D11.2

An Evaluation of Current Protocols based on Identified Model

Project number:	609611
Project acronym:	**PRACTICE**
Project title:	Privacy-Preserving Computation in the Cloud
Project Start Date:	1st November, 2013
Duration:	36 months
Programme:	FP7/2007-2013
Deliverable Type:	Report
Reference Number:	ICT-609611 / D11.2 / 1.0
Activity and WP:	Activity 1st / WP11
Due Date:	October 2015 - M24
Actual Submission Date:	3rd November, 2015
Responsible Organisation:	BIU
Editor:	Benny Pinkas
Dissemination Level:	PU
Revision:	1.0
Abstract:	This document describes a set of application scenarios for secure multi-party computation, a performance comparison of secure multi-party computation systems written by members of the PRACTICE project, and conclusions based on the results of the performance comparison.
Keywords:	Secure multi-party computation

This project has received funding from the European Union's Seventh Framework Programme for research, technological development and demonstration under grant agreement no. 609611.

Editor

Benny Pinkas (BIU)

Contributors (ordered according to beneficiary numbers)

Florian Kerschbaum (SAP)
Florian Hahn (SAP)
Thomas Schneider (TUDA)
Michael Zohner (TUDA)
Reimo Rebane (CYBER)

Executive Summary

This deliverable analyzes the performance of existing secure multi-party computation systems based on identified application scenarios and models. The systems that are compared are the ABY, FRESCO/SPDZ and Sharemind systems for generic secure multi-party computation, and the SEEED system for queries on encrypted data. All systems were implemented by members of the PRACTICE project. The deliverable benchmarks the run time of primitive operations in these systems, and uses an emulation to estimate, based on these results, the run time of complex protocols using these systems.

The main conclusions of the experiments are that (1) The performance of systems that are secure against malicious (active) adversaries is considerably slower than the performance of systems that are only secure against semi-honest (passive) adversaries. (2) There is therefore a need for improving the efficiency of systems providing security against malicious adversaries. (3) The performance of a protocol for a specific task can greatly exceed that of generic secure computation protocols that are applied to the same task. Therefore efforts should be invested in designing specific protocols for tasks of high importance where the required computation does not have an efficient representation in a format that is suitable for generic protocols for secure computation. This conclusion can apply to search on encrypted data, and for computing private set intersection.

Contents

List of Figures

List of Tables

Chapter 1

Application Scenarios and Models

Different application scenarios define different models for secure computation solutions. Deliverable D12.2, “Adversary, Trust, Communication and System Models”, specified such different models in different domains:

- The *adversary model* captures the strength of realistic adversaries,which typically can be either semi-honest (also known as passive adversaries), or malicious (also known as active adversaries). This model helps in providing an adequate level of security for a particular application.
- The *trust model* determines which levels of trust can be assumed. The possibility of leveraging trusted hardware in a scenario is also evaluated in the trust model.
- The *communication model* defines the different communication channels and their related explicit and implicit assumptions.
- The *system model* describes the capabilities and functional properties of different participants, including properties such as computation power, connection bandwidth, relevance of parallelism, etc.

1.1 Application Scenarios

Deliverable D12.2 defines thirteen application scenarios, covering a wide range of use cases for secure computing. The scenarios were collected among the project partners and represent real-world applications for which secure computation is an enabling technology, due to their intrinsic security and trust requirements.
The application scenarios are described in detail in Chapter 4 of D12.2. We provide here a concise description of the scenarios. The scenarios are grouped into four different categories: *joint business applications*, *joint studies applications*, *location sharing applications* and *end user applications*.
The first category, joint business applications, involves companies that are interested in cooperating with each other without revealing sensitive internal data of their company. Scenarios from this category use secure computation to jointly evaluate calculations, e.g., supply chain optimization, based on sensitive company data without revealing the data itself. Joint business applications that are investigated in this deliverable are:

- **Aeroengine Fleet Management:** This scenario describes a system that enables the optimization of the maintenance repair and overhaul process for the engine sector of the aeronautic supply chain. Maintenance plans can be calculated without revealing the participating companies data. An in-depth analysis of this use case as well as a prototype implementation are the objectives of Work Package (WP) 24.

- **Consortium Gathering Information from Its Members:** A consortium would like to gather information from its members, e.g., benchmarking economic results. Secure computation enables competing companies to contribute their private data to the consortium without risking disclosure of the individual data.
- **Platform for Auctions:** Multiple parties negotiate in an auction without revealing their bids. Exemplary markets are spectrum and electricity auctions.
- **Platform for Benchmarking:** A privacy preserving platform for benchmarking between business partners enables a trustworthy assessment. Partners can evaluate each other regarding different factors, i.e., credit card rating, without divulging losing sensitive company data.
- **Tax Fraud Detection:** Detecting tax frauds is an important scenario in which state entities are interested in analyzing precise financial data of companies. With the help of secure computation, a precise analysis of money flows can be executed without the necessity to reveal the companies' sensitive financial data to the revenue office.

In the second area, namely joint studies applications, sensitive data of many individuals or entities is used for studies and statistics without exposing the individual's data at any time. In this area we discuss the following scenarios:

- **Joint Statistical Analysis Between State Entities:** In some cases the law forbids the compilation of so-called super-databases from the individual datasets of different state entities. To enable a joint study across different entities, secure computation can be used to join databases in a privacy-preserving manner that fulfils the legal requirements.
- **Privacy-Preserving Genome Studies Between Biobanks:** Biobanks from different countries can perform a joint genome-wide association study using each other's data without breaching the donors' privacy using secure computation.
- **Privacy-Preserving Personal Genome Analyses and Studies:** Similar to the service offered by companies such as 23andMe, donors can submit their genome data and enter their phenotype data to receive feedback on genetic associations with specific illnesses and disorders. Secure computation can be used to prevent any mishandling of the donors' data.
- **Surveys on Sensitive Data:** A cloud system that provides a platform for privacy-preserving surveys. A survey creator submits a survey to the platform that is then filled with opinions from invited participants. Using secure cloud computing, the survey is evaluated and only the result is sent back to the creator. Thus, with the help of secure computation, the participants' input data can be protected.

Privacy-preserving location sharing is of relevance in the following two scenarios:

- **Location Sharing with Nearby Contacts:** Location information of smart phone users is sensitive, yet useful for social activities where contacts meet. With the help of secure computation, proximities can be calculated without revealing actual location data.

- **Privacy-Preserving Satellite Collision Detection:** Different countries wish to forecast collisions between their satellites without revealing the exact location and trajectory of their satellites.

The last group of scenarios is of end user applications. These scenarios aim towards increasing the end user's privacy when using cloud services. The described applications in this area are:

- **Key Management:** With the increasing number of devices used by an end user, cryptographic keys need to be shared between different devices more and more frequently. To avoid a centralized trusted third party, i.e., key server, a solution based on secure computation is preferable and is described in this scenario.
- **Mobile Data Sharing:** This scenario provides privacy-preserving data sharing between different mobile devices and users through the cloud. Data are stored in the cloud only encrypted (i.e., not inspectable by the cloud service provider) but still sharable between users, even if the users have their data stored on different cloud storage providers.

1.2 Models

Adversary model Different applications require different levels of security and thus different adversary models can be assumed for the underlying protocols so that the required security level for each application scenario is met. The adversary model identifies an adequate level of security, because a higher security level usually has negative impacts on the efficiency.
The security requirements in the setting of multi-party computation must hold even when some of the participating parties misbehave. Aumann and Lindell [1] distinguish three adversary models that are used to describe the attacker model in each scenario:

- *Malicious adversaries* (also known as active adversaries) are adversaries that may behave arbitrarily and are not bound in any way to follow the instructions of the specified protocol. Protocols that are secure in the malicious model provide a very strong security guarantee for the user.
- *Covert adversaries* have the property that they may deviate arbitrarily from the protocol specification in an attempt to cheat, but do not wish to be "caught" doing so. Protocols secure in the covert model guarantee that an adversary is caught cheating with at least a defined probability ϵ.
- *Semi-honest adversaries* (also known as passive, or honest-but-curious, adversaries) correctly follow the specified protocol, yet they may attempt to learn additional information by analysing the transcript of messages received during the execution. Security in the presence of semi-honest adversaries provides a weaker security guarantee, yet might already be sufficient if the adversary is given limited access to the computation, e.g. through defined interface to framework executed in isolation (like trusted hardware).

In settings with more than two parties, it is also possible to consider an adversary model with an *honest majority*, where multiple participants (of the same type) are involved in a protocol and it is assumed that most of them (the majority) act in a benign way.

Trust model Depending on the base problem of a scenario and the solution approach chosen in the scenario, different levels of trust in the participating components and parties can be assumed. The trust model identifies trusted parties and components and the degree of trust one can place in them. A party is called trusted, if it behaves exactly as requested by the protocol. It is important to note that sometimes there are some *implicit* trust assumptions, such as in Certificate Authorities when using Public Key Infrastructure.
A special case which should be considered in trust model is the use of *trusted hardware*, which can increase both security and efficiency. This refers to the usage of cryptographic functionalities in dedicated hardware devices such as smartcards, Hardware Security Modules (HSM) or integrated into complex hardware components like processors (like Trustzone, or Intel's SGX).

Communication model The communication model specifies which parties can communicate between themselves, and also finer characteristics of the communication channels. For example, the requirement that a cloud provider is always online. Or the requirement for simultaneous communication, or for messages to be transferred within a specific maximum delay.

System model The system model reflects the capabilities and properties of the parties participating in the application scenario. The model considers system benchmarks such as computational power, amount of memory, network connection properties, parallelism of computation, reuse of services, etc.

1.3 Relation

The application scenarios describe the most promising usages of secure computation technology. The adversary, trust, communication and system models were defined based on these applications, and model how different secure computation solutions should be evaluated. The task of the work reported here was to examine the existing secure computation solutions according to these models, and identify gaps where new and improved protocols are needed.

Chapter 2

Performance Comparison

Our performance analysis compared three systems for generic secure multi-party computation, that were designed by partners of the PRACTICE project: ABY, SPDZ/FRESCO and Sharemind. We provide in Section 2.1 a brief description of these systems. In addition, in Section 2.4 we describe the performance of the SEEED system for queries on encrypted databases. (This system is not for generic multi-party computation and therefore is no directly comparable with the other systems.)

2.1 The Systems that were Compared

This section describes the three systems for generic secure multi-party computation that were compared in our tests.

2.1.1 The ABY System

ABY(for Arithmetic, Boolean, and Yao sharing), introduced in [9], is a novel framework for developing highly efficient mixed-protocols that allows a flexible design process. ABY was designed using several state-of-the-art techniques in secure computation and by applying existing protocols in a novel fashion. It uses optimized sub-routines based on a detailed benchmark of the primitive operations. ABY is intended as a base-line on the performance of privacy-preserving applications, since it combines several state-of-the-art techniques and best practices in secure computation. The source code of ABY is freely available at `http://encrypto.de/code/ABY`. ABY provides security only against semi-honest adversaries.
On a very high level, the ABY framework works like a virtual machine that abstracts from the underlying secure computation protocols (similar to the Java Virtual Machine that abstracts from the underlying system architecture). The virtual machine operates on data types of a given bit-length (similar to 16-bit short or 32-bit long data types in the C programming language). Variables are either in Cleartext (meaning that one party knows the value of the variable, which is needed for inputs and outputs of the computation) or secret shared among the two parties (meaning that each party holds a share from which it cannot deduce information about the value). The ABY framework currently supports three different types of sharings (Arithmetic, Boolean, and Yao) and allows to efficiently convert between them, see Figure 2.1. The sharings support different types of standard operations that are similar to the instruction set of a CPU such as addition, multiplication, comparison, or bitwise operations. Operations on shares are performed using highly efficient secure computation protocols: for operations on Arithmetic sharings it uses protocols based on Beaver's multiplication triples [4], for operations on Boolean

sharings it uses the protocol of Goldreich-Micali-Wigderson (GMW) [34], and for operations on Yao sharings it uses Yao's garbled circuits protocol [74].

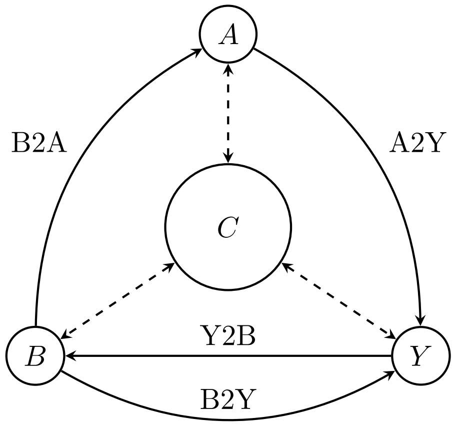

Figure 2.1: Overview of our ABY framework that allows efficient conversions between **C**leartexts and three types of sharings: **A**rithmetic, **B**oolean, and **Y**ao.

Flexible Design Process A main goal of the ABY framework is to allow a flexible design of secure computation protocols. The framework abstracts from the protocol-specific function representations and instead uses standard operations. This allows to mix several protocols, even with different representations, and allows the designer to express the functionality in form of standard operations as known from high-level programming languages such as C or Java. Previously, designers had to manually compose (or automatically generate) a compact representation for the specific protocol, e.g., a small Boolean circuit for Yao's protocol. As the framework focuses on standard operations, high-level languages can be compiled into our framework and it can be used as backend in several existing secure computation tools, e.g., L1 [44], [71], [72], SecreC [11], [12], or PICCO [75].
By mixing secure computation protocols, the ABY framework is able to tailor the resulting protocol to the resources available in a given deployment scenario. For example, the GMW protocol allows to pre-compute all cryptographic operations, but the online phase requires several rounds of interaction (which is bad for networks with high latency), whereas Yao's protocol has a constant number of rounds, but requires symmetric cryptographic operations in the online phase.

Efficient Instantiation and Improvements Each of the secure computation techniques is implemented in ABY using the most recent optimizations and best practices such as batch precomputation of expensive cryptographic operations. For Arithmetic sharing ABY generates multiplication triples via Paillier with packing or DGK with full decryption, for Boolean sharing it uses OT extension, and for Yao sharing it uses fixed-key AES garbling.

Feedback on Efficient Protocol Design The work on ABY performed benchmarks of the framework, from which new best-practices for efficient secure computation protocols were derived. it was shown that for multiplications it is more efficient to use OT extensions for pre-computing multiplication triples than homomorphic encryption. With the new OT-based conversion protocols of ABY, converting between different share representations is considerably cheaper than the methods used in previous works, and scales well with increasing the security parameter. In fact, on a low latency network, the conversion costs between different share representations are so cheap that already for a single multiplication it pays off to convert into a more suited representation, perform the multiplication, and convert back into the source representation.

2.1.2 The FRESCO System

FRESCO is a Java framework for efficient secure computation that is being jointly developed by The Alexandra Institute and Aarhus University. The goal of the FRESCO framework is to support the implementation of secure computation applications, and to make it easy to experiment with and compare different approaches to secure computation. To this end the framework is designed to be modular so that various components involved in a secure computation can be replaced and reused. These components include such things as

- Underlying secure computation protocols.
- Circuit construction and evaluation strategies.
- Network communication strategies.

A FRESCO application consists of two main parts: a circuit description of the function to be securely evaluated, and a run-time system that evaluates the circuit according to some underlying protocol for secure computation.

FRESCO Circuit description In FRESCO functions to be securely evaluated are described as circuits. In order to decouple the circuit description from the underlying protocol, the circuits are abstract in that they are not explicitly taken to be e.g. boolean or arithmetic circuits. The framework supplies a library of interfaces for basic circuits, such as circuits computing arithmetic and boolean operations. The application programmer can combine these basic circuits into a generic circuit that computes whatever function she desires. It is then up to the implementer of the run-time system to provide implementations of the circuits for the basic operations.

FRESCO Run-Time Systems Run-time systems in FRESCO specify how circuits are evaluated, and are thus highly dependent on the underlying protocol for secure computation that they support. The run-time system must define the notion of a gate used by the protocol and how each gate type is to be evaluated. There is no restriction that a gate must implement specific arithmetic or boolean operations. In fact a gate is simply seen as an unit of computation that requires at most a single round of communication. From the gates it provides a run-time system also provides implementations of (at least a subset of) the basic circuits described above. Additionally a run-time system may provide a number of strategies for gate evaluation and network communication. Such strategies may control how gates are scheduled for evaluation, whether they are evaluated sequentially or in parallel an many other aspects of the evaluation. Currently run-time systems written for FRESCO includes support for the following protocols for secure computation:

- The TinyOT protocol by Nielsen *et al.* for maliciously secure two-party computation based on boolean circuits [14].
- The Bedoza protocol by Bendlin *et al.* for maliciously secure multi-party computation based on arithmetic circuits [2].

- The SPDZ protocol by Damgård *et al.* for maliciously and covertly secure multi-party computation based on arithmetic circuits [8, 7].
- The protocol by Gennaro *et al.* for semi-honest secure multi-party computation based on arithmetic circuits [11].
- The protocol by Katz and Malka for semi-honest secure private function evaluation based on boolean circuits [13].

2.1.3 The Sharemind System

Sharemind [4, 5, 3] is a secure service platform for data collection and analysis. Designed as a distributed secure database and application server, it is capable of collecting, storing and processing confidential data without compromising the privacy of individual records.
At its core, Sharemind uses secure multiparty computation technology to achieve the necessary cryptographic security in data storage and computations. More specifically, it is based on 3-party *additive secret sharing scheme* in the ring of 32-bit integers, i.e., a secret $s \in \mathbb{Z}2^{32}$ is split into three random shares $s_1, s_2, s_3 \in \mathbb{Z}2^{32}$ such that $s_1 + s_2 + s_3 \equiv s \pmod{2^{32}}$. In this particular implementation the computation protocols are provably secure in the *honest-but-curious* security model with no more than one semi-honest corrupted party.
Sharemind can be programmed to perform various secure computations, thus enabling the development and execution of custom data processing applications. Its protocol suite is universally composable, allowing the basic secure operations to be composed sequentially to form programs, and in parallel to achieve efficient SIMD (single instruction, multiple data) operations on vectors. Sharemind implements a distributed virtual machine that provides a consistent instruction set for accessing secure computational resources, while abstracting away most of the low-level protocol implementation details. The secure computation algorithms can be specified either in the low-level Sharemind assembly language interpreted directly by the virtual machine, or in the high-level privacy-aware programming language called SecreC.
The protocol suite of Sharemind covers basic arithmetic and comparison on integers. All operations are designed to be performed pointwise on vectors of inputs. Both unary and binary operations are supported.
Sharemind enables users to choose which underlying secure computation method suits them best. In the following we describe the protection domain kinds currently implemented for Sharemind .

- **Public virtual machine** controls the public execution flow and powers the public protection domain in Sharemind 3, allowing to store and process data publicly. The VM supports signed and unsigned integers (8, 16, 32 and 64 bit) and floating point values (32 and 64 bit), as well as heap manipulation functionality. The booleans and public strings are simulated types on the SecreC level.
- **additive3pp** is the 3-party MPC protocol suite based on additive secret-sharing in the semi-honest model. The supported data types include booleans, signed and unsigned integers (8 to 64 bit), floating point values (32 and 64 bit) and xor-shared strings.
- **additive2pp** is the 2-party MPC protocol suite based on additive secret-sharing and additively homomorphic Paillier cryptosystem in the semi-honest model. It supports arithmetic on 32-bit integers. [18]

- **additive2pa and additive2pa_sym** are the 2-party MPC protocol suites similar to additive2pp, but achieve malicious security by protecting the shares with MACs. Both support arithmetic on 32-bit integers. [17]

2.2 Main Features of the Systems

There are major differences in the features and the security levels that are guaranteed by the different systems.
Before describing these differences, we quickly recall the notions of security against *semi-honest adversaries* (also known as passive adversaries, or honest-but-curious adversaries), and security against *malicious adversaries* (also known as active adversaries): Semi-Honest adversaries are guaranteed to operate according to the specification of the protocol that they should be running. Namely, it is assured that they will run the program that they are asked to run. However, they might examine the messages that they receive and try to obtain from these messages information about the inputs of other participants in the protocol. Malicious adversaries, on the other hand, may act arbitrarily. That is, they might run an arbitrary program, send arbitrary messages, and might not follow the operation specified for them by the protocol.
Malicious adversaries are often a more realistic threat model. However, protocols which guarantee security against malicious adversaries are typically considerably less efficient than protocols which only offer security against semi-honest adversaries.

The systems that we examined The systems that we examined differ in the setting in which they work and the security level that they guarantee:

- ABY is a system for *two-party* computation, which is secure against *semi-honest* adversaries.
- FRESCO/SPDZ can work both in the *two-party* and the *multi-party* settings (i.e., in a setting with strictly more than two parties). It is secure against *malicious* adversaries.
- Sharemind is mostly focused on a setting where data is shared between 3 parties (i.e., works in a 3-party setting). It mostly provides security against *semi-honest* adversaries.

It is therefore apparent that ABY and Sharemind work in different settings (two-party vs. 3-party computations), whereas FRESCO/SPDZ can work in both of these settings. Furthermore, ABY and Sharemind provide security against the weaker notion of semi-honest adversaries, whereas FRESCO/SPDZ provides security against stronger, malicious adversaries. This additional security of SPDZ/FRESCO obviously comes at a cost, which will be apparent in the performance comparison.

2.3 Performance Comparison of Systems for Generic Secure Computation

For each of the three secure computation systems that were examined, ABY, SPDZ/FRESCO and Sharemind, we used a benchmarking tool to measure the computation time of the primitive operations. We ran benchmarks for a range of input sizes starting from 1 operation to 1 million operations. We increased the inputs size by powers of 10, from 1 operation to 1 million

operations. The benchmarking was run on machines with 2x Intel X5670 2.93 GHz CPUs and 48GB RAM, and 1 Gbit network links between each of the machines.
The benchmarking provided the running time of the primitive operations for the different input sizes. With this information we built mathematical (regression) models to estimate the running time of these operations in an *emulator* which emulated time of complete protocols based on these operations.
We provide two type of performance results. First, we describe the run time of the basic operations. Then, we describe the emulated run time of complete protocol based on these operations.

2.3.1 The Performance of Primitive Operations

We describe in this section both the performance of single operations, and the amortized performance. The single operation performance is computed when running the operation on a single value (or a pair of values) at a time. This case parallelizes poorly and is shown as a worst case performance. An example of this case would be when it is needed to chain two multiplications, where the result of the first multiplication is the input for the second one.
The amortized performance is computed when running the operation on a larger number of input values that do not depend on each other. This case parallelizes very well and is shown as a best case performance. We have taken the best performance for each operation from all of the input sizes.

The tables A performance comparison between the primitive operations of the ABY, Sharemind and SPDZ/FRESCO secure computations systems is shown in Table 2.1 and Table 2.2. The results for SPDZ/FRESCO include only the runtime of the online phase.
The performance is shown in computed operations per second. Note that the number suffixes used in the table are $K = 10^3$, $M = 10^6$ and $G = 10^9$. Table 2.1 shows the performance of running the operation on a single value. The results in Table 2.2 show the amortized best performance for running the operation on 1 to 10^6 values.
The ABY system uses multiple secure computation schemes (such as arithmetic circuits, boolean circuits and Yao's garbled circuits). As a result, operations have implementations in more than one scheme, and we therefore only show the result for the best performing protocol. (When measuring the performance of a single basic operation, rather than their amortized overhead or an evaluation of many basic operations, there is no scheme that performs consistently better than the others.)
Sharemind does not have a separate protocol for the MUX operation. The operation is done using 1 MUL and 2 ADD operations.
A detailed analysis of the results appears in Section 2.5.1. We only comment FRESCO/SPDZ has lower performance than ABY, which has lower performance than Sharemind. The relatively low performance of FRESCO/SPDZ is obvious given the fact that it is the only system providing security against malicious adversaries. The implementation of the Sharemind system is the more mature of all protocol implementations and is therefore more efficient than the other implementations. Note that Sharemind does not work in a setting with only two parties, and therefore the performance in this setting is inferior to that in a setting with three or more parties.

Operation	Bit length	ABY (ops/sec)	SPDZ/FRESCO (ops/sec)	Sharemind (ops/sec)
ADD	8	204 (arith)	–	143K
	16	160 (yao)	–	333K
	32	116 (bool)	45	333K
	64	170 (yao)	50	333K
CMP	8	160 (bool)	–	1.40K
	16	168 (yao)	–	1.21K
	32	119 (bool)	4	950
	64	118 (bool)	3	914
EQ	8	149 (yao)	–	2.14K
	16	152 (yao)	–	1.44K
	32	137 (yao)	7	1.49K
	64	138 (bool)	8	1.24K
MUL	8	166 (arith)	–	3.98K
	16	144 (arith)	–	4.27K
	32	106 (yao)	43	4.27K
	64	142 (arith)	43	4.23K
MUX	8	154 (yao)	–	–
	16	171 (yao)	–	–
	32	177 (bool)	40	–
	64	128 (bool)	40	–

Table 2.1: Single operation performance

Operation	Bit length	ABY (ops/sec)	SPDZ/FRESCO (ops/sec)	Sharemind (ops/sec)
ADD	8	9.01M (arith)	–	617M
	16	3.86M (arith)	–	1.05G
	32	1.27M (arith)	800K	1.02G
	64	350K (arith)	746K	803M
CMP	8	255K (bool)	–	699K
	16	118K (bool)	–	410K
	32	56.5K (bool)	248	240K
	64	27.2K (bool)	208	129K
EQ	8	608K (bool)	–	3.00M
	16	310K (bool)	–	2.32M
	32	158K (bool)	137	1.68M
	64	79.1K (bool)	135	1.17M
MUL	8	486K (arith)	–	18.0M
	16	262K (arith)	–	11.3M
	32	132K (arith)	78.4K	6.56M
	64	60.8K (arith)	76.9K	3.45M
MUX	8	591K (bool)	–	–
	16	298K (bool)	–	–
	32	154K (bool)	991	–
	64	77.3K (bool)	1.00K	–

Table 2.2: Amortized operation performance

2.3.2 The Emulated Performance of Chosen Protocols

We emulated the run time of two simple protocols which are representative of the core tasks of secure computation applications:

- An auction. The auction has n inputs. The output is the largest value, and its index.
- A filtered average task. The inputs for this function are values $x_1, \ldots, x_n, y_1, \ldots, y_n$ and the output is the filtered average $\sum_{i=1}^{n} x_i y_i / \sum_{i=1}^{n} y_i$, where the x_i values are integers and the y_i values are equal to either 0 or 1.

The performance was emulated using the benchmarking results of the single operations. (Therefore, the performance corresponds to the same machines and setting that were used in the benchmarking.) The emulation includes only the run time of the secure computation, and not the run time of public computation and of the system overhead. The computations were done on 64 bit integers.
We note that the auction task could be implemented either by a multiplexer protocol (based on oblivious choice), or by a protocol based on additions and multiplications. For ABY and FRESCO/SPDZ we tried both variants and chose to report the results of the better performing variant: In ABY we implemented the multiplexer protocol, and in FRESCO/SPDZ we implemented the protocol based on additions and multiplications.
Table 2.3 reports the emulated run times of all three systems for each of the computation tasks. The results are reported for input sizes of up to 10^6 items. Tables 2.4,2.5 and 2.6, report the distribution of the running time between the different cryptographic operations that are used in each of the protocols.

Task	Input size	ABY (ms)	SPDZ/FRESCO (ms)	Sharemind (ms)
Auction	1	0	0	0
	10^2	194	4059	7
	10^4	1479	57122	90
	10^6	51681	19370383	8347
Filtered average	1	8	25	1
	10^2	208	413	1
	10^4	773	1224	3
	10^6	51681	15500	281

Table 2.3: Comparison of emulated task running times

2.4 SEEED Queries on Encrypted Data

2.4.1 SEEED Description

SEEED is a database that allows running SQL statements over encrypted data that is outsourced to the cloud, and achieve this functionality without intermediate decryption [12]. Data ownership is maintained by ensuring that only the client is able to access unencrypted data, and primary keys stay with the data owner.
SEEED is not a system for generic secure multi-party computation, like the other systems that we examined, but rather a system for the specific task of working on outsourced encrypted data. It is unsuitable for general computation but performs much better than generic systems

Task	Input size	Total time (ms)	Operation	Time per operation (ms)
Auction	1	0	CMP (bool)	–
			MUX (bool)	–
	10^2	194	CMP (bool)	68
			MUX (bool)	126
	10^4	1479	CMP (bool)	1003
			MUX (bool)	476
	10^6	51681	CMP (bool)	24323
			MUX (bool)	27358
Filtered average	1	8	ADD (arith)	–
			MUL (arith)	8
	10^2	208	ADD (arith)	198
			MUL (arith)	10
	10^4	773	ADD (arith)	404
			MUL (arith)	369
	10^6	51681	ADD (arith)	6166
			MUL (arith)	12427

Table 2.4: Emulated task running times for ABY

at the task it was designed for. We therefore analyzed and describe the performance of the SEEED system.
SEEED is based on the idea of adjusting the encryption levels with the help of onions of encryption presented by Popa et al. [16]. Different types of encryption mechanisms are used, each having different characteristics that SEEED makes use of:

Randomized encryption (RND) produces different ciphertexts for the same plaintext, and provides the strongest security of the used encryption schemes. (Example: encryption using AES-CBC.)

Deterministic encryption (DET) produces the same ciphertext for the same plaintext, and enables usage of the SQL expression = (equal, not join) and of `GROUP BY`. (Example: encryption using AES-ECB.)

Order preserving encryption (OPE) preserves the plaintext order on ciphertexts, and enables the usage of the SQL expression < and >, `ORDER BY` and `GROUP BY`. (Example: the encryption schemes of Boldyreva et al. [6].)

Homomorphic encryption (HOM) enables addition on ciphertexts, and therefore the SQL expression `SUM`. (Example: the encryption scheme of Paillier [15].)

An onion of encryption is a mechanism to make encrypted data available in a structured way by nesting the ciphertext of the encryption schemes, e.g. encrypting a plaintext with OPE, then with DET and finally with RND. (Namely RND(DET(OPE(plaintext)))). Due to the structure of HOM a separate onion is needed for data aggregation.
The SEEED driver analyzes the operator tree of a given SQL statement, and decrypts the onion by peeling off its layers until reaching the first encryption scheme that supports all SQL expressions specified in the statement. Then the driver rewrites the SQL statement by encrypting the statement values according to the uncovered encryption scheme, and runs the

Task	Input size	Total time (ms)	Operation	Time per operation (ms)
Auction	1	0	ADD	–
			CMP	–
			MUL	–
	10^2	4059	ADD	600
			CMP	2981
			MUL	478
	10^4	57122	ADD	1350
			CMP	53272
			MUL	2500
	10^6	19370383	ADD	10032
			CMP	19328253
			MUL	32098
Filtered average	1	25	ADD	–
			MUL	25
	10^2	413	ADD	352
			MUL	61
	10^4	1224	ADD	774
			MUL	450
	10^6	15500	ADD	5200
			MUL	10300

Table 2.5: Emulated task running times for SPDZ/FRESCO

statement on the encrypted SEEED database. In a final step, the retrieved (encrypted) result set is decrypted and processed on the client side.

2.4.2 The SEEED Hardware Testbed

We executed all experiments on an SAP HANA database (SP05 release) [10] running on an HP Z820 workstation with 128GB RAM und 16 dual cores (Intel Xeon CPU running at 2.60GHz). There was no network access, connections were performed via the loopback interface. Our performance measurement is solely based on the database execution time and hence independent of network performance. Our client is implemented in Java 1.7 as a JDBC driver and running on the 64-bit JVM. The crypto routines are implemented in C++, compiled with GCC 4.3 and accessed via JNI.

2.4.3 SEEED Performance Analysis

Two types of representative queries were considered in the performance analysis of the SEEED database: Equality queries such as

```
SELECT DEALS.DEAL_ID FROM TEST_SCHEMA.DEALS WHERE DEALS.PRODUCT_ID = 1
```

and Greater-Than queries, e.g.

```
SELECT DEALS.DEAL_ID FROM TEST_SCHEMA.DEALS WHERE DEALS.ORDER_QTY > 3.
```

Task	Input size	Total time (ms)	Operation	Time per operation (ms)
Auction	1	0	ADD	–
			CMP	–
			MUL	–
	10^2	7	ADD	0
			CMP	7
			MUL	0
	10^4	90	ADD	0
			CMP	88
			MUL	2
	10^6	8347	ADD	4
			CMP	7761
			MUL	582
Filtered average	1	0	ADD	–
			MUL	0
	10^2	0	ADD	0
			MUL	0
	10^4	3	ADD	0
			MUL	3
	10^6	281	ADD	2
			MUL	279

Table 2.6: Emulated task running times for Sharemind

The analysis was performed for 100, 1,000, 10,000 and 100,000 records as shown in the columns of Tables 2.7, 2.8, 2.9 and 2.10. The measurements are averaged values from multiple runs measured in milliseconds for different number of records. The first column shows the timing results for 100 records, the second column shows the results for 1,000 records and so on. The size of the result set of the Equal query was around 10% of the database records, and the results set of the Greater-Than query was approximately 60% of the database size.
In the first query run – see Tables 2.7, 2.9 – the encryption of the requested database records had to be adjusted for the query; e.g. the encryption layers RND (randomized) and DET (deterministic) had to be removed for the Greater-Than queries which require OPE (order-preserving encryption). Therefore, the SEEED interpreter, database updater and encrypter needed more time in the first run than in the second run – see Tables 2.8, 2.10. The last row shows how the workload was shared between the server and the client, e.g. in Table 2.7 for 100,000 number of records the server used 56% of the combined processing time (client and server) and the client the remaining 44%. It has to be noted that the client runtime heavily depends on the size of the result set, since some processing (especially decryption) can only be performed on the client.
In the following, a short description of all benchmarked components is given. For a more detailed description of the used algorithms, the used database scheme as well as a specification of the architecture we refer to Deliverables D22.2 and D22.1.

- SEEED total: The time consumed for complete query execution; i.e. this runtime is composed of the runtime of *SEEED interpreter*, and of the runtime of *SEEED plain query*.

- SEEED interpreter: This component analyzes the plain SQL query, transforms it to its encrypted version and updates the meta data; i.e. this runtime is composed of the SQL query analysis, the runtime of *SEEED encrypter* in addition to *SEEED dbstate updater*.
- SEEED encrypter: Encryption of all values that have occurred in plaintext in the initial query.
- SEEED dbstate updater: Updates of the metadata regarding the database structure, e.g. after "peeling off" one onion layer, the algorithm used for the newly extracted layer is stored for future SQL queries.
- SEEED plain query: Execution time of an unencrypted query utilizing the underlying database engine (e.g. MySQL, SAP Hana). Note, that after transforming the plain query to its encrypted form, this database engine is used.
- SEEED result set: The decryption time of the retrieved result set consisting of encrypted values.

number of records	100	1,000	10,000	100,000
SEEED total	1,892	21,837	169,154	2,740,506
SEEED interpreter	1,849	21,790	169,094	2,740,329
SEEED encrypter	146	213	186	185
SEEED dbstate updater	1,702	21,576	168,907	2,740,143
SEEED result set	1,819	17,876	181,676	1,780,472
plain query	42.97	47.48	60.57	176.72
Server	1,260	16,004	149,027	2,534,478
Client	2,451	23,709	201,803	1,986,500
Server/Client	33.95%	40.30%	42.48%	56.06%

Table 2.7: Greater-Than query (OPE), First Run (times in milliseconds)

number of records	100	1,000	10,000	100,000
SEEED total	66.95	78.34	76.00	142.91
SEEED interpreter	33.94	30.66	30.71	32.02
SEEED encrypter	33.31	30.11	30.11	31.46
SEEED dbstate updater	0.01	0.01	0.01	0.01
SEEED result set	1,943	17,651	176,989	1,838,091
plain query	32.88	47.57	45.18	110.75
Server	33.54	49.60	49.98	110.75
Client	1,976	17,679	177,015	1,838,123
Server/Client	1.67%	0.28%	0.03%	0.01%

Table 2.8: Greater-Than query (OPE), Second Run (times in milliseconds)

number of records	100	1,000	10,000	100,000
SEEED total	732	4,336	39,188	574,603
SEEED interpreter	699	4,291	39,155	574,548
SEEED encrypter	103	99	131	153
SEEED dbstate updater	594	4,190	39,023	574,393
SEEED result set	414	3,226	33,713	303,903
plain query	33.71	45.14	32.38	55.30
Client	675	3,819	36,551	340,108
Server	471	3,743	36,350	538,397
Server/Client	41.11%	49.50%	49.86%	61.29%

Table 2.9: Equal query (DET), First Run (times in milliseconds)

number of records	100	1,000	10,000	100,000
SEEED total	75.65	49.58	42.73	57.73
SEEED interpreter	31.48	31.54	30.17	33.81
SEEED encrypter	30.85	30.95	29.60	33.19
SEEED dbstate updater	0.01	0.01	0.01	0.01
SEEED result set	371	2,813	29,133	304,506
plain query	44.04	17.92	12.48	23.81
Server	44.18	18.57	16.83	36.31
Client	403	2,844	29,159	304,527
Server/Client	9.87%	0.65%	0.06%	0.01%

Table 2.10: Equal query (DET), Second Run (times in milliseconds)

2.5 Conclusions

2.5.1 The Results

When examining the results of the experiments, it is important to recall that the different systems work in different settings and guarantee different levels of security, as was described in Section 2.2.
Following are conclusions from the results of the experiments:

- In almost all experiments, FRESCO/SPDZ has a lower performance than ABY, which has a lower performance than Sharemind .

  FRESCO/SPDZ typically has a performance that is slower by orders of magnitude than the performance of the other systems. This result is explained by the fact that FRESCO/SPDZ is the only system which provides security against malicious (active) adversaries. As was described earlier, guaranteeing security against malicious adversaries comes at a performance cost. This is apparent in the performance results.

- The performance of the ABY system is relatively stronger in computing equality and comparison operations. This result is explained by the fact that ABY combines computation of Boolean and arithmetic circuits, whereas the other two systems work on arithmetic circuits which can only implement arithmetic operations in a field.

  Arithmetic circuits excel at computing addition and multiplication operations (which are each implemented using a single gate). However, they perform less well in computing operations which depend on the bit-wise representation of values, such as comparisons and equality checks. The ABY system is designed to easily translate data between the two representations and therefore use the best implementation of the two worlds.

- The MUL operation, consumes the bulk of the run time on ABY. This is not surprising, since multiplication is relatively inefficient on Boolean circuits. Similarly, the CMP (comparison) operation takes the bulk of the runtime on FRESCO/SPDZ and Sharemind . This is also not surprising, since this operation is hard to implement on arithmetic circuits.

2.5.2 Recommendations

- It is evident from the results that the performance of protocols that are secure against active adversaries (exemplified by the SPDZ/FRESCO protocol) is much slower than that of protocols that are only secure against passive. On the other hand, security against malicious adversaries is a much more realistic security model. Therefore there is a need for new and improved protocols with security against malicious adversaries.

- Even the performance of protocols secure against semi-honest adversaries only, should be improved. This is particularly true for the case of two-party protocols, since Sharemind , which had the best performance, only works in a setting with more than two parties.

- The performance of a specific protocol, such as SEEED, can greatly exceed that of generic protocols that are applied to the same task. Therefore efforts should be invested in designing specific protocols is cases where the following two conditions are met:

  – The task is of high importance.

  – The problem does not have an efficient representation in a format that is suitable for generic protocols for secure computation (i.e., as a Boolean or an arithmetic circuit).

  The outsourced encrypted database application obviously satisfies these two requirements, due to its importance and the large size of the database.

  Another specific task for which these conditions are met is private set intersection (PSI), where two parties with private input sets wish to compute the intersection of their sets. This problem is highly important for applications of joint research or information sharing. It also does not have an efficient representation as a circuit (all known circuit representations have a size of n^2 or $n \log n$ gates for a problem with n inputs). The PSI task is therefore a prime candidate for efficient specific secure computation protocols.

List of Abbreviations

2PC	Two party computation
ABY	Arithmetic-Boolean-Yao
AES	Advanced encryption standard
BMR	the Beaver-Micali-Rogaway protocol
DH	Diffie-Hellman
ECC	Elliptic curve cryptography
FHE	Fully homomorphic encryption
GC	Garbled circuit
GMW	the Goldreich-Micali-Wigderson protocol
GRR	Garbled row reduction
MPC	Multi-party computation
OT	Oblivious transfer
SCS	Sort-compare-shuffle
SPDZ	the Damgard-Pastro-Smart-Zakarias protocol
PRF	Pseudo random function
PSI	Private set intersection
ZK	Zero knowledge

Bibliography

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

[2] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Kenneth G. Paterson, editor, *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, volume 6632 of *Lecture Notes in Computer Science*, pages 169–188. Springer, 2011.

[3] Dan Bogdanov. *Sharemind: programmable secure computations with practical applications.* PhD thesis, University of Tartu, 2013.

[4] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A Framework for Fast Privacy-Preserving Computations. In Sushil Jajodia and Javier Lopez, editors, *Proceedings of the 13th European Symposium on Research in Computer Security - ESORICS'08*, volume 5283 of *Lecture Notes in Computer Science*, pages 192–206. Springer Berlin / Heidelberg, 2008.

[5] Dan Bogdanov, Margus Niitsoo, Tomas Toft, and Jan Willemson. High-performance secure multi-party computation for data mining applications. *International Journal of Information Security*, 11(6):403–418, 2012.

[6] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'Neill. Order-preserving symmetric encryption. In *Proceedings of the 28th International Conference on Advances in Cryptology*, EUROCRYPT, 2009.

[7] 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, *Computer Security - ESORICS 2013 - 18th European Symposium on Research in Computer Security, Egham, UK, September 9-13, 2013. Proceedings*, volume 8134 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2013.

[8] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer, 2012.

[9] Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY - A framework for efficient mixed-protocol secure two-party computation. In *22nd Annual Network and Distributed*

System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2014, 2015.

[10] Franz Färber, Norman May, Wolfgang Lehner, Philipp Große, Ingo Müller, Hannes Rauhe, and Jonathan Dees. The sap hana database–an architecture overview. *IEEE Data Eng. Bull.*, 35(1):28–33, 2012.

[11] Rosario Gennaro, Michael O. Rabin, and Tal Rabin. Simplified vss and fact-track multiparty computations with applications to threshold cryptography. In Brian A. Coan and Yehuda Afek, editors, *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing, PODC '98, Puerto Vallarta, Mexico, June 28 - July 2, 1998*, pages 101–111. ACM, 1998.

[12] Patrick Grofig, Isabelle Hang, Martin Härterich, Florian Kerschbaum, Mathias Kohler, Andreas Schaad, Axel Schröpfer, and Walter Tighzert. Privacy by encrypted databases. In *Privacy Technologies and Policy - Second Annual Privacy Forum, APF 2014, Athens, Greece, May 20-21, 2014. Proceedings*, pages 56–69, 2014.

[13] Jonathan Katz and Lior Malka. Constant-round private function evaluation with linear complexity. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology - ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, volume 7073 of *Lecture Notes in Computer Science*, pages 556–571. Springer, 2011.

[14] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*, pages 681–700. Springer, 2012.

[15] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of the 18th International Conference on Advances in Cryptology*, EUROCRYPT, 1999.

[16] Raluca Ada Popa, Catherine Redfield, Nickolai Zeldovich, and Hari Balakrishnan. Cryptdb: protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 85–100. ACM, 2011.

[17] Pille Pullonen. Actively Secure Two-Party Computation: Efficient Beaver Triple Generation. Master's thesis, Institute of Computer Science, University of Tartu, 2013.

[18] Pille Pullonen, Dan Bogdanov, and Thomas Schneider. The design and implementation of a two-party protocol suite for Sharemind 3. Technical Report T-4-17, Cybernetica, http://research.cyber.ee/, 2012.