pi_c^i \leftarrow$ generate new random permutation with size $n+1$;
 - 3: $[D']_{pk} \leftarrow \text{MIXANDPERMUTE}([D]_{pk}, \pi_r^i, \pi_c^i)$;
 - 4: $[I']_{pk} \leftarrow \text{MIXANDPERMUTE}([I]_{pk}, \pi_r^i, \pi_c^i)$;
 - 5: **if** not last party **then**
 - 6: **send** $[D']_{pk}$ and $[I']_{pk}$ to the next party;
 - 7: **else**
 - 8: **multicast** $[D']_{pk}$ and $[I']_{pk}$;
 - 9: **end if**
-

Algorithm 2 MIXANDPERMUTE($[A]_{pk}, \pi_r, \pi_c$)

Require: Encrypted matrix $[A]_{pk}$ and two permutation arrays π_r and π_c for rows and columns, respectively

Ensure: The input matrix mixed and permuted according to the given permutations

- 1: **for all** elements in matrix $[A]_{pk}$ **do**
 - 2: $[0]_{pk} \leftarrow \text{Enc}_{pk}(0)$; {random homomorphic encryption of zero}
 - 3: $[A'_{ij}]_{pk} \leftarrow [A_{ij}]_{pk} \oplus [0]_{pk}$; {homomorphic addition}
 - 4: **end for**
 - 5: **for** $0 \geq r \geq m$ **do** {all rows in $[A']_{pk}$ }
 - 6: $[A''_{r,\bullet}] \leftarrow \pi_r([A'_{r,\bullet}])$;
 - 7: **end for**
 - 8: **for** $0 \geq c \geq n$ **do** {all columns in $[A'']_{pk}$ }
 - 9: $[A'''_{\bullet,c}] \leftarrow \pi_c([A''_{\bullet,c}])$;
 - 10: **end for**
 - 11: **return** $[A''']_{pk}$;
-

In the end, the optimal solution is reached (admitting the problem is bounded and such solution exists) and the parties figure out the basic variables of the resulting matrix. However, one side effect of the matrix permutation is that the indexes of the basic variables are also permuted. This means that the parties need to find out the corresponding variables in the initial matrix. That's why the parties permute the second matrix $[I]$ within the Algorithm 1.

In order to retrieve the indexes of the initial system matrix, the parties run Algorithm 3. This is done only once after the optimal solution is found. The input matrix $[I]$ contains the permutations (for rows and columns) that map the initial system matrix to the final permuted matrix $[D]$ resulted from the last iteration of the secure LP.

The first line from Algorithm 3 invokes once again the linear permutation algorithm in order to permute one last time the system matrix. Note that the LP iterations have already finished, which implies in no further index selection or pivot operation. This last invocation intends to keep the permutations applied to the system matrix secret, as in [23]. By permuting one last time, no party is able to infer anything about the indexes of the system matrix and therefore, no extra information can be gained.

The result from line 1 includes the encrypted matrix $[I']_{pk}$ and hence, the overall combination of permutations. In the next step, the parties collaboratively decrypt the elements from such matrix enabling for the extraction of the (final)

Algorithm 3 Index Recovering Protocol for the Linear Round Permutation

Require: Two encrypted matrices $[D]_{pk}$ and $[I]_{pk}$ (with same size), where $[I]_{pk}$ contains the combination of all permutations applied to the initial system matrix

Ensure: The final matrix with indexes restored to the original position

- 1: $[D']_{pk}, [I']_{pk} \leftarrow \text{LINEARROUNDPERM}([D]_{pk}, [I]_{pk})$;
 - 2: $I' \leftarrow \text{DECRYPTMATRIX}([I']_{pk})$;
 - 3: **extract** Π_r and Π_c from I' ;
 - 4: $[D'']_{pk} \leftarrow \Pi_c(\Pi_r([D']_{pk}))$;
 - 5: **return** $[D'']_{pk}$;
-

permutation for rows and columns. The decryption process is conducted according to Algorithm 4, which in turn implements the threshold decryption algorithm described in [8] for the Paillier cryptosystem. Observe that even though the permutations are revealed (clear text after the decryption), they don't leak any extra information, proving to be secure.

Algorithm 4 DECRYPTMATRIX($[A]_{pk}$)

Require: An encrypted matrix $[A]_{pk}$ where $[A_{ij}]_{pk} \in \mathbb{Z}_{pk}$;

Ensure: The plain text corresponding to the given input;

- 1: **for all** i, j from A **do**
 - 2: $A_{ij} \leftarrow \text{DECRYPT}([A_{ij}]_{pk})$; {algorithm from [8]}
 - 3: **end for**
 - 4: **return** A ;
-

Now the parties are able to compute the inverse of the extracted permutations and consequently recover the indexes from the initial system matrix. The output of Algorithm 3 is a matrix containing the last iteration values for the LP in their original positions, allowing for the computation of the LP optimal values. One last remark: considering the overall LP computation, our solution adds one more iteration, in the end, referring to the index recovery protocol.

3.2.1 Analysis

The complexity analysis is divided in three parts: communication, round and computational complexity.

The communication complexity is the sum of all messages sent by one party during the whole protocol (Algorithm 1). The size of the messages from lines 6 and 8 is $2(nm)\mathcal{O}(c)$ bits, where m is the number of rows and n is the number of columns from both matrices $[D]$ and $[I]$. Each party sends one message during the sequential part and the last party sends $(p-1)$ messages for the multicast from line 8, which is the worst case. The overall communication complexity is $2(p-1)(mn)\mathcal{O}(c) = \mathcal{O}(p(mn)c)$ bits.

The round complexity is computed as the sum of the $(p-1)$ sequential messages and the last multicast message from party p . In total, the algorithm takes p rounds to apply the permutation, i.e. $\mathcal{O}(p)$. The MIXANDPERMUTE protocol dominates the computational complexity. In order to mix and permute a matrix, the protocol takes (mn) modular exponentiations plus (mn) modular multiplications. Applying the permutations to the matrices is costless. The all inclusive computational complexity of Algorithm 1, assuming both invocations of MIXANDPERMUTE can be done in parallel, is then $\mathcal{O}(mn)$ modular exponentiations.

At last, we consider the complexities regarding the index recovering algorithm.

Its communication complexity depends on the Algorithm 1 and on the DECRYPT protocol (from [8]) complexities. The decryption of one matrix with size $m \times n$ implies on (mn) singular decryptions. The decryption of one element corresponds to 2 multicast messages with size $\mathcal{O}(c)$ bits, i.e. $2(p-1)\mathcal{O}(c)$ bits. The communication complexity is $2(p-1)(mn)\mathcal{O}(c)$ plus $(2(p-1)(mn)\mathcal{O}(c))$, i.e. $\mathcal{O}(p(mn)c)$.

The round complexity is dominated by the Algorithm 1, since the decryption protocol is constant round. Algorithm 3 takes $(p+2)$ rounds, i.e. $\mathcal{O}(p)$ rounds.

And finally, both Algorithm 1 and the DECRYPT protocol are responsible for the computational complexity of the index recovering protocol. The decryption of one single element takes one modular exponentiation (for the shared decryption process) plus $(t+1)$ modular exponentiations plus $(2t^2 + 3t + 1)$ modular multiplications plus $(t^2 + t)$ modular inverses (for the shared combining process). The overall cost for one decryption is $(t+2)$ modular exponentiations plus $(t^2 + t)$ modular inverses plus $(2t^2 + 3t + 1)$ modular multiplications. Assume the (mn) decryptions (from matrix $[I]$) are executed in parallel. Then, the computational complexity of Algorithm 3 is $\mathcal{O}(mn)$ modular exponentiations and $\mathcal{O}(mn)$ modular multiplications for the permutation, plus $\mathcal{O}(t)$ modular exponentiations, $\mathcal{O}(t^2)$ modular multiplications and $\mathcal{O}(t^2)$ modular inverses for the decryption. Considering the threshold as $t = p - 1$ we have $\mathcal{O}(p + mn)$ modular exponentiations, $\mathcal{O}(p^2 + mn)$ modular multiplications and $\mathcal{O}(p^2)$ modular inverses.

3.3 Logarithmic Round Algorithm

Observing that the round complexity has great impact not only on the communication complexity but also on the absolute running time, we propose a logarithmic round algorithm as an alternative to the linear round complexity approach.

Instead of using the permutation represented as arrays, the protocol uses the matrix representation format as defined in Section 3.1. Notice that by representing permutation as matrices, we are able to compute the final permutation matrix by simply multiplying all the private permutations from the parties. All the (secure) computations performed during the *Log Round Permutation* protocol use encrypted permutation matrices.

The *Log Round Permutation* protocol is then depicted on Algorithm 5. All parties run the protocol in parallel and output the same result: the matrix $[D]$ permuted on rows and columns according to the combination of the permutations from all parties.

As the Algorithm 5 is executed in parallel, so are the multiplication of the elements from all permutation matrices. It can be divided in four main steps: (i) exchange of permutation matrices, where every party multicasts his own matrices and receives the matrices from all other parties; (ii) (secure) logarithmic round multiplication of the permutation matrices; (iii) update of the permutation matrix combination for the current iteration; and (iv) the final secure multiplication of the permutation matrices by the system matrix, producing the desired output. Step (iii) is used to optimize the index recovering protocol.

The improvement of such approach over the linear round algorithm is due to lines 6 and 7, where Algorithm 6 is in-

Algorithm 5 LOGROUNDPERM

Require: The encrypted system matrix $[D]_{pk}$

Ensure: The encrypted system matrix permuted according the composition of the permutations of all parties

- 1: $k \leftarrow$ current iteration number;
 - 2: ${}^k[R]_{pk}^i \leftarrow$ generate random permutation matrix for rows on party i for iteration k ;
 - 3: ${}^k[C]_{pk}^i \leftarrow$ generate random permutation matrix for columns on party i for iteration k ;
 - 4: **multicast** ${}^k[R]_{pk}^i$ and ${}^k[C]_{pk}^i$;
 - 5: **wait** until receiving ${}^k[R]_{pk}^i$ and ${}^k[C]_{pk}^i$ for $1 \geq i \geq p$;
 - 6: $[R]_{pk} \leftarrow$ LOGROUNDMULT(${}^k[R]_{pk}^1, \dots, {}^k[R]_{pk}^p$);
 - 7: $[C]_{pk} \leftarrow$ LOGROUNDMULT(${}^k[C]_{pk}^1, \dots, {}^k[C]_{pk}^p$);
 - 8: ${}^k[R]_{pk} \leftarrow$ MULTIPLYMATRICES($[R]_{pk}, {}^{k-1}[R]_{pk}$);
 - 9: ${}^k[C]_{pk} \leftarrow$ MULTIPLYMATRICES($[C]_{pk}, {}^{k-1}[C]_{pk}$);
 - 10: $[D']_{pk} \leftarrow$ MULTIPLYMATRICES($[R]_{pk}, [D]_{pk}$);
 - 11: $[D'']_{pk} \leftarrow$ MULTIPLYMATRICES($[D']_{pk}, [C^T]_{pk}$);
 - 12: **return** $[D'']_{pk}$;
-

voked (LOGROUNDMULT) in order to combine the private permutations from the parties in a logarithmic base. Algorithm 6 receives an array of permutation matrices as input and considers a binary tree over such array, multiplying pair-wise and updating the array to half of the values at every iteration. For simplicity assume that the input array has a power of two length. The protocol is then *recursively* invoked (line 9) until there is only one matrix left in the array. The output is such unique matrix, that represents the combination of all permutation matrices.

Algorithm 6 LOGROUNDMULT($\{P\}_{pk}$)

Require: An array of encrypted permutation matrices of size p , where p is a power of two: $\{P\}_{pk} = \{[P^0]_{pk}, \dots, [P^{p-1}]_{pk}\}$;

Ensure: An encrypted permutation matrix equal to the multiplication of all the matrices in the input array;

- 1: **if** $p = 1$ **then**
 - 2: **return** $[P^0]_{pk}$;
 - 3: **else**
 - 4: $\{B\}_{pk} \leftarrow \{[B^0]_{pk}, \dots, [B^{(\frac{p}{2}-1)}]_{pk}\}$; *{ create an array of matrices with half the size of the input array }*
 - 5: **for** $i = 0, \dots, (\frac{p}{2} - 1)$ **do**
 - 6: $[B^i]_{pk} \leftarrow$ MULTIPLYMATRICES($[P^{2i}]_{pk}, [P^{2i+1}]_{pk}$)
 - 7: **end for**
 - 8: **end if**
 - 9: **return** LOGROUNDMULT($\{B\}_{pk}$);
-

The standard matrix multiplication idea is used. However, as we are dealing with encrypted data, a slight extension is introduced in order to correctly compute the result. Algorithm 7 depicts how the multiplication is done.

Please note that due to the same reason, the multiplication of two encrypted values is computed according to the secure multiplication protocol proposed by [7] and is denoted as SECUREMULT.

On account of the permutation matrix properties, the index recovering is rather simple: in order to restore the final

Algorithm 7 MULTIPLYMATRICES($[A]_{pk}, [B]_{pk}$)

Require: Two encrypted matrices $[A]_{pk}$ and $[B]_{pk}$ where $[A_{ij}]_{pk}, [B_{ij}]_{pk} \in \mathbb{Z}_{pk} \forall i, j$;

Ensure: An encrypted matrix that is the result of the multiplication of the input matrices;

```
1: initialize  $[C]_{pk}$ 
2: if number of columns of  $A$  = number of rows of  $B$  then
3:   for  $i = 0$  to number of rows of  $A$  do
4:     for  $j = 0$  to number of columns of  $B$  do
5:       for  $k = 0$  to number of columns of  $A$  do
6:          $[C_{ij}]_{pk} \leftarrow [C_{ij}]_{pk} \oplus \text{SECUREMULT}([A_{ik}]_{pk}, [B_{kj}]_{pk})$ ;
7:       end for
8:     end for
9:   end for
10: end if
11: return  $[C]_{pk}$ ;
```

permuted system matrix to the initial index positions we simply computed two secure matrix multiplications.

First observe that on every round the parties update the combination of both rows and columns permutation matrices (lines 8 and 9 from Algorithm 5). Since we are dealing with matrices, such combination is a matter of multiplying the matrix from the current iteration with the last iteration matrix, so that in the end, every party has a final permutation matrix $[R]$ (for rows for example) such that $[R] = {}^k[R] * {}^{k-1}[R] * \dots * {}^0[R]$, where k is the number of iterations of the LP. The same applies for the columns permutation matrix. And two such matrices suffices to compute the overall inverse permutation, i.e. considering the permutation matrix for rows, $[R] * [D] = [D']$, which implies in $[D] = [R^{-1}] * [D']$.

Algorithm 8 Index Recovering Protocol for the Logarithmic Round Permutation

Require: Three encrypted matrices $[D]_{pk}$, ${}^f[R]_{pk}$ and ${}^f[C]_{pk}$ (with same size), where D is the final system matrix, fR and fC are the resulting permutation matrices for rows and columns, respectively, from the last iteration $k = f$

Ensure: The final matrix with indexes restored to the original position

```
1:  $[D']_{pk} \leftarrow \text{MULTIPLYMATRICES}({}^f[R^T]_{pk}, [D]_{pk})$ ;
2:  $[D'']_{pk} \leftarrow \text{MULTIPLYMATRICES}([D']_{pk}, {}^f[C]_{pk})$ ;
3: return  $[D'']_{pk}$ ;
```

Computing the inverse of an encrypted matrix is an extremely complex operation. However, observing that a permutation matrix is an orthogonal matrix by definition, and therefore special properties apply, i.e. $RR^T = I$ (the identity matrix), the inverse matrix exists and can be written as $R^{-1} = R^T$. This fact enables for a simple matrix multiplication solution as presented in Algorithm 8.

3.3.1 Analysis

Again, we divide the complexity analysis in three aspects: round, communication and computational complexity.

Algorithm 5 can be viewed as having three abstract rounds: (i) exchange of permutation matrices (lines 1-5); (ii) the logarithmic round multiplications (lines 6 and 7); and (iii) the

last matrices multiplications (lines 8-11). Abstract rounds because the second is not constant: it depends on the round complexity of the logarithmic round solution. Each secure multiplication protocol invocation takes 5 rounds and Algorithm 6 clearly takes $\mathcal{O}(\log(p))$ rounds. Assuming both invocations of Algorithm 6 are executed in parallel, as well as the four matrix multiplications from lines 8-11, the number of rounds required by Algorithm 5 is $(1 + 5\log(p) + 5)$, which is bounded by $\mathcal{O}(\log(p))$ rounds.

For the communication complexity, consider that all the messages are multicast messages, i.e. one multicast message corresponds to $(p-1)$ messages. Also consider that the messages are sent in parallel and have different sizes according to its type, and that both permutation matrices are of size $m \times n$ with $m = n = a$.

On line 4, each party multicasts both matrices, i.e. communication complexity of $(p-1)2a^2\mathcal{O}(c)$ bits. Lines 6 and 7 correspond to the same operation: for an array of size p , the LOGROUNDMULT protocol executes $(p-1)$ matrix multiplications. For each pair of matrices being multiplied, the MULTIPLYMATRICES protocol is invoked and a^3 secure multiplications are performed. For each secure multiplication, four multicast messages of size $\mathcal{O}(c)$ bits plus one multicast message containing a clear text integer (32 bits, i.e. constant) are exchanged. The communication complexity of the operations from lines 6 and 7 is bounded by $2(p-1)\{a^3[(p-1)(4\mathcal{O}(c) + \mathcal{O}(1))]\}$ bits. Lines 8-11 correspond to four parallel invocations of the MULTIPLYMATRICES protocol, exchanging $a^3[(p-1)(4\mathcal{O}(c) + \mathcal{O}(1))]$ bits. As being so, the complete communication complexity is the combination of the such complexities and hence, is bounded by $\mathcal{O}(p^2a^3c)$ bits.

For the computational complexity we consider three main operations: (i) permutation matrices generation, (ii) secure multiplication (pair-wise) and (iii) decryption of a single element, which is included in a secure multiplication.

- (i) requires $2a^2$ modular exponentiations;
- (ii) requires three modular exponentiations (one homomorphic encryption and two multiplications by constants) plus $(2p+3)$ modular multiplications (regarding the homomorphic additions) plus 2 decryptions;
- (iii) requires $(t+2)$ modular exponentiations plus (t^2+t) modular inverses plus $(2t^2+3t+1)$ modular multiplications (as described before);

The protocol requires $(p-1)$ matrix multiplications for the LOGROUNDMULT invocation and one matrix multiplication for the lines 8-11 (all the operations are executed in parallel). Assume square matrices of size axa , which means that to compute the product of two matrices, a^3 single secure multiplications are needed. The computational complexity is clearly dominated by the SECUREMULT protocol and it is bounded by $\mathcal{O}(p^2a^3)$ modular exponentiations plus $\mathcal{O}(p^3a^3)$ modular multiplications plus $\mathcal{O}(p^3a^3)$ modular inversions.

The index recovering protocol is constant round, i.e. one round for each matrix multiplication and five rounds for the SECUREMULT protocol invocations which are performed in parallel.

Since all computations are local, there is no communication complexity.

The computational complexity is dominated again by the complexity of the SECUREMULT protocol which depends on

the DECRYPT protocol. Considering again, $t = p - 1$, the computational cost of the index recovering protocol is $\mathcal{O}(pa^3)$ modular exponentiations.

3.4 Results and Considerations

In order to validate the proposed algorithms, a series of Monte Carlo experiments were conducted. The implementation is Java¹ based. Concurrent programming techniques were applied in order to reduce waiting periods and to optimize the use of CPU and memory. A total of 16 machines with 2.4GHz processor (dual core) and 3GB RAM memory were used.

The setup included keys provided by an implementation of the Paillier Threshold Homomorphic Cryptosystem – CS_s^t – where t is the threshold and s is the integer that defines the order of the modular exponentiations and hence, the order of the plain/cipher texts (as proposed in [8]). The keys size is 1024 bits and $s = 2$, i.e. the plain text size is $plain \in \mathbb{Z}_{mod^2}$ and the cipher text is $c \in \mathbb{Z}_{mod^3}$, where mod is the modulo of the cryptosystem.

The main point is to evaluate the permutation algorithms regarding the influence of the number of parties and rounds, in a real SCMP application, where the parties are eventually geographically dispersed.

The first results comprise the absolute running time taken by each of the secure operations performed, namely encryption, threshold decryption and multiplication. Regarding the size of the input, since the encryption process includes a modular exponentiation on the size of the plain text, variations on the latter are important. The plain text input for the encryption tests included random numbers with size varying from 2^1 to 2^{10} bits. The results are depicted on Figure 2.

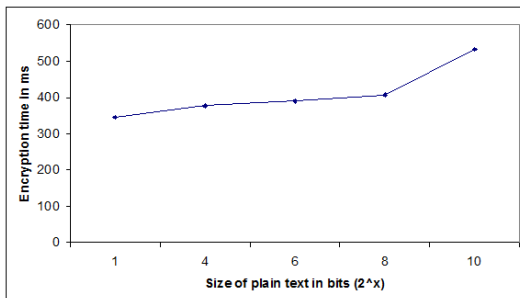


Figure 2: Encryption time for different plain text sizes.

Note that the time spent for encryption is limited above by the value of the modular exponentiations on the ring \mathbb{Z}_{mod^s} , i.e. approx. 2050 bits in this case.

As for the other operations, the input is a cipher text, which's size is bounded by the size of the modular exponentiations on the ring $\mathbb{Z}_{mod^{s+1}}$ defined by the cryptosystem. For a 1024 bits key, the cipher text size is approx. 3075 bits.

Observe that for the decryption of a given cipher text, the protocol combines exactly $t + 1$ secret shares, where $t = p/2$, which happens to be the minimum number of requested shares. Although only $t + 1$ shares are combined, each party eventually receives all $(p - 1)$ shares for every value that is going to be decrypted. In practice, for the 8 parties case,

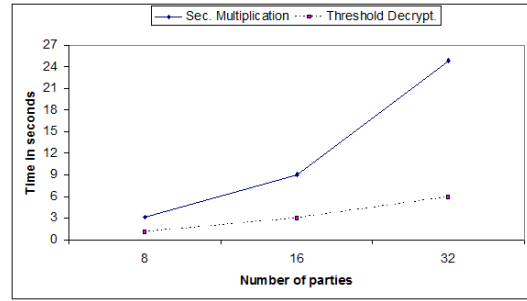


Figure 3: Absolute running time for decryption and multiplication of cipher texts.

only the first 4 received shares are combined, and for the 32 parties case, the first 16 shares are combined. This is done in order to save computational effort, since share combining involves $\mathcal{O}(t + 1)$ modular exponentiations.

Notice the implication of that fact when considering the semi-honest security model where the threshold is defined as $t = (p - 1)$. In such model, each party would have to combine all p shares, which would considerably increase the absolute running time of one single decryption.

How dispersed the parties are is another fact that must be taken into account. In a real SCMP scenario the parties are often geographically separated and all the communication is conducted over the Internet, for example. Note the implications of such fact: parties located in different sites carrying out secure computations (based on messages exchange) may be considerably affected by the delay imposed by the network.

Moreover, a delay in the communication would strongly influence the absolute running time of the protocol. The influence is directly proportional to the number of rounds taken by the protocol, since the rounds are counted based on the messages, i.e. sequential or multicast messages.

Having this important observation in mind, we propose the logarithmic round algorithm for the permutation procedure in the SCMP problem. The most significant realization is the reduction of the round number from a linear base to a logarithmic base, which in real applications can bring considerable improvement on the absolute running time and/or optimization of CPU/memory usage. Naturally, there's a price to be paid when reducing the number of rounds due to the introduction of more complex computations.

Consider the relation between the amount of time spent due to the delay in the communication and the time spent on the actual computations. The linear round algorithm has computational complexity of $\mathcal{O}(mn)$ modular exponentiations, i.e. encryptions, and the party 0 (for example) has to wait for the $(p - 1)$ parties to apply their permutations, so that he can continue with the LP. Assume a delay d on the network, the time spent for an encryption enc_{time} (from Figure 2) and a large number of parties. It is possible to have $(mn * enc_{time} \ll (p + 1) * d)$, reflecting how fragile is the linear round solution concerning a network delay: the parties will be idle most part of the time, waiting for messages. This is not desirable and directly affects the absolute running time of the protocol

The last discussion regards the communication cost. Although such measurement is not affected by an eventual de-

¹Version 1.5 – <http://java.sun.com/javase/>

lay, the point here is to depict the difference between both permutation approaches. While the logarithmic round algorithm saves rounds and therefore reduce the influence of the delay (e.g. regarding to absolute running time), it also introduces communication overhead that must be considered for real applications.

First measurement comprises the inherent overhead introduced by the RMI *infrastructure* itself. By observing the fact that all messages exchanged by the parties contain encrypted values, a primary estimation of the RMI overhead is straightforward: consider a cipher text size around 384 bytes and the simplest scenario with two parties permuting a system matrix of size 2×2 . The cipher text size is defined by the size of the plain text, i.e. integers in this case, with maximal value bounded by the modulo of the cryptosystem.

The theoretical communication cost of the Sequential Algorithm (Algorithm 1) is bounded by $\mathcal{O}(p(mn)c) = \mathcal{O}(2(2 * 2)384) = \mathcal{O}(3072)$ bytes. Each party send three messages with approximately the same size, i.e. $3 * (4 * 384) = 4608$ bytes. In practice, such scenario results in a communication cost of approximately 8172 bytes, due to the following reasons: (i) overhead of the RMI for each message, and (ii) the *lookup* message required for each invocation with practical size of approx. 369 bytes which is on the same size of the cipher text, for example.

Next, consider the communication cost of the Logarithmic Round Algorithm (Algorithm 5) for the same simple scenario depicted above. The communication complexity of Algorithm 5 is bounded by $\mathcal{O}(p^2 a^3 c) = \mathcal{O}(2^2 * 2^3 * 384) = \mathcal{O}(12288)$ bytes. Looking closely to the messages being exchanged, each party sends one message with the permutation matrices (2 matrices with 4 encrypted values each) of size $2 * 4 * 384 = 3072$ bytes. Later, the multiplication algorithm takes $2^3(2 * 384 + 3 * 384) = 15360$ bytes for each of permutation matrices (row and column). The overall communication cost is then 18432 bytes for each party. The experiments revealed that RMI as well as the data format used to represent the encrypted matrices introduce a considerable overhead on the communication. The first message, with the encrypted permutation matrices, costs 4504 bytes plus 369 bytes for the RMI lookup. During the multiplication procedure, the initial message has size $1405 + 369$ bytes and the last three have size $934 + 369$ bytes. In total, the practical communication cost for the logarithmic round algorithm is around 50337 bytes from which 12177 bytes are from the *lookup* calls.

Finally, Table 1 summarizes the overall complexities (in big- \mathcal{O} notation) for both permutation solutions. Assume $m = n = a$ and $t = p/2$ for all cases. The computational complexity includes only the modular exponentiations which are the most expensive operations.

Solution	Comm.	Round	Comp.
Linear	$\mathcal{O}(pa^2c)$	$\mathcal{O}(p)$	$\mathcal{O}(a^2)$
Logarithmic	$\mathcal{O}(p^2a^3c)$	$\mathcal{O}(\log(p))$	$\mathcal{O}(p^2a^3)$

Table 1: Complexities comparison between both permutation solutions for the multi-party case

Next, we summarize the indexing recovering protocols' complexities for both solutions (Table 2) under the same assumptions. The results showed how the implementation influences the practical costs, i.e. RMI overhead plus Java

Solution	Comm.	Round	Comp.
Linear	$\mathcal{O}(pa^2c)$	$\mathcal{O}(p)$	$\mathcal{O}(p + a^2)$
Logarithmic	-	$\mathcal{O}(1)$	$\mathcal{O}(pa^3)$

Table 2: Index recovering protocols' complexities comparison

objects serialization, and how much extra communication is introduced by using the logarithmic round permutation approach. In the end, the decision on which approach is the best depends on the nature of the problem: how many parties and how dispersed are they as well as the size of the system matrix.

4. OTHER SCENARIOS

There are many other scenarios than supply chain optimization for SMC in business applications. Many of those consists of collaborative applications that one company would not be able to execute alone.

The first application we studied well is collaborative benchmarking. In benchmarking a group of companies computes statistics about their key performance indicators (KPI). These KPIs can come from any branch of the company, e.g. financial (cash flow), production (machine cycle time) or human resources (employee fluctuation rate). SMC enables the computation of the statistics without revealing the KPI to anyone else.

We constructed protocols for securely computing these statistics in a centralized fashion [17]. These protocols can then be offered as a service, since every client only needs to communicate with the server. We also constructed a peer group formation algorithm that groups a set of companies into useful peer groups [15]. This enables building an enterprise information system over the SMC protocol. Finally we implemented the protocols over web services [18]. The interesting result is that the protocols can be implemented, such that communication cost is negligible.

The second application is collaborative fraud detection: a number of parties combine their log files in order to identify fraud cases they could not detect themselves. The main obstacle for collaborative fraud detection is time-based event correlation.

We present an algorithm for pseudonymizing timestamps, such that a trusted third can compare them if they are close, i.e. within a certain distance [16]. The problem is that any such algorithm is not secure, if the set of timestamps is dense, such that one can build clusters of adjacent timestamps. Furthermore timestamps require synchronized clocks to operate correctly which is a too strong requirement in many scenarios. It is therefore better to use logical clocks, such as vector clocks or Lamport's clocks. We present the first privacy-preserving solution for these clocks in [21].

Privacy can also be motivated by legislation, as in the case of collaborative social network analysis for criminal investigations. We present an algorithm to perform the most important metrics computation on a distributed social network without exchanging the information [20].

In this workshop we also present a scheme for remote auditing [22] In remote auditing an auditor wants to preserve the confidentiality of its queries and the audited wants to preserve the confidentiality of his database. There are more cases, such as auctions [3] or name correlation [13].

5. RELATION TO SERVICES

The trend towards service-oriented architecture (SOA) is strongly moving the industry. Future business software will be composed of services. So we can ask ourselves a few questions about the integration of service and SMC.

5.1 Encapsulation of Services

Services are seen as the main ingredient of SOA. A service is a self-contained, loosely coupled entity offered over the network. One of the main advantages is e.g. that is possible to combine services without the need to watch for distributed transactions.

SMC, on the other hand, is strongly coupled. No interaction makes sense without the others; interruptions and diversions are not foreseen. Is it possible to derive security services for composable SMC?

5.2 Web Services as Transportation

It is possible to implement SMC over web services as an additional layer. We have done so [18], but the implementation costs in terms of memory and performance are huge [14]. Given that no other benefits of web services, such as late binding, can be used, the penalty can be prohibitive.

5.3 Web Service Support for Security Against the Business Partners

The main security mechanisms for web services (standards WS-Security, etc.) target security against outsiders. There is no security mechanism for security against business partners; not even for the “simple” cases, such as reputation.

6. CONCLUSION

We argued that there is a growing trend towards security against the business partner. So far, only secure multi-party computation, seems to provide security guarantees.

We gave the example of secure supply chain management. It is an example where it is imperative to protect the data and not disclose it in any form. SMC can provide that security guarantee.

Secure supply chain management is an optimization problem (linear programming) and we compared the complexities of the important sub-operation of multi-party permutation. We gave a detailed analysis of two algorithms, their complexities and practical performance. In particular we estimated the effects of the different complexities on practical performance. We showed that the solution with the best round complexity is not necessarily the best solution overall.

We then showed several other examples: privacy-preserving benchmarking, collaborative fraud detection, social network analysis and remote auditing.

We then briefly compared the approaches of SOA and SMC. Our conclusion is that there is much research work to be done to secure against the business partner in cross-organizational SOA.

7. ACKNOWLEDGEMENTS

The developments presented in this paper were partly funded by the European Commission through the ICT programme under Framework 7 grant FP7-213531 to the SecureSCM project.

8. REFERENCES

- [1] BAKER, W., HYLENDER, D., AND VALENTINE, A. 2008 Data Breach Investigations Report. Available at <http://www.verizonbusiness.com/resources/security/databreachreport.pdf>, 2008.
- [2] BEN-OR, M., GOLDWASSER, S., AND WIGDERSON, A. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the 20th annual ACM symposium on Theory of computing*, 1988.
- [3] BOGETOFT, P., CHRISTENSEN, D., DAMGARD, I., GEISLER, M., JAKOBSEN, T., KROIGAARD, M., NIELSEN, J., NIELSEN, J., NIELSEN, K., PAGTER, J., SCHWARTZBACH, M., AND TOFT, T. Multiparty Computation Goes Live. Available at <http://eprint.iacr.org/2008/068>, 2008.
- [4] BONA, M. *Combinatorics of Permutations*. Chapman Hall-CRC, 2004.
- [5] CAMENISCH, J., AND VAN HERREWEGHEN, E. Design and Implementation of the idemix Anonymous Credential System. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, 2002.
- [6] CLARK, A., AND SCARF, H. Optimal policies for a multi-echelon inventory problem. *Management Science* 6(4), 1960.
- [7] CRAMER, R., DAMGARD, I., AND NIELSEN, J. B. Multiparty computation from threshold homomorphic encryption. In *Proceedings of Eurocrypt*, 2001.
- [8] DAMGARD, I., AND JURIK, M. A Generalisation, a Simplification and Some Applications of Paillier’s Probabilistic Public-Key System. In *Proceedings of 4th International Workshop on Practice and Theory in Public Key Cryptography*, 2001.
- [9] DANTZIG, G. B. *Linear Programming and Extensions*. Princeton University Press, 1963.
- [10] DEITOS, R., KERSCHBAUM, F., ROBINSON, P., AND HALLER, J. A comprehensive security architecture for dynamic, web service based virtual organizations for businesses. *Proceedings of the 3rd ACM Workshop On Secure Web Services*, 2006.
- [11] DINGLEDINE, R., MATHEWSON, N., AND SYVERSON, P. Tor: The Second Generation Onion Router. *Proceedings of USENIX Security Symposium*, 2004.
- [12] GOLDREICH, O., MICALI, S., AND WIGDERSON, A. How to play any mental game. *Proceedings of the 19th annual ACM conference on Theory of computing*, 1987.
- [13] IBM. IBM Anonymous Resolution Version 4.1 Technical Information. See <http://ibm.com/db2/eas/>, 2006.
- [14] JURIC, M., ROZMAN, I. BRUMEN, B., COLNARIC, M., AND HERICKO, M. Comparison of Performance of Web Services, WS-Security, RMI, and RMI-SSL. *Journal of Systems and Software* 79(5), 2006.
- [15] KERSCHBAUM, F. Building A Privacy-Preserving Benchmarking Enterprise System. In *Proceedings of the 11th IEEE International EDOC Conference*, 2007.
- [16] KERSCHBAUM, F. Distance-Preserving Pseudonymization for Timestamps and Spatial Data. In *Proceedings of the ACM Workshop on Privacy in the Electronic Society*, 2007.

- [17] KERSCHBAUM, F. Practical Privacy-Preserving Benchmarking. *In Proceedings of the 23rd IFIP International Information Security Conference*, 2008.
- [18] KERSCHBAUM, F., DAHLMEIER, D., SCHRÖPFER, A., AND BISWAS, D. An Experimental Study on the Practical Importance of Communication Complexity for Secure Multi-Party Computation Protocols. *SAP Internal Technical Report*, 2008.
- [19] KERSCHBAUM, F., HALLER, J., KARABULUT, Y., AND ROBINSON, P. PathTrust: A Trust-Based Reputation Service for Virtual Organization Formation. *In Proceedings of the 4th International Conference on Trust Management*, 2006.
- [20] KERSCHBAUM, F. AND SCHAAD, A. Privacy-Preserving Social Network Analysis for Criminal Investigations. *In Proceedings of the ACM Workshop on Privacy in the Electronic Society*, 2008.
- [21] KERSCHBAUM, F., AND VAYSSIÈRE, J. Privacy-Preserving Logical Vector Clocks using Secure Computation Techniques. *In Proceedings of the 13th International Conference on Parallel and Distributed Systems*, 2007.
- [22] KERSCHBAUM, F., AND VAYSSIÈRE, J. Privacy-Preserving Data Analytics as an Outsourced Service. *In Proceedings of the ACM Workshop on Secure Web Services*, 2008.
- [23] LI, J., AND ATALLAH, M. J. Secure and private collaborative linear programming. *In Proceedings of International Conference on Collaborative Computing*, 2006.
- [24] PADMANABHAN, H., AND WHANG, S. Information distortion in a supply chain. *Management Science* 43(4), 1997.
- [25] PAILLIER, P. Public-key cryptosystems based on composite degree residuosity classes. *In Proceedings of Eurocrypt*, 1999.
- [26] PIBERNIK, R., AND SUCKY, E. Centralised and decentralised supply chain planning. *International Journal of Integrated Supply Management* 2(1/2), 2006.
- [27] SANDHU, R., COYNE, E., FEINSTEIN, H., AND YOUMAN, C. Role Based Access Control Models. *IEEE Computer* 29(2), 1996.
- [28] TOFT, T. Primitives and Applications for Multi-Party Computation. PhD. Thesis of the University of Aarhus, 2007.
- [29] YAO, A. Protocols for Secure Computations. *In Proceedings of the annual IEEE Symposium on Foundations of Computer Science*, 1982.