Project N°: **FP7-284731**

Project Acronym: **UaESMC**

Project Title: **Usable and Efficient Secure Multiparty Computation**

Instrument: **Specific Targeted Research Project**

Scheme: **Information & Communication Technologies**

Future and Emerging Technologies (FET-Open)

Deliverable D5.2.3
Scientific Progress Analysis and Recommendations

Due date of deliverable: 31st July 2015

Actual submission date: 31st July 2015

Start date of the project: **1st February 2012** Duration: **42 months**

Organisation name of lead contractor for this deliverable: **CYB**

Specific Targeted Research Project supported by the 7th Framework Programme of the EC		
Dissemination level		
PU	Public	✓
PP	Restricted to other programme participants (including Commission Services)	
RE	Restricted to a group specified by the consortium (including Commission Services)	
CO	Confidential, only for members of the consortium (including Commission Services)	

Executive Summary:
Scientific Progress Analysis and Recommendations

This document summarizes deliverable D5.2.3 of project FP7-284731 (UaESMC), a Specific Targeted Research Project supported by the 7th Framework Programme of the EC within the FET-Open (Future and Emerging Technologies) scheme. Full information on this project, including the contents of this deliverable, is available online at `http://www.usable-security.eu`.

In this document we report on the current status of the various secure multiparty computation techniques we have investigated in the UaESMC project. We give benchmarking results wherever possible, and describe the theoretical and practical significance of the results we have achieved and are yet planning to achieve. We find that we have made significant progress in different kinds of techniques, filling the gaps identified in previous periods, and these techniques have a good chance of combining into a full-fledged theoretical framework for privacy-preserving computations of very different kinds.

List of Authors

Yiannis Giannakopoulos (UoA)
Roberto Guanciale (KTH)
Liina Kamm (CYB)
Peeter Laud (CYB)
Alisa Pankova (CYB)
Martin Pettai (CYB)
Pille Pullonen (CYB)
Sander Siim (CYB)
Yiannis Tselekounis (UoA)

Contents

Chapter 1

Introduction

The goal of UaESMC is to increase the use of secure multiparty computation (SMC) techniques both in numbers and in variety. The project works toward this goal by looking for real-life problems that could most benefit from SMC techniques, by determining the reasons why these techniques are not used, by proposing solutions that overcome these reasons, and by demostrating the usefulness of these solutions.

During the last reporting period of UaESMC, our main focus has been on proposing privacy-preserving protocols for many different computational problems, trying to achieve a wide variety in terms of the kinds of data and the structure of computations. We have worked on solving linear equation systems, on discrete algorithms working on graphs, strings, or state-transition automata, and on algorithms that select the representation of data based on its characteristics. We have also worked on increasing the privacy of the results of a statistical study by converting it to a differentially private mechanism, while still getting the computational privacy benefits of SMC. Several of our applications have benefitted from our novel protocols for reading or writing to an array according to a private index. Besides applications we have also continued the study of novel methods for building SMC protocols, in particularly making the protocols secure against stronger than passive adversaries.

We have also continued to study the limits of certain techniques, both for privacy-preserving computations and for certain mechanisms to be executed on top of it. We describe our results in detail in the respective deliverables [29, 21, 25], but an overview of them and their relationships is given in the next chapter.

We note the importance that the Bitcoin infrastructure [46] (or in general: the infrastructure provided by ledger-based cryptocurrencies) has started to play in strengthening the properties of SMC. The availability of such infrastructure gives a common reference point for the parties in a multiparty protocol, giving them a functionality for Byzantine agreement, as well as for incentives to faithfully participate in the protocol. While the full impact of such infrastructure on SMC and other cryptographic protocols will be mapped out in future scientific results, we have made contributions in UaESMC by investigating the fairness guarantees the infrastructure can give, as well as provide a game-theoretic robustness analysis of core Bitcoin protocols.

Chapter 2

Progress during third reporting period

2.1 Limits of private computation

2.1.1 Transformation-based private outsourcing of linear equation systems

Linear programming was selected as one of the problems that the work in UaESMC should concentrate on. We were particularly interested in privacy-preserving protocols for linear programming where the privacy is achieved through problem transformation. Using this approach, the original problem is scrambled in the private domain, and the resulting linear programming problem is made public. The published problem is solved using conventional algorithms, and the result is again unscrambled in the private domain, using the randomness generated during scrambling. Hopefully, this is more efficient than directly solving the problem in the private domain. Also, the scrambling transformation must be such, that not much about the initial problem is leaked through the scrambled problem, even to a party that knows a part of the formulation of the initial problem. A number of problems from linear algebra, including the solving of systems of linear equations *over finite fields*, are amenable to privacy preservation through problem transformation.

In previous year, we have found that the existing scrambling methods for linear programs are flawed [38] and improving them would be difficult [39]. Hence we planned to study novel methods of different kind for solving LP in privacy-preserving manner. Both the simplex method and interior point methods are iterative, hence they are not a perfect match for the secret-sharing based basic SMC protocols offered by the Sharemind SMC platform that we use in our work. Interior point methods usually perform less iterations, but each iteration is more complex and involves solving a system of linear equations over real numbers. We hoped that these systems could be privacy-preservingly solved by some transformation-based method. In particular, we asked whether a non-singular matrix over reals could be inverted in this manner.

We have obtained both positive and negative results. For scrambling the input matrix, we have considered random affine transformations. We have shown that the most general usable affine transformation is the one, where the scrambler generates two random non-singular matrices of the same size, and multiplies the input matrix with them from both sides. We have also shown that it is insufficient to multiply just from one side — if such scrambling were privacy-preserving for some set of possible input matrices, then they are of a special form that is easier to invert than to scramble. It is also likely that the same holds when we multiply from both sides, but we have not yet obtained a general result for this case.

The impossibility results named in previous paragraph only apply for perfect security. If only statistical hiding of the input matrix is desired, then, for certain sets of possible input matrices, there exist scrambling methods that are privacy-preserving, yet useful. We discuss them in Sec. 2.3.1.2.

We also note that the scrambled matrix definitely leaks the absolute value of the determinant of the original matrix. Hence this piece of information must be non-sensitive.

2.1.2 Bitcoin: A Game Theoretic Approach

One of the major concerns of the UaESMC project is the construction of SMC protocols that employ game-theoretic assumptions instead of using heavy cryptographic assumptions. With the advent of the Bitcoin protocol [46], SMC can be implemented using Bitcoin's global transaction ledger, while guaranteeing fairness in the presence of dishonest majority ([3, 4, 7, 34, 33]). This is achieved using a compensation mechanism which imposes fines to the parties that deviate from the prescribed protocol.

The application of ideas from the game-theoretic context in SMC provides robust SMC protocols which are built on top of Bitcoin. For this reason, analysing Bitcoin in the game-theoretic context is mandatory. Our ongoing work [32] aims to provide a game-theoretic modeling and analysis of the Bitcoin protocol, motivated by [20], in which the authors show that Bitcoin is not incentive compatible, i.e., participants that do not disclose information related to their actions, as it is dictated by the protocol, they receive higher payoff than following the protocol. Their main assumption is that participants are split into two groups. Those assumed to be the honest majority following the protocol, and a minority group that follows a strategy called "Selfish mining". In [32] we provide a game-theoretic model for Bitcoin, and we aim to analyze the circumstances under which following the protocol is a Nash equilibrium, while having no assumption on the network structure.

2.2 Constructing SMC protocols

2.2.1 From privacy to security

We construct larger SMC protocols for various computational tasks from smaller ones, typically for additions, multiplications, and other arithmetic and relational operations. For the security proofs of the large SMC protocols to be feasible to construct, the security properties have to be *compositional*, such that the security of the smaller protocols, and their black-box use by the large protocol, directly implies the security of the large protocol. In the protocol set of Sharemind, the small protocols often perform some extra steps for the purpose of satisfying composable security properties. It has been known that sometimes, some of these steps may be left out in composed protocols, without breaking their security.

We have studied [12] the simulation-based security of Sharemind's protocols and have identified a weaker property, called *input privacy*, which is often sufficient for the construction of large SMC protocols from smaller components. We say that a protocol is *input-private* if all that a party learns could be deduced from its inputs. This is weaker than simulatable security, which also requires that a party's output in the real protocol should be indistinguishable from its output if an ideal component had been used instead [14]. We have found that input privacy can also be stated in the UC framework, and it is composable, at least in the forms that we need for constructing large SMC protocols. Another important property is, that the sequential composition of an input-private protocol and a secure protocol is again secure. This allows us to construct the whole protocol for a sizable computational task using input-private building blocks, and perform some extra steps for security only at the end. The efficiency gain of this approach is significant for protocols with many intermediate steps.

Our current results only consider security against honest-but-curious adversaries. The notion of input privacy readily generalizes to malicious adversaries as well, and so does the composition theorem for input-private protocols. A full generalization of our results to malicious adversaries still needs a good formalization of correctness of protocols that have not yet delivered their outputs.

2.2.2 Verifiable protocols with preprocessing

SMC protocols secure against honest-but-curious adversaries are easier to construct than protocols secure against malicious adversaries. The first kind of protocols are typically also much more efficient in practice (by at least a couple of orders of magnitude), if considering the entire required computational and communicational effort. There are protocol sets, e.g. SPDZ [16] secure against malicious adversaries, that can deliver the outputs of a computation from its inputs with the effort similar to best honest-but-curious protocols,

but they also have a complex offline preprocessing stage that dominates the computations. Hence there has been interest in security against adversaries of intermediate power. We have considered *verifiable* protocols. These are protocols that catch the deviating, adversarially controlled parties after the end of its run, in a postprocessing phase.

We have previously presented a verification mechanism based on linear probabilistically checkable proofs [40]. In this mechanism, the majority of the protocol participants had to be honest, such that a sufficient number of parties flagging a particular party as dishonest is a proof of its dishonesty. The mechanism required the protocol participants to compute new outgoing messages from the existing ones using arithmetic circuits over a finite field. Post-execution, the parties secret-shared all their incoming and outgoing messages among others, and the correctness of computation could be verified.

We have now proposed a different verification mechanism [41], where the internal computations can be done over any finite ring, and actually several rings can be simultaneously used. This is a much better match for the protocol set of Sharemind that uses secret sharing over rings $\mathbb{Z}_{2^n}$. The post-execution effort is also much smaller thanks to a new preprocessing phase. In this phase, a sufficient number of *multiplication verification triples* are generated. This already verified triple can be used to verify that a party correctly performed a multiplication during the execution phase. The generation is similar to SPDZ that also generates multiplication triples. However, the use of these triples is rather different, with SPDZ using them to multiply secret-shared values during the execution. We also differ in the computational complexity of generating the triples — verification triple generation is much easier due to their content being known to the party whose behaviour will be verified.

We thus have a mechanism for turning passively secure (SMC) protocols into protocols secure against adversaries performing active attacks, who want to avoid being caught. The mechanism only negligibly increases the effort necessary during the execution phase of the protocol. It adds a post-execution phase, which is probably not more than an order of magnitude more complex than the execution phase. It also adds a pre-execution phase which requires a similar level of effort. The implementation of the proposed mechanisms is subject of work to be performed in near future.

2.2.3 Bitcoins for fair SMC

Inspired by the development of the field of Mechanism Design [45, 47] within Game Theory [51], that in the view of the seminal impossibility results of Arrow [5] and Gibbard-Satterthwaite [22, 50] provided a rich and powerful tool to implement social choice rules via the use of *monetary transfers*, we try to make a step towards an analogous development of a formal model for the deployment of cryptocurrencies to circumvent the classic impossibility cryptographic results of [15].

Our contribution in [33] is three-fold. Firstly, we put forth a new formal model of SMC with compensation and we show how the introduction of a suitable ledger and synchronization functionalities make it possible to express completely such protocols using *standard interactive Turing machines (ITM)*, circumventing the need for the use of extra features that are outside the standard model (in comparison, the only previous model [7] resorted to specialized ITM-s that utilize resources outside the computational model). Secondly, our model is expressed in the global universal composition setting and is equipped with a composition theorem that enables the design of protocols that *compose safely* with each other and within larger environments, where other protocols with compensation take place; a composition theorem for MPC protocols with compensation was not known before. And finally, we introduce the first robust MPC protocol with compensation, i.e. an MPC protocol where not only fairness is guaranteed (via compensation) but additionally the protocol is guaranteed to deliver output to the parties that get engaged and therefore the adversary, after an initial round of deposits, is not even able to mount a denial of service attack without having to suffer a *monetary penalty*. Importantly, our robust MPC protocol requires only a *constant number* of (coin-transfer and communication) rounds.

Num. variables.	Cramer	Gaussian elim.	LU decomp
2	0.6 s		
3	1.1 s		
4		2.9 s	3.1 s
7		9.4 s	8.7 s
10		21.5 s	19.1 s

Table 2.1: Performance of linear regression on 10000-element arrays (in seconds)

2.3 SMC Applications

2.3.1 Solving systems of linear equations

Privacy-preserving statistics has been one of the main foci of UaESMC. The statistical methods for discovering relationships between variables often require systems of linear equations to be solved. Hence we have studied privacy-preserving methods for solving these systems, where the coefficients (real numbers) are secret-shared among the computing parties, and the solution vector has to be obtained in a similarly secret-shared manner. We are interested in solving systems where the number of equations and variables is equal. If such systems are non-singular, then they have exactly one solution. We have studied both the translation of known algorithms into SMC protocols, as well as transformation-based methods.

2.3.1.1 Numeric methods

For small systems (up to 3 variables) we use Cramer's rule. We have implemented the computations of necessary determinants in the private domain. For these computations, as well as for the methods described below, we build on the representations and the set of protocols we have for expressing and computing on floating-point values on the private domain [30].

For larger systems, we have shown how the methods of Gaussian elimination and LU decomposition can be adapted to run on top of a SMC framework. Both methods heavily depend on pivoting that should take place around an element with the largest absolute value, in order to minimize numerical instabilities. We have thus proposed a method for finding this element among a vector of elements. We do not want to leak the position of this value, but nonetheless use the index of that position in subsequent pivoting. This would normally require techniques for oblivious array access (Sec. 2.3.2), but in our case, we fortunately can hide the position by permuting the columns of the matrix of the system of equations.

We have implemented the protocols for solving systems of linear equations on top of the Sharemind SMC framework, and have included them in the Rmind tool for privacy-preserving statistical analysis [11] as part of the protocol for linear regression. We tested the performance of the protocols on a Sharemind installation running on three computers with 3 GHz 6-core Intel CPUs with 8 GB RAM per core (a total of 48 GB RAM). The computers were connected using gigabit ethernet network interfaces. The running times of linear regression on 10000-element arrays, using either Gaussian elimination or LU decomposition, are given in Table. 2.1.

2.3.1.2 Transformation-based methods

In Sec. 2.1.1 we describe the principles of transformation-based outsourcing and its use in SMC protocols. We are interested whether the inversion of a matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ can be outsourced in a privacy-preserving manner. We have shown that in general, it is impossible. However, it is possible if the set $\mathcal{A}$ of possible matrices $\mathbf{A}$ has been restricted in some manner.

We have shown that for certain classes of matrices $\mathcal{A}$, we can define the distribution of matrices $\mathbf{P}, \mathbf{Q} \in \mathbb{R}^{n \times n}$ so, that the product $\mathbf{PAQ}$ *statistically* hides $\mathbf{A}$, while the entries of $\mathbf{P}$ and $\mathbf{Q}$ are not too much larger than the entries of $\mathbf{A}$. We require that for a certain matrix decomposition (QR-, or LU-, or singular value,

or eigenvalue, etc.), all $\mathbf{A} \in \mathcal{A}$ are such, that in the decomposed form, the entries on the main diagonal of (some of) the matrices in the decomposition are certain constants. Then we can statistically hide the component matrices by multiplying them from left or from right. This hiding is based on our ability to perfectly hide orthogonal matrices, and to statistically hide triangular matrices with fixed main diagonal by multiplying them with a suitably distributed matrix from one side only.

2.3.2 Parallel Oblivious Array Access

Suppose we have a private array of length n and a private index. We want to obtain a private representation of the array element under that index. This is a non-trivial task. The most straightforward privacy-preserving solution involves expanding the index into its characteristic vector of length n, and computing its scalar product of the original array, for a total work of $O(n)$, or $O(n)$ overhead compared to public look-up. Using the techniques of oblivious RAM [42, 31], the overhead can be reduced to around $O(\log^3 n)$. Similar overhead is associated with updating a private array at a private position.

Our secret-sharing based SMC protocols require significant parallelism from the higher-level applications in order to implement them efficiently. In this case, many accesses of a private array can also be assumed to take place in parallel. In [36] we give protocols for reading m values in parallel, or writing m values in parallel to an array of length n. Both our protocols have the asymptotic complexity $O((m+n)\log(m+n))$, or $O((1+n/m)\log(m+n))$ overhead per operation. We see that if m is small (e.g. constant), then the overhead may be even larger than for trivial protocols. However, if $m = \Theta(n)$, then the overhead is just $O(\log n)$. Also, the constants hidden in the O-notation are very reasonable.

Our protocols for both reading and writing have the following important property. They can be seen as the composition of two protocols, the first having the complexity $O((m+n)\log(m+n))$ and the second having the complexity of just $O(m+n)$. The first protocol takes as inputs the indices to be read or written, and the length of the array, but not the elements of the array or (in case of writing) the new values to be written into it. Hence, if an application reads or writes several times according to the same indices, accessing the same or different arrays, then the first, expensive protocol has to be invoked only once.

We have implemented the protocols on top of Sharemind SMC framework, providing a set of SMC protocols for three parties, and secure against a single honest-but-curious party. We measured the performance of our protocols. The measurements were taken on a cluster of three machines (each of them hosting one of the computing parties), connected to each other with dedicated links of bandwidth 1 Gbit/s. The running times of reading $n/2$ values from an array of length $n/2$ are given in the left part of Fig. 2.1. We have timed the two component protocols separately. We see that the second part (actually performing the read) is almost two orders of magnitude faster than the first part (preparing for reading). Hence the applications should minimize the invocations to the first part.

The right part of Fig. 2.1 gives the execution times for the two parts of writing protocol, again writing $n/2$ values to an array of size $n/2$. We see a similar proportion between the running times of preparing a write according to certain indices, and actually performing it.

2.3.3 Graph algorithms

2.3.3.1 Minimum spanning tree (MST)

We have used the oblivious array access protocols described in Sec. 2.3.2 to devise a protocol for privately finding the minimum spanning tree in a weighted graph that has been stored in the private domain, i.e. the description of the graph has been secret shared among the computing parties. In our setting, only the number of vertices and the number of edges are public. For each edge, its endpoints and its weight have been stored in the private domain.

The well-known algorithms of Kruskal and Prim are inherently sequential, hence unsuitable for being adapted as a protocol for privately finding the minimum spanning tree. We have adapted the parallel algorithm by Awerbuch and Shiloach [6] into an SMC protocol, which we describe in [36]. The required adaptations are relatively minor. We simplify the control flow of the algorithm — instead of iterating the

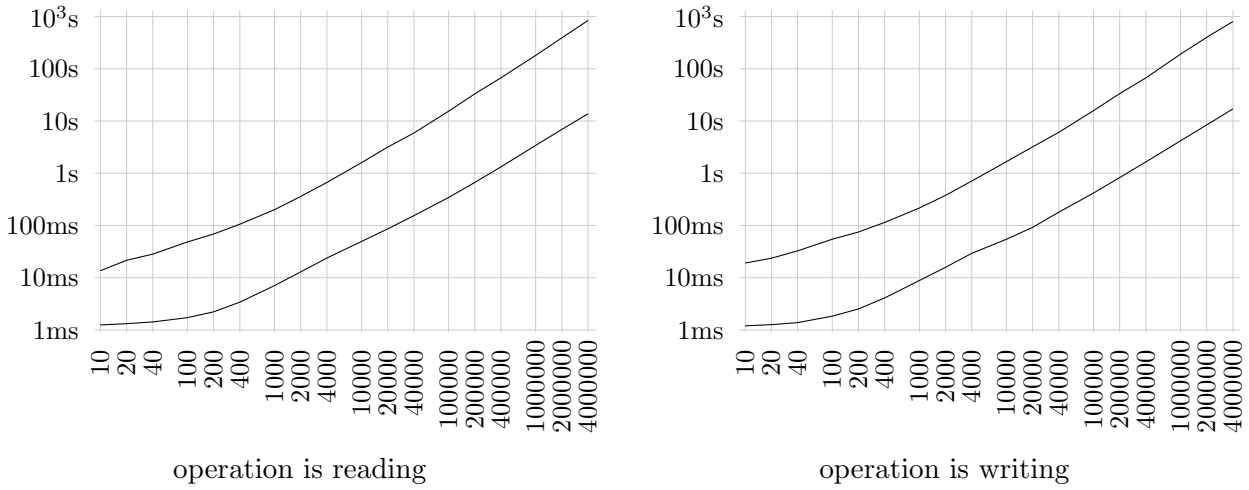

Figure 2.1: Times for preparing (upper) and performing (lower) a parallel operation, depending on the sum of vector length and the number of individual operations

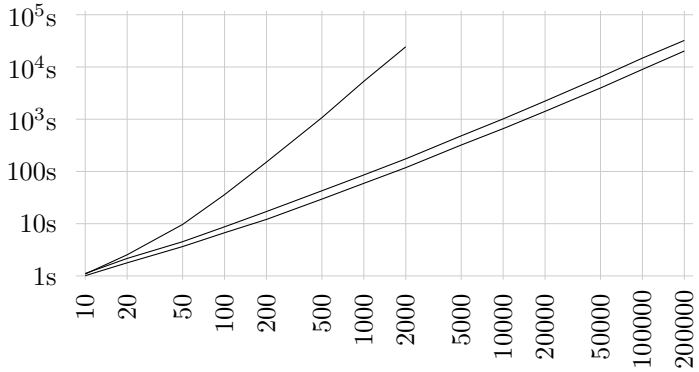

Figure 2.2: Running times for the private MST algorithm, depending on the number of vertices n. Number of edges is $m = 3n$ (lower line), $m = 6n$ (middle line), $m = n(n-1)/2$ (upper line)

main loop of the algorithm as long as there are changes, we iterate it the maximum necessary number of times.

The asymptotic complexity of the Awerbuch-Shiloach algorithm is $O(|E| \log |V|)$ work in $O(\log |V|)$ time. Our oblivious array access protocols induce a logarithmic overhead, hence the communication complexity of our MST protocol is $O(|E| \log^2 |V|)$ and the round complexity is $O(\log^2 |V|)$. We have measured the performance of our protocol on hardware described in Sec. 2.3.2. The running times are presented in Fig. 2.2. We see that we require less than 9 hours to privately find the minimum spanning tree of a graph with 200,000 vertices and 1,200,000 edges, showing that for MST-like graph problems, the SMC protocols definitely scale to sizes relevant in real-world applications.

2.3.3.2 Single-source shortest distances (SSSD)

Finding shortest distances in graphs in privacy-preserving manner has been previously investigated several times [13, 2, 42, 31], often serving as a common example for proposed SMC frameworks. When applied for sparse graphs with private structure, the SSSD often requires techniques of oblivious RAM, in order to propagate the distances along the edges of the graph. In previous year [10], we proposed a technique for looking up values from an array according to private indices, which pushed most of the work into a preprocessing stage and had no overhead during the actual execution of the algorithm. We have now used this technique to give a privacy-preserving implementation of the Bellman-Ford algorithm [35]. In

Table 2.2: SSSD execution benchmarks (times in seconds)

n	100	100	100	100	300	300	300	600	600	600	1000	1000	2000
m	100	600	1000	9900	300	1800	3000	600	3600	6000	1000	6000	2000
offline	0.3	1.3	1.9	19	5.2	31	52	41	240	400	190	1100	1540
online	6.0	7.9	9.2	68	10.4	49	72	39	190	310	110	580	550

previous works, Dijkstra's algorithm has been used for SSSD, but we find Bellman-Ford to be more suitable, because its data accesses and control flow depend less on the private structure of the graph. In particular, our implementation performs no array writes according to private indices (we have no preprocessing-based protocol for that), which would have been unavoidable in a protocol based on Dijkstra's algorithm.

We have implemented the Bellman-Ford algorithm on top of the Sharemind platform, hiding the structure of the graph, as well as the lengths of edges. In our implementation, the numbers $n = |V|$ and $m = |E|$ are public, and so are the in-degrees of vertices (obviously, these could be hidden by using suitable paddings). During the execution, we use private lookup to find $d_i[s(e)]$ for an edge e, where $s(e)$ denotes its source vertex, and $d_i[v]$ is the shortest distance from the source vertex to v when considering only paths with at most i edges. As the vectors d_i have to be computed one after another, but the elements of the same vector can be computed in parallel, our implementation has $O(n)$ rounds in the online phase.

We have measured the performance of our implementation on hardware described in Sec. 2.3.2. The execution times of both the offline preprocessing, and online phase are given in Table 2.2.

The parallel oblivious array access protocols described in Sec. 2.3.2 are also highly suitable for implementing the Bellman-Ford algorithm, even without leaking the in-degrees of vertices. We believe that in future, these protocols should be used in protocols for privately finding shortest distances.

2.3.4 String algorithms

2.3.4.1 String matching

The privacy-preserving implementations of string algorithms are common examples in SMC protocol research, particularly for the case of garbled circuits, as many well-known string algorithms (e.g. the Knuth-Morris-Pratt for string matching) are inherently sequential and thus unsuitable for implementation on top of a secret-sharing based SMC framework. A PRAM algorithm for string matching is described in the well-known textbook [28]. Using the protocols for oblivious parallel array access (Sec. 2.3.2), we have adapted this algorithm to run as a privacy-preserving protocol on top of the secret-sharing based Sharemind SMC framework. In our implementation, the text from which we search a pattern, and the pattern itself are input to the protocol in a secret-shared manner. Namely, each character of the text or the pattern has been shared separately. The lengths of the string and the pattern are public.

A key step of a parallel string matching algorithm is comparing two positions in the text for the purpose of excluding one of them as the starting position of the pattern. Repeated application of this "comparison" allows to significantly reduce the number of possible starting positions, such that the remaining positions can be trivially checked. When designing an SMC application, an important consideration is, how many positions to compare in a single shot. Using binary comparisons optimizes the amount of communication, but comparing more positions at once may reduce the number of rounds we need. We have implemented both the comparison strategy based on binary trees, and strategy based on doubly-logarithmic trees. We have also considered a mixed strategy, where first few comparison rounds are binary, and afterwards switch to comparisons according to the doubly logarithmic tree.

We have measured the performance of our protocols on hardware described in Sec. 2.3.2. The results of matching a pattern of variable length against a text of 1000, 10000, or 50000 characters are given in Fig. 2.3. Beside the running times of the PRAM algorithm adaptation using different comparison strategies we also present the running time of a brute-force algorithm that compares each position of the text with

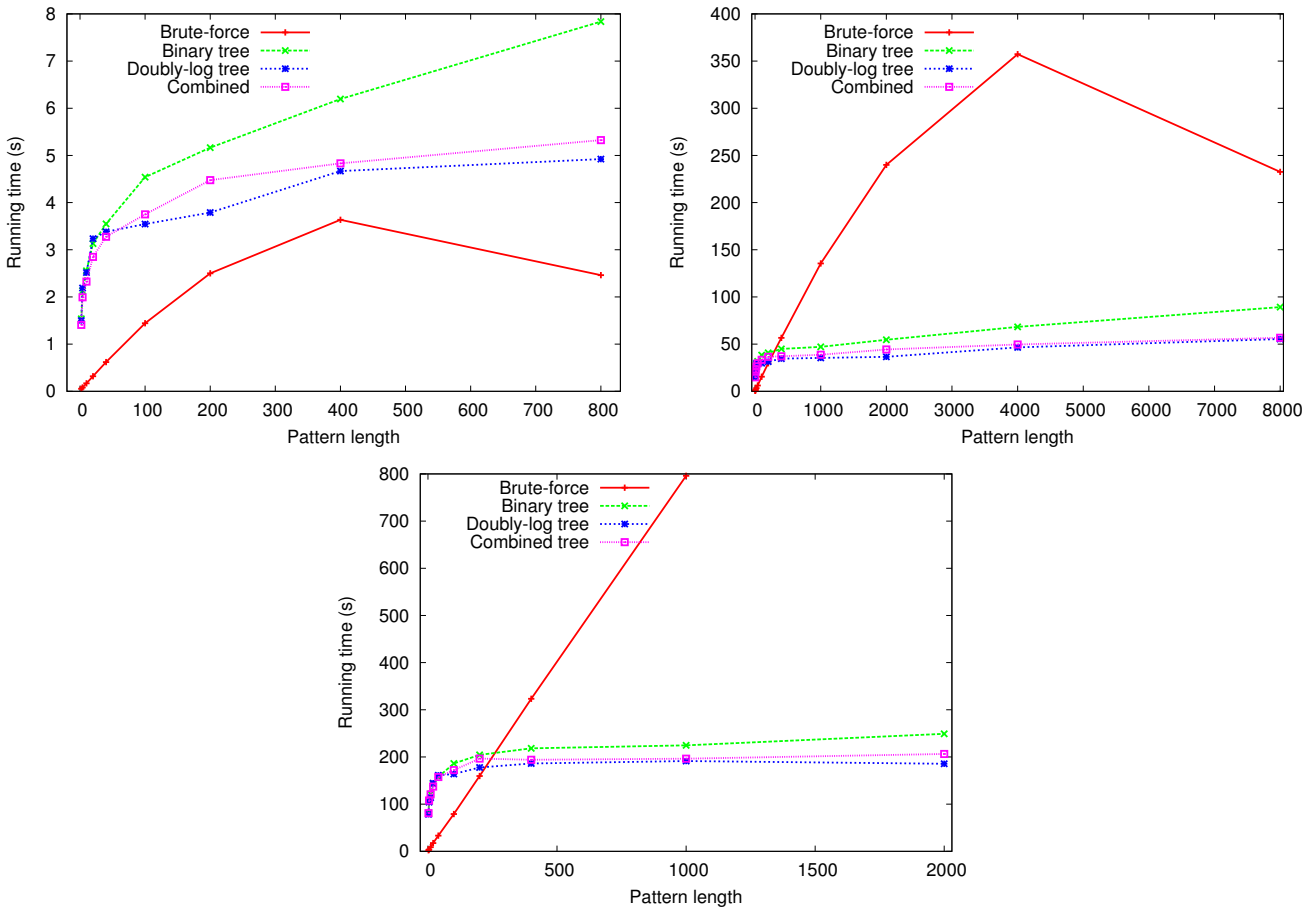

Figure 2.3: Running times of privacy-preserving string matching protocols

each position of the pattern and combines the results.

2.3.4.2 Log auditing

Spotting unexpected behaviors in activities that involve several businesses is important to discover failures, errors and inefficiencies. From a behavioral point of view, the "sequence" of activities performed by the involved parties should be consistent with the business processes of all partners.

Assume two enterprises a and b. For each of the two enterprises we are given events, Σ_a and Σ_b, respectively. Let $\Sigma = \Sigma_a \cap \Sigma_b$ be the events and interactions shared between the two enterprises, while $(\Sigma_a \cup \Sigma_b) \setminus \Sigma$ are internal tasks. Enterprise a recorded the log of a process instance $\omega \in \Sigma_a^*$ by monitoring enterprise activities. Enterprise b owns a local business process, representing all possible licit executions, that is given as a bounded labelled Petri net N_b defined over the corresponding local alphabet. Formally, the problem of *log auditing* is to compute whether

$$\mathbf{proj}_{\Sigma_a \cap \Sigma_b}(\omega) \in \mathbf{proj}_{\Sigma_a \cap \Sigma_b}(\mathcal{L}(N_b)) \ .$$

In Chapter 7.4 of the UaESMC book [37] we consider two mutually distrustful parties, one party owning the log and the other one owning the business process, assuming no trusted third party is available. We provide privacy preserving log auditing by implementing oblivious execution of deterministic finite automata on top of the SMC protocol for looking up values from an array according to private indices, which has been proposed in the previous year [10].

Our implementation is sub-optimal in the round complexity, as it faithfully implements the definition of the DFA execution. On the other hand, this also implies that many instances of log auditing run in parallel would have almost the same runtime as a single instance.

Table 2.3: DFA execution benchmarks (times in milliseconds, $\ell = 2000$)

	$(m, n) =$	$(3, 2)$	$(15, 10)$	$(100, 30)$	$(1000, 30)$
$GF(2^{32})$, additive	offline	12	72	1140	10600
	vector-only	5	110	2200	49000
	online	563	620	1230	6800
$GF(p)$, Shamir	offline	7	120	2600	27000
	vector-only	0	1	89	8100
	online	900	900	1230	4300
$GF(2^{32})$, Shamir	offline	12	200	3900	38000
	vector-only	0	2	310	32000
	online	900	940	1900	13000

Table 2.3 presents the actual running times of our DFA execution implementation. We measured the running time for different automaton sizes m and alphabet sizes n. The length of the input string was always 2000 — the work performed by the algorithm, as well as its timing behavior is perfectly linear in this length. We report our results for different fields and we subdivide the execution times in: (offline) part that can be done without knowing the actual inputs; (vector-only) part where the actual private DFA is available; (online) part where the private log is also available.

2.3.5 Analysing business processes

We studied business process engineering techniques in presence of privacy constraints. Our aim is to support the creation and management of virtual enterprises by a behavioral perspective without threating the participants' autonomy; i.e. without requiring the participant to expose their internal processes and business procedures.

2.3.5.1 Process fusion

In [23] we investigate a mechanism to establish cross-organizational business processes, or more precisely, to identify for each participant of a virtual enterprise (VE) which operations can be performed locally. In other words, we need to compute the contributing subset of the existing local business process that is consistent with the processes of the other VE constituents. We refer to this problem as *VE process fusion*.

Assume two enterprises a and b, with their own business processes, that cooperate to build a VE. For each of the two enterprises we are given a local alphabet, Σ_a and Σ_b, representing the set of possible events and tasks. Each enterprise also owns a local business process, representing all possible allowed executions, that is given as a bounded labelled Petri net (N_a and N_b, respectively) that is defined over the corresponding local alphabet. The problem of *VE process fusion* can be defined as computing the mapping:

$$N_i \mapsto N_i' \qquad (i \in \{a, b\})$$

where $N_i' \sim \mathbf{proj}_{\Sigma_i}(N_a \times N_b)$, $\sim$ stands for trace equivalence, $\times$ is the synchronous product and $\mathbf{proj}_{\Sigma_i}$ hides all transitions that are labeled with symbols not contained in Σ_i. This describes the problem of each participant computing from its local business process N_i a new process, consistent with the global VE business process represented by the synchronous composition.

We address VE process fusion by lifting the problem to privately intersect regular languages, which in [24] we address by using minimal deterministic finite automata (DFA) as a suitable, non-leaking representation of regular language intersection.

Additionally to the results of the previous year, we implemented an new prototype based on Moore's DFA minimization. We experimented our prototype on a cluster consisting of three computers with 48 GB of RAM and a 12-core 3 GHz CPU with Hyper Threading, connected by an Ethernet local area network

Table 2.4: Running times of one iteration of privacy-preserving service matching

service	$\|I\|$	$\|Q_{\mathrm{SA}}\|$	$\|\delta_{\mathrm{SA}}\|$	$\|Q_{\mathrm{OG}}\|$	$\|\delta_{\mathrm{OG}}\|$	$\|\Phi\|$	t_{prep}	t_{iter}
Online Shop	16	222	531	153	331	735	1m09s	2m29s
Internal Order	9	14	18	512	2304	19172	4m42s	2m33s
Purchase Order	10	137	437	168	548	2074	1m29s	4m14s

with link speed of 1 Gbps. On this cluster, if the minimization is executed on the product of two automata (i.e. the two automata representing the regular languages of the two participants) having 100 states and a common alphabet consisting of ten symbols, then one iteration in Moore's algorithm requires ca. 4.5 s. For input automata having 300 states this time is 40 s. In the worst case, the algorithm requires to converge a number of iterations equal to the product of the sizes of the input automata. While these are the execution times of single iterations, our experiments show that privacy-preserving minimization of DFA is feasible even for automata with 100,000 states, if the application producing these automata allows us to give reasonable bounds on the number of iterations necessary for these algorithms to converge.

2.3.5.2 Process Matching

A non-trivial process engineering task is to ensure soundness (i.e., interoperability) of enterprise collaborations. *Business process matching* consists in a bottom-up soundness check: each (collaborating) enterprise owns an existing private process, and the task is guaranteeing that their composition is sound. Such a bottom-up approach must face two problems: (i) the soundness of the composition of "secret" processes must be checked without revealing information about the constituents, and (ii) if the soundness check fails, a Boolean result is of little use, since the participants have no information about the global process and thus can not investigate what is wrong. This prevents the participants from adapting their local procedures to form a sound collaboration.

In [26] we present a bottom-up approach to checking soundness of interorganizational business processes that addresses both these problems. To address problem (ii) above, we introduce a measure for *behavioral similarity* between two Service Automata. Then, our goal is checking whether the composition of the automata A and B is sound, and in case it is not, to allow the participant owning A to discover, among all automata that can be soundly composed with B, the one that is most similar to A.

Our approach is based on the combination of three techniques: (i) we lift the check of the soundness of the automata composition to matching one of the automata against the *Operating Guideline* of the other, (ii) we introduce the notion of *weighted matching* as a measure for matching degree, and show how to compute this measure and at the same time extract the behaviourally most similar service that can soundly collaborate with the other party, and (iii) we implement an algorithm to compute weighted matching while preserving privacy by means of SMC techniques.

Table 2.4 reports some preliminary studies of the running time of our implementation. We used a Inter Core i5-4300U CPU running at 1.9 GHz (with four cores) and 8GB memory. The computing parties communicated with each other over the loopback interface. The traffic volume between two parties always remained below 1Gbit/s. We use the larger examples from [43]. For the examples we've considered, we give its name, its size (characterized by the number of labels, the number of vertices and edges of the service automata and the operating guideline. We report the time it took to run a single iteration of the algorithm; the necessary number of iterations for each task has to be determined from its parameters. We also report the time it took to set up the data used in the iterations.

2.3.6 Differentially private outputs of SMC protocols

Differential privacy [18, 44] is a property of information-releasing mechanisms which states that the output distributions of the mechanism may differ only a little for similar inputs. For statistical studies, it goes

well together with SMC — while the use of SMC to perform a study ensures that information is not leaked during the computations, the differential privacy of the study problem ensures that the results of the study, presumably made public, do not allow private information to be inferred. Among several formalizations of output privacy, differential privacy is one of the most preferred ones, due to its well-understood properties [27], and its support for simple arguments through composability.

There are several methods to turn a statistical function into differentially private, including the well-known methods of adding Laplacian noise [18]. In our work, we have considered the sample-and-aggregate method [48], smoothening the function, such that less noise has to be added in order to obtain the same level of privacy. We also consider the personal differential privacy (PDP) property [19], which allows to state more precisely, how much different components of the input (records of the database that is being analysed) may affect the output distribution.

Using the SMC framework Sharemind, we have implemented a number of statistical functions, and have applied to it the sample-and-aggregate, and provenance for PDP methods for achieving differential privacy [49]. We have compared the running times of these functions together, and without methods for differential privacy. The hardware used for benchmarks is described in Sec. 2.3.2. The running times are presented in Table 2.5. We have considered the following statistical functions:

- Count of rows. Due to the use of structures for privacy-preserving databases described in D2.2.1 [8], this is a non-trivial operation — the number of physical rows may be larger than the number of logical rows, and a private mask column indicates which rows are actually "active".
- Average and median value of a column (also together with the mask vector). Here ℓ is the number of samples used by the sample-and-aggregate method. For different functions, the best value of ℓ may be different. For average and count, $\ell = n$ gives the best precision for a given level of privacy, where n is the number of rows.
- Linear correlation coefficient. This is an example of a multivariable aggregation function, trying to find $c \in \mathbb{R}$, such that the values in one column could be approximated as c-fold values of another column.

The meanings of the columns in Table 2.5 are the following:

- **non-diff. private** gives the execution time of the SMC protocol without the use of differentially private mechanisms.
- The columns under "**budgets**" give the execution times, when the sample-and-aggregate mechanism is used. When the privacy budget is **global**, the personalized differential privacy mechanism is not used and the privacy cost of publishing the result of the query is applied to the whole database. When the budgets are **in-place**, the database table has an extra column where the remaining budget of each row is stored. When the **provenance** budgets are in use, then each row has a *provenance* (an identifier), and each provenance has a remaining budget (stored in a separate table).

The take-away of our experiments is, that the mechanisms of differential privacy do not cause a significant further overhead to privacy-preserving statistical analysis. When a separate table of provenances (which has to be joined to the main table during computations) is not used, then the running times are at most a couple of times higher than the times without differential privacy.

2.3.7 Frequent itemset mining

Frequently co-occurring events or actions often reveal information about the underlying causal dependencies. Hence, frequent itemset mining is often used as one of the first steps in the analysis of transactional data. Market basket analysis is one of the most well-known application areas for these algorithms. We have studied privacy-preserving frequent itemset mining previously [9]. In this study, the set of transactions was represented as a 0-1 matrix. This representation may be inefficient if the matrix is sparse, e.g. if the number

function	num. rows	non-diff. private	budgets: global	in-place	provenance
count	10000	392	759	1124	6096
	20000	405	768	1241	10672
	50000	460	827	1587	
	100000	594	962	2257	
	200000	866	1184	3452	
	500000			7234	
	1000000			13871	
average	10000	443	1099	1475	6598
	20000	461	1285	1753	11051
	50000	532	1774	2531	
	100000	694	2625	3873	
	200000	999	4426	6483	
	500000			15118	
	1000000			28995	
correlation ($\ell = 100$)	10000	822	2455	2826	7663
	20000	1001	2665	3157	12693
	50000	1556	3278	4092	26833
	100000	2473	4312	5525	54767
	200000	4443	6359	8548	112300
	500000	9721	12218	17895	
	1000000	19414	22608	33530	
correlation ($\ell = 1000$)	500000	9850	20115	25961	
	1000000	19380	30657	41363	
	2000000	37643	51381	72436	
median ($\ell = 100$)	10000	851	1332	1708	6548
	20000	1327	1737	2131	11786
	50000	2318	2137	3502	
	100000	4620	4075	5312	
	200000	6498	5559	9198	
	500000			24175	
	1000000			34583	

Table 2.5: Benchmarking results for differentially private statistical functions running on SHAREMIND (in milliseconds)

of items in each transaction is much smaller than the total number of different items. In this reporting period, we tried to come up with protocols that make use of the sparsity, also relying on techniques of oblivious array access (Sec 2.3.2).

An important operation in FIM algorithms is the intersection of two sets. If these two sets are represented as bit-vectors in an SMC protocol, then their pointwise multiplication gives the bit-vector corresponding to the intersection. In our current work, we have proposed a protocol for finding the intersection if the sets are represented as a sequence of indices (of elements belonging to them), or if one of the sets is represented as a bit-vector and the other one as a sequence of indices.

If we are willing to leak somewhat more about the densities of the columns of the matrix, then it is also possible to use a mixed representation, where some sets are represented as bit-vectors and some as index sequences. We have found that this representation indeed allows a speed-up for the privacy-preserving FIM computation.

In our benchmarks, we made the value of k (the maximal size of an itemset) large enough, so that all the frequent subsets of arbitrary size would be generated. We have found that it is not so easy to compare our new results to our previous results. Namely, in the results of [9] the tested threshold t (the minimal number of transactions that should contain a frequent itemset) is quite large, and using the same numbers would make no sense for sparse representation, as all the sparse columns are immediately discarded, leaving only dense columns behind. In addition, the programs of [9] use a much older version of Sharemind and are not fairly comparable. However, our new implementation has been tested on both sparse and dense representations.

The achievable theoretical results are far from those that we obtained in practice. As Sharemind supports data types of fixed size, it is not so easy to be flexible in making the communication dependent on the parameters of the initial database. In addition, local computation is sometimes more expensive than an online protocol, and hence in some cases we deliberately increased the communication to win in the computation time. Although our theoretical algorithm tries to use sparse approach only if it is indeed useful, in practice there turn out to be sequential jumps between bit and set representation which are sometimes more expensive than using just bits.

We have indeed obtained efficiency gain for very small values of t. Since small t leaves a lot of columns behind and makes the task very slow, we have tested only 5500 first transactions of the standard Retail dataset (which can be found in the Frequent Itemset Mining Dataset Repository [1]). Since it makes no sense to set t larger than the size of sparse column, we had to start from $t = 30$. In this case, the set approach was still slower than the bit approach, and the communication was larger, although it should be significantly smaller theoretically. For $t = 20$, the results were more or less the same. For $t = 10$, the straightforward bit approach has taken 5 hours while the sparse approach only 3.5 hours. For some strange reason, the sparse approach still reported more communication. At the same time, we also could not win in rounds since the bit approach takes clearly less rounds, even theoretically. In future, we are going to make the local computation of Sharemind more efficient, so that it indeed would make sense to parallelize rounds and decrease communication without losing in overall speed.

We have also tried whether Diffset gives advantage for dense matrices. We have taken the Chess dataset, which is one of the densest available at [1]. This is still far from the density that should be achieved to make sparse representation possible at all for Diffset, and the algorithm had to be started entirely on bit matrices, but we hoped that some sparse columns will emerge on further iterations, since Diffset makes the sizes of columns smaller. This time we used the same values of t as in our previous results in [9], as in Diffset the columns represent the difference between the column size and its prefix size, which may be small even for large t. The sparse columns indeed emerge, but for larger t they make the sparse approach only slower, so possibly there are not so many of them. So far we have not achieved the situation where Diffset would be faster using sparse representation.

The database Mushroom has been reported in [52] to be even worse with Diffset that with Eclat, and hence we tested it with Eclat only. Although all the columns are dense in the beginning, some of them may become sparse on further iterations. However, we see that they are still not sparse enough to make the sparse approach more efficient.

Database	t	**Eclat without sparsity**		**Eclat with sparsity**	
		Time	Comm	Time	Comm
Retail 5500	30	13.5 min	2.4 GB	15.5 min	4.1 GB
Retail 5500	25	22 min	4.1 GB	27 min	7.2 GB
Retail 5500	20	45 min	7.8 GB	46 min	13.2 GB
Retail 5500	15	1 h 40 min	17.8 GB	1 h 25min	27.1 GB
Retail 5500	10	5 h	53.3 GB	3.5h	70.9 GB

Database	t	**Eclat without sparsity**		**Eclat with sparsity**	
		Time	Comm	Time	Comm
Mushroom	2400	4 min	0.6 GB	5 min 15 s	1.2 GB
Mushroom	2200	5 min 30 s	0.8 GB	7 min 20 s	1.6 GB
Mushroom	2000	7 min 20 s	1.3 GB	12 min	2.6 GB
Mushroom	1800	24 min	2.7 GB	29 min	5.3 GB
Mushroom	1600	3 h 11 min	9.8 GB	3 h 33 min	19.3 GB
Mushroom	1400	3 h 40 min	11.1 GB	4 h 05 min	21.7 GB

Database	t	**Diffset without sparsity**		**Diffset with sparsity**	
		Time	Comm	Time	Comm
Chess	2600	5 min	0.6 GB	7 min	1.4 GB
Chess	2400	31 min	1.9 GB	37 min	4.6 GB
Chess	2200	3 h 40 min	5.3 GB	4 h	19.2 GB

Table 2.6: Performance of bit matrix approach vs sparse matrix approach

In addition, we have noticed that the initial order of columns matters, as it defines the intersections of Eclat and Diffset algorithms. In general, it is better to put sparse columns in front of bit columns, as it reduces the number of situations where a set column should be converted back to a bit column. Nevertheless, this is more difficult to trace on further iterations.

The results are shown in Tab. 2.6.

Bibliography

[1] Frequent itemset mining dataset repository. `http://fimi.ua.ac.be/data/`. Last accessed 2015-07-07.

[2] Abdelrahaman Aly, Edouard Cuvelier, Sophie Mawet, Olivier Pereira, and Mathieu Van Vyve. Securely solving simple combinatorial graph problems. In Ahmad-Reza Sadeghi, editor, *Financial Cryptography*, volume 7859 of *Lecture Notes in Computer Science*, pages 239–257. Springer, 2013.

[3] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek. Secure multiparty computations on bitcoin. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 443–458, May 2014.

[4] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Fair two-party computations via bitcoin deposits. In Rainer Böhme, Michael Brenner, Tyler Moore, and Matthew Smith, editors, *Financial Cryptography and Data Security*, volume 8438 of *Lecture Notes in Computer Science*, pages 105–121. Springer Berlin Heidelberg, 2014.

[5] K.J. Arrow. *Social Choice and Individual Values*. Yale University Press, 1951.

[6] Baruch Awerbuch and Yossi Shiloach. New connectivity and MSF algorithms for shuffle-exchange network and PRAM. *IEEE Trans. Computers*, 36(10):1258–1263, 1987.

[7] Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In JuanA. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014*, volume 8617 of *Lecture Notes in Computer Science*, pages 421–439. Springer Berlin Heidelberg, 2014.

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

[9] Dan Bogdanov, Roman Jagomägis, and Sven Laur. A universal toolkit for cryptographically secure privacy-preserving data mining. In *Proceedings of the 2012 Pacific Asia Conference on Intelligence and Security Informatics*, PAISI'12, pages 112–126, Berlin, Heidelberg, 2012. Springer-Verlag.

[10] Dan Bogdanov, Liina Kamm, Peeter Laud, Alisa Pankova, Pille Pullonen, Riivo Talviste, and Jan Willemson. Advances in SMC techniques, January 2014. UaESMC Deliverable 2.2.2.

[11] Dan Bogdanov, Liina Kamm, Sven Laur, and Ville Sokk. Rmind: a tool for cryptographically secure statistical analysis. Cryptology ePrint Archive, Report 2014/512, 2014. `http://eprint.iacr.org/`.

[12] Dan Bogdanov, Peeter Laud, Sven Laur, and Pille Pullonen. From Input Private to Universally Composable Secure Multi-party Computation Primitives. In Datta and Fournet [17], pages 184–198.

[13] Justin Brickell and Vitaly Shmatikov. Privacy-preserving graph algorithms in the semi-honest model. In Bimal K. Roy, editor, *ASIACRYPT*, volume 3788 of *Lecture Notes in Computer Science*, pages 236–252. Springer, 2005.

[14] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145. IEEE Computer Society, 2001.

[15] R Cleve. Limits on the security of coin flips when half the processors are faulty. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, STOC '86, pages 364–369, New York, NY, USA, 1986. ACM.

[16] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer, 2012.

[17] Anupam Datta and Cedric Fournet, editors. *IEEE 27th Computer Security Foundations Symposium, CSF 2014, Vienna, Austria, 19-22 July, 2014*. IEEE, 2014.

[18] Cynthia Dwork. A firm foundation for private data analysis. *Commun. ACM*, 54(1):86–95, 2011.

[19] Hamid Ebadi, David Sands, and Gerardo Schneider. Differential privacy: Now it's getting personal. In Sriram K. Rajamani and David Walker, editors, *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*, pages 69–81. ACM, 2015.

[20] Ittay Eyal and EminGün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *Financial Cryptography and Data Security*, volume 8437 of *Lecture Notes in Computer Science*, pages 436–454. Springer Berlin Heidelberg, 2014.

[21] Yiannis Giannakopoulos and Yiannis Tselekounis. Design of Protocols with a Mixture of Cryptographic and Game-Theoretic Assumptions, July 2015. UaESMC Deliverable 3.4.

[22] A. Gibbard. Manipulation of Voting Schemes: A General Result. *Econometrica*, 41(4):587–601, 1973.

[23] Roberto Guanciale and Dilian Gurov. Privacy preserving business process fusion. In Fabiana Fournier and Jan Mendling, editors, *Business Process Management Workshops - BPM 2014 International Workshops, Eindhoven, The Netherlands, September 7-8, 2014, Revised Papers*, volume 202 of *Lecture Notes in Business Information Processing*, pages 96–101. Springer, 2014.

[24] Roberto Guanciale, Dilian Gurov, and Peeter Laud. Private Intersection of Regular Languages. In *Proceedings of the 12th Annual Conference on Privacy, Security and Trust*, pages 112–120. IEEE, 2014.

[25] Roberto Guanciale, Dilian Gurov, Peeter Laud, Alisa Pankova, Martin Pettai, and Sander Siim. Algorithms for Large-Scale SMC Problems, July 2015. UaESMC Deliverable 4.2.2.

[26] Dilian Gurov, Peeter Laud, and Roberto Guanciale. Privacy Preserving Business Process Matching. In *Proceedings of the 13th Annual Conference on Privacy, Security and Trust*. IEEE, 2015.

[27] Justin Hsu, Marco Gaboardi, Andreas Haeberlen, Sanjeev Khanna, Arjun Narayan, Benjamin C. Pierce, and Aaron Roth. Differential privacy: An economic method for choosing epsilon. In Datta and Fournet [17], pages 398–410.

[28] Joseph JáJá. *An Introduction to Parallel Algorithms*. Addison-Wesley, 1992.

[29] Liina Kamm, Peeter Laud, Alisa Pankova, and Pille Pullonen. Advances in SMC techniques, July 2015. UaESMC Deliverable 2.2.3.

[30] Liina Kamm and Jan Willemson. Secure floating point arithmetic and private satellite collision analysis. *International Journal of Information Security*, pages 1–18, 2014.

[31] Marcel Keller and Peter Scholl. Efficient, Oblivious Data Structures for MPC. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II*, volume 8874 of *Lecture Notes in Computer Science*, pages 506–525. Springer, 2014.

[32] Aggelos Kiayias, Elias Koutsoupias, and Yiannis Tselekounis. Bitcoin: a game-theoretic approach. Unublished report, 2015.

[33] Aggelos Kiayias, Hong-Sheng Zhou, and Vassilis Zikas. Fair and robust multi-party computation using a global transaction ledger. Cryptology ePrint Archive, Report 2015/574, 2015. `http://eprint.iacr.org/`.

[34] Ranjit Kumaresan and Iddo Bentov. How to use bitcoin to incentivize correct computations. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 30–41, New York, NY, USA, 2014. ACM.

[35] Peeter Laud. A private lookup protocol with low online complexity for secure multiparty computation. In Siu Ming Yiu and Elaine Shi, editors, *ICICS 2014*, LNCS. Springer, 2014. To appear.

[36] Peeter Laud. Parallel Oblivious Array Access for Secure Multiparty Computation and Privacy-Preserving Minimum Spanning Trees. *Proceedings of Privacy Enhancing Technologies*, 2015(2):188–205, 2015.

[37] Peeter Laud and Liina Kamm, editors. *Applications of Secure Multiparty Computation*, volume 13 of *Cryptology and Information Security*. IOS Press, 2015. `http://www.iospress.nl/book/applications-of-secure-multiparty-computation/`.

[38] Peeter Laud and Alisa Pankova. New Attacks against Transformation-Based Privacy-Preserving Linear Programming. In Rafael Accorsi and Silvio Ranise, editors, *Security and Trust Management (STM) 2013, 9th International Workshop*, volume 8203 of *Lecture Notes in Computer Science*, pages 17–32. Springer, 2013.

[39] Peeter Laud and Alisa Pankova. On the (Im)possibility of Privately Outsourcing Linear Programming. In Ari Juels and Bryan Parno, editors, *Proceedings of the 2013 ACM Workshop on Cloud computing security, CCSW 2013*, pages 55–64. ACM, 2013.

[40] Peeter Laud and Alisa Pankova. Verifiable Computation in Multiparty Protocols with Honest Majority. In Sherman S. M. Chow, Joseph K. Liu, Lucas Chi Kwong Hui, and Siu-Ming Yiu, editors, *Provable Security - 8th International Conference, ProvSec 2014, Hong Kong, China, October 9-10, 2014. Proceedings*, volume 8782 of *Lecture Notes in Computer Science*, pages 146–161. Springer, 2014.

[41] Peeter Laud and Alisa Pankova. Preprocessing-based verification of multiparty protocols with honest majority. Cryptology ePrint Archive, Report 2015/674, 2015. `http://eprint.iacr.org/`.

[42] Chang Liu, Yan Huang, Elaine Shi, Jonathan Katz, and Michael W. Hicks. Automating efficient ram-model secure computation. In *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, pages 623–638. IEEE Computer Society, 2014.

[43] Niels Lohmann. Correcting deadlocking service choreographies using a simulation-based graph edit distance. In *Proceedings of BPM 2008*, volume 5240 of *Lecture Notes in Computer Science*, pages 132–147, 2008. Case studies available from `http://service-technology.org/publications/lohmann_2008_bpm/`.

[44] Prashanth Mohan, Abhradeep Thakurta, Elaine Shi, Dawn Song, and David E. Culler. GUPT: privacy preserving data analysis made easy. In K. Selçuk Candan, Yi Chen, Richard T. Snodgrass, Luis Gravano, and Ariel Fuxman, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 349–360. ACM, 2012.

[45] Roger B. Myerson. Optimal auction design. *Mathematics of Operations Research*, 6(1):58–73, 1981.

[46] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[47] Noam Nisan. Introduction to mechanism design (for computer scientists). In Noam Nisan, Tim Roughgarden, Éva Tardos, and Vijay Vazirani, editors, *Algorithmic Game Theory*, chapter 9. Cambridge University Press, 2007.

[48] Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. Smooth sensitivity and sampling in private data analysis. In David S. Johnson and Uriel Feige, editors, *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 75–84. ACM, 2007.

[49] Martin Pettai and Peeter Laud. Combining Differential Privacy and Secure Multiparty Computation. Cryptology ePrint Archive, Report 2015/598, 2015. `http://eprint.iacr.org/`.

[50] M.A. Satterthwaite. Strategy-proofness and arrow's conditions: Existence and correspondence theorems for voting procedures and social welfare functions. *Journal of Economic Theory*, 10(2):187–217, 1975.

[51] John von Neumann and Oskar Morgenstern. *Theory of games and economic behavior*. Princeton University Press, 3rd edition, 1953.

[52] Mohammed J. Zaki and Karam Gouda. Fast vertical mining using diffsets. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 326–335, New York, NY, USA, 2003. ACM.