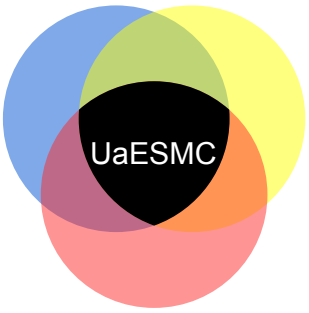


Project N°: **FP7-284731**

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 D5.2.2
Scientific Progress Analysis and Recommendations

Due date of deliverable: 31st January 2014

Actual submission date: 31st January 2014

Start date of the project: **1st February 2012** Duration: **36 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 D5.2.2 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 whereever 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, 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

Dan Bogdanov (CYB) Yiannis Giannakopoulos (UoA) Roberto Guanciale (KTH)
Dilian Gurov (KTH) Liina Kamm (CYB) Peeter Laud (CYB)
Alisa Pankova (CYB) Pille Pruulmann-Vengerfeldt (UT) Pille Pullonen (CYB)
Riivo Talviste (CYB) Yiannis Tselekounis (UoA) Jan Willemson (CYB)

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 second year of UaESMC, we have continued our work on several of the problems [3] selected on the basis of the interviews conducted at the beginning of the project [39]. These include the statistical analysis of structured data, linear programming, finding shortest paths in graphs, distributed network monitoring, and cryptographic tools for mechanism design; all with the aim to preserve the privacy of the participants. While the actual results are described in their respective deliverables [5, 19, 22], we will give an overview below and show how they fit together in enabling the ubiquitous use of SMC techniques.

In a sense, our focus has somewhat shifted from the goals we set for ourselves while setting up the project more than three years ago. We believed that it was important to demonstrate the possibility of sufficiently efficient privacy-preserving protocols for a large variety of tasks, in order to encourage the security R&D community to build well-engineered and usable SMC tools. We are now seeing the SMC field to develop faster than we imagined, with advances in both generic and specific techniques for different computational problems. We are thus focusing our research on our planned SMC framework, constructing tools and techniques to use SMC in novel contexts. Often, these are techniques that use existing infrastructure (e.g. PKI) in order to efficiently incentivize the parties to perform correctly in some sense. But we are also providing a number of computational techniques to speed up privacy-preserving computations for certain classes of problems.

The exact problems we've worked on during the second year and will continue working on during the third year of UaESMC, have been selected in discussions involving all partners of the project, with the UaESMC framework in mind. In this report, we show how our results fit together, enhancing each other and contributing to the UaESMC framework in the end. We give an overview of our results and our plans for extending them during the third year of the project. We discuss the theoretical significance and practical applicability of our results and their planned extensions. If possible, we complement this discussion with the results of benchmarking, showing the problem sizes that we have tackled in practice.

Chapter 2

Progress during second year

2.1 Statistical Analysis of Structured Data

2.1.1 Statistical analysis framework

To demonstrate the feasibility of privacy-preserving statistics, we design, implement and conduct an experimental study. In the scenario, we use a table of subjects and their demographic information from the Population Register, a table specifying whether a subject attended a city school from the Ministry of Education, and a table of taxed income payments for the same subjects from the Tax Office. We used artificially generated data in our experiments.

For our implementation, we use Sharemind and we implemented the statistical algorithms using the SecreC programming language. We uploaded data using a data importer application developed using the Sharemind controller library.

We conducted the experiments 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). While monitoring the experimental scenario, we did not notice memory usage above 500 MB per machine. The computers were connected using gigabit ethernet network interfaces.

Table 2.1 contains the operations, input sizes and running times for our experimental scenario. We see that most operations in our experimental study take under a minute to complete. The most notable exceptions is the group median computation, as median computation has to be applied to the payments of 2000 subjects. This time can be reduced by vectorising the median invocations or conduct this aggregation before the data is converted into secret-shared form.

To check scalability, we performed some tests on ten times larger data vectors. We found that increasing input data size 10 times increases running time about 5 times. Only histogram computation is actually slower, because it uses a more detailed frequency table for larger databases.

The improved efficiency per input data element is explained by the use of vectorised operations of the Sharemind framework. The operations in the Sharemind framework are more efficient when many are performed in parallel using the SIMD (single instruction, multiple data) model.

The strengths of our solution are generality, precision and practicality. First, we show that secure multiparty computation is flexible enough for implementing complex applications. Second, our use of secure floating point operations makes our implementation more precise. Third, we use the same algorithms as popular statistical toolkits like GNU R without simplifying the underlying mathematics.

In the third year we plan to develop the statistics framework even further to include more complex methods that are useful for data analysts such as FDR correction, ANOVA and more complex regression analysis.

Step 1: Data import

Operation	*Record count*	*Time*
Data import from offsite computer	2 000	3 s
	53 977	24 s

Step 2: Descriptive statistics

Operation	*Record count*	*Time*
5-number summary (publish filter size)	2000	21 s
	20000	97 s
5-number summary (hide filter size)	2000	27 s
	20000	107 s
Frequency table	2000	16 s
	20000	222 s

Step 3: Grouping and linking

Operation	*Record count*	*Time*
Median of incomes by subject	53 977	3 h 46 min
Linking two tables by a key column	2000×5 and 2000×3	28 s
Linking two tables by a key column	2000×7 and 2000×2	29 s

Step 4: Statistical tests

Operation	*Record count*	*Time*
Student's t-test, equal variance	2000	167 s
	20000	765 s
Student's t-test, different variance	2000	157 s
paired t-test, known mean	2000 and 2000	98 s
paired t-test, unknown mean	2000 and 2000	102 s
χ^2-test, 2 classes	2000	9 s
	20000	10 s
χ^2-test, n-class version, 2 classes	2000	20 s
χ^2-test, n-class version, 5 classes	2000	23 s
Wilcoxon rank sum	2000	34 s
Wilcoxon signed-rank	2000 and 2000	38 s

Table 2.1: Running times of privacy-preserving statistics (in seconds)

2.1.2 Sorting Methods

We analyzed the performance of all the oblivious sorting algorithms mentioned in UaESMC deliverable D2.2.2 [5] — naive comparison sort (NaiveCompSort), quicksort, radix sort and sorting networks. We implemented all algorithms in the SecreC programming language to run on the Sharemind secure multi-party computation system.

Our quicksort implementation is based on the work in [23] and personal communication with its authors. We implemented the algorithm as similarly as possible to achieve a fair comparison. The naive comparison sort and radix sort algorithms are implemented straightforwardly from algorithm descriptions given in deliverable D2.2.2. All implementations are vectorized to optimize running time.

The sorting network implementation consists of two parts. We implemented sorting network generation using Florian Forster's `libsortnetwork` library[1]. Sharemind generates and caches sorting networks and encodes them for delivery to SecreC programs. The evaluation of the sorting network is implemented in SecreC. Our implementation generates Batcher's bitonic mergesort networks, as they were the fastest to generate.

The experiments were conducted using a Sharemind installation consisting of three servers connected

[1] Available from `http://verplant.org/libsortnetwork/` in January, 2014.

by a 1 Gbps local area network. Each server was equipped with 48 GB of memory and a 12-core 3 GHz Intel processor.

We measured the running time and network usage using the profiling mechanism built into SHAREMIND. We marked code sections and SHAREMIND measured and logged the running time and network usage of each section invocation. We sampled the memory use reported for the SHAREMIND server process every one second and aligned this data with the running time data to find the peak memory usage for each experiment.

We ran each algorithm with bitwise-shared 64-bit unsigned integer data. We used worst-case data (all equal values) for each algorithm and additionally, used random data for the quicksort algorithm similarly to the experiments of [23].

Figure 2.1 shows the breakdown of the running times of oblivious sorting algorithms. We see that most of the time in naive sorting and quicksort is spent on comparisons. The time taken for sorting network evaluation begins with mostly comparisons and oblivious choice, but their importance is reduced as the time needed for generating the network increases. This is a strong motivator for the precomputing and caching of sorting networks. Radix sort has the most interesting profile, as it does not use comparisons. Instead, its most expensive part is oblivious choice.

Following are the comparisons of all the algorithms. Note that the axes of the comparison figures are on a logarithmic scale. Figure 2.2 shows the comparison of the running time. Naive comparison sort is very fast on small inputs, but its high complexity makes it infeasible for larger inputs. Quicksort is the fastest of all algorithms, but only in the random data vector experiment. When we run quicksort on data vectors with all equal values, it performs significantly slower. This can be explained by the need to actually go through all the subsets of the data. On randomized data, our implementation of quicksort achieves the same performance as reported in [23].

Radix sorting is not the most efficient on small inputs, but its use of cheap secure operations ensures that its running time does not grow as quickly as that of the other algorithms. Sorting networks are efficient early, but the time needed to generate the network starts to grow significantly as the data size grows. If the sorting network structure is cached, sorting network evaluation is almost as fast as radix sorting.

We see the network usage measurements in Figure 2.3. Naive sorting and quicksort on worst-case data require a lot of network communication. The other algorithms form a more efficient group, with quicksort on random data requiring the least communication and radix sort taking the second place.

Finally, Figure 2.4 shows the memory usage. The memory usage of the naive implementation grows squared in the size of data, making it infeasible for large inputs. The sorting network algorithm uses significant amounts of memory during the generation of the sorting network and reduced amounts after that. The memory requirements of oblivious radix and quicksort are low in comparison.

We also implemented oblivious matrix sorting for all sorting algorithms. However, for brevity these are excluded from this summary and included in the appendix of deliverable D2.2.2.

In conclusion, our performance analysis shows that even though naive comparison-based sorting is fast on small inputs, its $\mathcal{O}(n^2)$ complexity makes it slow for practical input sizes. While the oblivious version of quicksort is very efficient on random data, it performs poorly when the input contains many equal elements. Its increased running time on such inputs also leaks the number of equal elements.

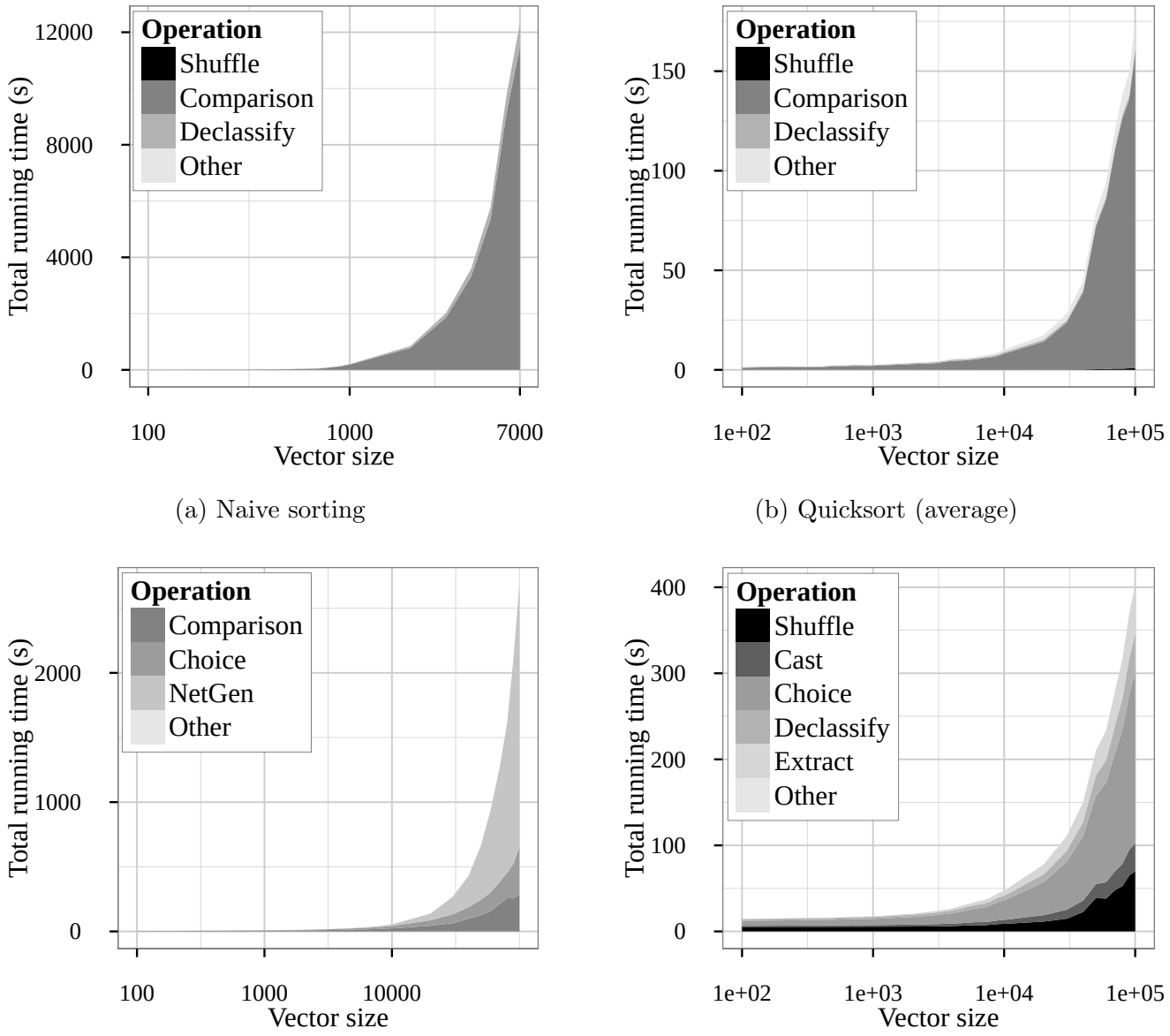
Oblivious sorting networks are a great choice when we can precompute or cache the network structure. In that case, the algorithm provides perfect privacy with a reasonable performance.

Our novel oblivious radix sorting algorithm leaks less information than constructions based on shuffling and declassified comparison results. As input sizes grow, its performance comes closer to that of quicksort on random data, because it does not need to use the relatively expensive comparison operations.

2.2 Regular languages

2.2.1 Execution of Finite Automata

Regular languages and finite automata appear in many tasks that handle data in textual form, including network monitoring tasks. A deterministic finite automaton (DFA) over an alphabet Σ has a set of states



(a) Naive sorting

(b) Quicksort (average)

(c) Sorting networks

(d) Radix sort

Figure 2.1: Running time breakdown for implemented sorting algorithms.

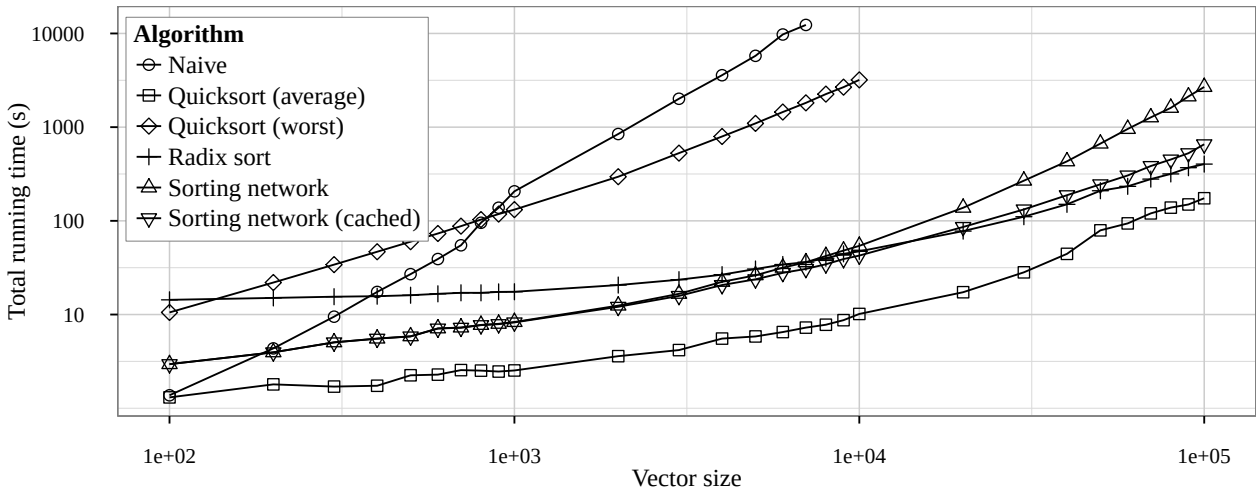


Figure 2.2: Comparison of the running time of oblivious sorting algorithms.

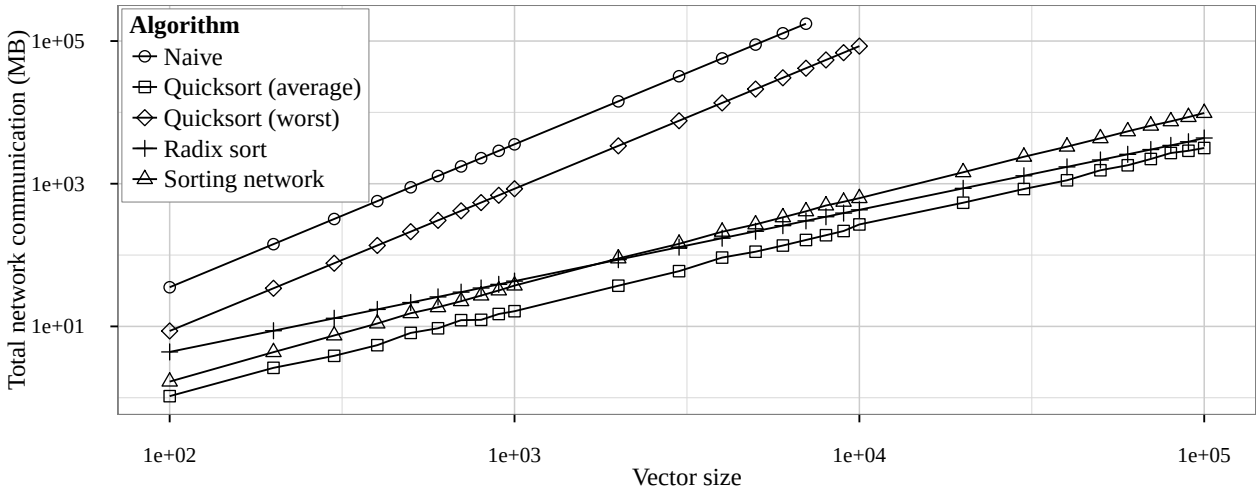


Figure 2.3: Comparison of the network usage of oblivious sorting algorithms.

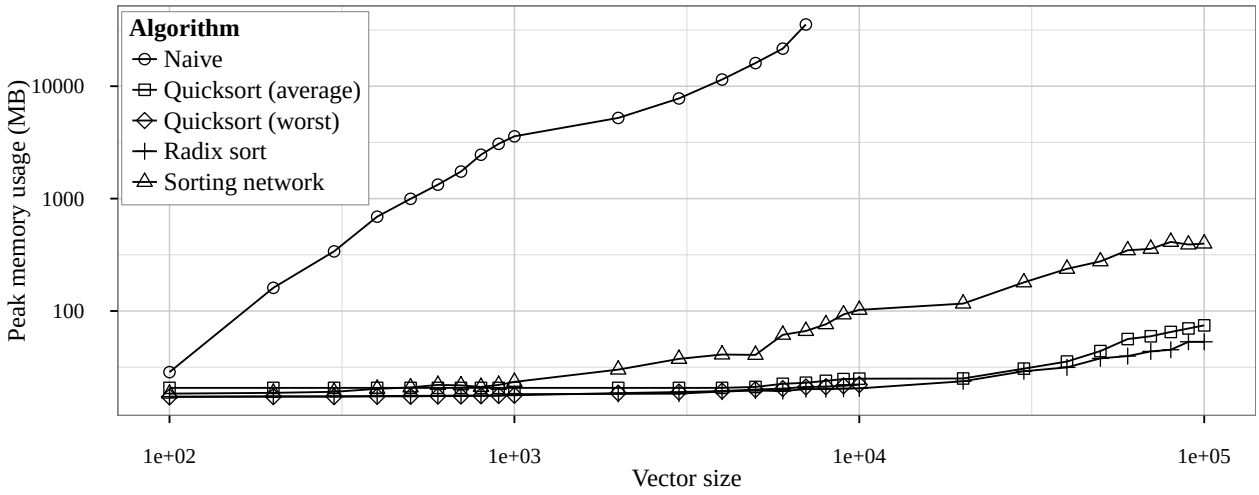


Figure 2.4: Comparison of the memory usage of oblivious sorting algorithms.

Table 2.2: DFA execution benchmarks (all times in milliseconds)

$(m,n) =$		$(3,2)$	$(15,10)$	$(100,30)$	$(1000,30)$
$GF(p)$, additive	offline	7	130	2700	22000
	FA-o.	4	70	2000	20000
	online	660	680	840	2600
$GF(2^{32})$, additive	offline	23	180	2600	24000
	FA-o.	9	180	4700	129000
	online	670	800	3400	14000
$GF(p)$, Shamir	offline	9	150	3800	22000
	FA-o.	1	12	330	6600
	online	990	1040	1300	3100
$GF(2^{32})$, Shamir	offline	29	560	11200	105000
	FA-o.	1	10	1300	94000
	online	1000	1200	5200	21000

and a transition function that maps the current state and the currently read symbol (from Σ) to the next state. The transition function is typically given in *tabular* form; in order to execute the automaton on an input string, one has to *select* from this table as many times as there are symbols in the given string. If the string is stored in the private domain, then such selection is a quite complex operation in existing SMC frameworks.

We have shown how to push almost all of this complexity of a *private selection* to the *offline* stage of the protocol [30]. In other words, most of the necessary computational effort can be expended during the precomputation, without seeing the actual input string or the automaton. Our techniques are generic and apply to any SMC framework that implements an Arithmetic Black Box [15]. In the *online* stage, the complexity of a single private selection requires just a couple of multiplications.

Table 2.2 presents our running times[2] for privately applying a m-state DFA to a 2000-character string over a n-letter alphabet. For finite automata execution, one can naturally consider even three stages — the offline, the automaton-only (where the description of the automaton is available to the protocol, but the input string is not) and the online. We have benchmarked two different ABB implementations, one based on additive sharing (as used in Sharemind), and one based on Shamir's secret sharing (as used in VIFF or SEPIA). We have also considered the representation of states and characters either as elements of $GF(2^{32})$ or $GF(p)$, where $p = 4294967291$ is the largest 32-bit prime number. Our implementations of arithmetic operations over binary fields are relatively unoptimized. Different ABB-s and fields enable different optimizations and trade-offs.

Our developed techniques will have a significant role in the UaESMC framework, allowing fast execution of a whole new class of algorithms, albeit at the cost of offline preprocessing. The techniques may also be directly applicable in the context of privacy-preserving network management, the research on which will continue during the third year of the project.

2.2.2 Intersection

Finite state automation, and the corresponding class of languages they recognize, the *regular languages*, are a wide adopted formalism to describe and analyze system behavior, textual data and DNA.

In Appendix A.3 of Deliverable D4.2.1 [22] we considered the problem of privately computing the intersection of regular languages. We assume there are two parties p_1 and p_2 knowing the languages $\mathcal{L}_1$ and $\mathcal{L}_2$, respectively, that want to compute their intersection $\mathcal{L} = \mathcal{L}_1 \cap \mathcal{L}_2$ without leaking more information than can be inferred from this intersection. We assume that the two languages are regular and defined over a

[2] All tests have been performed on the three-party setup of the Sharemind cluster in CYB [37]

Table 2.3: DFA intersection (all times in seconds)

$(m, m', n) =$	$(3, 5, 2)$	$(10, 10, 4)$
NFA product	0.5	5
NFA trim	3	31
NFA powerset	24	149
Intersection	28	185

Table 2.4: SSSP based on Bellman-Ford (all times in seconds)

$(v, e) =$	$(4, 6)$	$(8, 28)$	$(16, 120)$	$(32, 496)$
SSSP	4	24	120	747

publicly known alphabet L.

The two participants provide their language by secretly sharing the corresponding deterministic automation (DFA) and obtain back the minimal DFA recognizing the language intersection. Our implementation is based on the Sharemind SMC framework. The minimal DFA is computed using the Brzozowski's algorithm, which is based on three main building blocks: (i) computing the product of two NFA, (ii) trimming a NFA (removing all states from which no terminating state is reachable), and (iii) determinizing a NFA via the powerset construction. Table 2.3 presents our running times[3] for privately compute the language intersection of the $m-$state DFA and the $m'-$state DFA over $n-$letter alphabets. In all experiments the resulting minimal DFA consists of four states.

Our plan for third year of the project includes three main tasks: (i) compare the performance of alternative minimization algorithms (focusing on approaches that use partition refinement), (ii) investigate the benefit of allow partial information leakage (enabling the application of randomization techniques), and (iii) extend the approach to efficiently intersect multiple languages (with the specific goal of using language intersection as main building block for Distributed Monitoring).

2.3 Network management

2.3.1 Routing

In Appendix A.4 and Appendix A.5 of Deliverable D4.2.1 [22] we presented two approaches to compute single target shortest paths. These algorithms output for each gateway a single entry of the routing table, which allows to accomplish the most cost-effective delivery of packets targeted to a single destination. The algorithms uses two different strategies:

- the algorithm presented in Appendix A.4 implements a modified version of the Belman-Ford algorithm in Sharemind. The algorithm allows to compute the routing three using three servers (miners) that adopt secret sharing, requiring a centralized infrastructure
- the algorithm presented in Appendix A.5 implements a modified version of the Merlin-Segall algorithm. The algorithm computes the routing information by distributing the tasks among the involved network nodes and takes benefit of the network topology.

Table 2.4 presents the running times of the prototype implementation of the algorithm in Appendix A.4, which is based on the Sharemind SMC framework (here v and e represent the number of vertexes and edges respectively).

[3] All tests have been performed on a virtual machine

Our future plans include the design of a more efficient protocol that takes benefit of policies that allow to partially leak the information. We also plan to implement a prototype infrastructure to experiment our approach in small scale networks. Moreover, we are interested to handle systems where an autonomous system can be multi-homed and then manage several gateways. In this scenario our protocol must be refined to prevent information leakage.

2.3.2 Distributed monitoring

Assume a set of distributed agents that keep logs over locally chosen observable events. When things go wrong in the system, the usual question is the one of: "What happened?" Distributed monitoring is the problem of computing the possible (global) executions that are compatible with the (local) logs recorded by the agents. For very large distributed systems, however, the more meaningful problem is not the one of computing a global solution, but the one of computing local views of the solution, in a distributed fashion. In other words the problem is, for each agent, to infer what happened locally, that is, by communicating with the other agents to compute all possible local executions that are (i) locally consistent with the logs, and (ii) globally synchronizable. This problem is known as modular distributed monitoring. An important issue is the mathematical foundation on which to base modular distributed monitoring. Conceptually, it is natural to think in terms of partial orders between events (or event occurrences).

In Appendix A.2 of Deliverable D4.2.1 [22] we show how the distributed monitoring can be formalized in term of products and projections of languages. Moreover, we show how the problem of privacy-preserving modular distributed monitoring for two agents can be solved by employing private language intersection as the only required privacy-preserving operation. For finite languages/sets this problem has been solved in the literature. In Appendix A.2 of Deliverable D4.2.1 [22] we address the implementation of the required primitive for a class of infinite languages: regular languages. During the third year of the project we will investigate how to scale the proposed approach to interaction networks consisting of several partners. This research will be probably performed in two steps: first we will investigate tree shaped interaction networks, then we will focus on sparse networks.

2.4 Numeric algorithms

2.4.1 Linear programming

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, are amenable to privacy preservation through problem transformation.

A year ago, a number of transformations had been proposed for linear programming problems by various authors, albeit with unclear statements on their privacy properties. We intended to come up with constructions that have well-defined, cryptographic privacy guarantees according to the lines of indistinguishability under chosen-plaintext attacks. We proposed such a transformation, the security of which rested on well-defined computational assumptions [4].

During the last year, we studied these assumptions and found them to not hold. Even more, we discovered that no transformation from a large class (containing all transformations that have appeared in the literature so far) can provide sufficient privacy for a linear programming problem, unless the application of this transformation already requires one to solve the problem [28]. Previously proposed transformations can be

attacked, too [27]. Hence, barring any radically new ideas, privacy-preserving implementations of algorithms for solving linear programs remain the only possible method for privacy-preserving solutions.

In the third year of UaESMC, we hope to study such privacy-preserving implementations. In particular, we will consider *interior point methods*. These differ from the simplex method by the typically smaller number of iterations required to converge to an optimal solution. Each step, however, is more complex, requiring one to solve a large system of linear equations. But as we mentioned before, problem transformation can be used to solve systems of linear equations in the privacy-preserving manner, at least when one looks for solutions over a finite field. We are actually interested in solutions over the field of real numbers, and in the coming year we intend to study the transformations from this point.

2.4.2 Trilateration

We investigated one of the applications proposed in the Deliverable D4.1 [21]: a peer to peer positioning system. In this scenario several mobile peers dynamically discover their positions (e.g. by using the existing infrastructures or the p2pps itself) and act as further information sources for other peers.

We assume a set of peers knowing their own position. A further participant enters in the system and we assume that he has a mechanism to estimate the its distance from all other peers. The new participant wants to discover its own location by adopting "trilateration".

In Appendix A.1 of Deliverable D4.2.1 [22] we show how the quadratic system of equation can be linearized (under the assumption that at least four peers know their own position) and we describe a mechanism to outsource the computation of the resulting linear system of equations. The approach is based on problem transformation and partially homomorphic cryptosystems.

Our plan for the next year includes (i) the formal verification of soundness of the transformation that we applied to the system of equations and (ii) investigating if the approach can be extended to manage overdetermined system of equations (when the integer to real transformation is sound).

2.5 Analysis and transformation of protocols

2.5.1 Quantification of leaks in SMC protocols

We have seen examples where SMC protocols that leak a little bit of their inputs are significantly more efficient than protocols that completely preserve the privacy. Protocols for database operations with multiple tables are a concrete example [31]. Another area where such effects are likely to surface, are the privacy-preserving protocols for computations with real numbers.

We are interested in quantifying the loss of privacy, if such protocols are combined into larger applications. We intend to study such compositions using the methods of quantitative information flow analysis (see e.g. [9]). We have adopted the analysis methods to our setting and, during the third year of UaESMC, intend to study their practical performance on the kinds of problems that we have selected [3]. This study will allow us to tweak our methods, in order to provide more precise bounds for leakage.

2.5.2 Public verifiability of distributed computations

Most of the actual SMC protocols we've used in UaESMC applications have been secure only against semi-honest adversaries. There have been several reasons for that, the main of them being the efficiency of such protocols. Also, in some real-life situations one can argue that assumption of semi-honesty is natural. Protocols secure only against semi-honest adversaries may deliver wrong results, and cause the leakage of private data if some party is not following the protocol.

Protocols secure against active adversaries deliver their guarantees even if parties misbehave. An intermediate notion, introduced by Aumann and Lindell [1], is the security against *covert adversaries*. Such protocols may still deliver wrong results or expose private data if some party misbehaves, but in this case, there is a non-negligible chance that at least one of misbehaving parties is exposed. Thus one can use

other existing infrastructures (e.g. the justice system) in order to discourage misbehaviour. Security against covert adversaries is considered to correspond very well to the requirements in many real-life situations.

How cheap can protocols secure against covert adversaries be, when compared with protocols secure against passive adversaries? Damgård et al. [12] have proposed a transformation from semi-honest security to covert security that splits the execution of the protocol into two phases — the *working* phase and the *honesty verification* phase. The result of the computation is available already after the end of the working phase. Hence it is particularly important to keep the overhead of the working phase small, as long as the honesty verification can also be executed with practically reasonable resources.

The transformation of Damgård et al. [12] achieves the misbehaviour exposure probability of $(1 - 1/n)$, at the cost of n-fold increase of the execution cost (we are mostly concerned of the *communication* costs of the protocols) of the working phase of the protocol, in comparison to the execution time of the semi-honest protocol. We have significantly improved upon these results — in [29], we propose a transformation that does not significantly increase the communication of the working phase, while achieving the exposure probability of $(1 - \alpha)$, where α is negligible with respect to the security parameter. These complexity estimates hold if we consider the number of parties in the protocol to be constant.

During the third year of the project, we intend to further optimize our construction, as well as implement it in practice. We believe we can reduce the constants in the complexity measures even for larger number of involved parties, by switching from a trivial secret-sharing scheme to Shamir's scheme. The construction is going to be integrated with the Sharemind platform, making its benefits available to all users of that platform already in a relatively short time.

2.5.3 Tamper-resilient circuits

Traditionally, cryptographic algorithms are designed under the assumption that adversaries have *black box* access to the algorithms' implementation and private input. In this setting, the adversary chooses an input, supplies the algorithm with it, receives the corresponding output, and it is not allowed to alter the algorithm's internals during its execution. This mode of interaction is usually being modeled as a security game (e.g., chosen-ciphertext attack against an encryption scheme or chosen message attack against a digital signature) and the underlying cryptographic scheme is proven secure based on it. In reality though, besides observing the algorithms' input-output behaviour, an adversary may also land physical attacks on the algorithm's implementation, e.g., by inducing faults to the computation [2, 7, 26].

The holy grail in this line of research is to construct efficient compilers that transform any circuit into a tamper resilient one. Towards that direction, the solutions proposed during the last years consider proofing the circuits against attackers who tamper with circuit wires [24, 16, 10], while some of them do heavily exploit tamper-proof gates [16, 10]. This fact suggests a fundamental issue since an attacker may also land physical attacks against circuit gates [38].

During the last year, we initiated the investigation of *gate*-tampering attacks, we gave an impossibility result on tamper-resilience that applies to both gate and wire tampering adversaries by relating the amount of tampering with the depth of the circuit, we studied the relation between *wire* and *gate* tampering attackers and we proved that gate attackers are strictly stronger than the wire ones, and finally, we gave feasibility results on tamper-resilience against gate attackers by proving security for a construction that employs SMC techniques [25].

During the third year of the project, we hope to improve our results w.r.t. the size of resulting circuit, e.g., by constructing compilers that produce circuits that are larger than the original circuit only by a polylogarithmic amount. Again, we intend to employ SMC and verifiable computation techniques.

2.6 Two-party SMC protocol suites for Sharemind

This chapter introduces the details and benchmarking results of our implementation of two-party protocols and corresponding precomputation as introduced in [34] and deliverable D2.2.2. More specifically we

introduce the full asymmetric protection domain consisting of the main computation protocols and the precomputation phase. We also include an online phase of the symmetric setting and various ideas for building precomputation phase for the symmetric setting. We can conclude that the asymmetric protection domain is too inefficient for practical usage, however the symmetric protection domain is efficient enough for practical usage. In addition, our proposals for secure triple generation have varying efficiency, but the packing ideas prove to be promising.

2.6.1 Implementation platform

Our protocols are part of Sharemind 3 which is implemented in C++ as are our protection domains. Sharemind currently uses RakNet [36] as a network layer and Boost [8] for multi-threading and configuration. We used a popular free C++ cryptography library Crypto++ [11] for the functionality of the elliptic curves. In addition, the GNU Multiple Precision library (GMP) [20] was used to get unbounded integers needed to represent shares and ciphertexts in our implementation. The implementation of the Paillier cryptosystem is similar to [35], but ported to GMP.

The performance tests were executed on the Sharemind cluster where each miner ran in a different machine and they were communicating over LAN. Each of the cluster machines had 48 GB of RAM, two Intel Xeon X5670 CPUs and were connected with 1 GB/s LAN connection.

2.6.2 Benchmarks

All the given results are average running times of the operations over at least ten repeated tests, more tests were used for faster operations. Column *length* denotes the length of the input and output vectors, other columns in the following tables denote various implemented protocols. All protocols, despite being described for single values, are implemented in vectorised manner applying operations to input vectors component-wise.

All the experiments were executed using a SecreC script. We recorded the running times of each independent execution of separate operations. These results are fixed at a miner level, thus allowing us to get separate measurements from both miners. The latter is mostly important for the online protocols of the asymmetric protocol set where the workload of the miners varies significantly. It is important to note that the precomputations are running in parallel with online operations during the measurements of the online phase. This mostly affects the multiplication operation because it uses up a lot of precomputed triples that need to be replaced.

2.6.2.1 Online protocols

This section analyses the time requirements of the online phase of the asymmetric and symmetric protocol sets. We use the asymmetric setting with 2048-bit key and give the symmetric setting for 2048-bit prime as a comparison to that, as they represent similar data types. In addition, we compare the efficiency of the two computing parties in the asymmetric setting and give a 65-bit version of the symmetric PD.

Table 2.5: Time requirements of asymmetric computation protocols for party $\mathcal{CP}_1$ in Sharemind (milliseconds)

Length	Publish	Add	Subtract	ConstAdd	ConstMult	Multiply
1	21.28	0.03	0.06	0.002	0.15	218.57
10	197.67	0.10	0.41	0.008	1.37	572.57
100	1974.02	0.62	3.93	0.037	13.49	4135.15
1000	19732.16	6.27	38.87	0.170	134.19	39866.97
10000	197276.02	72.75	400.92	3.652	1343.81	392461.09

Table 2.6: Time requirements of asymmetric computation protocols for party $\mathcal{CP}_2$ in SHAREMIND (milliseconds)

Length	Publish	Add	Subtract	ConstAdd	ConstMult	Multiply
1	24.76	0.02	0.11	0.003	0.47	222.54
10	210.76	0.15	0.98	0.004	4.65	599.92
100	2103.54	1.38	9.64	0.025	46.19	4399.50
1000	20919.80	13.92	96.69	0.227	461.83	42510.33
10000	209190.81	172.94	989.36	5.749	4613.70	418776.28

Tables 2.5 and 2.6 illustrate the time requirements of the two computing parties in the asymmetric setting. Theoretical analysis in [34] indicated that this setup results in unbalanced workload for the two computing parties, and our measurements also reflect this. Local protocols of $\mathcal{CP}_1$ are two to three times faster than the same protocols for $\mathcal{CP}_2$ who also has to compute with ciphertexts. There is less difference for publishing or multiplication protocols as those are collaborative and it is likely that $\mathcal{CP}_1$ has to wait until $\mathcal{CP}_2$ finishes some computations and answers on the network, before the parties can continue. Time requirements of both miners demonstrate a linear growth as the test inputs increase, illustrating that we actually do not gain much from vectorisation and that the computations are more likely to be CPU than network bounded.

The asymmetric setting can be compared to the symmetric setting with a 2048-bit modulus. Comparing the asymmetric results in Tables 2.5 and 2.6 to those of the symmetric protocols in Table 2.7 reveals that the gain from the symmetric protocol is significant. The declassifying and, thus, also multiplication protocols have gained most as there are no more encryption operations involved in the symmetric setting.

Table 2.7: Time requirements of symmetric computation protocols for 2048-bit modulus in SHAREMIND (milliseconds)

Length	Publish	Add	Subtract	ConstAdd	ConstMult	Multiply
1	10.28	0.01	0.01	0.003	0.01	110.48
10	10.56	0.03	0.05	0.004	0.03	112.36
100	9.94	0.26	0.39	0.024	0.24	127.89
1000	11.27	2.71	3.89	0.175	1.41	223.09
10000	22.65	34.83	48.63	2.534	12.27	1147.97

A new trend in the symmetric setting is that the times to declassify a value or multiply shares do not increase linearly as the input size grows, at least for small input sizes. This probably indicates that these protocols depend more on the network speed than computation power. The sudden growth in multiplication cost for *length* 10000 can be explained by the fact it has to perform several Publish operations and the network capacity may become a bottleneck. In addition, it requires as many triples as the input length and, thus, there is continuous precomputation in the background to replace those triples. These trends can be especially well seen from Table 2.8 which also includes longer input lengths.

The comparison of Table 2.7 to Table 2.8 shows, that the considerable differences in the data type size affect the running time less than we might expect. According to Tables 2.7 and 2.8, computation with 65-bit modulus in only two to three times faster than computing with 2048-bit modulus. The difference between using 65-bit and 33-bit modulus illustrated the same trend where 33-bit modulus is only slightly faster than 65-bit. The surprising result that ConstMult is faster than Add results from the specifics of our setup where the public value is a uniformly random 32-bit element, which is small compared to general tested values. Measuring the symmetric setup with 33-bit prime gives a better estimate where ConstMult is actually approximately three times slower than Add.

These results clearly show that the symmetric setting can be more efficient than the asymmetric one, as expected. However, the symmetric PD can only be made usable if there also exists a reasonably efficient

Table 2.8: Time requirements of symmetric computation protocols for 65-bit modulus in Sharemind (milliseconds)

Length	Publish	Add	Subtract	ConstAdd	ConstMult	Multiply
1	10.51	0.02	0.01	0.005	0.01	55.79
10	10.27	0.04	0.02	0.007	0.01	56.76
100	10.16	0.23	0.19	0.023	0.05	54.33
1000	11.01	1.37	1.75	0.188	0.62	65.84
10000	24.77	13.56	17.84	0.886	4.49	203.48
100000	102.27	146.48	185.64	10.462	46.20	1880.76
1000000	846.05	1467.25	1682.50	97.189	460.60	14084.73

precomputation phase. In conclusion, the protocol set for the symmetric setup is a reasonable focus for future developments.

For simple comparison, in traditional Sharemind three miners PD multiplication of vectors of length 10000 took less than 100 milliseconds and was close to that also for shorter input lengths of 32-bit secrets [6]. Our asymmetric protocol set is significantly slower than that, but actually our symmetric protocol set can show similar speeds for 65 or 33-bit moduli. The main difference here is of course that [6] does not do precomputations. Covertly secure SPDZ [13] for two-parties reports doing 64-bit multiplications of input length 10000 in about 76 milliseconds for one thread and vectorised inputs. Our symmetric protocol set is currently slightly slower than that, but seems to be a good step from the asymmetric version.

2.6.2.2 Precomputation protocols

This section analyses the behaviour of our precomputation protocols. Table 2.9 gives the results of the time requirements of the precomputation of the asymmetric protection domain. The precomputation phase of the asymmetric protocol set is clearly less efficient than the online phase. In addition, measured results also indicate that the zero-knowledge (ZK) proofs are the most expensive part of these protocols as also noted in the theoretical analysis. The proofs take approximately $\frac{4}{5}$ of time in the singles protocol and $\frac{3}{4}$ of total time in the triples protocol. We need approximately 1.6 seconds for one 2048-bit triple, whereas SPDZ [13] can prepare one 128-bit triple in 0.4 seconds.

Table 2.9: Time requirements of asymmetric precomputation protocols in Sharemind (milliseconds)

Length	Singles with ZK	Triples with ZK	Singles	Triples
1	315	1852	42	529
10	2335	15786	402	4699
100	22492	154853	4014	46487
1000	226923	1544571	40257	464853
10000	2233351	15464414	402658	4678799

Protocol B-Triples is used exactly as given in its protocol description in [35] as packing smaller data types was native to this algorithm. The ShareConv-Triples algorithm is benchmarked using the packing idea based on the Chinese remainder theorem. We only consider packings where all packed moduli are of equal bit length for simpler exposition and comparison. We chose 65-bit and 33-bit moduli as they are sufficient to keep traditional 32-bit or 64-bit integers in them.

The CRT packing enables us to pack 15 elements of length 65-bits and 31 elements of length 33-bits into one ciphertext for 2048-bit modulus. This also explains the phenomena in Table 2.10 that lengths 1 and 10 take the same time for ShareConv-Triples—in both cases they are packed into one ciphertext and the main algorithm has the same workload. Difference between packing efficiency results in the approximately double difference between efficiency of 33-bit and 65-bit versions of these algorithms. Theoretical analysis showed

that ShareConv-Triples is the most efficient of our proposals and the measurements clearly illustrate this. ShareConv-Triples can prepare about 186 packed 65-bit triples in a second, which is approximately 12 triple generation operations. In comparison, this means that ShareConv-Triples can prepare a semi-honestly secure 65-bit triple in 0.005 seconds, and SPDZ can prepare an actively secure 64-bit triple in 0.027 seconds [13].

Table 2.10: Time requirements of Beaver triple protocols with packing in Sharemind (milliseconds)

	B-Triples		ShareConv-Triples	
Length	33-bit	65-bit	33-bit	65-bit
1	63	64	152	155
10	287	311	153	153
100	2617	2767	398	661
1000	25686	27199	2789	5458
10000	256775	270903	26948	53674

For linear packing in B-Triples, we use a security constant $\sigma = 112$, which enabled us to pack 11 elements of 33-bits and 8 elements of length 65-bit into 2048-bit of plaintext space. Both this packing inefficiency and considerably higher requirements on the network made this less efficient than ShareConv-Triples. These packing counts also explain the relatively small difference in runningtimes for 33 and 65-bit cases. For both of these moduli, $\mathcal{CP}_1$ has to encrypt all *length* elements and the gain of packing only comes from a shorter result it gets back from $\mathcal{CP}_2$ which also lessens the amount of decryptions. Hence, the effect the packing has on the overall performance is substantially smaller than for packing with CRT, but the latter gain most from reducing the amount of necessary encryption and decryption functions.

In conclusion, it seems realistic to combine one of our Beaver triple protocols with CRT packing and share conversion to use it as full precomputation in the symmetric setting. The main open issue is defining efficient general share conversion that applies to additive shares and protection mechanisms.

2.7 Applications of SMC in mechanism design

2.7.1 Privacy preserving and truthful allocation of edges in a weighted graph

Section 2.2 and Appendix A.4 of Deliverable D4.2.1 [22] describes a prototype system to that allocates edges in a weighted graph preserving the participant privacy and guaranteed truthfulness.

Weighted graphs are common abstractions to formalize a wide range of applications, including network routing, route planning and task allocation. As example, Inter-autonomous networks are managed by commercial entities that negotiate economical agreements. Each party (a commercial entity that manages one or more network link) has its own cost for packets that traverse its network and this can be taken into account in the selection of routing strategies. Each link cost represents the damage the gateway incurs to exchange a packet (e.g. network congestion). In this network we want to deliver a packet between two gateways and we want to maximize of the social welfare, that in this example is to minimize the total damage we cause. In this scenario the network costs are commercial and sensitive information and can not be publicly disclosed. Moreover, the parties can lie about their own costs, reporting higher values that cause the link to not be allocated in any packet delivery. To prevent this behavior, a payment can be guarantee for incentive the parties (and compensate their costs).

The implemented system computes the best edge allocation in a weighted undirected graph to reach all possible nodes starting from a common root (all pair shortest paths). Moreover, the prototype computes a reward for each involved edge to guarantee that there is no benefit for each party (here reporting the cost of a single edge) to lie. The prototype guarantees that the player reported costs (edge costs) are kept secret, but is allowed to disclose the computed shortest paths. This information leakage reflects the fact that in several application scenarios (e.g. network routing) it is usually possible (e.g. monitoring the traffic) to discover the resulting edge allocation.

Table 2.11: Truthful allocation of edges (all times in seconds)

$(v, e) =$	$(4, 6)$	$(8, 28)$	$(16, 120)$
total execution time (paths and payments)	15	194	2046

Table 2.11 presents the running times of the prototype implementation of the algorithm in Appendix A.4, which is based on the SHAREMIND SMC framework (here v and e represent the number of vertexes and edges respectively).

The proposed mechanism is suitable for small networks, but its centralized design can not scale to handle large and sparse networks. We plan to handle this issue during the third year. The implementation can be optimized by taking into account properties of sparse graphs, for example by limiting the number of connecting edges for each vertex. Moreover, we plan to investigate the application of Dynamic Single Source Shortest Paths algorithms in order to prevent the complete re-execution of the SSSP algorithm. Finally, we are interested in application of the presented approach to routing scenarios, where the distributed nature of the participants and their commercial relations play an important role.

2.7.2 The effect of private payments in auctions' revenue

Two critical factors that the designer of any SMC framework must take into consideration are, of course the fact that participating parties have private information that they are not willing to share and also the serious computational restrictions that this privacy enforces on any potential implementation of a specific algorithm or application. This is a backbone idea of this project, and so in this year's (second half of) Deliverable D3.3 [19] we set out to quantify the effect that selfish/private behavior has in the revenue of "simple", easy to implement auction mechanisms: this is very important for the research front in investigating potentially further integration between Game Theory and SMC. If no "acceptable" approximation ratio guarantees can be provided by such "simple" auctions, then we need to look for other candidates or techniques in case we want to use them as components in SMC systems or applications.

So, in our paper [18] we study multiple-items auctions where some prior distributional knowledge of the bidders willingness to pay is known to the seller (Bayesian setting), and we concentrate in analyzing the performance of simple, natural selling mechanisms, namely the ones that either sell each item independently or all of them in a single full-bundle. We choose the uniform and the exponential distributions as "canonical" examples from which the players' bids are drawn (they are the maximum entropy distributions in bounded and unbounded intervals, respectively).

The main tool to provide solid approximation ratios for particular algorithms, is to be able to give good upper bounds on the optimal possible revenue. This has not been achieved up to now for such multi-item auctions, neither in the Economics or CS communities. We manage to provide solid, closed-form formula bounds by utilizing the duality-theory framework for Mechanism Design developed very recently by Giannakopoulos and Koutsoupias [17] and plugging-in appropriate instantiations of dual-variables. The overall results are positive, indicating that such simple, deterministic auctions can perform very well.

Our plan for the third year of the project towards this direction is to study Mechanism Design frameworks where cryptographic and security assumptions are present in a more straightforward way, beyond just traditional truth-telling requirements in auctions, thus contributing more into the challenging area at the interface of Game Theory and Cryptography.

2.7.3 Making the payments private

The mechanisms we have designed may require one of the parties to make payments to others. E.g. to route a packet between networks, the operators of intermediate links should be payed by the party requesting the

delivery. While it is natural to assume that the requester will learn how much the delivery will cost to him, it is quite likely that he shouldn't learn how much each intermediate operator receives.

We have found that the Bitcoin infrastructure provides us with sufficient means to split one's payment into several different parts, the sizes of which become known to their receivers only. Roughly, the payment process works as follows. Here, values in the "private domain" are stored inside the SMC engine, according to the protocol set currently in use.

1. In private domain, generate m public-private key pairs $(pk_1, sk_1), \ldots, (pk_m, sk_m)$. Here m depends on the number of receivers and on the possible sizes of payments. Make the public keys public.

   Currently, Bitcoin uses the RSA signature scheme, for which the private generation of public-private key pairs is a rather complex procedure [14]. If Diffie-Hellman-based signature schemes are supported in future, then this step will become much more straightforward.

2. The payer transfers c_i credit units to the public key pk_i, such that $\sum_{i=1}^{m} c_i$ is equal to the total payment it has to make. All other parties to the computation verify that these transfers are indeed made. Here the values c_i again depend on the sizes of potential payments to receivers. In simplest case, if the total payment is P and each single payment is integral, we can take $m = P$ and $c_1 = \cdots = c_m = 1$. Optimizing the number m, based on the potential sets of payments, may be an interesting combinatorial problem.

3. Let $p_1, \ldots, p_k$ be the actual payments to be made to the parties $P_1, \ldots, P_k$. Here $p_1, \ldots, p_k$ are stored in the private domain. It is known that $\sum_{i=1}^{k} p_i = \sum_{i=1}^{m} c_i$. In the private domain, compute the array $Q = (q_{ij})_{i,j=1,1}^{m,k}$, such that

   - each secret key sk_i is stored exactly once in the array Q;
   - the rest of the elements of Q are not privacy-sensitive (they may be constants, or random numbers);
   - if $sk_{i_1}, \ldots, sk_{i_\ell}$ are stored in the j-th column of Q, then $c_{i_1} + \cdots + c_{i_\ell} = p_j$.

4. For each j, make the j-th column of Q known to the j-th receiver P_j.

The details of this scheme will be worked out during the third year of UaESMC, taking into account the needs of the tasks that require payments.

As described above, the total sum payed by the payer is known to everyone. We can hide this quantity as well, by letting the payer also be among the receivers.

Chapter 3

Towards the UaESMC Framework

The major task of the third year of the project is to systematize the work done in this project and elsewhere, giving instructions on how to solve very different kinds of computational problems with the help of SMC. During the first two years of UaESMC, we have made progress in a number of directions, matching the several of the axes outlined in the Description of Work of UaESMC for characterizing computational problems. We identified the following axes:

Data Types The values used as inputs, intermediate results, and outputs of different computational problems may be of different types, enumerative (including booleans and strings) or numeric (integral or fractional). In UaESMC, we have improved the handling of enumerative types, by proposing fast (in the online phase) methods for array accesses. Similar precomputations are also useful for certain operations with numeric values, including equality checks [32] or conversions from one bitwidth to a different one.

Control flow complexity Certain algorithms are *control-intensive* — they make a lot of comparisons and select the execution path according to their results. Others are *data-intensive* — they perform mostly the same operations, possibly very complex ones, on any input data. Using state-of-the-art technologies, the second kind of algorithms are much easier to execute in privacy-preserving manner. In UaESMC, we have not made much progress in private executions of control-intensive algorithms. Still, the methods for privacy-preserving array access can be applicable here. Also, a more careful decomposition of problems may help, demonstrating that the control decisions made during the execution do not have to remain secret at all.

Complexity of computation Different computational tasks require a different amount of computational resources, also depending on the required precision of the answer. To solve more complex tasks in privacy-preserving manner, we have to decompose the task, separating the privacy-sensitive parts. In UaESMC, we have tackled more complex problems from the area of collaborative intrusion detection. We have seen that if the result of the computation can be made public, then there are often a number of intermediate results that can be public as well, thereby allowing the task to be decomposed.

Benefit of obtaining the result Different parties to the computation may see different benefit from the computed result. In order to incentivize their participation, some sort of payments may be necessary. We have seen that there are ways to do such payments, even with the preservation of privacy.

Cost of information leakage Some leakage of private information may be tolerable, if it leads to faster algorithms. The UaESMC framework will contain methods to estimate that leakage.

Benefit of misbehaving A party may misbehave by giving "wrong" inputs to the computation, or by not following the protocol it is supposed to execute. The first kind of misbehaviour has to be fought with the methods of mechanism design, while the second can be dealt using cryptographic means. In UaESMC, we have studied the incentives for parties to give "correct" inputs to different kinds of

computational problems. Our results on security against covert adversaries take away any motivation for the parties to not follow the cryptographic protocols they are supposed to execute, as long as some judicial power is wielded over them.

Connectivity We usually assume that each pair of computing nodes can directly exchange information between each other. Such assumption may be unrealistic in certain settings, including tasks related to network management. We have seen that for relevant tasks, it is possible to perform the privacy-preserving computations in a manner that a full mesh connectivity is not needed.

Additional infrastructure While planning the project, we believed that the use of available public key infrastructures would help privacy-preserving protocols for certain tasks, e.g. those used for agreeing on some further activities. Hence we have proposed methods to construct valid signatures directly inside SMC protocols. During the project, we have also found that the available digital currency (Bitcoin) infrastructures can be highly useful for executing mechanisms involving payments in privacy preserving manner. We see the protocols making use of such infrastructures as a significant component of the UaESMC framework.

We thus consider to have made good progress towards an expansive framework for privacy-preserving computations. During the third year of the project, we have to further systematize our results, align different techniques with different characteristics of computational problems, identify any remaining gaps and attempt to fill those, and give a number of recipes usable in different settings. According to the project plan of UaESMC, we are going to perform a second round of interviews with potential end-users of SMC techniques. Their perspective on the functional and non-functional requirements on the problems that interest them allows us to align our techniques with real-world needs once more before finalizing the UaESMC framework.

Bibliography

[1] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. *J. Cryptology*, 23(2):281–343, 2010.

[2] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In *Advances in Cryptology-CRYPTO'97*, pages 513–525. Springer, 1997.

[3] Dan Bogdanov, Yiannis Giannakopoulos, Roberto Guanciale, Liina Kamm, Peeter Laud, Pille Pruulmann-Vengerfeldt, Riivo Talviste, Kadri Tõldsepp, and Jan Willemson. Scientific Progress Analysis and Recommendations, January 2013. UaESMC Deliverable 5.2.1.

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

[5] 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.

[6] Dan Bogdanov, Margus Niitsoo, Tomas Toft, and Jan Willemson. High-performance secure multi-party computation for data mining applications. *Int. J. Inf. Sec.*, 11(6):403–418, 2012.

[7] Dan Boneh, Richard A DeMillo, and Richard J Lipton. On the importance of checking cryptographic protocols for faults. In *Advances in Cryptology-EUROCRYPT'97*, pages 37–51. Springer, 1997.

[8] Boost - C++ libraries. `http://www.boost.org/`. Last accessed 2013-04-02.

[9] David Clark, Sebastian Hunt, and Pasquale Malacaria. A static analysis for quantifying information flow in a simple imperative language. *Journal of Computer Security*, 15(3):321–371, 2007.

[10] Dana Dachman-Soled and Yael Tauman Kalai. Securing circuits against constant-rate tampering. In *Advances in Cryptology–CRYPTO 2012*, pages 533–551. Springer, 2012.

[11] Wei Dai. Crypto++ library. `http://www.cryptopp.com/`. Last accessed 2013-04-02.

[12] Ivan Damgård, Martin Geisler, and Jesper Buus Nielsen. From passive to covert security at low cost. In Micciancio [33], pages 128–145.

[13] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical Covertly Secure MPC for Dishonest Majority - Or: Breaking the SPDZ Limits. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *ESORICS*, volume 8134 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2013.

[14] Ivan Damgård and Gert Læssøe Mikkelsen. Efficient, Robust and Constant-Round Distributed RSA Key Generation. In Micciancio [33], pages 183–200.

[15] Ivan Damgård and Jesper Buus Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 247–264. Springer, 2003.

[16] Sebastian Faust, Krzysztof Pietrzak, and Daniele Venturi. Tamper-proof circuits: How to trade leakage for tamper-resilience. In *Automata, Languages and Programming*, pages 391–402. Springer, 2011.

[17] Yiannis Giannakopoulos and Elias Koutsoupias. Duality and optimality for uniform auctions. Manuscript, 11 2013.

[18] Yiannis Giannakopoulos and Elias Koutsoupias. Bounding optimal revenue in multiple-items auctions. Manuscript, 2014.

[19] Yiannis Giannakopoulos, Yiannis Tselekounis, and Add More. Quantification of the Effects of Partially Revealing Private Data, January 2014. UaESMC Deliverable 3.3.

[20] Torbjörn Granlund. GMP: The GNU multiple precision arithmetic library. `http://gmplib.org/`. Last accessed 2013-04-02.

[21] Roberto Guanciale. Identification of application scenarios, January 2013. UaESMC Deliverable 4.1.

[22] Roberto Guanciale, Dilian Gurov, and Add More. Algorithms, January 2014. UaESMC Deliverable 4.2.1.

[23] Koki Hamada, Ryo Kikuchi, Dai Ikarashi, Koji Chida, and Katsumi Takahashi. Practically Efficient Multi-party Sorting Protocols from Comparison Sort Algorithms. In *Proc. of ICISC'12*, volume 7839 of *LNCS*, pages 202–216. Springer, 2013.

[24] Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and David Wagner. Private circuits ii: Keeping secrets in tamperable circuits. In *Advances in Cryptology-EUROCRYPT 2006*, pages 308–327. Springer, 2006.

[25] Aggelos Kiayias and Yiannis Tselekounis. Tamper resilient circuits: The adversary at the gates. In *Advances in Cryptology-ASIACRYPT 2013*, pages 161–180. Springer, 2013.

[26] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in Cryptology-CRYPTO'99*, pages 388–397. Springer, 1999.

[27] 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.

[28] 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.

[29] Peeter Laud and Alisa Pankova. Verifiable computation in multiparty protocols with honest majority. Cryptology ePrint Archive, Report 2014/060, 2014. `http://eprint.iacr.org/`.

[30] Peeter Laud and Jan Willemson. Universally composable privacy preserving finite automata execution with low online and offline complexity. Cryptology ePrint Archive, Report 2013/678, 2013. `http://eprint.iacr.org/`.

[31] Sven Laur, Riivo Talviste, and Jan Willemson. From Oblivious AES to Efficient and Secure Database Join in the Multiparty Setting. In *Applied Cryptography and Network Security*, volume 7954 of *LNCS*, pages 84–101. Springer, 2013.

[32] Helger Lipmaa and Tomas Toft. Secure equality and greater-than tests with sublinear online complexity. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *ICALP (2)*, volume 7966 of *Lecture Notes in Computer Science*, pages 645–656. Springer, 2013.

[33] Daniele Micciancio, editor. *Theory of Cryptography, 7th Theory of Cryptography Conference, TCC 2010, Zurich, Switzerland, February 9-11, 2010. Proceedings*, volume 5978 of *Lecture Notes in Computer Science*. Springer, 2010.

[34] Pille Pullonen. Actively secure two-party computation: Efficient Beaver triple generation. Master's thesis, University of Tartu, Aalto University, 2013.

[35] Pille Pullonen, Dan Bogdanov, and Thomas Schneider. The design and implementation of a two-party protocol suite for Sharemind 3. Technical report, Cybernetica AS Infoturbeinstituut, 2012. `http://research.cyber.ee`.

[36] RakNet - multiplayer game network engine. `http://www.jenkinssoftware.com/`. Last accessed 2013-04-02.

[37] Reimo Rebane. A feasibility analysis of secure multiparty computation deployments. Master's thesis, University of Tartu and Aalto University, 2012. `http://comserv.cs.ut.ee/forms/ati_report/index.php?year=2012`.

[38] Sergei P Skorobogatov and Ross J Anderson. Optical fault induction attacks. In *Cryptographic Hardware and Embedded Systems-CHES 2002*, pages 2–12. Springer, 2003.

[39] Kadri Tõldsepp, Pille Pruulmann-Vengerfeldt, and Peeter Laud. Requirements specification based on the interviews, July 2012. UaESMC Deliverable 1.2.