DISSERTATIONES
MATHEMATICAE
UNIVERSITATIS
TARTUENSIS
104

RIIVO TALVISTE

Applying Secure Multi-party Computation in Practice

DISSERTATIONES MATHEMATICAE UNIVERSITATIS TARTUENSIS
104

DISSERTATIONES MATHEMATICAE UNIVERSITATIS TARTUENSIS
104

RIIVO TALVISTE

Applying Secure Multi-party Computation in Practice

UNIVERSITY OF TARTU
Press

Institute of Computer Science, Faculty of Mathematics and Computer Science, University of Tartu, Estonia

Dissertation is accepted for the commencement of the degree of Doctor of Philosophy (PhD) on February 3, 2016 by the Council of the Institute of Computer Science, University of Tartu.

Supervisors:

Dr. Tech.	Sven Laur University of Tartu Tartu, Estonia
Ph.D	Dan Bogdanov Cybernetica AS Tartu, Estonia

Opponents:

Prof. Ph.D	Kurt Rohloff New Jersey Institute of Technology Newark, NJ, United States of America
Prof. Dr.	Stefan Katzenbeisser Technische Universität Darmstadt Darmstadt, Germany

The public defense will take place on March 14, 2016 at 16:15 in J. Liivi 2–405.

The publication of this dissertation was financed by Institute of Computer Science, University of Tartu.

ISSN 1024-4212
ISBN 978-9949-77-047-2 (print)
ISBN 978-9949-77-048-9 (pdf)

University of Tartu Press
http://www.tyk.ee

Contents

ABSTRACT

The only way one can benefit from stored data is by using it. This is especially true if data from several sources are combined. For example, by combining data from several of its institutions, a state can discover trends or pin-point problematic issues. However, this is often forbidden due to privacy concerns, as the combined data set becomes an attractive target for both insider and outsider attacks.

Secure multi-party computation is a technology that allows data to be processed so that the computation servers see no actual data values. With the first practical implementations emerging in the 2000s, the technology is now mature enough to be used for privacy-preserving data analysis on real data.

This thesis looks at the technical and organisational challenges that arise from developing secure multi-party computation from a lab prototype to a real-world system. First, we give a brief overview of the two secure multi-party applications that were first used on real-data: the Danish sugar beet auction and the ITL financial benchmarking application in Estonia. We address several shortcomings of these applications. Among others, we concentrate on challenges specific to web-based data gathering and result sharing for such applications, and integrating them with existing information systems.

Our main achievement is the world's first large-scale registry-based statistical study on linked databases using secure multi-party computation technology. We discuss the technical and legal issues that rise in such deployments. Finally, we propose to deploy secure multi-party technology as a service on a federated database infrastructure. As an example, we describe a deployment for the Estonian governmental data exchange layer X-Road. This makes similar registry-based privacy-preserving studies more affordable and transparent in the future.

CHAPTER 1

INTRODUCTION

1.1 Data privacy and secure computation

Already in ancient Greece, the Spartans were worried about the privacy of their messages during military conflicts and used a transposition cipher (*scytale*) to render their messages illegible to others. By now, the need to protect the privacy of one's data has transferred from the military to the civil sphere.

With the advent of *big data*, more and more organisations do not have in-house resources to store and process the collected data. Storing data in the cloud is seen as a valid solution, but in many cases the data itself is confidential or sensitive. Encrypting data (on the client side) to store it in the cloud solves the problem of protecting data at rest, but also renders it unusable for further processing. Downloading a copy of the data and decrypting it for processing is not a viable option for most organisations, taking into account today's data volumes. With the growth of computational power and network throughput, privacy-preserving computation is an option for securely outsourcing computations so that no parties learn individual input values.

Secure multi-party computation (SMC) is a distributed computation model, where several parties collaboratively compute a common function on each other's inputs, while keeping their own inputs private and only learning the computation result. SMC protocols are privacy-preserving to the extent that they leak nothing about the input values other than what can be deduced from the output or other explicitly published values during the protocol run. Thus, SMC does not hide input values that can be directly inferred from the computation result.

Even though secure multi-party computation has been around for more than 30 years [129, 43, 67], its adoption in practice has been scarce. Although some of the SMC related research papers contain benchmarking results, these are mostly academic prototypes tailored for a specific purpose. The first truly practical large-scale SMC application was the Danish sugar beet auction in 2008 [32]. Since then,

SMC has been used in practical applications, where the computation is performed over the public internet [31], and even provided as part of a service (Energiauktion.dk[1], Dyadic[2], Sepior[3]).

As the SMC technology becomes more practical and economically interesting, new challenges in its deployment arise. The security requirements and the distributed nature of SMC make it more difficult to build reliable and robust systems. We cannot underestimate the fact that the concept of privacy-preserving computations is hard to grasp for the industry. Moreover, in many cases there are no regulations nor legal cases that cover how such technology should be handled in conjunction with data protection laws.

1.2 Claims of this thesis

The author has developed, deployed and evaluated SMC applications for both data collection and for combining many data sets for statistical analysis. The main contribution of this monograph is an end-to-end process for deploying and managing the life cycle of a secure multi-party computation application. To accomplish that, the author had to address three basic shortcomings from the state of the art in 2011 when starting his Ph.D studies. First, we show how to integrate SMC applications with existing (including legacy) systems and demonstrate that by developing generic re-usable software libraries, similar deployments are repeatable with less effort. Second, we describe how we implemented privacy-preserving database linking and oblivious sorting that are essential operations in large-scale statistical studies. Third, we discuss organisational steps necessary to satisfy the stakeholders in an SMC deployment.

We validate the described process on the first large-scale registry-based privacy-preserving statistical study using secure multi-party computation technology. In this study, we show that following the described process, it is possible to obtain a positive resolution from the Estonian Data Protection Agency. Moreover, we demonstrate that it is feasible to process practical-size data sets using SMC. Nevertheless, we also bring out several shortcomings in deployment and monitoring that surfaced during the study.

1.3 Outline and author's contributions

Chapter 2 introduces secure multi-party computation and gives a brief overview of different types of SMC – garbled circuits, fully homomorphic encryption and

[1] EnergiAuktion.dk – `https://energiauktion.dk/`
[2] Dyadic – `https://www.dyadicsec.com/`
[3] Sepior – `https://sepior.com/`

linear secret sharing. All of the practical solutions proposed in this thesis are based on the SHAREMIND SMC framework that we also briefly describe in this chapter. Finally, we give some considerations on integrating SMC technology into modern information systems.

Chapter 3 gives an overview of two real-world SMC applications – the Danish sugar beet auction and the financial benchmarking application. The latter is based on the Master's thesis of the author that was later refined into and the following publication [31]:

- Bogdanov, D., Talviste, R., Willemson, J.: Deploying secure multi-party computation for financial data analysis (short paper). In: Proceedings of the 16th International Conference on Financial Cryptography and Data Security. FC'12. Lecture Notes in Computer Science, vol. 7397, pp. 57–64. Springer (2012)

We bring out some of the shortcomings of the described applications that we address in the following chapters.

Chapter 4 describes challenges and possible solutions for using web-based applications as input or result parties for the secure multi-party computation. Many proposed ideas were initially implemented by the author of the thesis in a JavaScript library developed for the financial benchmarking application [31]. The second part of the chapter gives an overview how the web components (JavaScript library and server-side components) have evolved during subsequent prototype applications. The outlined applications were not developed by the author of this thesis. However, these applications reused and refined the aforementioned JavaScript library and the author has been in a consulting role in many cases.

Chapter 5 describes an oblivious AES block cipher implementation and how it can be used to provide a privacy-preserving database join operation. The chapter is based on the author's previously published work [96] and its extended version [97]:

- Laur, S., Talviste, R., Willemson, J.: From Oblivious AES to Efficient and Secure Database Join in the Multiparty Setting. In: Applied Cryptography and Network Security. ACNS'13, Lecture Notes in Computer Science, vol. 7954, pp. 84–101. Springer (2013)

The non-vectorised version of AES S-box implementation based on oblivious selection was designed and prototyped by Jan Willemson. All other versions of oblivious S-box evaluation as well as vectorised versions of the cipher were implemented and evaluated by the author of this thesis. Similarly, the size unification

protocol proposed for hiding the number of colliding values in a database join was designed by Sven Laur. All other proposed methods were designed and evaluated by the author of this thesis.

Chapter 6 gives an overview of oblivious implementations of sorting algorithms and compares their performance. The chapter is based on the author's previously published work [27, 28]:

- Bogdanov, D., Laur, S., Talviste, R.: Oblivious Sorting of Secret-Shared Data. Tech. Rep. T-4-19, Cybernetica, `http://cyber.ee/en/research/publications/`. (2013)
- Bogdanov, D., Laur, S., Talviste, R.: A Practical Analysis of Oblivious Sorting Algorithms for Secure Multi-party Computation. In: Proceedings of the 19th Nordic Conference on Secure IT Systems. NordSec'14, Lecture Notes in Computer Science, vol. 8788, pp. 59–74. Springer (2014)

Chapter 7 gives a lifecycle overview of the PRIST project, where SMC technology was used to perform a registry-based statistical study on linked government databases. The privacy-preserving statistical algorithms and methods used in this study were designed by Liina Kamm and are detailed in her thesis [88]. The author of this thesis collaborated with the project team during the analysis phase and was in the role of the delivery engineer during the rest of the project. Thus, this work gives an overview of outcomes and shortcomings of the project from a more technical perspective. An overview of the PRIST project will also be published in the *Proceedings on Privacy Enhancing Technologies* [22]:

- Bogdanov, D., Kamm, L., Kubo, B., Rebane, R., Sokk, V., Talviste, R.: Students and Taxes: a Privacy-Preserving Social Study Using Secure Computation. Proceedings on Privacy Enhancing Technologies (PoPETs) 2016(3) (2016), (to appear)

Chapter 8 proposes a solution how SHAREMIND SMC technology can be incorporated with the Estonian X-Road data exchange layer that connects the government databases. We show how providing secure multi-party computation as a service makes studies based on linked databases like PRIST more convenient and replicable. The architecture proposed in this chapter is based on an idea initially proposed in a previously published paper of the thesis author [23]:

- Bogdanov, D., Kamm, L., Laur, S., Pruulmann-Vengerfeldt, P., Talviste, R., Willemson, J.: Privacy-preserving statistical data analysis on federated databases. In: Proceedings of the Annual Privacy Forum. APF'14. Lecture Notes in Computer Science, vol. 8450, pp. 30–55. Springer (2014)

The first authorship of this published paper is shared with Liina Kamm who is the first author of Sections 5–7. The author of this thesis is the first author of Sections 2–4, where the idea of combining SMC and the Estonian X-Road is proposed.

CHAPTER 2

INFORMATION SYSTEMS AND SMC

2.1 Data security in modern information systems

Modern information systems are built on a client-server model where a single server (or a cluster of servers) provides a service for a large amount of clients. As shown in Figure 2.1, an end user uses a client application to send a query to the server. The server contains the business logic of the service and uses it to process the client query, accessing one or more databases if needed. The result of this process is sent back to the client as a response to the query. The client application here can be anything from mobile or web-based user interfaces to desktop applications and even other services.

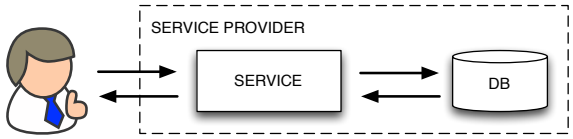


Figure 2.1: Client-server model with database backend.

All of the components shown on Figure 2.1 can be considered data access points for various external and internal parties with respect to the service provider. For external clients, the access rights should be validated on the perimeter of the service provider or by the service itself based on the data access policy and identity of the authenticated client. These client authentication and authorisation means are not covered by this work, as they should be addressed by following current best practices. Instead, we focus on the access to the service and database by internal parties (employees, support staff) as well as unauthorised external parties (hackers).

The industry standard at the moment is to encrypt data in the database at rest, if at all. Unfortunately, encrypting the data with standard tools (e.g. AES block

cipher in GCM mode) renders it unusable as it has to be decrypted whenever it is used in computations. If these computations are performed by the business logic deployed in the service, then the decryption key must also reside in the service. Hence, such encryption only helps in case of a very contained security breach, e.g. if an attacker is able to break in only to the database server and not the service.

A step forward is to use an encryption scheme that preserves some functionality of the encrypted data, e.g. searchable symmetric encryption [119, 66] and order preserving encryption [4, 33]. Such encryption schemes allow the business logic to carry out some specific operations (e.g. range queries) on encrypted data without decrypting the database first. Solutions like CryptDB [113] and SAP SEEED [69] help against corrupted database administrators.

Finally, secure computing makes it possible to obliviously hide the whole business logic. Such a use case, where the service provider never sees the data, is especially compelling for cloud deployments. By secure computing, we mean computations on encrypted or otherwise illegible (e.g. garbled or secret-shared) data. In this work, we only consider secure computing made possible by cryptographic means and do not cover hardware-based trusted execution environments (e.g. Intel SGX, Secure Enclave in iOS devices) or heuristic methods that rely on adding noise.

2.2 Secure computation

In the following, we briefly introduce three approaches to secure multi-party computation: Yao's garbled circuits, fully homomorphic encryption and lastly linear secret sharing, that this work focuses on. Secure multi-party computation (SMC) formalises a scenario, where n parties collaboratively compute a function

$$f(x_1, x_2, \ldots, x_n) = (y_1, y_2, \ldots, y_n) \ ,$$

such that each party $\mathcal{P}_1, \ldots, \mathcal{P}_n$ knows its corresponding input x_i and receives it's result y_i. Parties learn nothing about other parties' inputs nor their part of the result.

2.2.1 Yao's garbled circuits

The garbled circuits (GC) approach, introduced by Yao [129, 130] and later detailed my many others [67, 12, 104], is a cryptographic technique to achieve secure function evaluation. In the approach, there are two parties $\mathcal{P}_1$ and $\mathcal{P}_2$ and the functionality f they want to compute is expressed as a Boolean circuit, i.e. an electronic circuit containing AND, OR and other types of gates. Parties' inputs x_1 and x_2 are represented by bit strings.

In GC, party $\mathcal{P}_1$ encrypts ("garbles") the original circuit to protect the privacy of it's own input by doing the following. For each wire in the circuit, two random strings are generated, one of which represents the bit value 1 and the other 0. Then, for each gate g with input values b_1 and b_2 the corresponding output string is encrypted with a composite key (b_1, b_2). Only by knowing both parts of the key, i.e. both input values, it is possible to decrypt the output value of the gate. Hence, for each gate, there are four different ciphertexts and each combination of input can correctly decrypt only one of them. To hide the operation of the gate, this ciphertext table is also randomly permuted. Party $\mathcal{P}_1$ encodes its input to the input wires in a similar manner and then sends this garbled circuit to party $\mathcal{P}_2$. Party $\mathcal{P}_2$ uses *oblivious transfer* (OT) to receive the corresponding random strings for its input bits from party $\mathcal{P}_1$. After evaluating the circuit, party $\mathcal{P}_2$ learns the output of the computation and, if needed by the application, forwards it to party $\mathcal{P}_1$.

This description suggests that the computations are CPU-bound with only a few communication rounds. The communication load is also asymmetrical, as $\mathcal{P}_2$ starts its computations only after $\mathcal{P}_1$ has finished. Nevertheless, there are designs, where the garbled circuit is streamed to $\mathcal{P}_2$ to reduce this asymmetry and the memory footprint at $\mathcal{P}_1$ [102].

A more in-depth descriptions of the garbled circuit model along with security proofs are given in [100, 14]. Some practical implementations based on garbled circuits include Fairplay [103, 15], TASTY [75], FastGC [77] and ABY [52].

2.2.2 Fully homomorphic encryption

Fully homomorphic encryption (FHE) is a client-server model, where the server performs computations on encrypted data without access to the private key held by the client. FHE takes advantage of the homomorphic properties of the underlying encryption scheme. Homomorphic public key cryptosystems allow modifying the ciphertext in order to change the value under encryption in a desired way without decrypting the value in the process. For example, Paillier, lifted ElGamal and Damgård-Jurik cryptosystems are additively homomorphic. Thus, it is possible to modify ciphertexts so that the values under encryption are either summed together or multiplied by a public constant. However, for multiplying values under encryption, one needs to interact with the party having access to the private key. Similarly, there are multiplicatively homomorphic encryption schemes where values under encryption can be multiplied but not summed.

The first cryptosystem that allowed both additions and multiplications with constant-size ciphertext was introduced by Boneh, Goh and Nissim [34] and was based on pairings on elliptic curves. However, while the BGN cryptosystem allowed an unlimited number of homomorphic addition operations to be performed,

it only supported one multiplication operation. Such a cryptosystem is called *somewhat homomorphic*.

The first implementable fully homomorphic encryption scheme that was able to both add and multiply values under encryption was created by Gentry in 2009 [59, 60]. Similarly to the BGN system, Gentry's somewhat homomorphic scheme was only able to evaluate low-degree polynomials on encrypted data before the noise under encryption grew too large and made it impossible to correctly decrypt the result. Gentry proposed to alleviate this problem by *bootstrapping* – homomorphically re-encrypting the ciphertext under encryption yielding a fresh noise free ciphertext. FHE has received a lot of attention and alternative constructions from the research community [37, 62, 61, 63, 65, 118]. Some of the practical implementations include HElib[1], hcrypt[2], Stanford's FHE library[3] and FHEW[4].

The main problems of FHE are that both the homomorphic operations as well as bootstrapping are costly operations. Moreover, the ciphertext size blow-up is significant (10^3–10^9) and makes the scheme impractical for large applications. Nevertheless, FHE is used as a sub-protocol for non time-critical pre-computations in other secure computation systems, e.g. SPDZ [51]. Moreover, Archer and Rohloff have demonstrated its use in filtering spam and securing VoIP calls [6].

2.2.3 Linear secret sharing

In cryptography, *secret sharing* is a concept where a secret value x is split into n values called "shares" $x_1, \ldots, x_n$ such that each share is indistinguishable from a random value. The shares are then distributed between independent parties such that party $\mathcal{P}_i$ gets share x_i. The idea is that for a given *threshold* t ($t \leq n$), having less than t shares gives no information about the initial secret value. The concept of secret sharing was independently invented by Shamir [116] and Blakley [17].

In [116], Shamir proposes a t-out-of-n secret sharing scheme based on the fact that t points uniquely determine a $(t-1)$-degree polynomial. To share a secret value x, a random $(t-1)$-degree polynomial f is chosen such that $f(0) = x$. The n shares are then computed as

$$x_1 = f(1), x_2 = f(2), \ldots, x_n = f(n) \ .$$

One can use Lagrange interpolation with any set of t or more of these shares to reconstruct the polynomial and thus learn the secret value x. This forms the theoretical basis of many secure multi-party computation implementations.

[1] HElib – `https://github.com/shaih/HElib`

[2] hcrypt project – `https://github.com/hcrypt-project`

[3] Stanford's FHE library – `http://cs.stanford.edu/people/dwu4/fhe.html`

[4] FHEW – `https://github.com/lducas/FHEW`

The rest of this thesis uses SMC mechanisms based on more simple n-out-of-n *additive* and *bitwise* secret sharing schemes. An additive secret sharing scheme works in a ring $\mathbb{Z}_N$, where a secret value $x \in \mathbb{Z}_N$ is secret shared among n parties by first choosing $n-1$ random numbers modulo N and the last share is found by subtracting the $n-1$ random values from the initial secret (modulo N):

$$\begin{aligned} x_1 &\leftarrow \mathbb{Z}_N \\ &\vdots \\ x_{n-1} &\leftarrow \mathbb{Z}_N \\ x_n &\leftarrow x - x_1 - \ldots - x_{n-1} \pmod N \ . \end{aligned}$$

It is easy to see how adding the shares up modulo N yields the initial secret x.

In bitwise sharing scheme a secret value $x \in \mathbb{Z}_{2^k}$ is secret shared similarly. The first $n-1$ values are chosen at random from $\mathbb{Z}_{2^k}$, where $k \in \mathbb{N}$, and the last share is obtained by applying exclusive or (XOR) operation on them and the original secret value:

$$x_n \leftarrow x \oplus x_1 \oplus \ldots \oplus x_{n-1} \ .$$

As each bit is shared independently, bitwise secret sharing schemes are useful for bit-level operations.

In the following, we denote secret-shared values in double brackets, so $[\![x]\!]$ stands for a tuple $(x_1, \ldots, x_n)$, where party $\mathcal{P}_i$ holds a share x_i. Secret-shared vectors of k elements, written in bold face, are secret-shared element-wise:

$$[\![\boldsymbol{x}]\!] = ([\![x_1]\!], \ldots, [\![x_k]\!]) \ .$$

As the simplest example, consider adding together two secret values x and y secret-shared additively between n parties. As additive secret sharing is additively homomorphic, adding is a local operation that can be done independently by each party using its shares x_i and y_i:

$$\begin{aligned} [\![x]\!] + [\![y]\!] &= (x_1, \ldots, x_n) + (y_1, \ldots, y_n) \\ &= x_1, \ldots, x_n + y_1, \ldots, y_n \\ &= (x_1 + y_1) + \ldots + (x_n + y_n) \ . \end{aligned}$$

For multiplying additively secret-shared values, there are two options. First, it is possible to convert the shared values to use *replicated secret sharing*, where each party holds a different $(n-1)$-element subset of all of the shares. For example, in 3-party case, to compute the product $[\![x]\!] \cdot [\![y]\!]$, each party first sends their shares of x and y to the next party (in a circle). By knowing x_i, y_i, x_{i-1} and y_{i-1},

each party independently computes the sum of three terms $x_i y_i + x_i y_{i-1} + x_{i-1} y_i$ out of the total of nine terms:

$$\begin{aligned}\llbracket x \rrbracket \cdot \llbracket y \rrbracket &= (x_1 + x_2 + x_3)(y_1 + y_2 + y_3) \\ &= (x_1 y_1 + x_1 y_3 + x_3 y_1) \\ &+ (x_1 y_2 + x_2 y_1 + x_2 y_2) \\ &+ (x_2 y_3 + x_3 y_2 + x_3 y_3) \ .\end{aligned}$$

This idea is used, for example, by SHAREMIND in its 3-party implementation based on additive secret sharing scheme.

An alternative option is to use Beaver triples [13, 45]. Beaver triples are random triples $(\llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket c \rrbracket)$, such that $c = a \cdot b$. With the help of such triple, $\llbracket x \rrbracket \cdot \llbracket y \rrbracket$ can be computed by each party locally computing $\llbracket e \rrbracket = \llbracket x \rrbracket - \llbracket a \rrbracket$ and $\llbracket d \rrbracket = \llbracket y \rrbracket - \llbracket b \rrbracket$. The values e and d are then opened (declassified) to each party and used by each party to compute their share of the result:

$$\llbracket x \rrbracket \cdot \llbracket y \rrbracket = \llbracket c \rrbracket + e \cdot \llbracket b \rrbracket + d \cdot \llbracket a \rrbracket + e \cdot d \ .$$

As the triples themselves are independent of x and y, they can be pre-computed in bulk and stored for later use. This idea is, for example, used in the SPDZ [51] implementation.

There are several practical implementations of secure multi-party computation based on linear secret sharing. Examples include VIFF [48], SEPIA [38], SHAREMIND [29, 19], ShareMonad [95], FRESCO [46], SPDZ [51] and ABY [52].

2.2.4 SHAREMIND SMC framework

In this work, we use the SHAREMIND platform [19] for our designs and practical experiments. SHAREMIND is a application and database server developed by Cybernetica, capable of running secure multi-party computation protocols. Applications for SHAREMIND are written in the SECREC programming language that allows to manage public and private data in parallel [26]. For private data types, SECREC uses the concept of *protection domains* that are protocol suites to handle different SMC techniques. For example, 3-party additive secret sharing scheme with semi-honest security setting is one protection domain and 2-party garbled circuits based approach with active adversary would be another one. SECREC runtime guarantees that operations carried out on data belonging to a specific protection domain use corresponding SMC protocols and secret values are moved to the public domain only through the explicit `declassify` operation.

Currently, the most advanced protocol suite in SHAREMIND handles the 3-party additive secret sharing scheme with semi-honest adversary model. The fundamental protocols used in this protection domain are described in [29] and [30] with detailed composability proofs given in [25].

SHAREMIND versions

The first version of SHAREMIND was an academic prototype developed by Bogdanov for his Master's thesis in 2006 [18]. Later, its fundamental protocols with security proofs were published in [29].

SHAREMIND version 2 was the first version to be used in practical applications, e.g. the financial benchmarking application described in Section 3.1.2. Initially, it supported a single data type: 32-bit additively shared unsigned integer. Later, it was extended with support for 32-bit signed integers and also bitwise shared integers. SHAREMIND version 2 also introduced the first version of the SECREC language [84]. However, since it supported a single data type and only additively and bitwise shared data, it did not include the concept of protection domains. SECREC version 1 only separeted computations in the "public" and "private" domains.

SHAREMIND version 3 is a complete re-write of the platform and also introduces new SECREC language with support for different protection domains. In addition to additively and bitwise shared integers of different bit lengths, its 3-party protocol suite for semi-honest adversary model also supports fixed [93] and floating point numbers [89], and conversions between them. SECREC 2 also includes a standard library that collects most-used algorithms in a re-usable package. These include methods for common vector and matrix manipulations, sorting methods as well as statistical analysis algorithms[5].

After bringing over its networking library from RakNet[6] to Boost ASIO[7] for performance reasons, SHAREMIND 3 was renamed to SHAREMIND Academic Server and is used mainly for academic prototyping.

The latest version is SHAREMIND Application Server that shares the same SECREC language and standard library with the Academic Server. Hence, SECREC applications developed for Academic Server run also on Application Server and vice versa. SHAREMIND Application Server uses its own networking library that, like Boost ASIO, uses mutually authenticated TLS tunnels for communication.

2.2.5 Security model for SMC

As this work focuses on using secure multi-party computation in real-world applications, in many cases we consider SMC to be a black box and refer reader to the previous publications for detailed security proofs. Nevertheless, we briefly

[5] SECREC language and standard library reference – `http://sharemind-sdk.github.io/stdlib/reference/index.html`

[6] RakNet - Multiplayer game network engine – `http://www.jenkinssoftware.com/`

[7] Asio C++ Library – `http://think-async.com/`

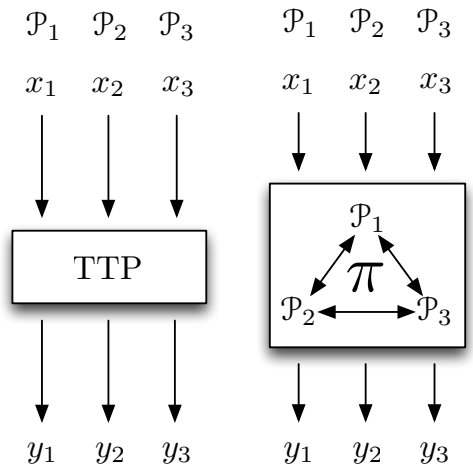
Version	Application language	Year
SHAREMIND 1	C++ library	2006
SHAREMIND 2	SECREC 1	2009
SHAREMIND 3	SECREC 2 + stdlib	2011
Application Server	SECREC 2 + stdlib	2013
Academic Server	SECREC 2 + stdlib	2014

Table 2.1: Overview of SHAREMIND and SECREC versions, including the standard library (stdlib).

describe the security model that SHAREMIND and many other SMC implementations are based on.

First, the SMC protocols in SHAREMIND are *universally composable* [25], meaning that they remain secure even if multiple instances of the same protocol or other protocols are run in parallel or in sequence. Moreover, protocols that use only universally composable sub-protocols are also universally composable themselves [41, 40, 110]. This allows us to combine individual protocols to build algorithms and combine algorithms to build privacy-preserving applications.

The security proofs for individual SMC protocols use the ideal-real world paradigm [39]. In the ideal world (Figure 2.2a), the required functionality f is implemented by an incorruptible trusted third party (TTP) that gathers inputs from the input parties, carries out the computation and publishes the result. Parties only learn what is deducible from the explicit computation result. In contrast, in the real world the required functionality is implemented by a protocol π that includes parties exchanging messages with each other (Figure 2.2b).



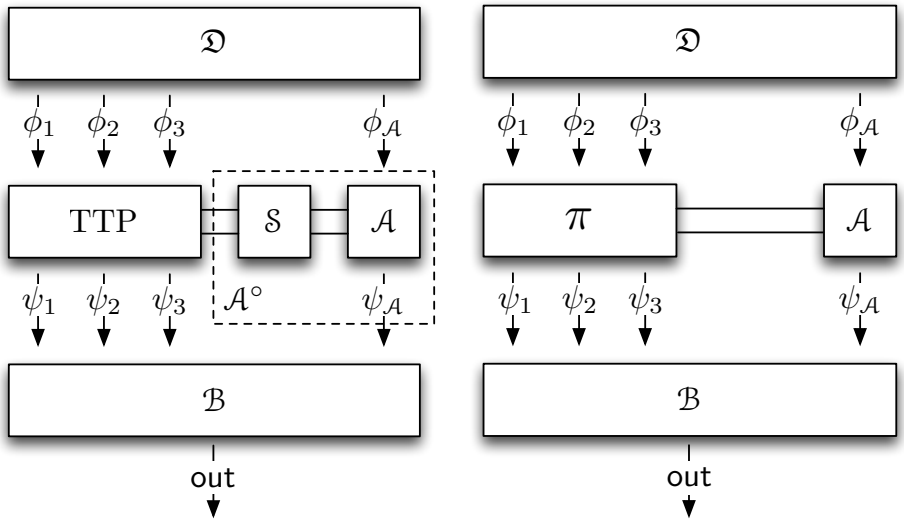
(a) Ideal world (b) Real world

Figure 2.2: The ideal-real world paradigm.

To argue about security of such protocols, we replace inputs of parties with a more abstract input distribution $\mathfrak{D}$ and introduce a distinguisher $\mathcal{B}$ as shown on Figure 2.3. The distinguisher $\mathcal{B}$ must output a single bit based on the output vector $(\psi_1, \ldots, \psi_n, \psi_{\mathcal{A}})$ corresponding to the outputs of computing parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ and the adversary. The aim of the distinguisher $\mathcal{B}$ is to guess whether it is in the ideal or real world. Let $\mathcal{A}$ and $\mathcal{A}^\circ$ denote adversaries in the real and ideal world, respectively. Additionally, let $\Pr[\mathcal{B}_{ideal} = 1]$ denote the probability of distinguisher $\mathcal{B}$ returning 1 in the ideal world and $\Pr[\mathcal{B}_{real} = 1]$ the probability that it outputs 1 in the real world. Then we say that the protocol π is ε-secure if

$$\forall \mathcal{A} \in \mathfrak{A},\ \exists \mathcal{A}^\circ \in \mathfrak{A}^\circ,\ \forall \mathfrak{D},\ \forall \mathcal{B} \in \mathfrak{B}:\ |\Pr[\mathcal{B}_{ideal} = 1] - \Pr[\mathcal{B}_{real} = 1]| < \varepsilon\ .$$

The sets of plausible adversaries $\mathfrak{A}$, $\mathfrak{A}^\circ$ and distinguishers $\mathfrak{B}$ in this logical formula determine the exact nature of security guarantees. Usually one considers only efficient adversaries, which are modelled as non-uniform polynomial-time algorithms. The set of distinguishers is either the set of all computable predicates (statistical security) or efficiently computable predicates (computational security). It is important to note that the adversary $\mathcal{A}^\circ$ cannot depend on the input distribution or the distinguisher. In most security proofs the construction of the adversary $\mathcal{A}^\circ$ is explicitly given through a simulator construction $\mathcal{A}^\circ = \mathcal{S}^{\mathcal{A}}$, where this notation means that the simulator $\mathcal{S}$ can use $\mathcal{A}$ in a black-box manner.



(a) Distinguisher in ideal world

(b) Distinguisher in real world

Figure 2.3: Distinguisher $\mathcal{B}$ has to distinguish between ideal world with simulator $\mathcal{S}$, and real world with protocol π.

Another detail that determines security properties is the ratio of running times of $\mathcal{A}^\circ$ and $\mathcal{A}$. If $\mathcal{A}^\circ$ runs ρ times slower than $\mathcal{A}$, then by participating in the protocol π, the adversary $\mathcal{A}$ may speed up its computations up to ρ times compared to ideal world. As a result, if the simulator $\mathcal{S}$ runs exponentially slower than $\mathcal{A}$, then the gain is so significant that we no longer can guarantee security of other cryptographic primitives (e.g. encryption, hashing). Therefore, it is commonly required that the slow-down factor ρ is polynomial in the running time of $\mathcal{A}$.

The model of tolerated corruption is another important factor required for the completeness of the definition. In this work, we only consider static corruption model, where an adversary chooses parties to corrupt before the computations are executed. Alternatively, the adversary can choose parties to corrupt dynamically based on the information received during the protocol. This is known as adaptive corruption. The corruption can be either passive or active. Passive corruption means that all parties carry out protocol without deviating from the protocol specification, whereas active corruption allows arbitrary deviations.

Finally, we emphasise that the security definition sketched above captures the security of a protocol in stand-alone setting, where parties do not carry out other computations concurrently with the protocol. As such, this definition is useful for assessing security of the entire computation procedure. Usually, complex protocols are composed from smaller sub-protocols. For those protocols, it makes sense to use more strict notions of security, which guarantee that security of a protocol is preserved even if other protocols are executed in parallel. Universal composability [41, 40, 110] is a security notion that provides such level of security. As standard result [40] assures that a secure protocol is also universally composable if the black-box simulator is non-rewinding, we omit the formal definition of universal composability.

If we consider static corruption, then the simulation procedure can be split into two conceptual steps. First, the simulator must somehow provide "correct" inputs to the trusted third party. After that the simulator receives corresponding outputs from the TTP and must create protocol messages to $\mathcal{A}$ that are consistent with these outputs. In particular, if the adversary follows the protocol, then the corrupted parties must obtain outputs as prescribed by the trusted third party. The first part of the simulation is known as *input extraction* and the second part as *output equivocation*.

In security proofs for protocols that compose other sub-protocols, we use the notion of the *hybrid world*. In this model, sub-protocols are replaced by their corresponding ideal functionalities for easier argumentation. In the corresponding security proofs one shows that ideal world and hybrid world are indistinguishable in the sense of SMC. That is, there exists a simulation strategy that allows us to simulate messages of sub-protocols in the hybrid world. Note that simula-

tion of hybrid world is much easier, as parties submit their inputs and get their outputs from TTP to evaluate a sub-protocol. As a result, the simulator directly gets inputs of sub-protocol and can freely prescribe corresponding outputs for the sub-protocol. Hence, input extraction and output equivocation are much simpler.

If the ideal and hybrid world are indistinguishable, then the security follows from the universal composability of sub-protocols. In a nutshell, universal composability assures that real and hybrid world are indistinguishable. This together with indistinguishability of the hybrid and ideal world implies indistinguishability of the ideal and real world, which is sufficient for security.

2.2.6 Roles in SMC deployment

We can assign each participant in secure multi-party computation one or more roles from the following [23]. Input party ($\mathcal{I}$) is a participant who either owns the data or is deletegated by the data owner to perform operations on the data. Usually, an input party can be offline for the rest of the process lifecycle. An input party is the only party that has access to the input data. Computation party ($\mathcal{C}$) invokes SMC protocols on secret-shared data and, in the process, exchanges messages with other $\mathcal{C}$-s. A computation party also manages storage of the secret shared or encrypted data if persistence is used. A result party ($\mathcal{R}$) is a participant who receives computation result from computation parties. In some applications, there is also a separate verifier party ($\mathcal{V}$) that audits if the secret sharing and computation is carried out correctly according to the protocol. Verification can be carried out either during or after the computation based on audit logs [111].

Example: Millionaire's problem

The garbled circuits approach was initially motivated by the so-called millionaires' problem [129] – two millionaires want to know which one of them has more money without revealing their exact wealth to the other.

In this scenario (see Figure 2.4), both of the involved parties have their own secret input (their wealth) so they are both input parties ($\mathcal{I}$). By construction, the amount of computation in GC is asymmetric between the parties. However, it can be argued that both garbling the circuit as well as evaluating it takes computation power so both parties can also be considered as $\mathcal{C}$. Also, by construction, only the evaluator gets the result of the computation and is thus the result party ($\mathcal{R}$). It is up to the specific application if the result should be forwarded to the other party. It is also possible for the computation result to be encrypted with only the first party having the key or secret-shared between several output parties [114].

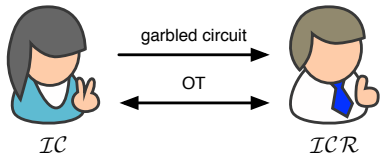


Figure 2.4: Roles in the garbled circuits construction.

Example: Outsourced computations

The killer application for fully homomorphic encryption is a case where computation on sensitive data is outsourced, e.g. to a cloud service. To protect the privacy of the data, it is first encrypted by the input party ($\mathcal{I}$) and then uploaded to a service provider ($\mathcal{C}$). The encrypted result is sent back to the initial party, as it has the corresponding decryption key, hence input party is also in the role of the result party. The interaction is shown on Figure 2.5.

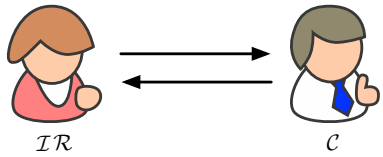


Figure 2.5: Roles in the fully homomorphic encryption construction.

Example: Outsourced services

In practical applications, secret sharing based SMC is most suitable for analysing data gathered from multiple sources (see Chapters 4 and 7). This category includes online questionnaires, auctions, but also combining whole data sets.

Figure 2.6 shows a combination of possible input parties ($\mathcal{I}$): desktop applications, web (including mobile) forms, structured files and data sets. Each data value is independently secret shared and the shares are distributed among computation parties ($\mathcal{C}$). The result party ($\mathcal{R}$) can be a statistician, a researcher or even an automated data publishing program.

2.3 Combining SMC into information systems

2.3.1 Input parties

Based on the way input parties handle their data, we can divide practical SMC applications into two categories. First, a setup where each data owner provides his or her data directly into the system by secret sharing it between the computation

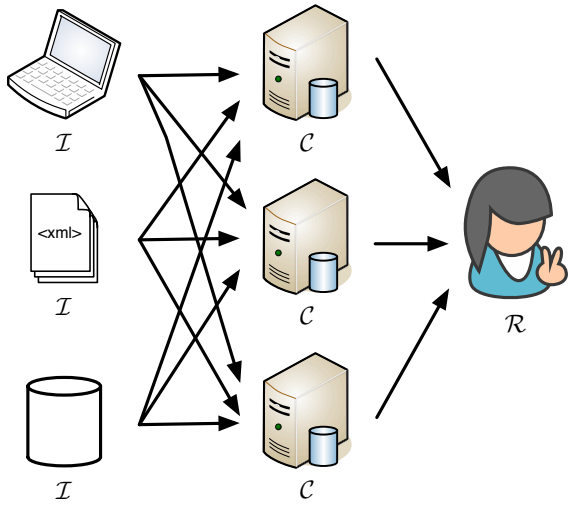


Figure 2.6: Roles in the secret sharing based 3-party SMC construction.

parties. In this case, the input party is usually a person and his or her input is one or a few records. Such examples include online questionnaire systems, auctions, voting, etc. The data is created or input for this specific purpose.

Secondly, we can consider a setup where input data is gathered from existing databases. In this case, the actual data owners are not directly involved in the process and the role of input party is given to whoever manages the database. Usually, in this setup there are only a few input parties but each of them have many records. Examples here include combining private or public sector databases for statistical studies or (financial) benchmark analysis.

In either case, the input party must send the value shares to the computation servers directly so that no single proxy can read those. This is accomplished by secure connections to each of the computation servers or encrypting each share with the public key of the corresponding computation party. Hence, in both cases, the input party must authenticate each computation party separately.

Furthermore, as the secret sharing of input data has to be performed at the control boundary of the input party, special care must be taken when using web-based application for data input. The web server serving the application must not be considered an input party as it is not under the control of the input party. Only the content downloaded to the web browser is allowed to process input data. More details on handling sensitive input data in web applications is given in Chapter 4.

2.3.2 Computation parties

Integrating computation parties into an information system involves finding an independent organisation for each of the computation nodes. For public sector information systems, other public institutions and/or private companies are good candidates. Competing companies are a good option for private sector information systems. For the latter option, it is best if all the computation parties share the incentive to host such a privacy-preserving information system, e.g. they all benefit from the outcome or have a legal obligation to participate.

Each computation party handles its security perimeter and storage of its shares, including backups, independently. In principle, for n-out-of-n secret sharing scheme with passive adversaries, we can differentiate between two attack scenarios.

First, for long-time storage of shares, i.e. when no secure multi-party computation protocols are running, it is enough for one party to remain secure in order to protect the privacy of the shared values. This is true as no combination of $n-1$ shares give any information about the actual secret value. Hence, when a data breach is detected by at least one of the computation parties, a *resharing* protocol must be initiated before the shares at all other computation parties are also compromised. Resharing protocol securely replaces all shares with fresh ones without reconstructing the initial data values or contacting the input parties. After resharing, all computation parties must securely destroy all copies of the old shares as they pose a confidentiality threat [21].

Secondly, a computation node may be compromised during the secure multi-party computations. In a passive security model, this would mean that an attacker has gained access to the computation node and can read its memory and network messages exchanged with other computation parties. At the same time, efficient SMC protocols require honest majority of computation parties to remain secure. Hence, using the 3-party additive secret sharing scheme with SMC protocols that are secure in the semi-honest setting in SHAREMIND, no more than a single computation party may be compromised this way. Upon detecting such an attack, the owner must first abort the computation and regain exclusive control over its computation node to mitigate further leaks. Next, the computation nodes can initiate the resharing protocol to invalidate any shares that the attacker might have collected during the breach.

2.3.3 Result parties

The role of a result party can be assigned to either a separate user of the information system or the information system itself can be the result party if the secure computation result is used in further (public) computations by the system. In the

latter case, the information system contacts the computation parties itself, receives the result shares and recombines them to learn the result of the computation. In case of a separate user as a result party (e.g. a person authorised to access privacy-preserving survey results), the communication pattern depends on the sensitivity of the computation result. If only the result party is allowed to see the result then he must contact all of the computation parties directly, as the input party must do. Otherwise, the information system is allowed to cache the result value for the result party.

There is a big difference whether all computation parties are also result parties or not. If desired outputs are public, then some intermediary results can be declassified to gain efficiency. For example, if we have a secret-shared table and in the following computations we are only interested in its rows, where a given predicate holds, the following steps are commonly performed. First, we obliviously shuffle (randomly permute) the rows in the table to hide the initial order of rows. Then we obliviously compute the predicate on a given column and declassify the computation results. Finally, we discard all rows, where the declassified predicate value is false. It is easy to see that this way the computation parties learn the number of rows that satisfy the predicate. If computing parties are not allowed to learn the number of rows that satisfy the predicate, then we must obliviously compute the predicate on a given column and keep the result as a secret-shared binary *availability vector*. Further privacy-preserving computations would be done on the full-sized table, while also obliviously taking into account the values in the availability vector. Hence, this efficiency-privacy trade-off has to be considered when designing SMC applications and assigning roles to parties.

CHAPTER 3

CHALLENGES IN DEVELOPING REAL-WORLD SMC APPLICATIONS

3.1 State of the art in real-world SMC

3.1.1 Danish sugar beet auction

By real-world applications the author considers practical SMC applications that process real sensitive data. Thus, academic prototypes working in a lab setting or using generated or public data have been left out from this chapter.

Secure multi-party computation was first used in such large-scale real-world application in Denmark in 2008 when it was used to implement a double auction to reallocate sugar beet production contracts between farmers and the sugar beet production company [32]. In a double auction, each seller specifies how much goods he or she is willing to sell for each price from a fixed set of potential prices. Similarly, for each potential price, each buyer specifies how much goods is he or she willing to buy for that price. The task for an auctioneer is to collect these bids from all of the buyers and sellers and calculate a *market clearing price* – a price where total demand equals total supply. Using SMC to calculate this market clearing price mitigates the risk that any information learned from either buyers' or sellers' bids is misused, for example, in similar subsequent auctions.

In the Danish sugar beet auction, the role of the auctioneer was carried out by three computation parties: the sugar production company Danisco, the representative of the farmers (DKS, the sugar beet growers association) and the SMC technology provider (SIMAP project). The auction was divided into two logical phases: bidding and tallying, where the latter involved secure multi-party computation, see Figure 3.1.

As there were 1229 bidders (input parties) in the setup, it was important to make the bidding application easily accessible to the farmers and the Danisco

company. This was accomplished by distributing the application as a Java applet available from a web page that required user authentication. This application allowed each bidder to place a bid for either buying, selling or both for each of 4,000 pre-configured prices. Each bid was then shared using Shamir's 3-out-of-3 secret sharing scheme over a field $\mathbb{Z}_p$, where p is a 65-bit prime number [116]. In this application the three computation parties were not online during the first phase and the input shares were stored in an intermediary proxy database. Before sending the input shares to the proxy database, the bidding application encrypted each share of an input with the public key of the corresponding computation party. This was needed, as all of the shares were sent to a single proxy database that otherwise could easily reconstruct confidential bids.

In the second phase, the representatives of the three computation parties downloaded their corresponding shares from the proxy database and used their private key to decrypt the shares of bids. These decrypted input shares were then used to initiate the secure multi-party protocol to find the result of the double auction. The market clearing price itself was calculated using binary search, i.e. about 12 private comparisons were done during the computation phase. The second phase of the auction was carried out over a 100 Mbit LAN connection and took about 30 minutes. However, most of it was spent on decrypting the individual input shares by the computation parties. The SMC protocols were designed for semi-honest security setting. As a result of this first SMC double auction, about 25 000 tons of sugar beet production rights changed owner.

Since its first use in 2008, the same technology has been used again in the following years to reallocate sugar beet growing contracts in Denmark. The group of people behind the sugar beet auction founded a company named Partisia with the aim of commercialising the underlying technology [125]. In 2011 Partisia went live with a secure software-as-a-service auction platform Energiauktion.dk, where electricity contracts are exchanged between small and medium sized companies. The technology is similar to the one used in the sugar beet auction, but the computation parties are deployed in Amazon cloud and the secure computation is carried out over the Internet.

3.1.2 Financial benchmarking with SMC

The author of the thesis developed another SMC application working on real data and deployed it in Estonia in 2011 [121, 31]. The Estonian Association of Information Technology and Telecommunications (abbreviated as ITL) is a consortium that consists of local ICT companies and promotes their cooperation. As ICT is a rapidly evolving field, ITL members feel the need to access financial benchmarking results of each other to see how the economic sector is doing. There are the official and very detailed economic benchmarking results released by the Tax

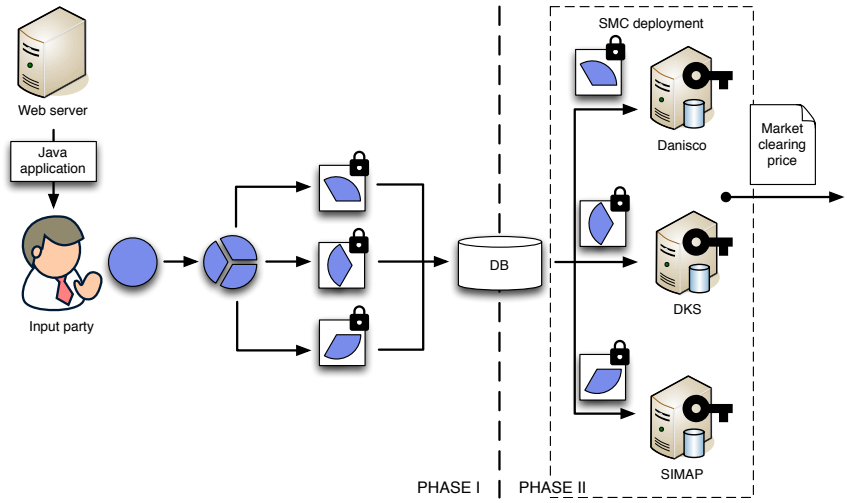


Figure 3.1: Deployment of SMC for Danish sugar beet auction. In phase I, each input share is encrypted with the corresponding computation party's public key and stored in a central server. In phase II, each computation party downloads and decrypts its shares and participates in the market clearing price computation.

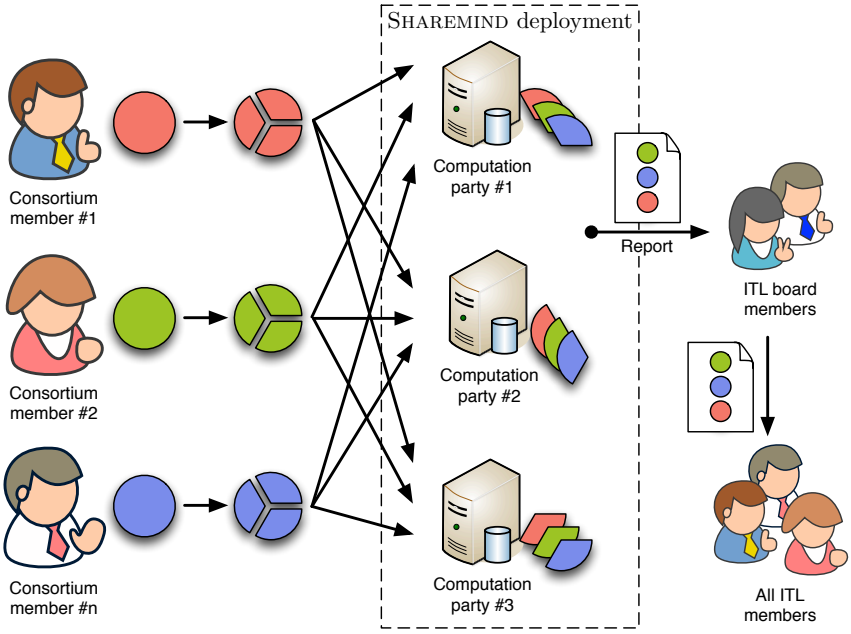


Figure 3.2: Deployment of the financial benchmarking application. Each consortium member submits several financial indicators. SHAREMIND is used to sort values of each financial indicator separately and the result is published to the ITL board as a report. The board shares the report with the rest of ITL members.

and Customs Board annually, but this is too infrequent. ITL members agreed that they will benchmark the sector themselves, by periodically collecting financial indicators from ITL members and releasing an overview for themselves.

Initially, the ITL board were to collect the financial indicators from its members and release the report for the rest of the members. However, as the ITL board also consists of board members of ICT companies, there was a fear of them misusing the data. This might also give the rest of the ITL members an incentive to lie about their data or not to participate in the process at all. To mitigate that risk, ITL decided to use secure multi-party computation to collect and process the data so that only overview is released to the ITL board who is in the role of result party.

The financial benchmarking application was built using the SHAREMIND 2 platform and the role of the computation parties was taken by three ITL members Zone Media, Microlink and Cybernetica. In order to reduce the maintenance cost for the hosts, Cybernetica helped to set up the other two computation nodes and thus had administrative access to these servers.

To make the data collection process easier for the ITL members (input parties), the input was securely collected using a web-based form that was integrated into ITL's web page where the representatives of ITL members had access to. The deployment of the components is shown on Figure 3.2, technical aspects of the benchmarking application are covered in more detail in Chapter 4.

The data processing step itself was fairly simple. The values for each of the collected financial indicators (total return, number of employees, percentage of export, profit, etc.) were sorted using an oblivious bubble sort implemented in SECREC 1 and the resulting sorted vector was published. On one hand, independently sorting each indicator meant that the connections between different indicators of a single company were erased. On the other hand, publishing all of the original values to the ITL board gave it an opportunity to further process the data with their favourite tools.

This is the first practical SMC application where the actual computation took place over the public Internet. Compared to the the sugar beet auction described before, the financial benchmarking application used web forms with JavaScript for data input and secret sharing, not relying on any third-party plug-ins. Moreover, the input shares were distributed among the computation parties by the data owner without using any central servers for storage.

ITL used the application twice to collect data from its members in 2011. On the second run, some enhancements were made. For example, the oblivious bubble sort was replaced with oblivious sorting network implementing Batcher's odd-even merge sort algorithm [11]. Additionally, a user feedback survey was also conducted together with the data collection. Although only 12 companies filled in the feedback survey, the results can be meaningful as the companies are influ-

ential players in local ICT market. First, the results (see [31, Fig. 2]) show that ICT companies value this kind of financial benchmarking. Secondly, ICT companies value privacy as 2/3 of the participants familiarised themselves with the SMC-related materials provided together with the benchmarking application and for about half of the participants the enhanced privacy measures were the reason they submitted their data. Moreover, 2/3 of the participants were willing to submit even more data (e.g. average salary or employee turnover). The user feedback questionnaire was also implemented using the SHAREMIND platform.

ITL board decided not to use the SMC-based system for subsequent data collection as they concluded that Estonian ICT companies are too unique for such anonymised survey. According to an ITL board member, from 70 ITL members, only up to 18 would have been comparable by such a survey. When taking into account that not all of these companies participated, the number of remaining participants was too small to draw any statistical conclusions [83].

3.2 Missing capabilities and algorithms

These first practical applications of SMC brought out some of the technical capabilities that were still missing from the implementations. First, there were no easy-to-use secure sorting methods implemented. Although in the financial benchmarking application, the quadratic complexity bubble sort was later replaced by a sorting network, it was not a generic solution. The sorting network structure had to be generated separately and taking into account the number of participants. As sorting is a widely used subroutine in many algorithms, an efficient and generic oblivious sorting method implementation is essential. The author addressed this challenge and implemented oblivious sorting methods described in Chapter 6.

Secondly, both of the applications described here make use of a single database table. In both applications, there is a single table with predefined set of columns and each input party (bidders in the case of Danish sugar beet auctions and company representatives in the case of financial benchmarking application) adds a single record to this table. However, there is another kind of application scenario where multiple data sets need to be combined together. Imagine a use case where input parties do not supply a single record but rather a whole data set and each one of them has a different structure (so they cannot be concatenated). In this case we need a privacy-preserving method to link records from one data set with the records from the other data set according to some predicate, similarly to the table join operation in SQL. Such secure database linking method was also developed by the author and is introduced in Chapter 5.

3.3 Lack of best practices in delivering and administration

For practical SMC application deployments, there are two main requirements. The computation parties must not collude with each other or any single outsider; and all of the communications channels between all of the input, computation and result parties must be secure.

In a distributed computing model such as SMC, parties have to communicate over secure (i.e. encrypted and authenticated) channels so an adversary cannot intercept or exchange messages. In practice, this is done by using TLS or similar public key cryptography setup. As a prerequisite, parties have to exchange public keys in order to know which incoming connections to trust. This key exchange has to be done using a separate authenticated out-of-band communication channel. It is also worth mentioning that although the underlying secret sharing based SMC schemes are information-theoretically secure, the communication channels in practice are secure only against computationally bounded adversary. Moreover, for performance reasons, protocol implementations use computationally secure pseudo-random number generators instead of true randomness.

In the two example applications described here, the most important requirement of SMC – non-collusion of computation parties – was not formally enforced. Moreover, non-collusion is extremely difficult, if not impossible, to enforce with technical measures. Instead, it should be covered with either contractual obligations or better yet, one's motivation to participate and keep their own data private.

As both applications were the first practical deployments of SMC by the respective research groups, there was less bureaucracy and more personal relationships involved: neither of the applications had any contracts between the involved parties. The fact there were no SMC-related contracts formed in the Danish sugar beet auction was confirmed in a private conversation among the SHAREMIND and sugar beet auction team members [108].

3.4 Limited practical validation

In the Danish sugar beet auction, the computation parties themselves did not deploy SMC software, their only responsibility was to keep their private key secret throughout the bidding and tallying phases. Similarly, in ITL financial benchmarking application, the chosen ITL members being in a computation party role provided a virtual machine for the deployment, but actually didn't install and manage SMC software themselves. This was done by a single administrator. Hence, large stakeholders, e.g. governmental institutions, had not yet deployed SMC for practical applications themselves. Such organisations have the responsibility to

handle their client or citizen data with care. Here lie interesting legal questions. For example, is sensitive data that is secret shared into random pieces and distributed among many parties still considered sensitive data from the legal point of view? Can secret sharing be considered a form of encryption and if so then where is the key? These questions have to be answered before it is possible to deploy secure multi-party computation applications for such stakeholders.

By the deployment model of SMC, each computation party is independent in following the protocol that computes something on the shares. If it encounters a protocol message that looks suspicious then it can – and should – halt the protocol to protect its shares. However, this model assumes that each computation party actually knows what the SMC software is doing. Provided that not every organisation will implement their own SMC software, there has to be a way to verify the software, either by code review, audit or having access to the source code.

From a technical point of view, IT departments of large organisations may not be as volatile as university research groups or SMEs that are more eager to deploy new software in their premises. Even with modern virtualisation technology, these organisations cannot deploy any bleeding-edge Linux distribution as they have specific processes in place to support and manage something that they are used to or have support contracts for. Also, learning a completely new platform may take too much resources. This is especially true for governmental organisations. Moreover, these organisations usually require more than a verbal promise that other involved parties allocate their resources to carry out the computation. Hence, a contract has to be formed between the computation parties that at least specifies the availability of the computation node as a service.

CHAPTER 4

DEPLOYING SMC FOR WEB APPLICATIONS

4.1 Data flow

In this chapter, we look at SMC applications where the role of input and/or result party is implemented as a web-based application. We do not consider deployments where web-based applications are computing nodes.

In SMC applications, the role of the input party is carried out by either the data owner or a delegate that has permission to process this input data. In web applications, this means that input data must be secret shared on the client side, i.e. in the web browser. Technical problems that this introduces are covered in Section 4.2.

In the simplest use case with an anonymous non-authenticated user, the user secret shares her input and distributes the shares among the computation parties as shown on Figure 4.1. The session ID is needed to synchronise the order of input records between the computation parties as, with many concurrent users, the order of received shares might get mixed up depending on the network configuration and delays between clients and computation parties. The session ID could also be generated by the server and given together with the application itself. However, in such case the application must forbid the user to use the same session ID twice.

If authentication is required, it generally does not make sense to authenticate to every computation party separately as this would require replicating user authentication code and user credentials database to each computation party. The latter, in turn, increases the risk of the user information database be stolen or leaked. Moreover, for an enhanced user experience, the communication with multiple computation parties should be transparent for the user. Therefore, the user should rather authenticate herself to a central authority and provide some kind of proof of authentication to the computation parties.

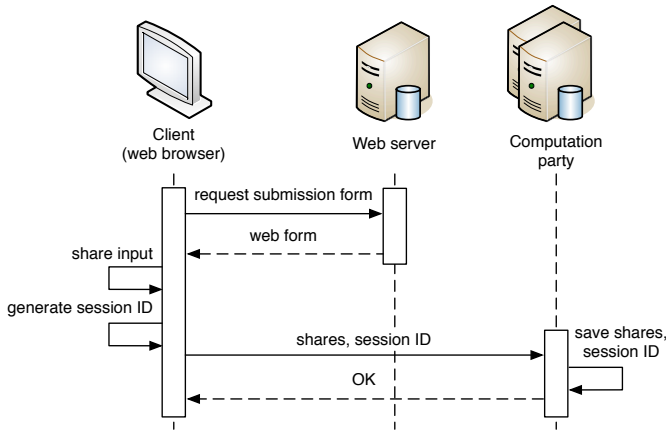


Figure 4.1: Saving shares for anonymous input party.

We can assume that the user authenticates herself with the same web page where she retrieves the application itself (Figure 4.2). In this case, this web page can verify if the user is eligible for accessing the application and if so then generate a unique token that is then embedded within the application. In case of an HTML form, this token can be a hidden form element. On the server side the token is tied with the user account.

In addition to the shares and a generated session ID as in the non-authenticated case, the web browser now also sends the authentication token to each computation party. The computation party itself must then contact the authentication server to check if the access token is valid (i.e. generated by that server and not expired). Upon a positive result, computation party stores the shares and replies to the user with a success. The user (client web application) shall, after receiving such confirmation from all the computation parties, notify the authentication server of the success. The authentication server then invalidates the access token. The access token could also be used as a session ID, but then the application must guarantee that the same session ID is never used to send different sets of shares on retries.

4.2 Overcoming barriers

4.2.1 Secret sharing in web browsers

First, secret sharing requires random number generation. It is possible to obtain cryptographically secure randomness directly from the operating system, or alternatively, use a pseudo-random number generator (PRNG) seeded with such strong randomness. Unfortunately, at the time of developing [121], web browsers did not have a standard (HTML or JavaScript) interface for such PRNG. More-

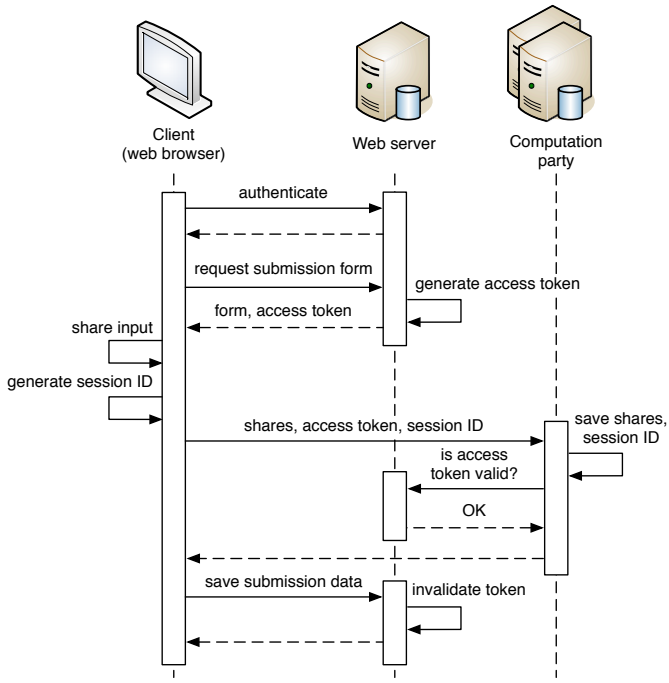


Figure 4.2: Saving shares for authenticated user.

over, JavaScript runs in a sandbox and therefore cannot communicate with operating system directly to gather randomness. Using browser plug-ins like Flash, Silverlight or Java applets, as used in the Danish sugar beet auction application described in Section 3.1.1, would have solved this problem. However, we did not want to introduce dependencies that would force the user to install third-party software. We felt that it reduces the applicability and makes them more reluctant to participate.

Notice that by design of SMC and in the example of using SHAREMIND with its 3-party protocol suite, we have a deployment with three separate servers hosted by independent non-colluding parties. Hence, we can ask each server for some amount of randomness by a simple server-side script and combine them together. The server has access to its operating system's entropy collection mechanism that is able to generate cryptographically secure randomness. On the client side, in the web browser, we use the bitwise XOR operation to combine the randomness received from each server (see Figure 4.3). Thus none of the servers know the resulting randomness that we are going to use as long as there is at least one non-colluding server.

If we need to share many inputs in the web browser, it makes sense not to use the randomness gathered from the servers directly. Instead, we ask the in-

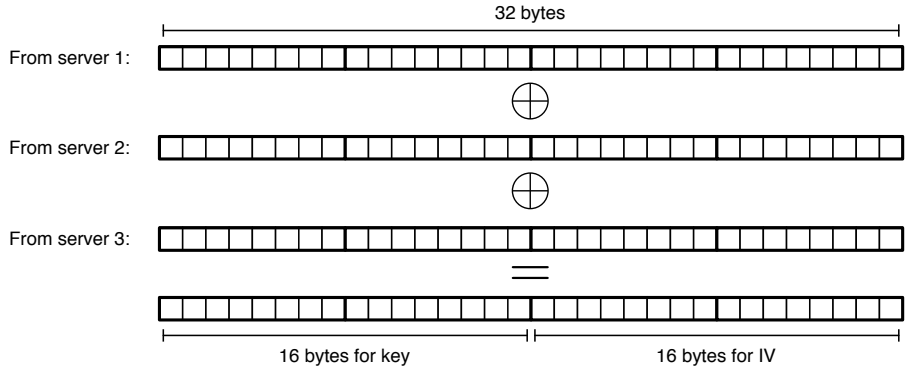


Figure 4.3: Combining randomness from three computation parties to obtain secure keying material for AES in counter mode.

dependent servers enough cryptographically secure randomness to instantiate a PRNG in JavaScript. For example, for AES-128 in counter mode, we need 128 bits of randomness for the key and another 128 bits of randomness for the initialisation vector (IV) [10]. Figure 4.3 illustrates this visually. As per [10, Appendix D.3], we can generate at most 400 million bits with this PRNG before it has to be reseeded with fresh randomness. As the additive secret sharing scheme used in SHAREMIND requires $2n$ bits of randomness to secret share an n-bit value, AES-128 in counter mode can be used to secret share 25 MB of data before new random seed is required from the servers. Hence, in most applications where a user fills in and submits a form, reseeding is not needed.

4.2.2 Communicating with computation parties

The Same Origin Policy

After the secret input is shared in the web browser, its shares have to be distributed among the computing parties, i.e. uploaded to the servers. We also need to be able to get information from the server, e.g. whether the authentication token was valid and the request was successful. Moreover, we might require even more than a request status from the server, e.g. in more advanced applications, computation nodes might reply with the shares of SMC computation result. Hence, we need to both upload and download data from the computation servers.

While it is possible for a web page to include resources from different servers, the Same Origin Policy[1] enforced by web browsers prevents JavaScript from accessing resources loaded from other origins than the one where the running code

[1]Same Origin Policy – `http://www.w3.org/Security/wiki/Same_Origin_Policy`

was loaded. In particular, this policy restricts JavaScript from making new HTTP requests to other origins. Simply put, "other origin" is defined as having a different domain, port or protocol scheme. Since the computation parties must be independent, it is natural to assume that they have different origin, e.g. they are hosted on different domains.

Therefore, there are two options to communicate with the computation parties from a web page. First, using a central proxy server so that the web page communicates with a single server, that in turn distributes the messages among the computation parties. This is similar to the proxy used in the Danish sugar beet auction described in Section 3.1.1. Second, using facilities provided by HTTP protocol or modern web browsers to bypass the Same Origin Policy.

Using a central proxy server

The central proxy approach requires each computation party to have an asymmetric key pair with the public key embedded in the web application. Each party's public key is used on the client side in the web browser to encrypt shares meant for this computation party. The encrypted shares of all parties are then sent to the central proxy server that distributes them among the independent computation parties. Due to the Same Origin Policy restrictions, the role of the central proxy should be carried out by the server where the web application is loaded from. This solves the problem of uploading data to the computation parties.

At the same time, the confidentiality of computation servers' replies must also be protected. Using public key cryptography for this communication direction might be cumbersome as each user would need to generate a new key pair in their web browser. Alternatively, the user may generate a symmetric encryption key and send it to the computation parties by encrypting it with their public keys. That way, only the user and computation parties know the common symmetric key to use in further communication. Once the generated symmetric key is exchanged, this hybrid cryptography approach may also be used to distribute shares as symmetric encryption is more resource-efficient.

However, using the central proxy server has some drawbacks. First, in order to receive confidential data from the computation parties, the application in the user's web browser has to generate a symmetric encryption key and this requires cryptographically secure randomness. The method of gathering randomness from the computation parties described in previous section does not apply here as the central proxy server would learn the combined random value. So the application has to depend on the user having a recent enough web browser that provides a cryptographically secure PRNG.

Secondly, central proxy server is a black box to the user and it can easily replace the public keys of the computation servers with its own. In order to avoid

this, the certificates containing the public keys must be signed by a trusted authority. However, since the keys are used in JavaScript then the verification of the certificates must also be carried out in JavaScript. Normally, this is done automatically by the web browser. Hence, it can be argued that communicating with computation parties directly is more transparent to the user as he can verify that the domains in use belong to the advertised computation parties. This can be done by either studying the downloaded application code or reviewing web browser's network console.

Bypassing the Same Origin Policy

There are three methods for bypassing the Same Origin Policy restrictions. Two of them, namely HTML5 Web Messaging API[2] and Cross-Origin Resource Sharing[3], are supported by modern web browsers and require the requested resource to know that it can be accessed from another origin. The third is a workaround using a dynamically created `script` tag with a callback and works in virtually every web browser. Next, we will give a brief overview of each of them.

In HTML, an `iframe` tag allows the browser to load a whole web page as a subpage of the current web page. This web page may be loaded from a different origin. However, the contents of the pages are strictly separated and the Same Origin Policy does not allow these two pages to communicate with each other. Nevertheless, with HTML5 Web Messaging application programming interface (API) support, it is possible for a web page to communicate with the content of an `iframe` in a controlled fashion. In our scenario, it is sufficient to load a simple web page in an `iframe` from every computation party and using the HTML5 Web Messaging API, distribute the shares among the `iframe`-s. Each `iframe` can then use its own JavaScript code to send the share to its server over a secure connection. Since each `iframe` communicates only with the server where it was loaded from, no Same Origin Policy restrictions apply here. Similarly, each `iframe` can forward each computation party's reply to the main web page.

Cross-Origin Resource Sharing (CORS) adds special headers to the HTTP protocol so a web page can explicitly state from which other origins it can be requested. When accessing resources from other origins with CORS, a browser makes a *preflight* request to see if the resource has the special CORS headers set to allow the request go through. With CORS headers set, cross-origin requests from JavaScript are enabled using ordinary means, e.g. the `XMLHttpRequest` method.

At the time of developing the ITL financial benchmarking application described in Section 3.1.2, the penetration of both HTML5-capable and CORS-

[2]HTML5 Web Messaging – `http://www.w3.org/TR/webmessaging/`
[3]Cross-Origin Resource Sharing – `http://www.w3.org/TR/cors/`

enabled web browsers was very low so a fallback solution was also needed. A well-known workaround with the HTML `script` tag was used to accomplish that. Loading resources (e.g. images and scripts) in HTML is not affected by the Same Origin Policy, so they can be loaded from any origin. Hence, we can dynamically create a new HTML `script` element with a source address of our choice and insert it into the Document Object Model (DOM) tree of the current document. Upon this, the requested file is automatically loaded and, if it is valid JavaScript file, it is also automatically executed. As we need to receive data this way, the server can send the values in JavaScript Object Notation (JSON) that is given as an argument to a function call:

```
someFunction({foo: 'This is a JSON object'});
```

This is a so-called "JSON with padding" format, or simply JSONP [79].

With this `script` tag workaround, we can send data to the server by encoding it into the URL of the requested JavaScript file, for example:

```
<script src="file.js?k1=value1&k2=value2"></script>
```

It is important to note here that one can only make HTTP GET request in this way as resources are loaded using GET requests. As GET request size (length of parameters) limit is much smaller than for POST request, one can only send a small amount of data with each request using this method. The GET request size limit is configurable on the server side and in practice, web browsers set this limit to 2–8 kilobytes. It was not a limiting factor in the financial data benchmarking application, but has to be considered in larger applications.

The HTTP protocol nor web browsers provide any facilities to guarantee authenticity of web pages, e.g. by signing the web page or JavaScript code. Hence, we have to rely on communication channel security provided by HTTPS. The connection to the initial web server as well as to the three web servers of computation parties use encrypted and authenticated HTTPS connections. Using any of the three methods mentioned above, the connections to the web servers of computation parties are made in the background (either by JavaScript using `XMLHttpRequest` method or by a hidden `iframe`). Hence, it is important that these servers use HTTPS certificates that are automatically accepted by all relevant web browsers – since the loaded web page is not shown to the user, he or she cannot take any explicit action to accept an untrusted certificate. It is sufficient if every computation party's web server uses a certificate that is signed by a trusted root certificate authority or it's intermediate authority. Modern web browsers bundle about 50 trusted root certificates and some of them, e.g. the *Let's Encrypt* initiative[4], even provide free web server certificates so this requirement can be easily fulfilled.

[4]Let's Encrypt – `https://letsencrypt.org/`

4.3 Prototypes

For the ITL financial benchmarking application [121], the author of this thesis developed a JavaScript library based on the Dojo Toolkit[5] that was used in the web browser to secret share input data and securely distribute the shares among the web servers of computation parties. The library was able to switch between the HTML5 Web Messaging API and the `script` tag workaround as necessary, depending on the version of the web browser.

This library made it possible to easily create other web-based frontends to SMC applications. The following is an overview of some such applications that used and enhanced this library. The author of this thesis did not develop these applications. However, they use the same ideas and architecture and the author of this thesis was in the role of a consultant.

4.3.1 Cloud demo

A meeting of the European Cloud Partnership Steering Board consisting of representatives and decision makers from the IT and telecommunications sector was scheduled to be held in Tallinn in July 2013. The Estonian Information System Authority (RIA) was interested in demonstrating new innovative cloud technologies to the board members and contacted Cybernetica.

We decided to create a web application that uses SMC to compute various statistics on the income of Estonian public sector employees. This data is freely available in Estonia, and as such did not require us to obtain any permissions from the data owners. Consequently, we could show real results instead of depending on synthetic data. At the same time, in some countries this data is not public and so using secure multi-party computation to publish only aggregate values on such financial data serves as a good demonstration for the usefulness of SMC.

The salary information for all of the positions in local governments and ministries was collected and uploaded to the SHAREMIND installation using a web application developed with the JavaScript library described earlier.

The computation parties were hosted by three organisations: RIA, the IT Center of the Ministry of Internal Affairs (SMIT) and Cybernetica. Each of them hosted a SHAREMIND (version 2) computation node and an instance of Apache web server. However, as the event was about cloud computing, the computation parties did not host the servers in their premises but rather rented a cloud server for independent cloud providers in Europe. RIA hosted its server in Microsoft

[5] Dojo Toolkit – `https://dojotoolkit.org/`

Azure[6], SMIT in Zone Media[7] and Cybernetica in Linode[8].

We decided to collect input data without the local proxy database used in the financial benchmarking application. Instead, we created an Apache web server module so that an Apache server at each computation party received the input shares from the web browser and forwarded them to the local SHAREMIND instance without saving them in any intermediary database. As such, Apache acted as a HTTP interface for the SHAREMIND server.

In the case of the ITL financial benchmarking application (Section 3.1.2), a web frontend was used only to input data into the system. The actual SMC process was initiated by running a command-line application and the results were saved as a file (although accessible from the web). In the public sector income analysis demo, the computation was also initiated from the web and when finished, the results (frequency tables and histograms) were also sent back to the web application. This was done to demonstrate that, on smaller data sets, SMC can be done in near real time and web-based user interface is a convenient way to interact with such a setup.

After demonstrating the web application to the European Cloud Partnership Steering Board members, Cybernetica decided to make it available as a public demo. Since the partners hosting the computation servers could not commit to hosting indefinitely, it was decided that Cybernetica will take over managing all three servers. Three independent cloud hosting providers were chosen: Amazon EC2, Microsoft Azure and Zone Media.

For the new deployment, the Apache proxy was replaced by a Node.js proxy that allowed to keep persistent connections between the SHAREMIND server and the proxy. Previously, with the Apache module, a new connection to SHAREMIND server was initiated on each web request, making it slower. Moreover, the Node.js proxy uses the WebSocket API [55] to exchange data with the web browser. WebSockets allow to directly establish cross-domain connections as they are not restricted by the Same Origin Policy. The WebSocket connections are managed by Node.js socket.io library that falls back to Flash or JSONP-based solutions on older web browsers. Consequently, communicating with the computation parties directly form the web page was faster and more robust than before.

In June 2014 the public cloud demonstration application was upgraded to use SHAREMIND Application Server.

[6]Microsoft Azure: Cloud Computing Platform & Services – `https://azure.microsoft.com/en-us/`

[7]CloudServer - Zone.ee – `https://www.zone.ee/en/service/cloudserver/`

[8]SSD Cloud Hosting - Linode – `https://www.linode.com/`

4.3.2 Internal employee satisfaction survey

In January 2014, Cybernetica conducted its annual company-wide employee satisfaction survey using SMC. The deployment was very similar to that of the ITL financial benchmarking application – Cybernetica employees used a web-based form to participate in a survey. The three computation nodes were hosted by Cybernetica's own employees. The survey system was not tied to company's central authentication system. Each employee was sent a unique random numerical token that served as an identifier. This token allowed the company's human resources specialist to track who had already participated and also allowed the employee to re-submit the form so that only the last version was taken into account.

The SMC-based employee satisfaction survey was used again in 2015 and is the foundation of a cloud-based software-as-a-service survey system described in Section 4.3.4.

4.3.3 Tax fraud detection prototype

The Estonian Tax and Customs Board has estimated that the government loses over 220 million euros every year due to some companies avoiding value-added tax (VAT) [74]. In 2013, the government proposed a change in legislation that would make it compulsory for all companies to report all transactions with any partner with whom the monthly sum of transactions is at least 1000 euros. However, the president of Estonia vetoed the proposal and sent it back to the parliament. In his decision, among other reasons, the president also expressed concerns about the security needs of a new database that would contain a large portion of all the transactions in Estonia [123].

Inspired by the raised privacy concerns, Cybernetica decided to build a prototype application that uses SMC to hide individual transactions while disclosing companies with a fraud risk [20]. The application performs the risk analysis obliviously, flagging only companies with suspicious transactions. In the end, only the flagged companies are revealed giving the tax official a well-founded justification for seeing the full data of these companies. In this process, the privacy of honest taxpayers is protected.

The user interfaces of the prototype for input parties and result party were web-based. Companies could use a web-based form to upload their transaction data in an XML format where each value was then secret shared in the web browser using an updated version of the JavaScript library. The shares were then distributed among the three computation parties. Similarly, an official could start the risk-analysis process and retrieve the results in a web application.

On a test set of 2.6 million transactions from 2000 companies, the privacy-preserving analysis took 43 minutes when deployed on Cybernetica's local cluster.

Based on the information from the Tax and Customs Board that data about a total of 100 million transactions is provided by 80 000 companies each month, it was estimated that the analysis would take ten days to complete.

After addressing some of the concerns, the updated version of the tax legislation was passed and the Tax and Customs Board started requesting the tax declarations as initially planned. One of the concerns of the Tax and Customs board was that they would have to share the fraud detection algorithm with other computation parties, as currently the computed algorithm itself is not hidden from the parties. Nevertheless, the Tax and Customs Board recognised the need to protect privacy of the data owners and agreed to reconsider adopting the technology in the future if its deployment cost is further reduced [20].

The updates in both the JavaScript library and server-side components developed for the tax fraud detection system prototype application made it even easier to create privacy-preserving web-based sample applications. For example, the author of this thesis were able to create a working voting application for the 2014 Eurovision Song Contest in just one evening.

4.3.4 Secure survey system

A secure online survey platform is implemented and deployed by PRACTICE project partners Alexandra Institute and Partisia (Denmark) and Cybernetica (Estonia) since 2013 [124, 85]. The survey system is publicly available online at `https://practice-survey.eu/` and allows organisers to design surveys, let people participate in the survey and see visual reports.

The user inputs are secret-shared in their web browser and distributed among computation servers using JavaScript `XMLHttpRequest` calls with CORS HTTP headers. All of the aggregated survey results are computed using SMC. The secure survey system allows to choose between two different SMC run-times: a 3-party SHAREMIND system with passive security, or a 2-party FRESCO backend using SPDZ protocol suite with active security developed by Alexandra Institute and Aarhus University.

4.4 Best practices

As seen from Table 4.1, the client side JavaScript library for developing privacy-preserving web applications has been gradually upgraded to support more data types and newer SHAREMIND versions. Most importantly, the initial need for a proxy database has been removed so the web server (Node.js or Apache module) forwards the shares directly to the SHAREMIND computation node. For ease of deployment and administration, a possible step forward would be to integrate web

server into SHAREMIND so that it has a native HTTP interface itself.

By now, most up-to-date web browsers support using different technologies (HTML5 Web Messaging API, CORS headers, WebSockets) to bypass the Same Origin Policy in a controlled fashion. These also support using HTTP POST requests with practical size limits in gigabytes. Hence, using several requests, there is virtually no limit on the amount of exchanged data. Similarly, in addition to asking cryptographically secure randomness from the computation parties, newer web browsers support the Web Cryptography API[9] that also provides a cryptographically secure PRNG via JavaScript.

However, there is still an inherent problem with deploying SMC applications in the web. The idea of secure multi-party applications is to eliminate the need to trust any single entity. Instead, the data owner trusts that a group of entities (computation parties) behaves according to a protocol. The problem with web-based SMC applications is that the application code itself (namely the JavaScript) is loaded from a single server in the form of a web page. As web applications cannot be signed, the user must now either trust the web server providing the page or audit the entire client-side code. In a sense, it is a variant of the "trusting trust" problem [122].

[9] Web Cryptography API – `http://www.w3.org/TR/WebCryptoAPI/`

Date	Browser side tech.	Server side tech.	Transport protocol	Used in
2011.02	JavaScript, HTML5 Web Messaging or dynamic `script` tag.	Apache server with PHP and proxy database, SHAREMIND 2.	HTTP/JSONP.	ITL financial benchmarking app, STACC private survey.
2013.05	JavaScript, HTML5 Web Messaging or dynamic `script` tag.	Apache module, SHAREMIND 2.	HTTP/JSONP.	Cloud demo (initial version).
2013.10	JavaScript, `socket.io`.	Node.js with `socket.io`, SHAREMIND 2.	WebSocket, fallback to `XMLHttpRequest` polling or Flash.	Cloud demo (public version).
2014.01	JavaScript with `socket.io`, supports floating point data types.	Node.js with `socket.io`, SHAREMIND 3.	WebSocket, fallback to `XMLHttpRequest` polling or Flash.	Cybernetica employee satisfaction survey 2014.
2014.04	JavaScript with `socket.io`, supports all data types supported by SHAREMIND.	Node.js with `socket.io`, SHAREMIND 3 or Application Server.	WebSocket, fallback to `XMLHttpRequest` polling or Flash.	Cloud demo (public version), Cybernetica employee satisfaction survey 2015, tax fraud detection system prototype, Eurovision voting demo.
2015.04	JavaScript with AngularJS, supports all data types supported by SHAREMIND.	Java, SHAREMIND 3 or FRESCO/SDPZ.	`XMLHttpRequest` with CORS headers.	PRACTICE Secure Survey System.

Table 4.1: Upgrades to the JavaScript library and related server-side components.

CHAPTER 5

PRIVACY-PRESERVING DATABASE LINKING

5.1 Introduction

By database linking, we mean merging databases horizontally so that the records from different input databases belonging to the same entity are merged into one record. An identifier for this *entity* has to be present in each input database and we call it a *key column* or a *key value*.

Such merging operation allows us to combine customer data from different organisations or link together state databases for research and supporting governance decisions. For example, for statistical analysis of health insurance, one can link together insurance records and hospital patient records based on a Social Security Number.

In this work we only consider the case where the linking is done based on the equality of the key values in different data sets. In relational algebra and Structured Query Language (SQL), this is called *equi-join*. Although there are several different types of join operations, depending on how missing values are handled, we consider here only three of them that are most commonly used in applications. These are all defined for two input tables ("left" and "right", respectively) and can be cascaded to support more inputs.

- *Inner join* considers only records that have a matching key value in both the left and right data tables. Formally, in relational algebra, inner join is a subset of the Cartesian product of the two input relations R and S and can be expressed as

  $$\sigma_\theta(R \times S),$$

  where σ is a selection of elements based on a predicate θ, e.g. $R.key = S.key$.

- *Left outer join* takes the left input table and complements each of its record with the values from the right table if a matching key value is found, or a special "empty value" (e.g. "NULL", "N/A", etc.) if a matching key is not found.

- *Right outer join* is a reverse to the left join. The right input table is taken as-is and complemented with records from the left table that have matching key values.

If a key value appears more than once in either input table, it is linked independently, e.g. all pairs are formed between the sets containing the occurrences of this key value from the left and right input tables. Note also that a key may consist of multiple attributes (e.g. a person name and a zip code) and all of the attributes have to match in order to consider it being equal to another key value.

In this chapter, we will introduce a privacy-preserving equi-join functionality based on an oblivious pseudo-random permutation function. The author has published the described protocols and their initial benchmarking results in [96] with its extended version available at the Cryptology ePrint Archive [97].

5.2 Privacy-preserving join operation

An ideal functionality for database join operation takes two secret-shared input data sets and outputs a new randomly ordered secret-shared data set where the join predicate holds. No new knowledge, except the number of rows in the joined data set should be published. While it is possible to hide the number of rows in the result data set by adding dummy records, it is not practical as the size of the output would have to be the product of the number of records in the input data sets. Randomising the order of output records is necessary for limiting leakage, if at some point a part of either the join operation input or output is published.

There is a straightforward privacy-preserving solution that naïvely follows the database join algorithm. We can create a Cartesian product of the two input tables by comparing each possible key value pair and adding the secret-shared comparison result to the table. The resulting table is then shuffled, i.e. its rows are randomly reordered by using an oblivious shuffling protocol described in [98]. Then the values in the comparison column are opened and all rows with the comparison result bit set to zero are removed. The remaining table is exactly the inner join of the two input tables. Notice that this method (referred to as NAIVEJOIN), is not limited to using only an equality predicate, other oblivious operations (e.g. less-than or greater-than) are also possible. As the oblivious shuffling protocol has a complexity of $\Theta(m \log m)$ for m-element vector, the asymptotic complexity for the NAIVEJOIN algorithm is $\Theta(m_1 m_2 \log (m_1 m_2))$, where m_i is the number of

rows in the i-th input table. However, in practice, its run-time is dominated by the $m_1 m_2$ oblivious comparisons. We will treat NAIVEJOIN as a baseline solution.

Theorem 1. *If a secure multi-party computation framework provides universally composable protocols for database shuffle and oblivious comparison, then the* NAIVEJOIN *protocol is universally composable in the information theoretical model.*

Proof sketch. The formal proof relies on three facts. First, the oblivious comparison leaks no information and the extra column can be simulated without problems. Second, the shuffle completely hides the order of the database rows. Third, the number of rows can be deduced from the ideal output and it coincides with the number of ones in the publishing phase.

In the simulation construction, we fake all shares until the shuffle phase without any knowledge of the true output. Next, we use the oblivious shuffle protocol to extract input shares of all malicious parties and submit them to the trusted third party who outputs the corresponding output shares. The simulator assigns them to random rows for $m_1 m_2$ shuffle outputs and sets the corresponding comparison column to 1. Remaining rows are filled with shares of zeros. After the shuffle simulation is completed, the simulator opens the values in the comparison column. □

5.2.1 Database join for unique key values

Electronic Codebook (ECB) is a block cipher mode of operation that encrypts each block of plaintext independently. Hence, for the same encryption key, equal plaintext values yield equal ciphertext values. If we apply such encryption to the values in the key column with an encryption key s, that none of the computation parties know, it is secure to reveal those encrypted key values. Without the knowledge of the encryption key s, no party can trace any elements back to the real values. As different keys map to different ciphertexts, we can perform the actual join operation on declassified ciphertexts. However, non-key columns must remain secret-shared.

The ECB encryption mode can be formalised as a pseudo-random permutation family (π_s), where the encryption key s uniquely defines the applied permutation. In our case, this encryption key s is secret-shared among the computation parties.

However, this database join leaks some information as it is possible to track which rows from the first table are linked to particular rows in the second table. By obliviously shuffling both input tables, this kind of leakage is avoided [98]. Algorithm 1 lists the steps of the resulting oblivious database linking protocol PRPJOIN based on pseudo-random permutation (PRP).

Algorithm 1: Secure implementation of PRPJOIN operation.

Data: Secret-shared database tables T_i with key columns $\boldsymbol{k}_i$
Result: Secret-shared equi-join T^* of the input tables

Database shuffling phase:

1 Computation parties obliviously shuffle each database table T_i
Let T_i^* denote the resulting shuffled table with a key column $\boldsymbol{k}_i^*$

Encryption and join phase:

2 Miners choose a pseudo-random permutation π_s by generating a shared key s

3 Miners obliviously evaluate π_s on all shared key columns $\boldsymbol{k}_i^*$

4 Miners publish all values $\pi_s(k_{ij}^*)$ and use standard database join to merge the tables based on columns $\pi_s(\boldsymbol{k}_i^*)$. Let T^* be the resulting table.

Optional post-processing phase for colliding keys:

5 If there are some non-unique keys in some key column $\pi_s(\boldsymbol{k}_i^*)$, miners should perform additional oblivious shuffle on the secret-shared table T^*

All the steps in Algorithm 1, except finding matching keys among published values in step 4, are performed on secret-shared data. Most importantly, the resulting joined table T^* is also secret-shared and can be used in further computations. Note the optional last step for obliviously shuffling the output in case there are colliding keys in the input tables. This is further discussed in Section 5.2.3.

Note that since the actual join operation is performed on published values, it is also possible to do left or right outer joins instead of the inner join. For outer joins, it might be necessary to add "empty values" to records that do not have a matching key in the other table. In practice, these empty values are (application-specific) secret-shared constants. Adding these to the relevant rows can be done as part of the public join operation, as the final oblivious shuffle hides their location in the final protocol output.

Theorem 2. *Let (π_s) be a pseudo-random permutation family. If a secure multi-party computation framework provides universally composable protocols for database shuffle and oblivious evaluation of $\pi_s(x)$ from secret shared values of x and s, and there are no duplicate key values in any of the input tables, then the* PRPJOIN *protocol is universally composable in the computational model.*

Proof sketch. For clarity, let us analyse the security in the modified setting where (π_s) is the set of all permutations and Steps 1–4 are performed by a trusted third party. Let m_1, m_2 and m be the number of rows in the input tables and in the final database table, respectively. Let $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$ be the vectors of encrypted values

published during PRPJOIN protocol. For obvious reasons, $|\boldsymbol{y}_1 \cap \boldsymbol{y_2}| = m$ and the set $\boldsymbol{y}_1 \cup \boldsymbol{y}_2$ consists of $m_1 + m_2 - m$ values, which are chosen randomly from the input domain without replacement. As Step 1 guarantees that the elements in $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$ are in random order, it is straightforward to simulate $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$ given only the number of rows m.

The simulation of the protocol is straightforward. As the protocol starts with the secure oblivious shuffles, the simulator first executes sub-simulators of these protocols. The input extraction phase allows us to recover all input shares of the corrupted parties. Hence, the simulator can forward these input shares to the trusted third party, who replies with output shares of resulting join table. From these shares it can trivially learn the number of rows m in the final table.

For the output equivocation phase of the shuffle sub-simulators, the simulator has to construct tables with the initial number of rows. For that, it splits the result table into columns for input tables and duplicates the key column for both of them. As $m \leq m_1, m_2$, the simulator has to fake the rest of the remaining $m_1 - m$ and $m_2 - m$ rows, respectively. For that, it generates valid shares of zeros for all the values in the remaining rows. The simulator randomly permutes both tables and gives them to the respective sub-simulator for output.

Next, the adversary starts to evaluate PRP on these key columns. The simulator uses output equivocation to specify the resulting published encryptions $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$. The simulator chooses the values for $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$ so that correct rows are joined. For this, the simulator first generates m unique ciphertexts that correspond to matching rows. Additionally, the other $m_1 - m$ and $m_2 - m$ ciphertexts are generated so that there are no collisions with any of the other ciphertexts.

Note that the rows obtained from the TTP are perfectly simulated. As the adversarial coalition is small enough, the adversary cannot distinguish the fake shares of remaining rows from the actual shares in the protocol.

It is easy to see that the simulation holds in the semi-honest model. The same is true for the malicious model with an honest majority, since honest parties can always carry out all the computations without the help from the adversarial coalition. In case of a dishonest majority, the adversarial coalition is allowed to learn its output and then terminate the protocol. In our case, the simulator must terminate the execution when the adversarial coalition decides to stop after learning the encrypted vectors $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$.

We can use the same simulation strategy for the original protocol where the trusted third party uses a pseudo-random permutation family. As the key s is unknown to all parties, the joint output distributions of the real and hybrid worlds are computationally indistinguishable. The latter is sufficient, as security in the hybrid model carries over to the real world through universal composability of share shuffling and oblivious function evaluation protocols. □

By combining oblivious AES (see Section 5.3) as a pseudo-random permutation function π_s and oblivious database shuffle protocol from [98] we get an efficient instantiation of the PRPJOIN protocol. The resulting protocol performs $\Theta(m_1 + m_2)$ oblivious PRP evaluations and $\Theta(m_1 \log m_1 + m_2 \log m_2)$ public computation operations, where m_1 and m_2 denote the number of rows in input tables[1].

5.2.2 Handling unique multi-column key values

In some data sets, the identifier for an entity might consist of several attributes. For example, a person might be identified by the combination of name and a birth date or a name and a ZIP code. We can deduce this case of using multiple columns for a key value to the previous case by hashing values in multiple columns together to obtain a single value. An *ε-almost universal hash function* is a function $h : \mathcal{K} \times \mathcal{M} \to \mathcal{T}$ that compresses message into shorter tags so that the following inequality holds:

$$\forall x, x' \in \mathcal{M} : x \neq x' \Rightarrow \Pr\left[k \leftarrow \mathcal{K} : h(k, x) = h(k, x')\right] \leq \varepsilon.$$

Such hash function should support efficient oblivious evaluation to be usable in our scenario. The Carter-Wegman construction [42]

$$h(\boldsymbol{k}, \boldsymbol{x}) = x_s k_s + \cdots + x_2 k_2 + x_1 k_1$$

is a good candidate for our application as it requires only a few simple operations and is $(2^{-\ell})$-almost universal when computations are done over the field $\mathbb{F}_{2^\ell}$. Alternatively, we can also make use of several independent Carter-Wegman functions over $\mathbb{F}_2$. For ℓ independently chosen keys, we still get the collision probability of $2^{-\ell}$. In the semi-honest model, the resulting protocol retains the same communication complexity as the complexity scales linearly w.r.t. the bit length. In the malicious model, the communication complexity depends on the implementation of the oblivious multiplication protocol. Compared to other constructions based on pseudo-random functions, the Carter-Wegman function has a low multiplicative complexity, which makes it a good choice for reducing the block cipher input size in our scenario. Oblivious hashing algorithm OHASH steps are listed in Algorithm 2.

Theorem 3. *If a secure multi-party computation framework provides universally composable protocols for addition and multiplication over $\mathbb{F}_2$, the* OHASH *protocol is universally composable in the information theoretical model. For ε-almost*

[1]The theoretical asymptotic complexity is higher, as the size of the database can be only polynomial in the security parameter and thus oblivious PRP evaluation takes $\mathsf{poly}(m)$ steps. Consequently, the protocol is asymptotically more efficient than the naive solution as long as the PRP evaluation is sub-linear in the database size.

Algorithm 2: Oblivious hashing OHASH.

Data: A secret-shared s-bit message $\boldsymbol{x}$
Result: A secret-shared ℓ-bit hash value $(h(\boldsymbol{k}_\ell, \boldsymbol{x}), \ldots, h(\boldsymbol{k}_1, \boldsymbol{x}))$

Offline phase:

1 Generate shared random keys (k_{ij}) for the Carter-Wegman construction

Online phase:

2 Treat each key tuple as a long bit string $\boldsymbol{x} = (x_s, \ldots, x_1)$
3 Use secure scalar product algorithm to compute the secret shared hash code:

$$h(\boldsymbol{k}_j, \boldsymbol{x}) = x_s k_{sj} + \cdots + x_1 k_{1j}, \text{ for } j = 1, \ldots, \ell$$

universal hash function and m invocations of OHASH*, the probability that two different inputs lead to the same output is upper bounded by $\frac{1}{2}m^2\varepsilon$.*

Proof sketch. The claim about security is evident as multiplication together with addition is sufficient to implement scalar product over $\mathbb{F}_2$. The collision probability follows from the union bound $\Pr[\mathsf{collision}] \leq \binom{m}{2} \cdot \varepsilon \leq \frac{m^2\varepsilon}{2}$. □

Security of the hash function

By using a hash function, we introduce a possibility that two different key values have an equal hash value. Such collision among the hashed key values of a single table invalidates the uniqueness assumption of Theorem 2, whereas a colliding hash value for non-equal keys from different tables introduces a false entry in the resulting joined table. Hence, the key length of the chosen hash construction has to be chosen so that the probability of collisions would be negligible.

For example, by accepting a failure probability of 2^{-80}, a 128-bit Carter-Wegman construction allows us to use tables with up to 33.5 million entries. This is enough for many practical applications. For smaller data sets with up to a million entries, even an 119-bit Carter-Wegman construction is sufficient.

Optimising oblivious hashing

We noticed that, if implemented naïvely, the OHASH may become the bottleneck in the PRPJOIN protocol. This is due to the fact that Algorithm 2 computes each bit of the hash separately and thus duplicates the data vector $\boldsymbol{x}$ for each output bit, resulting in ℓ copies of $\boldsymbol{x}$. This yields larger communication complexity than our chosen pseudo-random permutation function, AES.

However, as the Carter-Wegman construction is essentially a matrix multiplication over the field $\mathbb{F}_2$, we can take advantage of a common matrix multiplication optimisation used in SMC. First, values are converted from 3-out-of-3 secret sharing scheme to the replicated secret sharing scheme by each party sending its shares to the next one in a circle. This allows all parties to compute all the necessary scalar products locally. Finally, to guarantee uniform share distribution, and thus security [25], the output shares are reshared by masking them with random values. The result is depicted as Algorithm 3 for three computation parties. For double indices, the first index shows the output bit being computed while the second shows which party holds the bitstring. All inputs are bitwise shared and the superscript index notes the bit operated on.

Algorithm 3: More efficient protocol for Carter-Wegman hash.

Data: Secret-shared s-bit value $[\![m]\!]$ and shared s-bit keys $[\![k_1]\!], \ldots, [\![k_\ell]\!]$
Result: Secret-shared ℓ-bit hash value $[\![c]\!]$

Precomputation phase:

1 Each party $\mathcal{P}_i$ generates ℓ random bits $r_i^1, \ldots, r_i^\ell \leftarrow \mathbb{Z}_2$

Data distribution phase:

2 $\mathcal{P}_1$ sends s-bit shares $m_1, r_1, k_{1,1}, \ldots, k_{\ell,1}$ to $\mathcal{P}_2$
3 $\mathcal{P}_2$ sends s-bit shares $m_2, r_2, k_{1,2}, \ldots, k_{\ell,2}$ to $\mathcal{P}_3$
4 $\mathcal{P}_3$ sends s-bit shares $m_3, r_3, k_{1,3}, \ldots, k_{\ell,3}$ to $\mathcal{P}_1$

Post-processing phase:

5 Each $\mathcal{P}_i$ computes

$$w_{ij}^t \leftarrow m_i^t \wedge k_{j,i}^t \oplus m_{i-1}^t \wedge k_{j,i}^t \oplus m_i^t \wedge k_{j,i-1}^t$$

for each key $j \in \{1, \ldots, \ell\}$ and bit $t \in \{1, \ldots, s\}$ and sums them up together with re-randomisation $c_i^j \leftarrow w_{ij}^1 \oplus \cdots \oplus w_{ij}^s \oplus r_i^j \oplus r_{i-1}^j$

Theorem 4. *Assume that the shares of m are correctly generated. Then Algorithm 3 is correct and secure against a single passively corrupted party.*

Proof sketch. For each bit c^j of MAC the correctness follows from

$$\begin{aligned}
[\![c^j]\!] &= \bigoplus_{i=1}^{3} \left(\bigoplus_{t=1}^{s} m_i^t \wedge k_{j,i}^t \oplus m_{i-1}^t \wedge k_{j,i}^t \oplus m_i^t \wedge k_{j,i-1}^t \right) \oplus r_i^j \oplus r_{i-1}^j \\
&= \bigoplus_{i=1}^{3} \bigoplus_{t=1}^{s} \left(m_i^t \wedge k_{j,i}^t \oplus m_{i-1}^t \wedge k_{j,i}^t \oplus m_i^t \wedge k_{j,i-1}^t \right) \\
&= \bigoplus_{t=1}^{s} (m^t \wedge k_j^t) = h(k_j, m)
\end{aligned}$$

since the inner most sum contains all combinations of $m_a \wedge k_b$.

For the security analysis, it is sufficient to consider the corruption of $\mathcal{P}_2$ who receives all shares owned by $\mathcal{P}_1$. Note that two shares out of three always have a uniform distribution. Hence, it is trivial to simulate all messages received by $\mathcal{P}_2$. Since $\mathcal{P}_2$ is semi-honest, the simulator can extract shares of the message and keys from the input of $\mathcal{P}_2$ and submit them to the trusted party who will return shares $c_2^1, \ldots, c_2^\ell$. Since the simulator knows what random values $r_2^1, \ldots, r_2^\ell$ $\mathcal{P}_2$ is going to use, it can pick $r_1^1, \ldots, r_1^\ell$ so that $\mathcal{P}_2$ will indeed output $c_2^1, \ldots, c_2^\ell$. □

Benchmarks show that for 288-bit input and 128-bit output, the running time of a single OHASH as shown in Algorithm 2 is 25 ms, while the amortised running time per hash is 5.7 ms. For the optimised version shown as Algorithm 3 these figures are 11 ms and 0.012 ms, respectively. As the optimised version uses network more efficiently, its optimal amortised cost is reached by doing 4096 hash computations in parallel. For the unoptimised version, the stable network throughput is reached already with 16 parallel invocations. In the context of PRPJOIN, the former is about 10 times slower than AES, while the optimised version has only a minimal impact, counting for about 5% of the running time of the whole protocol.

5.2.3 Database join for non-unique key values

As the PRPJOIN publishes pseudo-randomly permuted key values, it is trivial to see that it leaks the coinciding keys in input tables. Moreover, without the extra shuffle at the end of the protocol it shows how many records match for each particular key in the output table. If the joined table is used in further computations and some of its values are eventually published, this might be enough to deduce some personal information.

The extra shuffle step at the end of the protocol destroys part of this link, but some information is still leaked. One can still observe how many keys from one table are matched against a key in another table. More formally, let the *occurrence signature* for a key be the number of occurrences in each table. We can then formulate the following security claim.

Theorem 5. *If the secure multi-party computation framework satisfies the same assumption as in Theorem 2, the* PRPJOIN *protocol leaks only the final size of the database and occurrence signatures without the corresponding key values.*

Proof sketch. The proof is similar to the proof idea used for Theorem 2 and we use the same notation here. First, the simulator forwards all input shares to the trusted third party and gets back the joined table together with the occurrence signatures. The simulator learns the number of output rows m from the output table. Additionally, it learns which keys from the first table matched with how many keys from the second table by looking at the occurrence signatures. Next, the simulator generates the encrypted columns $\boldsymbol{y_1}$ and $\boldsymbol{y_2}$ by choosing random values from the proper domain and setting some of them equal so that the occurrence signatures hold. These vectors together with the faked shuffled databases (shares of zeros) are sent to the adversary. Finally, the simulator can also forward the real output table to the adversary. Note that, unlike for Theorem 2, the simulator does not have to align the output shares with the encrypted vectors as this relation is destroyed by the final shuffling step. □

If leaking the *occurrence signatures* is acceptable for a given application, then the PRPJOIN protocol with the additional shuffle step at the end can also be used with non-unique keys. Unfortunately, in some applications, the number of occurrences for a key might already leak sensitive data. For example, in the presence of auxiliary information, the number of visits to a doctor may uniquely identify a person.

In many cases, the number of colliding keys may be small in practice. Moreover, the upper bound ℓ to the number any key may occur in a table may be publicly known. Then it is possible to hide colliding key values by obliviously adding fake rows to input tables such that each key occurs exactly ℓ times. This size unification idea is given as Algorithm 4.

The mask-and-mix phase uses two shared binary vectors $\boldsymbol{b}$ and $\boldsymbol{c}$ to keep track of fake rows. The former is only used inside the protocol while the latter is added to the output table. In the output table of the protocol, $\boldsymbol{c}$ can be used to filter out fake rows: $c_i = 1$ for either one or both input tables, depending on the used join method (inner, left or right outer join). The oblivious sorting step has to lexicographically order the rows according to the predicate

$$(k_i, b_i) \preceq (k_j, b_j) \qquad \Leftrightarrow \qquad k_i \leq k_j \wedge b_i \leq b_j$$

using, e.g. radix sort or any other compatible oblivious sorting algorithm from Chapter 6. For step 5, we need a secret-shared counter n and the update rule:

$$n = \begin{cases} n+1, & \text{if } k_{i-1} = k_i \ , \\ 1, & \text{otherwise} \ , \end{cases} \qquad b_i = \begin{cases} 0, & \text{if } n \leq \ell \ , \\ 1, & \text{otherwise} \ . \end{cases}$$

Algorithm 4: Size unification protocol SUNIF.

Data: Secret-shared table with key column $\boldsymbol{k}$
Result: Secret-shared table with exactly ℓ copies of each key value

Mask-and-mix phase:

1 Add two secret-shared bit columns $\boldsymbol{b}$ and $\boldsymbol{c}$ for marking fake rows. Initially they are filled with shares of zeros.
2 For each row, add $\ell - 1$ fake rows with the same key and flags $b_i = 1$ and $c_i = 1$
3 Apply oblivious shuffle to the database table to hide the content

Sort-and-filtering phase:

4 Sort all rows according to the key value and the flag pairs so that fake entries with same keys are always after the non-faked ones
5 Linearly scan key-flag pairs (k_i, b_i) and set flag b_i to zero if less than ℓ rows with the same key precede the current row
6 Apply oblivious shuffle to the table and open the index column $\boldsymbol{b}$
7 Delete all rows, which are still marked as fake, i.e., $b_i = 1$

The counter n shows how many elements with the value of k_i there are up to the current i-th position. The values in $\boldsymbol{b}$ show if the number of consecutive equal elements in $\boldsymbol{k}$ is greater than our maximal allowed number of colliding keys ℓ. An example of counter n and vector $\boldsymbol{b}$ for a sorted vector $\boldsymbol{k}$ using $\ell = 2$ is shown below:

k :	1	1	2	2	2	3	4	4	4
n :	1	2	1	2	3	1	1	2	3
b :	0	0	0	0	1	0	0	0	1

As seen, instead of a counter, we can also encode n as a vector. Consequently, one possible optimisation for SUNIF is to construct the whole vector $\boldsymbol{b}$ in parallel. This can be done obliviously by comparing each value in $\boldsymbol{k}$ with ℓ previous values. This requires ℓ comparisons, which can be done in parallel for each element of $\boldsymbol{k}$, and $\ell - 1$ logical conjunctions, that can be done using a binary tree approach. As a result, the round complexity of computing $\boldsymbol{b}$ is $\mathcal{O}(\log \ell)$.

Theorem 6. *Assume that a secure multi-party computation framework provides universally composable protocols for arithmetic operations and protocols for oblivious sort and shuffle and the maximal number of keys with the same value is below ℓ. Then the* SUNIF *protocol produces a database where each key appears exactly ℓ times and leaks only the number of distinct keys.*

Proof sketch. For the proof, note that the third and sixth step destroy all information about the location of fake rows and the number of ones revealed at the last

step is determined only by the database size and the number of distinct keys. The simulation construction is analogous to the previous proofs. □

Instead of removing the rows with unused keys ($b_i = 1$) in the last step of Algorithm 4, we can assign them new unused keys such that each key occurs exactly ℓ times. For example, one way to accomplish that is to replace all of the keys with subsequent values from a global counter d. We know that for each unique key value, there are exactly ℓ non-fake rows. Hence, each unique key value of pairs $(k_i, b_i = 0)$ is replaced by a subsequent value from the counter d. For all the fake rows ($b = 1$), we can assign $k_i = d$ and increment d after ℓ assignments. As initially we added $\ell - 1$ new rows for each existing database row, then the total number of rows is divisible by ℓ and thus each value of d is assigned exactly ℓ times. Let us denote this protocol as SUNIF$^+$.

Corollary 1. *Let the number of key occurrences be bounded by a public constant ℓ. Then by combining* SUNIF$^+$ *and* PRPJOIN *protocol we get an equi-join protocol that leaks no information besides the number of rows in the output table.*

Proof sketch. The PRPJOIN protocol leaks no information as each key occurs exactly ℓ times. However, the resulting table after the join phase contains rows that consist partly of fake entries. Then, we can use all fake flags c_i to compute whether the row is valid or not in the post processing step and eliminate invalid database entries. Again, this leaks no information as the number of invalid entries is determined by the number of rows in the output table. □

The combination of SUNIF$^+$ and PRPJOIN has the complexity of roughly $\Theta(\ell m \log m)$ and is thus much faster than the baseline solution NAIVEJOIN with quadratic complexity.

5.2.4 Related work

As a very practical task, privacy-preserving database linking task has been studied before. Unfortunately, none of the solutions are applicable in our composable model where both the input and output tables are secret-shared. One of the pioneering works in privacy-preserving data-mining described how exponentiation can be used in order to compute equi-join of data tables [3]. However, the idea worked in a two-party setting and the resulting table was published.

Freedman et al. showed how to implement secure set intersection with oblivious polynomial evaluation and balanced hashing [58]. Unfortunately, their ideas are also not directly applicable here as their two-party protocol assumes that key values are a local input, whereas in our setting the data is secret-shared. Moreover, oblivious polynomial evaluation allows to check if a given element belongs to a set. However, in PRPJOIN similar tests are actually done on published data.

The idea of Hazay and Lindell [73] of using pseudo-random permutation to hide the key values and perform set intersection on published ciphertexts is the closest to us. However, in their two-party protocol one of the parties learns the intersection.

Perhaps the most compatible solution to ours is given by Hadavi et al. in [70], where they use a searchable secret sharing scheme that allows to translate user queries into queries over shared values by the user himself.

5.3 Oblivious AES

Advanced Encryption Standard (AES) is a symmetric block cipher algorithm approved by the National Institute of Standards and Technology (NIST) [106]. AES works on 128-bit blocks of data, i.e. both its plaintext and ciphertext blocks are 128 bit long. The NIST FIPS-197 standard states that AES can be used with key lengths of 128, 192 or 256 bits. In this work, we will use AES-128, which denotes AES with 128-bit keys. However, the results here also apply for other key lengths as it only affects the number of rounds in the key derivation and encryption phases.

In this section, we will describe and implement AES that works on secret-shared data. In this setting, the plaintext, key and ciphertext are all bitwise secret-shared and not known to any single party.

The main motivation for implementing such oblivious AES is to use it as a pseudo-random permutation function to encrypt the key values in the privacy-preserving database join protocol PRPJOIN. Moreover, a similar idea of using oblivious AES as a pseudo-random permutation is used to implement privacy-preserving *group by* operation in [88]. Both of these operations are used in the privacy-preserving extract, transform, load (ETL) step of the statistical study described in Chapter 7.

However, AES has other application areas besides the obvious encryption. For example, in counter mode, AES can be used as a pseudo-random number generator [10]. Moreover, implementing AES on secret-shared data is interesting in its own right as it is widely adopted as a benchmarking test [112, 49, 77, 95, 50, 107, 78, 92, 63, 90, 64]. Hence, it allows us to compare our secure multi-party computation platform with other implementations. Finally, we can think of oblivious AES as a trusted hardware on a cloud environment that reduces the security requirements needed for symmetric key management [49].

5.3.1 Oblivious implementation of the S-box

AES algorithm overview

We have designed our oblivious AES protocol by following the NIST FIPS-197 standard [106]. The main part of the AES cipher is the round function that consists of four byte-oriented transformations (see Algorithm 5). The number of rounds Nr is determined by the key length. For example, using 128-bit keys, a total of 10 rounds are executed, with the final round missing the `MixColumns()` operation. Before the actual cipher rounds, the encryption key is expanded into round keys using similar byte transformation operations.

Algorithm 5: AES cipher structure [106].

Data: 16-byte state and round keys ($4 \cdot (\mathsf{Nr} + 1)$ bytes)
Result: 16-byte state

AddRoundKey(state, roundKey[0])
foreach $r \in \{1, \ldots, \mathsf{Nr} - 1\}$ **do**
    SubBytes(state)
    ShiftRows(state)
    MixColumns(state)
    AddRoundKey(state, roundKey[r])
end
SubBytes(state)
ShiftRows(state)
AddRoundKey(state, roundKey[Nr])

Each round starts by `SubBytes()` operation that applies a one-for-one substitution table (S-box) independently for each byte of the 16-byte cipher *state*. Applying the S-box (see Figure 5.1) is the only non-linear operation in the AES algorithm and it is also used by the `SubWord()` operation in the key expansion phase that applies it to each byte in its 4-byte input.

The other three round transformations use linear operations on the cipher state. `ShiftRows()` shifts bytes in the last three rows of a (4×4)-byte state in a cyclic manner using different offsets. `MixColumns()` mixes the columns of the state using linear operations over GF(2^8). Finally, `AddRoundKey()` combines the round key into the current state using XOR operation.

All linear operations can be carried out independently by each computation party on their shares. Hence, applying the S-box substitution table is the only operation that requires network communication and determines the complexity of the oblivious implementation.

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figure 5.1: S-box: substitution values for the byte xy (in hexadecimal format) [106].

S-box by oblivious selection

Basically, S-box can be thought of as a 256-element lookup table containing a substitution value for every possible byte value. In our setting, the selection index is secret-shared. Hence, we need an oblivious array selection that can be achieved by combining techniques described in [98].

Given a byte value x that we want to substitute, we have to construct a zero-one index vector $\boldsymbol{z}$ that has a value 1 at position z_x and all of the other positions are filled with zeros. Let $x_7x_6\dots x_0$ be the bit-representation of the input value x and $i_7i_6\dots i_0$ be the bit-representation of index value $i = 0,\dots,255$. Then, each position of the index vector can be expressed as a conjunction of equality predicates $z_i = [x_7 = i_7] \wedge \dots \wedge [x_0 = i_0]$ and the corresponding shares can be computed by evaluating multinomials

$$z_i = (x_7 \oplus i_7 \oplus 1)\cdots(x_0 \oplus i_0 \oplus 1) \ .$$

Hence, the first value in the index vector is $z_0 = (1 \oplus x_7)(1 \oplus x_6)\dots(1 \oplus x_0)$, the second value is $z_1 = (1 \oplus x_7)(1 \oplus x_6)\dots(1 \oplus x_1)x_0$, etc.

Note that each of the 256 multinomials z_i is of degree 8 and thus, in total, this requires 1792 multiplications over $\mathbb{F}_2$. To optimise the number of communication rounds, we group the terms $b_{ij} = x_j \oplus i_j \oplus 1$ by the j-th bit in each index position:

$$\boldsymbol{b}_7 = (b_{0,7},\dots,b_{255,7}),\dots,\boldsymbol{b}_0 = (b_{0,0},\dots,b_{255,0}) \ ,$$

and using vectorised bitwise multiplication evaluate these in a row. Doing this sequentially, one bit position at a time, would take seven multiplication rounds.

However, using a tree-style evaluation strategy by folding the vector into two after each round, we can accomplish this in three multiplication rounds:

$$\boldsymbol{z} = ((\boldsymbol{b}_7 \cdot \boldsymbol{b}_6) \cdot (\boldsymbol{b}_5 \cdot \boldsymbol{b}_4)) \cdot ((\boldsymbol{b}_3 \cdot \boldsymbol{b}_2) \cdot (\boldsymbol{b}_1 \cdot \boldsymbol{b}_0)) \ .$$

The communication complexity of creating the index vector can be further decreased by using the recursive nature of the index vector as proposed by Launchbury *et al.* [95]. Instead of using a top-down approach of folding a large vector together each round, it is possible to use a bottom-up method of recursively extracting and replicating smaller vectors to combine the same index vector. For reference, we also implemented the idea proposed in [95].

As a second step, the index vector is used to obliviously choose the right value from the S-box array. However, since the constructed index vector is binary, each bit of the output value has to be calculated separately. This can be done by computing a scalar product of the index vector $\boldsymbol{z}$ and the S-box array $\boldsymbol{y}$. Let $\boldsymbol{y}_j = (y_{0,j}, \ldots, y_{255,j})$ denote the vector of j-th bits of $\boldsymbol{y}$. Then, the j-th bit of the S-box output, f_j, can be computed as

$$f_j = \langle \boldsymbol{z}, \boldsymbol{y}_j \rangle = \sum_{i=0}^{255} z_i \ y_{ij}$$

over $\mathbb{F}_2$. As the output table $\boldsymbol{y}$ is public, this step can be done locally by each computation party and does not introduce any extra communication complexity.

S-box by circuit evaluation

The S-box evaluation by constructing an oblivious selection index is bound to give us sub-optimal performance results as we need to "touch" all of the elements in the output table although we are interested in just one.

Alternatively, we can solve this problem by using secure computation techniques based on branching programs [80]. This is done by converting the expression for f_j to a binary decision diagram $\mathcal{B}$ with minimal number of decision nodes. Diagram $\mathcal{B}$ is then evaluated bottom-up by a corresponding arithmetic circuit. Let c be the number of decision nodes and d be the longest path in $\mathcal{B}$. Then, as oblivious evaluation of each decision nodes takes two secure multiplications, the resulting circuit evaluation protocol consists of $2c$ secure multiplications over $\mathbb{F}_2$ that are arranged into d parallel rounds. Hence, the shape of $\mathcal{B}$ determines the performance of the evaluation.

Luckily, AES S-box evaluation circuits are thoroughly studied by the hardware optimisation researchers. In this work, we use the designs by Boyar and Peralta [35, 36]. However, their motivation is minimising the total number of gates in the circuit and it's overall depth. For us, it is most important that the

circuit contains as few AND (multiplication) gates as possible as XOR gates can be evaluated locally as we use bitwise shared values. Hence, their newer circuit design with 128 gates may not be the most suitable for us as it contains 34 AND gates and has a multiplicative depth of 4. Their previous work [35] with 32 AND gates and multiplicative depth of 6 may be more suitable when we do many S-box evaluations in parallel as is the case in privacy-preserving database join operation[2].

Since extended versions of both mentioned articles contain straight-line C-like descriptions of the circuits, it is possible to implement the corresponding secure evaluation protocols. However, these circuits are designed to work with individual bits whereas the smallest data type sent over a network is a byte. Hence, to avoid wasting 7 bits for each bit of useful data, we pack several S-box evaluations into parallel executions. This is trivial to accomplish for `SubBytes()` as it works with 16-byte states and thus executes 16 S-boxes in parallel anyway. In `SubWord()` that does only 4 S-box evaluations in parallel, group whole two AES blocks together if possible.

5.3.2 Security analysis

All of the aforementioned methods for evaluating the AES S-box are arithmetic circuits using only multiplication and addition gates. Hence, it is straight-forward to prove the following result.

Theorem 7. *If a secure multi-party computation framework provides universally composable protocols for bitwise addition, bitwise multiplication and bit decomposition, then all three AES S-box implementations are universally composable. Any universally composable AES S-box implementation gives rise to a universally composable SMC protocol for the AES block cipher.*

Proof sketch. Let us consider the hybrid model where each protocol instance is replaced with an ideal implementation that gathers all input shares and distributes shares of correct outputs. Then it is easy to see that output shares of any arithmetic circuit are correctly computed and coincide with the case where the trusted third party outputs just the shares of the circuit output. Due to the properties of secret sharing, all shares of intermediate results received by the adversarial coalition leak no information and can be easily simulated. Hence, the simulation construction in the hybrid model is straightforward and the security follows from the universal composability of addition and multiplication. □

[2]After publishing our results in [96], Peralta has updated his web page (`http://cs-www.cs.yale.edu/homes/peralta/CircuitStuff/CMT.html`) with a new AES S-box circuit with 113 gates. We do not consider this construction here separately, as it also consists of 32 AND gates.

Note that Theorem 7 holds not only for the honest-but-curious security model of SHAREMIND but also for any active model supporting verifiable bitwise addition and multiplication operations on shares. There are two principal ways to accomplish this.

First, we can embed elements of $\mathbb{F}_2$ into a larger finite field $\mathbb{F}_{2^t}$ with an extension element α and use a verifiable secret sharing scheme supporting multiplication over $\mathbb{F}_{2^t}$. Then it is possible to use universally composable bit decomposition [47] that splits a secret $x \in \mathbb{F}_{2^t}$ into a vector of shared secrets $x_{t-1}, \ldots, x_0$ such that $x = x_{t-1}\alpha^{t-1} + \cdots x_1\alpha + x_0$. Consequently, all requirements of Theorem 7 are satisfied and we have a secure AES implementation in an active model. However, due to embedding, all of the shares are now longer.

Alternatively, we can use oblivious message authentication codes [51] to guarantee the integrity of individual bits without extending the shares themselves. Unfortunately, the message authentication code itself can be much longer than the share of a bit that it protects. The solution is to use a single authentication code for larger groups of bits. This works well in our model as, in most applications, we evaluate many circuits in parallel.

5.3.3 Performance tweaks

The data to be encrypted is rarely as short as only a single AES block length of 128 bits. Hence, in practical applications where the encryption of subsequent blocks do not depend on each other (e.g. Electronic Codebook (ECB) and Counter (CTR) modes of operation) it makes sense to encrypt several data blocks in parallel. In SHAREMIND, this is naturally possible as protocols work on vectors of share values. Hence, vectorised AES operation takes a vector of plaintext shares together with a vector of key shares as input and outputs the vector of ciphertext shares.

Moreover, in many applications the same key is used to encrypt several blocks of data, for example in the case of PRPJOIN. Then the key expansion routine in AES has to be evaluated only once and the array of secret-shared round keys stored for later use. Such separation of pre-processing (key expansion) and online (encryption) phases decreases the amortised cost for the oblivious AES evaluation.

5.4 Benchmarking results

5.4.1 Test setup

We carried out performance tests for both the privacy-preserving database join operation and oblivious AES on a cluster of three nodes, each having its own SHAREMIND installation. Each cluster node had a 12-core 3 GHz CPU with *Hyper-Threading* and 48 GB of memory. Cluster nodes were connected by a

1 Gbps local area network and communication channels between them were encrypted using 256-bit elliptic curve key agreement and the ChaCha stream cipher [16, 109] used in the RakNet networking library that powered SHAREMIND 2. At the time of writing this thesis, the ChaCha stream cipher has been adopted in many security-related applications and attacks published against it are considered infeasible in practice [7, 81, 117].

The performance results presented here are from 2012 when oblivious AES and privacy-preserving database join were first implemented on SHAREMIND 2. For oblivious AES, we also give a comparison on how its performance have changed in the newer SHAREMIND 3.

5.4.2 AES performance

We implemented all of the four versions of oblivious S-box evaluations described in Section 5.3.1. We measured the running time separately in two situations: evaluating a single `SubBytes()` function and evaluating 4096 `SubBytes()` operations in parallel. The first case gives us the running time profile for the case where various delays play a significant role, whereas the second case is dependent on communication complexity. The results together with the most important theoretical properties are shown in Table 5.1. The OBSEL protocol stands for the top-down approach to construct the oblivious index vector, while LDDAM uses the bottom-up approach described in [95]. The oblivious S-box evaluation based on circuits designed by Boyar and Peralta are shown as BCIRC-1 and BCIRC-2. The multiplicative complexity stands for the total number of multiplication operations over $\mathbb{F}_2$. As some of these operations are done in parallel, then multiplicative depth denotes the number of sequential multiplication rounds.

Protocol	Mult. depth	Running time (1 evaluation)	Mult. compl.	Running time (4096 evaluations)	Ratio
OBSEL	3	32.5 ms	1792	9051 ms	5.05
LDDAM	3	**31.1** ms	304	1109 ms	3.65
BCIRC-1	6	69.6 ms	32	148 ms	4.63
BCIRC-2	4	40.8 ms	34	**127** ms	3.74

Table 5.1: Performance results of AES `SubBytes()` with various S-box evaluation algorithms. The ratio in the last column is amortised running time divided by multiplicative complexity and shows the amortised cost added by a single AND gate. Similarly, from multiplicative depth we get that each communication round adds 10–12 ms in a single operation mode.

As all of the multiplications are carried out over $\mathbb{F}_2$, we do not have to compensate for various input lengths. Hence, multiplicative depth and complexity are good candidates for estimating the real-life performance of such protocols.

Next, we measured the amortised cost of AES encryption function with precomputed round keys. Figure 5.2 shows that different S-box evaluation methods yield different network saturation points where further parallelisation does not decrease the amortised running time further. The OBSEL protocol constructs huge initial vectors for the index vector and thus uses a lot of bandwidth even with few blocks of inputs. The LDDAM protocol is much more efficient in building the same index vector and is actually the best protocol in case only a few AES blocks are processed in parallel. When encrypting about 100–10 000 blocks in parallel, the newest circuit design BCIRC-2 with the lowest multiplicative depth outperforms the previous circuit. After that, the two extra multiplication gates start to affect the performance and the BCIRC-1 protocol with smallest number of AND gates slightly outperforms the other.

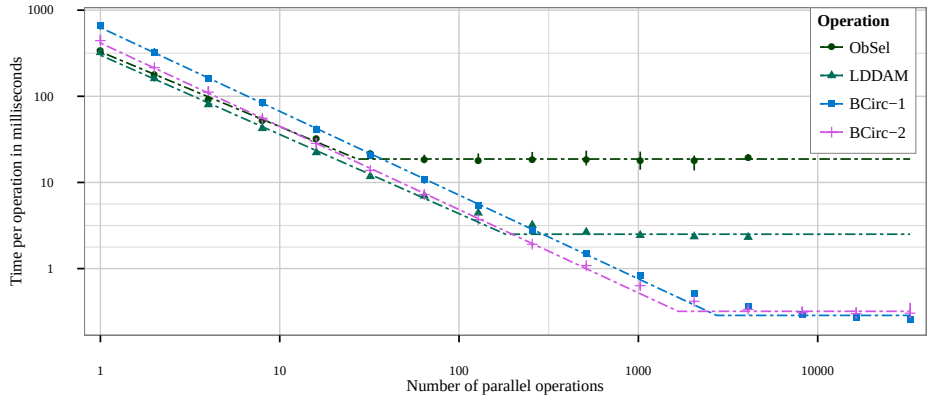


Figure 5.2: Performance of AES evaluation protocols using precomputed round keys.

Finally, for completeness, we studied the running time of oblivious AES with and without key scheduling. As before, we are interested in two cases: the running time of a single invocation and the amortised cost. The results are shown in Table 5.2, where mode I stands for the case with key expansion and mode II uses the pre-expanded key. As expected, key expansion counts for about half of the time when encrypting a single block of data as each encryption round requires a key expansion round. As a further optimisation, it is possible to compute the round key for the next encryption round in parallel with the current encryption round, decreasing this ratio to about 1.2, given that network bandwidth is not the bottleneck. For the amortised cost, the theoretical speedup should be 1.25 as in each round there are 20 S-box evaluations in mode I (`SubWord()` and `SubBytes()`)

and 16 S-box evaluations in mode II (only `SubBytes()`). The difference in the actual ratio suggests the existence of other bottlenecks in our implementation.

	Single operation			Amortised cost		
	Mode I	Mode II	Ratio	Mode I	Mode II	Ratio
OBSEL	682 ms	343 ms	1.99	20.34 ms	18.69 ms	1.09
LDDAM	**652** ms	**323** ms	2.02	4.16 ms	2.51 ms	1.66
BCIRC-1	1329 ms	664 ms	2.00	0.48 ms	**0.29** ms	1.68
BCIRC-2	890 ms	443 ms	2.01	**0.37** ms	0.32 ms	1.17

Table 5.2: Performance results for various AES evaluation algorithms. In both cases, ratio shows the difference between mode I and mode II.

Figure 5.3 shows benchmarking results for the oblivious AES with BCIRC-1 circuit for the S-box evaluation with pre-computed keys from the beginning of 2015 compared to the initial results published in [96]. As seen, a complete re-design of the SHAREMIND software with Boost ASIO as a networking layer can boost performance up to an order of magnitude.

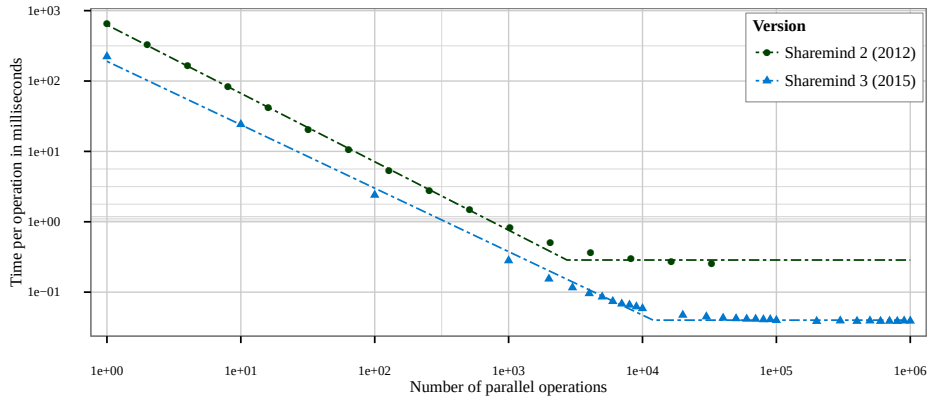


Figure 5.3: Comparison of time-per-operation for AES-128 with BCIRC-1 S-box and pre-expanded key on SHAREMIND 2 (2012) and SHAREMIND 3.

Table 5.3 compares our AES-128 performance results with the results reported by others. To keep the results comparable, we have only included implementations based on linear secret sharing, both in passive and active security settings. A more thorough overview of AES-128 performance results is given in [50]. In [96], we have reported the time of LDDAM protocol for the single operation case and BCIRC-1 for the amortised cost as it it possible to switch between different S-box implementations at run-time based in the length of input vector. Although we

use the same idea, we cannot fully explain the 20-times difference between our implementation of LDDAM and the implementation by its authors in [95]. Possibly, it comes from the difference in benchmarking setups and the way different implementations are tuned. Our implementation is optimised for the amortised case. However, as mentioned above, we have since re-designed the SHAREMIND platform and the updated running times (both with BCIRC-1 S-box evaluation) are considerably better. Similarly, the implementation of [90] uses the same algorithms as in [50] and obtains an order of magnitude better run-time by just scheduling SMC tasks more efficiently. Both of these implementations are based on the SPDZ protocols [51] that use pre-computed Beaver triples [13] for multiplication. As shown in Section 2.2.3, using Beaver triples for multiplication replaces replicating shares with a declassification of two values. However, in both cases, multiplication is still a single round protocol. Although only online computation time is included in the table for these implementations, the result of Keller et al. demonstrate the importance of implementation details.

At the end of Table 5.3, we have included recent performance results of AES-128 evaluation using 2-party garbled circuit approach (Frederiksen et al. [57]) and fully homomorphic encryption (Gentry et al. [64]) as reference. All results shown, except the first one and the one using garbled circuits, make use of the pre-expanded key (mode II).

Authors	Ref.	Parties	Security	Single op.	Amort. cost
Damgård and Keller	[49]	3	passive	2000 ms	— ms
Launchbury et al.	[95]	3	passive	14.28 ms	3.10 ms
Laur et al.	[96]	3	passive	323 ms	0.29 ms
Laur et al.	2015	3	passive	223 ms	**0.04** ms
Damgård et al.	[50]	2–4	active	250 ms	240 ms
Keller et al.	[90]	2	active	**12** ms	1 ms
Frederiksen et al.	[57]	2	active	810 ms	— ms
Gentry et al.	[64]	2	passive	— ms	2000 ms

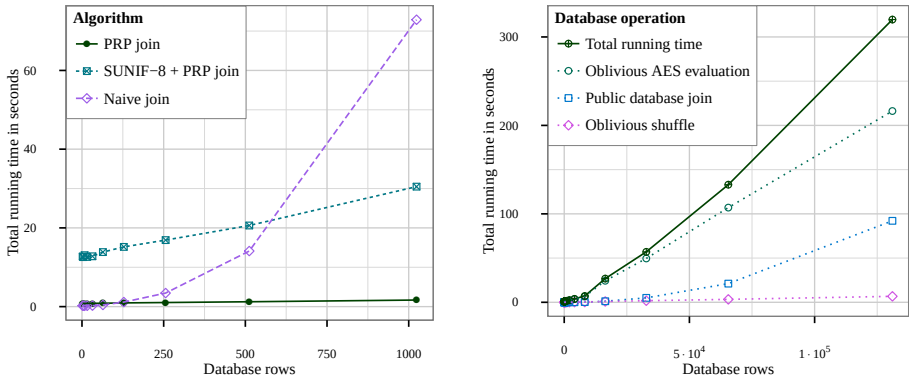
Table 5.3: Comparison of various secure AES-128 implementations based on linear secret sharing. For reference, the last two are recent results obtained with garbled circuits and fully homomorphic encryption, respectively.

5.4.3 Secure database join

We measured the performance of privacy-preserving database join operation by testing how much time it takes to join two database tables, each containing five 32-bit columns, including a single key column. Both tables had the same number

of rows and each key value in one table had exactly one corresponding key value in the other table. For the oblivious AES implementation, we chose the one based on BCIRC-1 as it yields the lowest amortised cost on thousands of input blocks.

The profile of PRPJOIN on Figure 5.4b shows that it scales nearly linearly with database rows. More precisely, the only non-linear component is the actual join operation on published ciphertexts that is known to have the complexity $\Theta(m \log m)$. The exact balance between oblivious AES and oblivious database shuffle [98] operations depends on the number of columns in the input tables as the latter operation depends linearly on it. To put it into context, for input tables with 100 000 rows each, it would take 180 columns in both tables for the oblivious shuffling operation to take as much time as the oblivious AES.



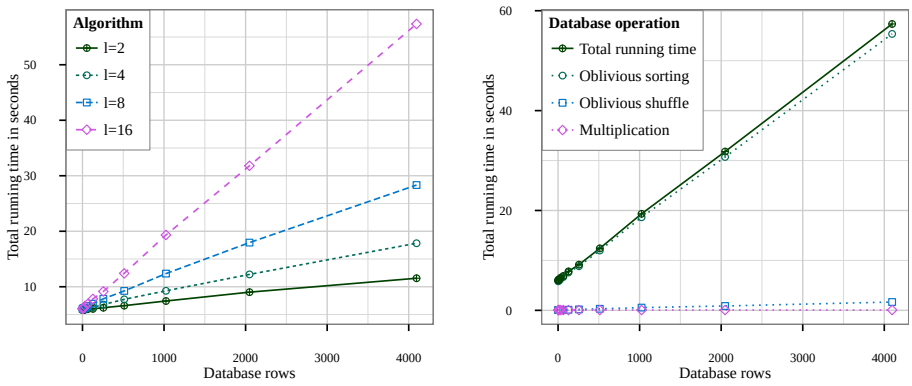
(a) Different join protocols.

(b) Breakdown of PRPJOIN.

Figure 5.4: Benchmarking results for the oblivious database join operation.

Figure 5.4a shows that with its quadratic complexity, NAIVEJOIN is not usable in practical applications. For input tables with 1000 rows, NAIVEJOIN has to perform a million equality comparisons and shuffle a database with 1 000 000 rows. To demonstrate the complexity-leakage tradeoff, Figure 5.4a also shows the running time of a combined protocol with SUNIF preprocessing that leaks nothing if there are no more than eight collisions per key value in the input tables. For small database sizes, it is very slow due to the expensive oblivious sorting, however, for practical database sizes it clearly outperforms NAIVEJOIN. The initial slowdown depend on the maximum numbers of allowed collisions per key ℓ.

Secondly, we also measured the running-time of the protocols with non-unique keys. As size unification protocols SUNIF and SUNIF$^+$ are independent from PRPJOIN and can be considered as pre-processing, we tested the performance of SUNIF separately. For testing, we used a single input table that was otherwise similar to the one used in PRPJOIN tests by consisting of five 32-bit columns in-



(a) Performance with different ℓ values.

(b) Breakdown of SUNIF, $\ell = 16$.

Figure 5.5: Benchmarking results for the SUNIF operation.

cluding a single key column and a varying number of rows. SUNIF also introduces a new variable ℓ – the maximum number of collisions for a key value.

As Figure 5.5a shows, SUNIF scales almost linearly with the number of database table rows as expected. Figure 5.5b shows that most of the time in SUNIF is spent on oblivious sorting. In our implementation of the sort-and-filter phase, we used oblivious radix sort (see Section 6.2.3) that scales near-linearly with the database size and is thus a good candidate even for large databases.

CHAPTER 6

OBLIVIOUS SORTING

6.1 Introduction

Sorting is an important primitive. Besides its obvious use, sorting is a necessary sub-operation in many algorithms and statistical tests, e.g. finding ranked elements (top-k, quantiles) or creating subgroups for aggregation methods.

An oblivious sorting algorithm implementation takes a vector of secret-shared data as input and outputs a secret-shared vector containing the same values in sorted order. To implement an oblivious sorting algorithm, the algorithm itself should be *data-independent*. This means that intermediate values nor output should leak any information about the input. Moreover, for a given input vector length, the algorithm runtime should not depend on the input values. Ideally, nothing besides the length of the input and output should be leaked. It is possible to also hide the vector length by stretching the vector to some known constant length by adding dummy values. However, this wastes computation resources and is rarely needed in practice. Hence, we consider algorithms that leak the input and output size to still be data-independent.

In principle, there are two ways to implement a privacy-preserving sorting algorithm. First, we can take a sorting network that is data-independent by design and use secret sharing and secure multi-party computation to hide the data values [87, 126]. Alternatively, we can take a data-dependent sorting algorithm as a basis and use oblivious shuffling to randomly permute the data vector before sorting. Hamada *et al.* [72] use this idea to propose a blueprint on how to turn any comparison-based sorting algorithm into an oblivious one. However, this construction leaks the number of equal elements in the vector. We will go over this idea in the following section.

6.2 Oblivious sorting algorithms

6.2.1 Constructions based on oblivious shuffling

It is possible to run any comparison-based sorting algorithm on a secret-shared input vector $[\![x_1]\!], \ldots, [\![x_n]\!]$ and use SMC protocols to compute comparisons $[\![g_{i,j}]\!] = [\![x_i]\!] < [\![x_j]\!]$. However, as the algorithm needs to make a decision based on such comparison result, these $g_{i,j}$ must be published and tracking those throughout the algorithm leaks the permutation applied during the sorting process. To mitigate this, the input vector must be obliviously shuffled before sorting [72]. This construction yields a secure randomised sorting algorithm with data-independent running time, provided that there are no equal elements in the input vector.

Theorem 8. *Assume that a secure multi-party computation framework provides universally composable protocols for comparison and oblivious shuffle and that there are no equal elements in a secret-shares input vector. Then obliviously shuffling the input vector and declassifying the comparison results made by a correct sorting algorithm yields a secure and correct sorting algorithm.*

Proof sketch. We give the proof idea in two parts. In the first part, we show how to simulate the trace of comparison results declassified by the sorting algorithm. In the second part, we extend the construction given in the first part to the full-fledged simulator.

PART I. Let $\boldsymbol{x}$ be an n-element vector of unique values that we want to sort. For each pair of values $i, j \in \{1, \ldots, n\}$, we can define a tertiary predicate that defines an order relation of those values:

$$b_{i,j} = \begin{cases} 0, & \text{if } x_i = x_j \ , \\ 1, & \text{if } x_i < x_j \ , \\ -1, & \text{otherwise} \ . \end{cases}$$

All possible values of this predicate for a given input vector $\boldsymbol{x}$ can be illustrated by an $n \times n$ table. For example, Figure 6.1 shows such tables for all possible 6 input permutations of a 3-element vector.

Any data-dependent comparison-based sorting algorithm starts by looking at a value in a specific cell from this table and moves to the next cell depending on the obtained value. The number of accessed cells depends on the values in those cells. Figure 6.1 shows the cells accessed by quicksort algorithm.

In the case of a secret-shared input vector $[\![\boldsymbol{x}]\!]$, we start by applying an oblivious shuffle on the input vector. Therefore, the sorting algorithm starts with one of the randomly chosen tables out of all $n!$ possible tables. As a result, the comparisons $b_{i,j}$ done by the algorithm are independent of the actual values in $[\![\boldsymbol{x}]\!]$.

	x_1	x_2	x_3
x_1	0	(1)	(1)
x_2	-1	0	(1)
x_3	-1	-1	0

	x_1	x_3	x_2
x_1	0	1	(1)
x_3	-1	0	(-1)
x_2	-1	1	0

	x_2	x_1	x_3
x_2	0	(-1)	(1)
x_1	1	0	(1)
x_3	-1	-1	0

	x_2	x_3	x_1
x_2	0	1	(-1)
x_3	-1	0	(-1)
x_1	1	1	0

	x_3	x_1	x_2
x_3	0	-1	(-1)
x_1	1	0	(1)
x_2	1	-1	0

	x_3	x_2	x_1
x_3	0	-1	(-1)
x_2	1	0	(-1)
x_1	1	1	0

Figure 6.1: Predicate b values for all possible permutations of a 3-element vector $\boldsymbol{x}$, where $x_1 < x_2 < x_3$. Circled cells indicate b values used by the quicksort algorithm. Cells marked with thicker circle are checked twice.

Hence, such construction is data-independent, although the running time is not constant for a given vector length.

PART II. As the oblivious sorting algorithm starts with oblivious shuffle, our simulator $\mathcal{S}$ uses the input extraction to obtain adversary's input shares. The simulator forwards these inputs to the trusted third party and gets back shares for the sorted vector in the correct order. Next, $\mathcal{S}$ randomly permutes a vector $(1, \ldots, n)$ to fake matrix $(b_{i,j})$. Note that this uniquely determines how the inputs are permuted by the sorting algorithm. By reversing this permutation, we know how to reorder shares returned by the trusted third party. The simulator $\mathcal{S}$ can use this order for output equivocation of the shuffle simulator. After that we must simulate outputs of comparison protocols according to the fake matrix $(b_{i,j})$ fixed before. □

It is possible to guarantee the uniqueness of values by applying a pre-processing conversion on the input vector (see Section 6.3).

Many secure multi-party computation implementations have efficient SIMD-style (*single instruction, multiple data*) operations. Hence, highly parallelisable algorithms with higher computation complexity may, for small inputs, outperform algorithms with lower computation complexity but more rounds. To demonstrate this, we propose an oblivious sorting algorithm that is based on oblivious shuffling and vectorised comparison operations. The NAIVECOMPSORT (see Algorithm 6) first obliviously shuffles the input vector and then compares each vector element with each other element in the vector in parallel. The sorted output vector is obtained by rearranging the elements according to the declassified comparison results. This algorithm always works in the worst case time of $\mathcal{O}(n^2)$ and its running time is, therefore, data-independent.

Algorithm 6: NAIVECOMPSORT

Data: Input array $[\![\boldsymbol{x}]\!] \in \mathbb{Z}_{2^k}^n$
Result: Sorted array $[\![\boldsymbol{x'}]\!]$
1 Let $[\![\boldsymbol{x}]\!] = \mathsf{Shuffle}([\![\boldsymbol{x}]\!])$
2 Compute in parallel values $[\![g_{i,j}]\!] = [\![x_i]\!] \leq [\![x_j]\!]$ for $1 \leq i < j \leq n$.
3 Declassify the values $[\![g_{i,j}]\!]$ and sort $[\![\boldsymbol{x}]\!]$ according to them, obtaining $[\![\boldsymbol{x'}]\!]$

6.2.2 Sorting networks

A sorting network consists of several stages of compare-and-exchange functions. A compare-and-exchange (CompEx) function is a function that takes two values as input, compares the values and swaps them if necessary. For example, a CompEx function that outputs the values in ascending order can be described as following:

$$\mathsf{CompEx}(x, y) = (\mathsf{Min}(x, y), \mathsf{Max}(x, y)) \ .$$

Input values for the CompEx function can be addressed by their index in the initial input vector. Hence, an m-stage sorting network may be written as an array $\mathcal{N} = (\mathcal{L}_1, \ldots, \mathcal{L}_m)$, where each stage $\mathcal{L}_i = (\mathbb{N} \times \mathbb{N})^{\ell_i}$ consists of ℓ_i pairs of indices. Each index pair (l, r) stands for an operation $\mathsf{CompEx}(x_l, x_r)$. See Figure 6.2 for an example. After applying all CompEx operations of all stages in a sorting network, the whole vector is sorted according to the CompEx function. A more detailed overview of sorting networks can be found in [91].

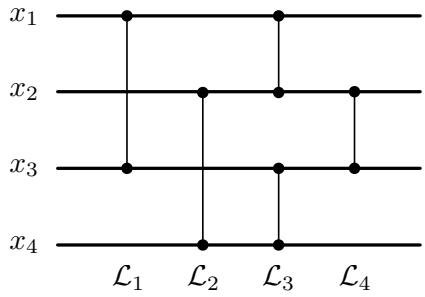


Figure 6.2: As example sorting network for sorting a vector with 4 values. Vertical connectors stand for CompEx operators. This 4-stage sorting network can be expressed as $\mathcal{N} = (((1, 3)), ((2, 4)), ((1, 2), (3, 4)), ((2, 3)))$.

Although, in theory, there are sorting networks constructions with the complexity $\mathcal{O}(n \log n)$, the involved constants are large and practical constructions have a complexity of $\mathcal{O}(n \log n^2)$. For efficiency, we also prefer constructions where any index is used at most once in each stage. This allows us to vectorise

CompEx functions for each stage (Steps 2–4 in Algorithm 7) and use the initial input data vector for storing changes.

Algorithm 7: Basic algorithm for sorting with a sorting network.

Data: Input array $[\![\boldsymbol{x}]\!] \in \mathbb{Z}_{2^k}^n$ and a sorting network $\mathcal{N} = (\mathcal{L}_1, \ldots, \mathcal{L}_m)$.
Result: Sorted output array $[\![\boldsymbol{x}]\!] \in \mathbb{Z}_{2^k}^n$.

```
1 foreach L_i ∈ N do
2 |   foreach (l, r) ∈ L_i do
3 |   |   (x_l, x_r) ← CompEx(x_l, x_r)
4 |   end
5 end
```

As sorting network structure is the same for all input vectors of given length, Algorithm 7 is trivially data-independent given a data-independent implementation of CompEx function. The latter is enabled by an oblivious implementation of the min-max construction given above.

6.2.3 Radix sort

For bitwise shared data we can take advantage of the fact that accessing individual shared bits is a local operation. Hence, we can use count and radix sort algorithms. Count sort [44, 54] sorts values from a small range by constructing a frequency table for the input vector and combining output based on this table.

Radix sort [76] works on a vector of integers and uses count sort as a subroutine. Most commonly, radix sort sorts its input one digit at a time, rearranging values based on counting sort output on the vector of digits in the given position. Starting with the least significant digit, radix sort gives the right output as the underlying count sort is a stable sorting algorithm.

An oblivious radix sort based on binary count sort is shown as Algorithm 8. The counting sort is made data-independent by keeping the counters for zeros and ones ($[\![c_0]\!]$ and $[\![c_1]\!]$) as well as the order vector $[\![\boldsymbol{ord}]\!]$ in secret-shared form. As the radix sort itself is just a for-cycle over the counting sort, this is enough to make the whole algorithm data-independent. However, notice that the elements of digit vector $\boldsymbol{d}$ are used in addition and multiplication operations. As these operations are expensive on bitwise-shared values, we convert vector $\boldsymbol{d}$ into additively shared secret. The data vector $\boldsymbol{x}$ and the output stays in bitwise shared form.

In his work, Zhang [131] also describes several sorting methods that sort values in a given range and are suitable for SMC. He also proposes to use radix sort on top of them to handle wider range of values. Unfortunately, he does not give any benchmarking results. Independently from us, Hamada *et al.* have designed

Algorithm 8: Data-independent radix sort.

Data: Bitwise shared input array $[\![\boldsymbol{x}]\!] \in \mathbb{Z}_{2^k}^n$.
Result: Bitwise shared sorted array $[\![\boldsymbol{x}]\!] \in \mathbb{Z}_{2^k}^n$.

```
// Iterate over all digits starting with the least
   significant digit:
```

foreach $m \in \{1, \ldots, k\}$ **do**
  Let $[\![\boldsymbol{d}]\!] = ([\![d_1]\!], \ldots, [\![d_n]\!])$ contain m-th bits of $([\![x_1]\!], \ldots, [\![x_n]\!])$.
  Convert elements of $\boldsymbol{d}$ from $\mathbb{Z}_2$ to additively shared $\mathbb{Z}_{2^k}$.
  `// Count number of zeros:`
  $[\![n_0]\!] \leftarrow n - sum([\![\boldsymbol{d}]\!])$
  `// Keep counters for processed zeros and ones:`
  $[\![c_0]\!] \leftarrow 0$; $[\![c_1]\!] \leftarrow 0$
  `// Keep` n`-element shared order vector. It may be initialised with shares of zero:`
  $[\![\boldsymbol{ord}]\!] \leftarrow ([\![0]\!], \ldots, [\![0]\!])$
  `// Put each element in the right position:`
  foreach $i \in 1 \ldots n$ **do**
    $[\![c_0]\!] = [\![c_0]\!] + 1 - [\![d_i]\!]$
    $[\![c_1]\!] = [\![c_1]\!] + [\![d_i]\!]$
    `// Obliviously update order vector:`
    $[\![ord_i]\!] = (1 - [\![d_i]\!]) \cdot [\![c_0]\!] + [\![d_i]\!] \cdot ([\![n_0]\!] + [\![c_1]\!])$
  end
  `// Shuffle two column database:`
  $([\![\boldsymbol{x}]\!], [\![\boldsymbol{ord}]\!]) \leftarrow \mathsf{Shuffle}([\![\boldsymbol{x}]\!], [\![\boldsymbol{ord}]\!])$
  $\boldsymbol{ord} \leftarrow \mathsf{Declassify}([\![\boldsymbol{ord}]\!])$
  Rearrange elements in $[\![\boldsymbol{x}]\!]$ according to $\boldsymbol{ord}$.
end

an oblivious radix sort construction very similar to our's and implemented it using Shamir secret sharing scheme [71]. We compare the benchmarking results of the implementations in Section 6.5.

6.3 Optimisations

6.3.1 Vectorisation

As most SMC protocols are network-bound, SIMD-style parallelisation helps to save network communication rounds and bring down the amortised cost per operation. The proposed NAIVECOMPSORT algorithm demonstrates this by doing all

the possible comparisons at once and constructing output vector from this result. Similarly, the quicksort implementation in [72] parallelises all comparisons on the same depth in the recursion tree.

As mentioned before, we also compute all of the CompEx operations of a stage in a sorting network in parallel. This is made possible by the assumption that no element is used in more than one CompEx function at any stage.

Radix sort is also vectorised by doing the whole count sort sub-procedure (Steps 7–11 in Algorithm 8) in parallel for each digit. Note, that it is possible to further save communication rounds in radix sort by applying count sort on more than one digit at once. For example, we can save half of the communication round count by replacing binary count sort with count sort that works on four values (two bits). However, this would mean replacing the efficient oblivious choice operation (Step 10 in Algorithm 8) with a more expensive comparison operation.

6.3.2 Share representation

All of the sorting algorithms mentioned here, except radix sort with binary count sort, use comparison operation to reorder values. Oblivious comparison operation is more efficient on bitwise shared data than on additively shared data. Hence, we might benefit from converting additively shared input vector to bitwise shared values and run the chosen sorting algorithm on this vector. This conversion requires an expensive bit extraction operation, but on large enough inputs the saving from comparisons can outweigh one vectorised conversion. Converting the shares back to additively shared data after sorting also requires network communication, but is not as expensive.

6.3.3 Assuring uniqueness

Obliviously shuffling the input vector helps to hide the permutation applied by the sorting algorithm in comparison-based algorithms where the comparison results are declassified. However, the number of equal elements may still leak from these comparison results. It is possible to alleviate this problem by making all of the input elements unique.

One method to accomplish this proposed in [72] is to append each element with bits representing its position in the initial vector. Alternatively, we can keep this index separately and obtain a secret-shared pair $(\llbracket x_i \rrbracket, \llbracket i \rrbracket)$ for each element. Oblivious shuffle at the beginning of the algorithm should be applied on the value-index pairs, obtaining a permutation

$$(\llbracket x_{\pi(1)} \rrbracket, \llbracket \pi(1) \rrbracket), \ldots, (\llbracket x_{\pi(n)} \rrbracket, \llbracket \pi(n) \rrbracket) \ .$$

In this case the oblivious comparison operation ">" also has to work on these value-index pairs:

$$(x_1, y_1) > (x_2, y_2) \Leftrightarrow x_1 > x_2 \vee (x_1 = x_2 \wedge y_1 < y_2) \ .$$

Note, that both of these methods also make the sorting algorithms stable. This property is important, while sorting matrices by more than one column as described in Section 6.4, because there we need to apply oblivious shuffling to hide the initial order of matrix rows.

6.3.4 Optimising sorting networks

Generating sorting networks is time-consuming. As sorting networks are data-independent, we can amortise this cost by caching the generated sorting network structure to use it again with different input vectors of the same size. If the input vector length is known in the application, the network generation can be moved to pre-processing phase.

If the oblivious implementation of the CompEx function is too expensive in a given SMC solution, it is possible to use the same approach as for the comparison-based algorithms. By obliviously shuffling the input vector, we can implement CompEx operation by choosing the correct output based on a declassified comparison result. This reduces the CompEx complexity to the complexity of comparison operation, which is a considerable saving e.g. for wide matrices, and may outweigh the cost of oblivious shuffle.

6.4 Sorting secret-shared matrices

In this context, we think of sorting matrices as rearranging the rows in two-dimensional vectors with n rows and m columns according to values in one or more columns. To sort by more than one column, e.g. by a column pair (i, j), where $i, j \leq m$, we have two options. We can either redefine the comparison operator to work on tuples of values, or first sort the rows by values in the j-th column and then again by values in the i-th column. The latter uses an approach similar to the one used by radix sort and needs the sorting algorithm to be stable. As the number of columns we want to sort by may dynamically change during the run-time of an application, it is cumbersome to implement the comparison operator needed for the former case. Thus, we decided to implement only the method that sequentially sorts each column using stable sorting algorithms.

Assume, that we want to sort the rows in a matrix according to the values in column k. First, we extract the k-th column as an n-element secret-shared vector and make the sorting algorithm stable by creating a secret-shared index vector

$[\![1]\!], \ldots, [\![n]\!]$. Together, these two vectors form a vector of element-index pairs and the sorting algorithm uses the comparison operation shown in Section 6.3.3. However, before sorting, we obliviously shuffle the matrix rows together with the values in the chosen column and the constructed index vector to hide the initial order of rows. Note, that this oblivious shuffling step replaces the required initial shuffling step in comparison-based algorithms like quicksort and NAIVECOMPSORT.

In comparison-based sorting algorithms, some elements may be moved several times. In matrices, swapping two rows means swapping all elements in these rows and this takes extra computation time, although local. Thus, we reduce sorting the matrix to sorting the individual column. After shuffling the matrix rows, we create an n-element permutation vector $1, \ldots, n$ and pass it to the sorting algorithm together with the k-th column and the index vector. Instead of swapping the rows in the matrix, the sorting algorithm makes the changes in the vector of element-index pairs and also in the permutation vector. When the sorting algorithm is finished, the values in the permutation vector can be used to reorder the matrix rows. Both the index vector and the permutation vectors may then be discarded. If the matrix has to be sorted by more than one columns, the same steps have to be repeated for the next column.

For sorting networks, instead of constructing a CompEx function that exchanges elements in both data and permutation vectors as described above, we can construct a CompEx function that works on whole matrix rows instead. Such CompEx function takes as input two rows as vectors $\mathcal{A}$ and $\mathcal{B}$, and the column index k that is used for sorting:

$$\mathsf{CompEx}(\mathcal{A}, \mathcal{B}, k) = \begin{cases} (\mathcal{B}, \mathcal{A}), & \text{if } \mathcal{A}_k > \mathcal{B}_k \\ (\mathcal{A}, \mathcal{B}), & \text{otherwise .} \end{cases}$$

An oblivious implementation of such CompEx function is shown as Algorithm 9.

Algorithm 9: Obliviously comparing and exchanging two rows in a matrix.

Data: Two input vectors $\mathcal{A}, \mathcal{B}$ of length m, column index $k \in \{1 \ldots m\}$.
Result: Pair of vectors $(\mathcal{A}', \mathcal{B}') = \mathsf{CompEx}(\mathcal{A}, \mathcal{B}, k)$.

1 $b \leftarrow \begin{cases} 1, & \text{if } \mathcal{A}_k > \mathcal{B}_k \\ 0, & \text{otherwise.} \end{cases}$
2 **foreach** $i \in 1 \ldots m$ **do**
3 $\quad \mathcal{A}'_i = (1-b)\mathcal{A}_i + b\mathcal{B}_i$
4 $\quad \mathcal{B}'_i = b\mathcal{A}_i + (1-b)\mathcal{B}_i$
5 **end**

Note, that using this CompEx operation for rows does not require obliviously shuffling the rows in the matrix as the first step. Hence, this construction also gives

us matrix sorting algorithm where the relative order of rows with equal element on k-th column is preserved.

6.5 Benchmarking results

6.5.1 Algorithm implementations

We implemented all of the sorting algorithms described in this chapter and shown in Table 6.1. All algorithms were implemented in the SECREC language using the SHAREMIND Application Server's protocol suite for honest-but-curious adversarial model available in fall 2013, when preparing the first version of the results published in [28].

Algorithm	Data-independence and leakage	Ref.
Quicksort	Comparison results are declassified. Running time is data-dependent. Leaks the number of equal elements.	[72]
Naive comparison sort (Algorithm 6)	Comparison results are declassified. Running time is data-independent. Leaks the number of equal elements.	[27, 28]
Sorting network sort (Algorithm 7)	Fully data-independent	[87, 126]
Radix sort (Algorithm 8)	Shuffled reordering decisions are declassified. Running time is data-independent. Does not leak the number of equal elements	[27, 28]

Table 6.1: An overview of implemented oblivious sorting algorithms.

The quicksort algorithm was implemented based on its description in [72] and personal communication with the authors. All other implementations are based on descriptions provided in the respective papers and this chapter. As many of these algorithms use oblivious shuffle as a sub-operation, it is worth to mention that its implementation is based on construction by Laur *et al.* [98]. All of the implementations take advantage of the vectorisation techniques described in Section 6.3.

Sorting network structure is generated using Florian Forster's `libsortnetwork` library [56]. The generated sorting network structures are cached and evaluated by SECREC. We benchmarked sorting network generation to choose the most suitable network generation algorithm for our implementation. Our goal was to create a network with the minimal number of rounds in a minimal time and, preferably, have a low number of CompEx functions.

We evaluated Batcher's bitonic mergesort network, Batcher's odd-even mergesort network and Parberry's pairwise sorting network. For each kind of network, we used the library to generate networks of various sizes, measured the time and counted the number of CompEx functions needed to evaluate it.

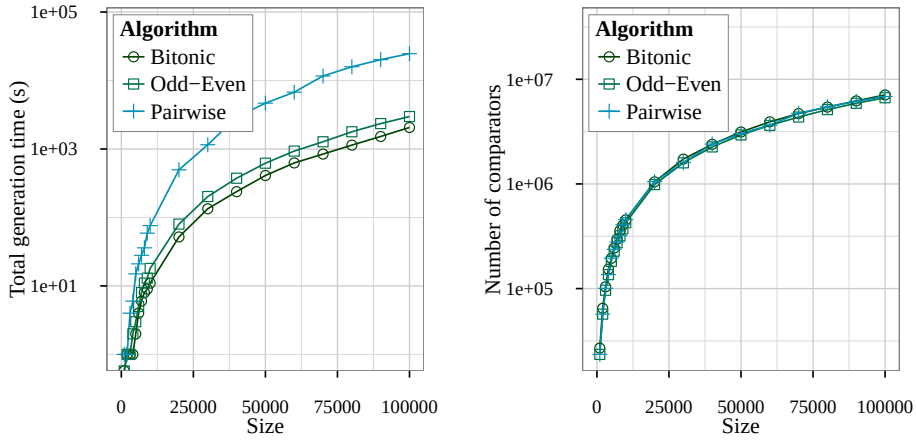


Figure 6.3: Benchmarking results for sorting network generation.

We found that the number of comparators is very similar for all algorithms and the same in many cases. Figure 6.3 shows a comparison of the running times and CompEx function call counts. We chose the bitonic mergesort algorithm as the library generates this network structure in the shortest time and while it has slightly more CompEx gates, the round count is the same and, therefore, the number of vector operations will be the same.

6.5.2 Test setup

As with privacy-preserving database linking (Section 5.4), the sorting algorithms were tested on the same cluster of three nodes each having its own installation of SHAREMIND. Each cluster node was powered by 12-core 3 Ghz CPU with *Hyper-Threading* and had 48 Gb of RAM. Nodes were connected by a 1 Gbps LAN connection. However, this SHAREMIND version used a custom built networking library where all communication channels were protected by TLS.

We tested each sorting algorithm on vectors of various lengths containing 64-bit bitwise secret-shared integer values. We used test vectors with all equal values (worst case) for all of the four algorithms. Additionally, a vector with all unique values was used for quicksort to get the average case running time. No share conversion was used to assure uniqueness of the values.

In addition to measuring running time and network usage that are provided

by the built-in profiling facilities in SHAREMIND, we also measured the peak memory usage. We logged the memory usage of a SHAREMIND process with 1 second interval, aligned the data with each test and found the peak memory usage.

6.5.3 Sorting vectors

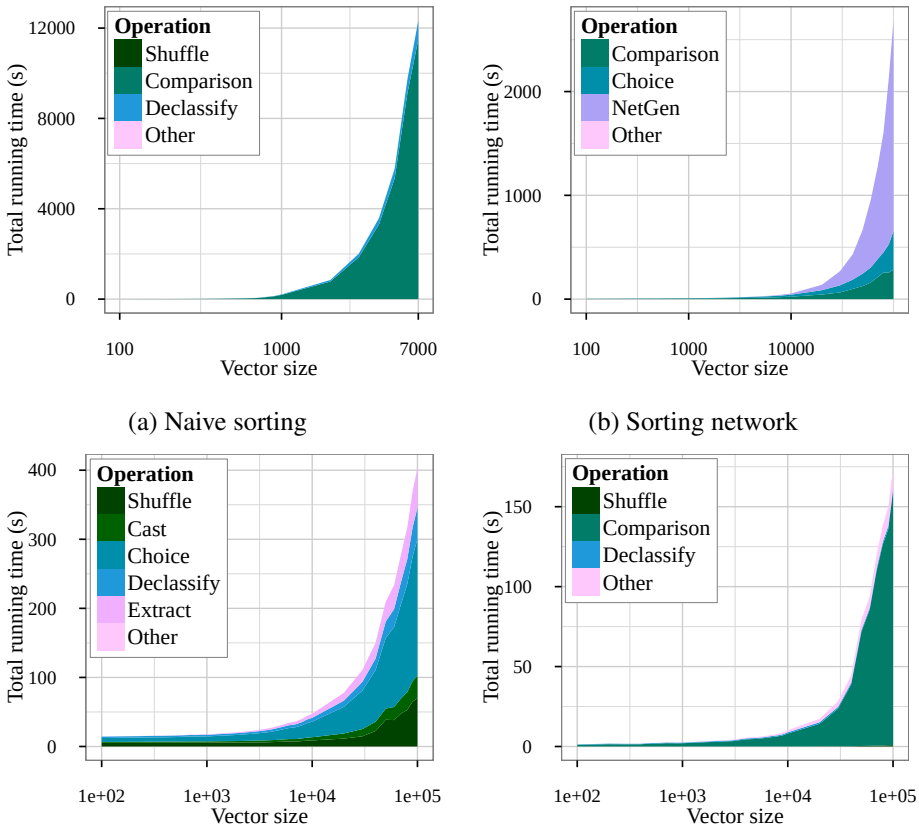
First, let us consider the complexities of the described sorting algorithms. Instead of giving the exact communication and computation complexity in bits, Table 6.2 gives the complexities as a combination of sub-protocols. For example, $m\mathsf{Protocol}(n)$ means that protocol $\mathsf{Protocol}$ is invoked m times, each time on n input values in parallel. This format allows for a more general overview, as the concrete sub-protocols may be swapped by other implementations.

Algorithm	Secure operation complexity
Quicksort (unique)	$\mathsf{Shuffle}(n) + \mathcal{O}(\log n)\mathsf{Comp}(\mathcal{O}(n))$ $+ \mathcal{O}(\log n)\mathsf{Declassify}(\mathcal{O}(n))$
Naive comparison sort	$\mathsf{Shuffle}(n) + \mathsf{Comp}(n(n-1)/2)$ $+ \mathsf{Declassify}(n(n-1)/2)$
Sorting network	$\sum_{i=1}^{m} \mathsf{Comp}(\ell_i) + \mathsf{Mult}(4\ell_i)$
Radix sort	$k \cdot (\mathsf{ShareConv}(n) + \mathsf{Mult}(n)$ $+ \mathsf{Shuffle}(n, 2) + \mathsf{Declassify}(n))$

Table 6.2: Secure operation complexity of oblivious sorting algorithms. n is the number of elements to sort and k is number of digits, where applicable. For sorting networks, m is the number of stages in the network and ℓ_i is the number of CompEx operations on the i-th stage.

Figure 6.4 gives the component breakdown for the four implemented algorithms. As expected, NAIVECOMPSORT and quicksort running times are dominated by comparison operations. Comparisons together with oblivious choice also take most of the time in sorting networks (although it is not clearly seen on Figure 6.4b). However, for longer input vectors, generating the network structure takes most of the time. This is a good indicator for the need to pre-generate and cache network structures if possible. Radix sort does not use comparison operations at all. Instead, oblivious choice and shuffle take the most time here.

The running time comparison of the implemented sorting algorithms is given on Figure 6.5. Note, that both axis are given in a logarithmic scale. We can immediately see that NAIVECOMPSORT is practical for only very short input vectors because of its quadratic complexity. Quicksort is the fastest algorithm, but only in case of unique element values. With colliding values, its performance is somewhere between its average and worst case.



(a) Naive sorting

(b) Sorting network

(c) Radix sort

(d) Quicksort (unique)

Figure 6.4: Cumulative running time breakdown by sub-components for implemented sorting algorithms.

For shorter input vectors, radix sort is not the most efficient algorithm, but at the same time its running time does not grow as quickly as it takes advantage of easier oblivious operations. In [71], Hamada et al. report better performance results with their implementation of radix sort that uses a similar construction. The difference can be explained by two facts. First, in our tests, we use 64-bit input values whereas they use 32-bit values. It not only means twice as much network traffic but also twice as many rounds for radix sort. Secondly, we implemented the algorithms in a high-level SecreC language while the implementation in [71] was written in C++.

Sorting networks perform very favourably on shorter input vectors, but the time to generate the network structure takes a lot of time for longer inputs. If caching can be used, sorting networks become almost as good as radix sort.

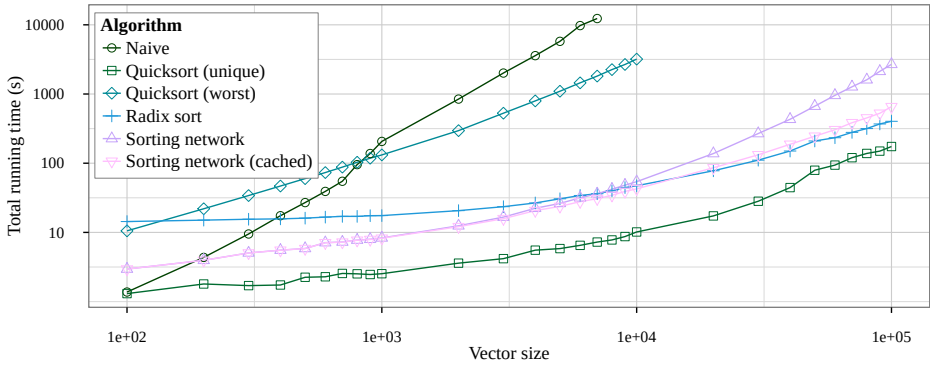


Figure 6.5: Comparison of the running time of oblivious sorting algorithms.

The comparison of network usage on Figure 6.6 shows that the two algorithms with quadratic computation complexity also send more data over the network. The rest of the algorithms are comparable with radix sort sending slightly more data for shorter input vectors and quicksort on unique values being the most efficient one.

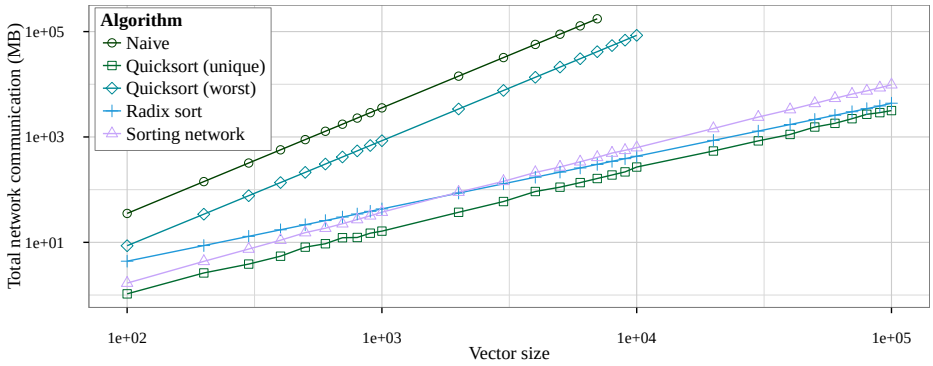


Figure 6.6: Comparison of the network usage of oblivious sorting algorithms.

Once again, memory usage (Figure 6.7) shows that NAIVECOMPSORT is impractical as it has to do all possible comparisons in parallel and thus allocates a lot of memory at once. Sorting networks also tend to need more memory for longer input vectors as the network structure is generated statically and kept in memory. The rest of the algorithms form a stable group.

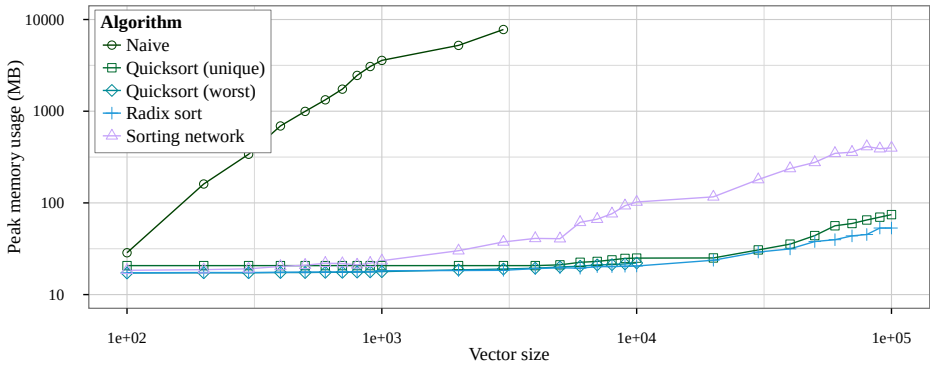


Figure 6.7: Comparison of the memory usage of oblivious sorting algorithms.

6.5.4 Sorting matrices

In addition to measuring the performance of sorting vectors, we also benchmarked sorting matrices with n rows and 10 columns. For sorting networks, we used the CompEx function designed for matrix rows. All other implementations use the permutation vector based approach.

Sorting by one column shows that even though there are 10 times more data, the running time and network usage of sorting does not increase tenfold (Figures 6.8 and 6.9). This can be explained by the fact that the sorting in actually performed on an index vector and not the whole matrix. Reordering the rows at the end takes constant amount of time. Although vectorised, the oblivious exchange has to performed on every column and thus the sorting networks do not benefit so much from caching any more. Naturally, as seen from Figure 6.10, the memory consumption of the algorithms is increased. Otherwise, the relation between the different algorithms remains the same.

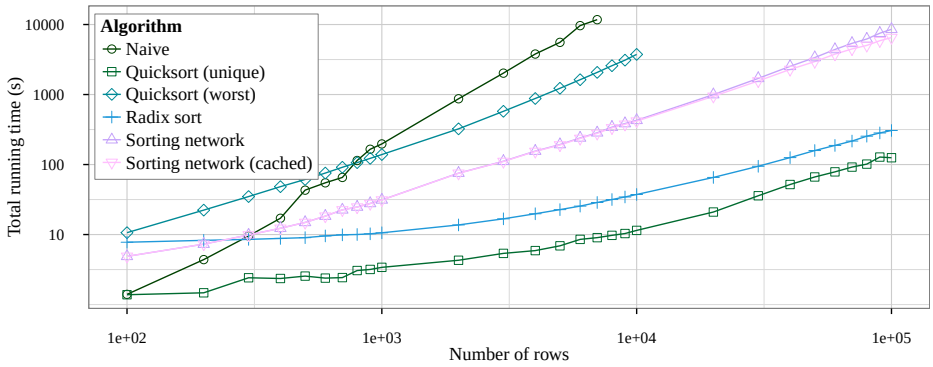


Figure 6.8: Running time of oblivious sorting algorithms on matrices.

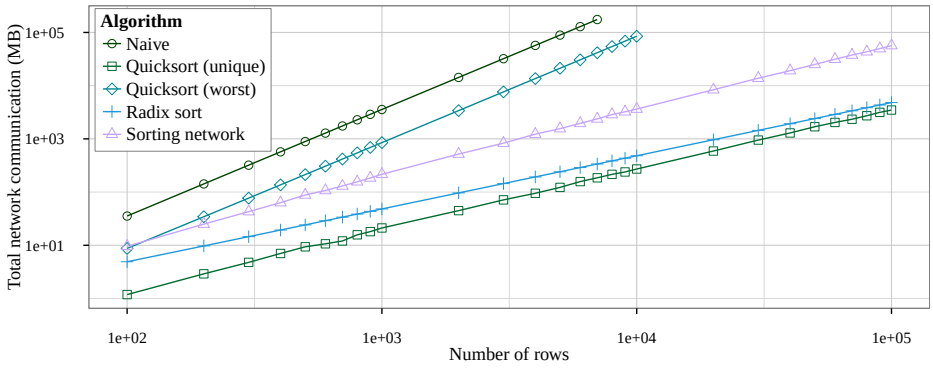


Figure 6.9: Network usage of oblivious sorting algorithms on matrices.

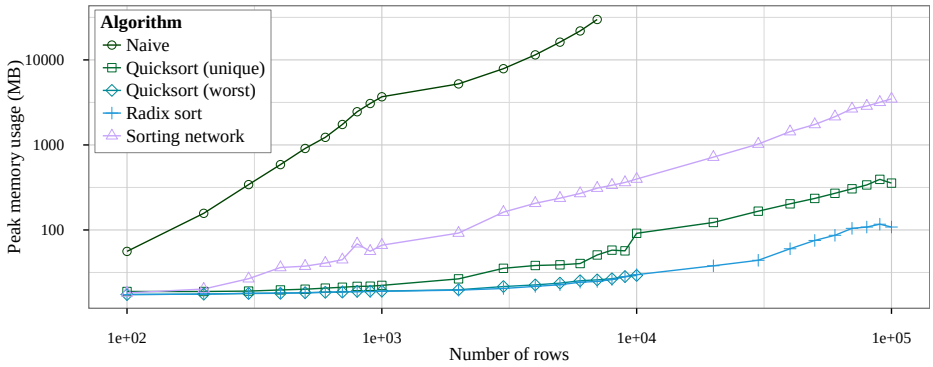


Figure 6.10: Memory usage of oblivious sorting algorithms on matrices.

Additionally, for sorting networks, we measured the performance of sorting the matrix by two and three columns. Fortunately, as seen from Figure 6.11, sorting by multiple columns does not have significant effect on the running time.

6.6 Conclusion

We have implemented and benchmarked four oblivious sorting algorithms. Clearly, NAIVECOMPSORT should be considered as a theoretical example construction and is not usable in practice.

On average, quicksort has the best performance results but only if the uniqueness of elements is guaranteed by the application itself or converting the shares by adding extra bits. Radix sort is competitive on longer inputs as it does not depend on the rather expensive oblivious comparison operation.

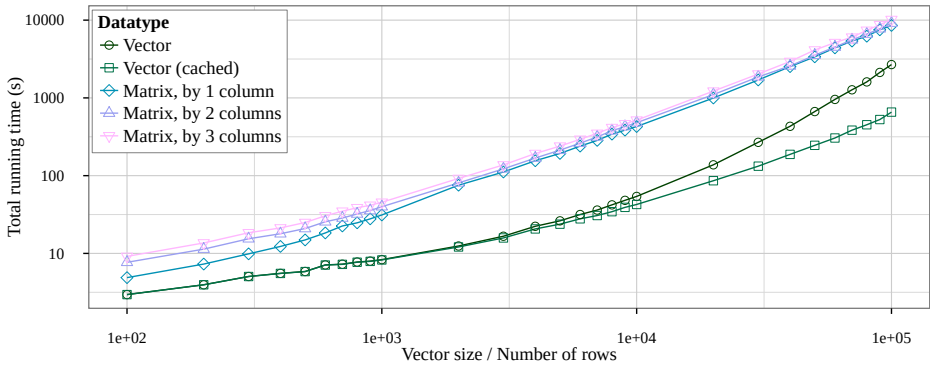


Figure 6.11: Sorting networks running time on vector and matrix inputs. For matrix inputs, sorting is performed by one, two and three columns.

Sorting network is the only design that is data-independent by design and does not rely on any declassifications. However, generating the network structure may take a lot of resources for longer input vectors.

The author recommends to use quicksort on both vectors and matrices if there are no equal elements or revealing the number of equal elements is acceptable. Otherwise, radix sort should be considered as the next alternative.

CHAPTER 7

DEPLOYING SMC FOR DATA INTEGRATION

7.1 Motivation

Informed decisions are the foundation of good governance. Smart governments analyse information about the state and use it to steer their decisions. There are two main methods for gathering information to answer a given question about the state. First, there is the possibility to conduct a survey among the interest group and analyse the results. This method allows to gather exact and up-to-date data for even very specific questions. However, in most cases, only a small sample of the target group can be surveyed and it might be biased in an unknown direction.

Alternatively, the government may take the *big data* approach and use the data it already has collected and that is stored in various databases hosted by different government institutions. For example, the state usually has information about its population, its health status, income (or at least taxes), education and criminal records. Making decisions based on analyses carried out on these databases means that the whole population is included and, at the same time, the analysis results are received faster without disturbing the people. This of course holds only if the data sets are up-to-date and managed properly.

These data sets need to be analysed somehow. In a common scenario, the state would contact its Statistics Board and ask it to conduct the necessary analysis. The latter has to contact the hosts of relevant databases and ask them for the data. Notice, that the data used by the Statistics Board might contain private information, even more so if it is a compilation of several data sets. This holds even if pseudonymisation is used [9, 105, 86]. Guarding this database against both outside attackers (hackers) and inside attackers (curious analysts or database administrators) is a non-trivial task. In some countries, the Statistics Board is in a favourable position as it is trusted by the government and can access tax

records for conducting analysis. However, it is important to notice that there are no technical means in place that make it impossible for a statistician to misuse that data. There are only the professional ethic and contractual obligations that forbid the statistician from doing that.

There is an alternative where no specific party is in a favoured position to access the data for analysis – the *open data* initiative. Lane, Heus and Mulcahy [94] bring out five reasons to promote data openness:

1. Data utility – data only have utility when it is used.
2. Replicability – scientific results must be replicable and validated by other researchers.
3. Communication – scientific results may be interpreted in many ways so the applied scientific concepts must be shared.
4. Efficiency – collecting data is costly, so it should be reused.
5. Capacity-building – researchers and policy makers need to go beyond aggregate tables and be able to access micro data to make informed decisions.

It is easy to see that these arguments also hold for state-collected data. However, it is also clear that the state cannot make health, income or criminal records public. It could publish aggregate data to hide individual records in the data set, but this introduces a *privacy-utility* trade-off and decreases the usefulness of the data. Since the aggregation has to be performed on each data set independently by its host, data sets from different hosts can not be linked together anymore on individual subject level. Moreover, all of pseudonymisation, k-anomymity [120] and ℓ-deversity [101] have been shown to provide inadequate privacy when an attacker has access to auxiliary data [105, 99].

7.2 The PRIST project

In this chapter, we describe a pilot project PRIST (Privacy-preserving statistical studies on linked databases), where secure multi-party computation was used to securely link and perform statistical analysis on state databases [22]. The research question motivating the project was to study the relationship between working during university studies and graduating on time. In Estonia, there is a problem that students are not graduating from their studies on time, especially in the ICT field. At the same time, there is a known deficit of work force in the ICT field so it is hypothesised that well-paid professional jobs lure students away from their studies. The main objective of the PRIST project was to validate if and to what

extent this is true and how ICT and non-ICT curricula relate to each other in that matter.

To study the relationship between students' academic progress and their career, an analyst ultimately needs a table with students educational history and working history. In Estonia, the Education Information System hosted by the Ministry of Education and Research (HTM) has all the education records for recent years. Income records are not available in any government databases, however, such data can be inferred from the tax payment records kept by the Tax and Customs Board (EMTA). The required analysis table can be compiled by linking each student's educational records from one database with tax (income) records from the other database.

In parallel to the SMC-based study, the analysts carried out the same study using conventional methods, such as k-anonymisation to protect individuals' privacy. First, under a non-disclosure agreement (NDA), the analysts received pseudonymised education data from the Ministry of Education and Research and divided the students into groups based on demographic attributes. The grouped pseudonyms were then sent to the Tax and Customs Board. By using the pseudonym and real identity matching table obtained from the Ministry of Education and Research, the Tax and Customs Board updated each group with income data for the corresponding students. However, there was no link between a student and income, i.e. the income data was randomly shuffled for each group. To obtain 3-anonymisation, groups with less than 3 individuals were left empty. Finally, the analyst conducted the study on the groups containing education and income data.

This pre-aggregation meant that many students with unique background were left out of the study. In general, such method caused a sample loss of 10%–30%, depending on a demographic group. Initially, the Tax and Customs Board wished to use similar k-anonymity measures on the tax data as well. However, this approach was discarded as it would have resulted in sample loss of 84%–97%. Thus, secure multi-party computation based study was seen as an alternative to enhance the privacy of the underlying data, while at the same time obtaining more accurate non-biased study results.

7.3 Overcoming barriers

7.3.1 Tools for statisticians

Statisticians are used to statistical analysis tools like SAS, SPSS, Stata or R. For privacy-preserving statistical studies like PRIST, we cannot assume that statisticians are going to develop or combine SMC protocols to perform the necessary aggregations or tests, as they are not cryptographers. On the other hand, cryptographers may develop specific protocols, but only if they have a detailed study plan

to follow – they may not be experienced enough to design the study themselves.

RMIND is a statistical analysis tool that performs computations on secret-shared data [24]. On the client side, RMIND provides a user interface with a command language that is inspired by R [115], so it is familiar to statistical analysts. However, the data is not loaded to the client application, but RMIND rather points to the data secret-shared in a SHAREMIND deployment. On the server side, each SHAREMIND computation node has SECREC programs that implement the necessary statistical functions [23, 24] – descriptive statistics, filtering, statistical tests, linear regression, etc. Among other, RMIND incorporates the database linking and sorting functionality described in Chapters 5 and 6, respectively.

RMIND never publishes individual values and employs a technique where it refuses to compute aggregates on data sets with a number of records below a given threshold. For example, it is not possible to draw a scatter plot with elements shown on two-dimensional grid. Instead, the individual elements are assigned to dynamically-sized buckets on the server side and the number of elements in each bucket is returned to the client. The result can be displayed as a heatmap. However, it may be possible in some cases to cleverly create filtered data sets that are large enough but differ only in one record. Comparing the aggregates computed on these data sets may reveal some information on the one added record. In principle, it is possible to apply *differential privacy* [53] on RMIND output, but this depends on the exact application area and should be optional. Limiting leakage and the privacy-utility trade-off is discussed in more detail in [24]. Alternatively, server-side query logging may help to mitigate this kind of behaviour.

7.3.2 Data protection regulation

Both the education information from the Education Information System and tax records from Tax and Customs board fall under the Personal Data Protection Act [1] as they contain personal data. Hence, data owners carefully considered the possible implications before accepting to participate in the study. With the help of a covering letter from the Ministry of Economy and Communications, initially meant for the pre-aggregated study, and a meeting where we explained how secret sharing and SMC works, the Ministry of Education and Research was willing to import the education data into the system. However, the Tax and Customs Board was still reluctant to do so with their data and asked us to get an approval from the Estonian Data Protection Agency (DPA).

In December 2013, we compiled a letter of explanation for the DPA where we explained how secret sharing and SMC work and how these technologies would be used in the study. We compared the conventional and SMC-based studies, and brought out how the high sensitivity data moves between the parties and stages in the study. In the conventional study, the analyst hosts a compiled database con-

taining pseudonymised pre-aggregated personal data. However, in case of using SMC, the high sensitivity data ceases to exist at the secret sharing process which is executed by the respective data owners. Hence, with SMC-based study, a copy of sensitive data does not exist outside the data owner's organisation.

After consideration, the Estonian DPA found that secret sharing should be considered to be a form of encryption and SMC a computation on encrypted data. As the individual records remain secret shared ("encrypted") throughout the whole process, the DPA concluded that we are not processing personal data in the study and thus do not need their permission [5]. However, sufficient organisational and physical information security measures must be taken by each computation party to protect their respective shares.

7.3.3 Tax secrecy

In addition to the Personal Data Protection Act, the income data from the Tax and Customs Board is also covered by tax secrecy in the Taxation Act [2, §26–§30]. This means that the Tax and Customs Board is the source of the most sensitive data in the PRIST study. To alleviate the risk of misusing the data, four controls were put in place.

First, the Tax and Customs Board wanted to look through the source code of the SHAREMIND system and other custom-made programs used in the study (RMIND and SECREC scripts). In October 2014, the technology provider Cybernetica AS signed an NDA with the IT department of the Ministry of Finance (abbr. RMIT) that were to conduct the code review. The code review were to be conducted by the IT department of the Ministry of Finance because the Tax and Customs is in the jurisdiction of the Ministry of Finance.

We delivered the source code of the SHAREMIND platform, RMIND statistics tool and SECREC scripts used in the study along with documentation to RMIT in November 2014. They soon concluded that the code follows modern standards and software engineering best practices. However, RMIT pointed out that as they do not have in-house competence to evaluate the cryptographic functions, this can only be considered as a code review and not a full audit.

Secondly, as RMIT already conducted a code review, it wanted to be sure that the same code base is used to build the binaries that all involved parties are going to use. Hence, RMIT was chosen as the party to build all the necessary binaries for computation parties, input parties and the result party.

Thirdly, the IT department of Ministry of Finance agreed to host one of the three computation servers in the study. This gave them greater control as even after the input data is secret shared between the computation parties, they can halt the further computation by shutting down their server if they observe the protocol and fear that the data is being misused. This is possible as for every operation,

all the three computation parties have to agree on the computation to be done. If one of them deviates from the previously agreed plan, each one of them has the opportunity to halt the operation simply by not communicating with the others.

Fourthly, a representative of the Cybernetica's SHAREMIND team demonstrated the analysis tool RMIND to the oversight department of the Tax and Customs Board. Using a test data set, he explained which operations are possible with RMIND and what kind of output is shown to the analyst.

7.3.4 Contracts

Secure multi-party computation technology gives us excellent privacy guarantees, but only if its deployment requirements are fulfilled. The most important being that the computation parties must follow the protocol and must not collude with each other to exchange their shares of the data. Additionally, as a distributed computing model, SMC requires availability of the relevant parties during various steps of the process. Most notably, computation parties must be available throughout the data input and analysis phase and they must agree on a provided bandwidth so no-one becomes a communication bottleneck.

For these and additional reasons described below, the following contracts were established during the PRIST project. Figure 7.1 gives an overview of contractual connections between the participating parties.

- As the source of the most sensitive data in the study, RMIT was to conduct a code review and was responsible for delivering the binaries to the rest of involved parties. Hence, a non-disclosure agreement and a source code license was signed by Cybernetica and RMIT.

- The three computation parties (RMIT, RIA and Cybernetica) signed an agreement with the Ministry of Education and Research stating that they will provide the necessary resources to host the SHAREMIND server and that they will not collude with each-other. Additionally, it was stated that the computation parties must delete the shares after the analysis was done. This contract was requested by the Ministry of Education and Research as it did not participate as a computation party itself and thus had no other means to enforce its data usage policy. To the best of our knowledge, this is the first contract supporting an SMC deployment.

- CentAR, the analyst and result party in the project, signed an agreement with the input parties (HTM and EMTA) stating that they will use the data only for the given purpose and will not try to misuse RMIND to gain access to sensitive data of individuals. With technical measures, this was enforced

by the query restrictions built into RMIND. However, differential privacy was not assured as we were interested in exact statistical aggregates.

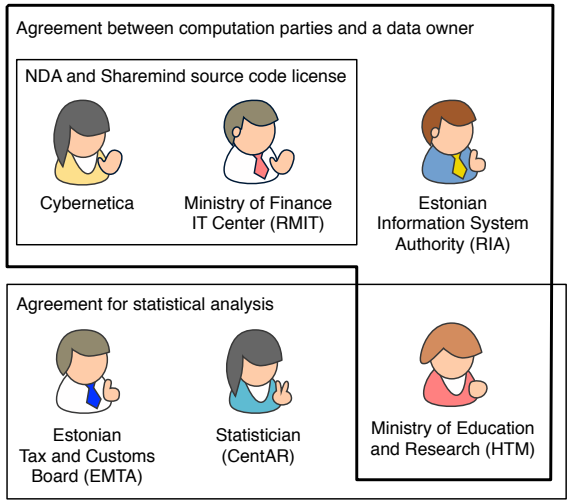


Figure 7.1: Contractual relationships between the parties in the PRIST project.

7.4 Project life cycle

7.4.1 Development and testing

The first input to the SHAREMIND team was the analysis plan put together by the analyst in the PRIST project, CentAR. As PRIST was the first study where RMIND tool [24] was used, we used this document as a reference to develop privacy-preserving versions of the necessary statistical functions that were still missing from RMIND at this point [88]. Based on the analysis plan, we developed Extract-Transform-Load (ETL) scripts for RMIND that cleaned the input education and tax data records, extracted relevant data fields, performed privacy-preserving linking and transformed the result table into a format that could be easily analysed [88].

The database schema included in the analysis plan contained a list with all the data fields and their format in the data exports that the input parties were to do. This information was used to create data models for the data importer tool used by the input parties.

Based on the data size estimations, we generated two test sets that we used for testing purposes throughout the project. The test sets were generated by simulating a working student and using statistical distributions based on a small sample set of data provided by CentAR. A smaller test set (test set A) was used to test

	Education records	Tax records
Test set A	354	8201
Test set B	831 424	16 205 641
Real data	623 361	10 495 760

Table 7.1: Number of records in the test data sets and final imported data.

locally during algorithm development as it was small enough to trace the origin of mistakes. A larger data set (test set B) was generated in order to mimic the actual data volume and was used for performance testing on Cybernetica's local cluster. CentAR also used both test data sets as an input to Stata – the statistical analysis tool used for the conventional study. The output was used to validate the output of RMIND. See Table 7.1 for the size comparison of the test sets and the final imported data set.

7.4.2 Delivery and setup

There were three types of application binaries distributed in the PRIST project (see Figure 7.2):

1. SHAREMIND Application Server – the software for storing shares of the input data and participating in secure multi-party computation. Used by all three computation parties: RIA, RMIT and Cybernetica.

2. Data import tool – a command-line tool that takes a data set in a comma-separated format (CSV) and by using a data model description, secret shares each value in the data set and distributes the shares between the computation nodes (SHAREMIND servers). The data model is an XML-file mapping the data types and field names of input and secret-shared data sets. The data import tool was used by input parties HTM and EMTA.

3. RMIND – statistical analysis tool used by the analysts: CentAR.

Each application binary was distributed as a separate compressed archive including all of the necessary third-party libraries. Each application also included an installation, configuration and a usage manual.

Code delivery

As RMIT was to review the source code and build binaries, they needed access to the source code of all of the components. RMIT set up a Git code repository on their infrastructure where Cybernetica's delivery engineer could push the code. Using a code versioning system made it possible to correct bugs after the initial

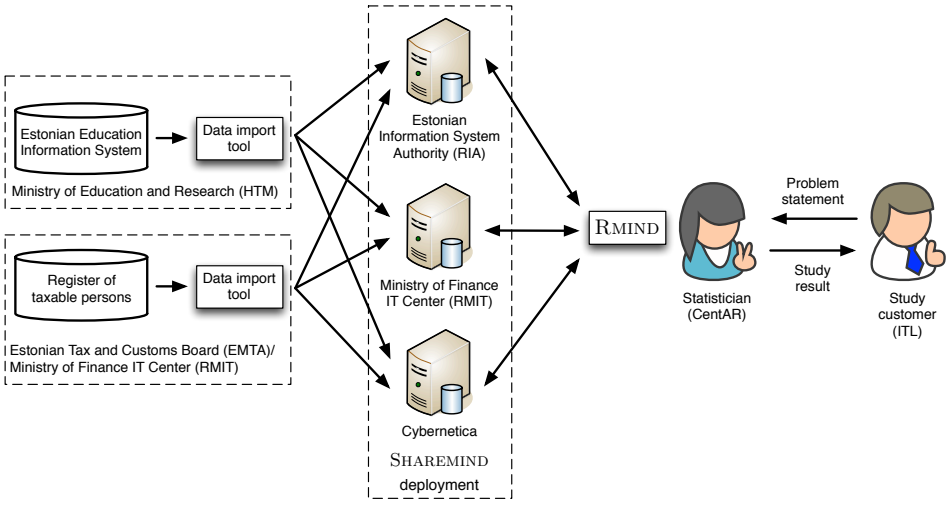


Figure 7.2: PRIST application binaries and their users.

version that was reviewed as each change in code could be easily tracked and verified by RMIT. The first version of the source code was delivered to RMIT on November 10, 2014, after signing an NDA and source code license with Cybernetica.

Choosing a distribution

SHAREMIND Application Server is written in C++ 11 adhering to the standard ISO/IEC 14882:2011 [82]. Moreover, as a security critical software, SHAREMIND and its supplemental components depend on the latest versions of many security libraries (i.e. GnuTLS). For this reason, the team mainly uses either testing versions of Linux distributions (Debian, Ubuntu) or rolling release distributions (Gentoo) for development and testing. At the time of the PRIST project, the only officially supported platform was stable Debian 7 (wheezy).

As a code reviewer, RMIT was chosen to be responsible for building and distributing application binaries to all parties, except Cybernetica. In Cybernetica, we could build our own binaries as we have access to the source code. However, even then the binaries have to be compiled from the same code base as the network level and protocol messages have to coincide with other parties. The same holds for SECREC scripts.

Initially, we planned that the binaries will be built on Debian 7 so they would run at least on Debian itself and Ubuntu. We had successfully used this model in the cloud demonstration (see Section 4.3.1) so we already knew that Ubuntu was acceptable for RIA. However, it turned out that RMIT uses only SUSE Linux

Enterprise Server (SLES) and deploying a VM running Debian or Ubuntu would mean too much extra effort. At the time of writing the documentation and delivering the code in October 2014, the latest version available was SLES 11 SP3, which didn't have a recent enough compiler. As a replacement RMIT agreed to use openSUSE that is a free and open version of SLES. OpenSUSE 12.3 has the necessary compiler version and binaries built on it ran on Ubuntu 14.04 LTS. Cybernetica and RIA chose to run the binaries on Ubuntu (14.04 LTS) as we had prior experience with it. This demonstrates that large organisations are not as volatile in their platform choices and that their deployment requirements should be taken into account early in order to avoid such confusions.

Key exchange

Communication channel security is one of the prerequisites to deploy secure multiparty computation. SHAREMIND Application Server establishes encrypted TLS tunnels with each party it needs to communicate with. This holds for data importer and RMIND client applications, as well as other SHAREMIND servers. SHAREMIND Application Server uses X.509 certificates for ease of management and interoperability in the future.

In PRIST, each participating party generated a self-signed certificate with a 2432-bit RSA keypair that is considered secure according to current best practice [8]. This was done with the help of GnuTLS `certtool` application bundled with SHAREMIND, RMIND and data importer program. The public keys were exchanged by e-mail. We chose not to deploy an application-specific Certificate Authority (CA) to sign the certificates. Instead, we bootstrapped the trust of each component's certificate from the national public key infrastructure of Estonia. Each party's representative (except HTM) signed their certificate (public key) using the Estonian ID-card.

7.4.3 Setup and administration

Setup

After reviewing the source code, RMIT used it to build application binaries for themselves and also other parties. As the building process on the specific Linux distribution was tested and documented by the author of this thesis, there were no major problems during this process. At the same time, all three computation parties started to set up their production environments and exchange public keys. On December 10, the three computation servers of the production environment were connected together for the first time.

As part of the review process, RMIT requested to play through the whole analysis process in the production environment before the actual data is imported. To

accomplish this, RMIT generated another key pair for both the data importer and RMIND that they had built. These key pairs were whitelisted by the computation nodes. In the second half of December 2014, RMIT imported test set A (both education and tax records) into the production environment and carried out the full ETL process in RMIND.

In the beginning of March 2015, both HTM and RMIT used the data importer to import the actual education and tax records to the production environment. Computation parties were restarted with whitelist configuration that only allowed CentAR's client application. The analysis step had begun.

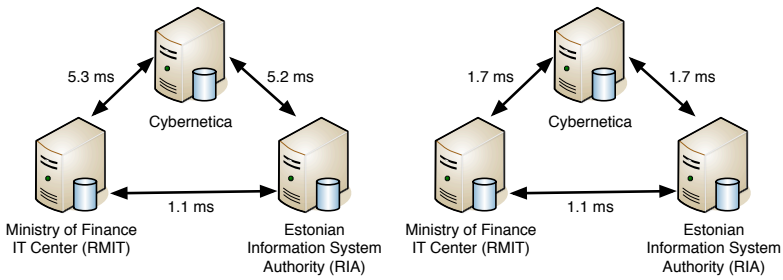
Maintenance

Even before the actual data was imported into the production environment, RMIT proposed that Cybernetica would take over its task of distributing application binaries for all parties. For the rest of the project, Cybernetica distributed updated binaries using its HTTPS-enabled web server. Access to the files was password protected and each party could only access files required for its role.

Between the beginning of the analysis phase in the beginning of March and its end on July 1, SHAREMIND computation nodes had to be restarted about 20 times. Most of these restarts were caused by the fact that during longer analysis steps, the client connection was often dropped, causing the computation nodes to hang. However, there were also other reasons, for example maintenance of network hardware at the hosting organisations.

Based on the benchmarking results on our local cluster with 1 Gbps LAN network connections, we asked each computation party to reserve at least 200 Mbps of their external link for the SHAREMIND server. However, during the analysis process we discovered that some steps require very small but very many (hundreds of thousands) sequential operations. As Cybernetica's server was in Tartu and the other two in Tallinn, network latency started to become a bottleneck. Hence, we migrated our SHAREMIND installation to our office in Tallinn that had smaller network bandwidth but we managed to reduce network latency three-fold, from 5.2 ms to 1.7 ms (See Figure 7.3).

Even when taking into account the difference in network parameters (bandwidth and latency) between the test cluster and the production environment, we still could not explain the up to twenty-fold difference in the running time of analysis scripts. At setup we had requested each SHAREMIND host to provide a (virtual) machine with 2-core CPU and 8 GB of memory. We now decided to run a simple benchmarking tool on all of the machines to see if maybe one party was a bottleneck, making all of the distributed protocols run slower. The results of



(a) Cybernetica's server in Tartu. (b) Cybernetica's server in Tallinn.

Figure 7.3: Latencies between the computation nodes in PRIST.

benchmarking tool UnixBench[1] showed that the server at RIA had the most resources as it was a physical machine. However, there was nothing to suggest that one of the servers was a bottleneck. Hence, we continued to make the privacy-preserving aggregation more parallel so less round-trips are needed.

ETL running times

The majority of the analysis phase was running the Extract-Transform-Load (ETL) RMIND scripts that took the two imported data sets, linked them together by person and converted the result to a format that was easy to use for specific questions. The overview of the ETL steps is given on Figure 7.4, detailed information about PRIST ETL process is given in [88, Section 6.5].

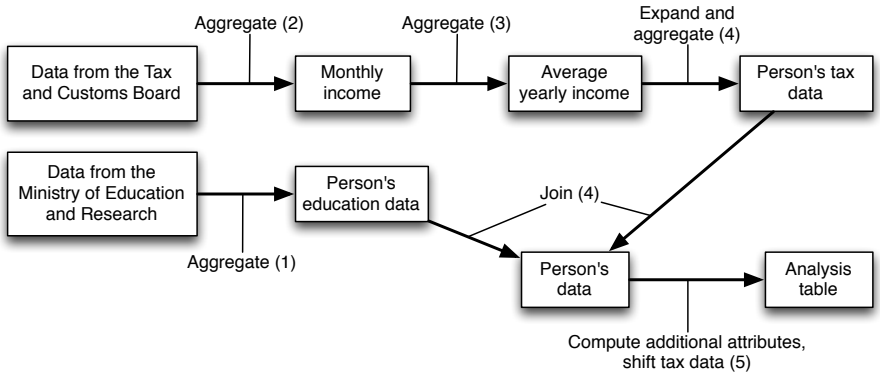


Figure 7.4: The privacy-preserving extract, transform and load process in PRIST with the corresponding ETL script numbers shown in parenthesis.

It should be noted that since we had problems with client disconnections when

[1] https://code.google.com/p/byte-unixbench/

running long SECREC scripts, some of the ETL steps were split into many parts. The final running times given in Table 7.2 are approximate, as some of the parts had overlaps. For example, the 5th step of ETL was done in 10 parts. Running all the ETL steps from beginning to end in one go would take a bit less time.

Uninstall

Initially, the analysis process in the PRIST project was supposed to be finished by March 1, 2015. However, due to technical difficulties and human errors that required running the analysis ETL process multiple times, this date was postponed until July 1, 2015. According to the contract signed by computation parties, each SHAREMIND host were to destroy all of the secret shares in their disposal. On July 1, Cybernetica stopped its SHAREMIND server and used secure deletion program `srm`[2] to destroy all shares stored in the VM. Even if we cannot be sure that the other computation parties destroyed their shares, this is enough to protect the confidentiality of the data, given that nobody has stored the exchanged network messages and has obtained some other party's shares.

7.4.4 Post mortem

On June 30, less than a day before erasing the shares, we discovered an inconsistency in the analysis results. Comparing with the analysis study results carried out with conventional methods, it seemed like the classifier "ICT"/"non-ICT" had been switched on some figures, but otherwise the figures were comparable. It is important to notice here that the results from the study with conventional methods and with secure multi-party computation are not directly comparable as, in the former, some data is removed by applying k-anonymity on the imported data set. Since RMIND also outputs the corresponding `gnuplot`[3] command line in addition to the figure, the analysts decided that, in the interest of using the remaining time productively, it is reasonable to fix the inconsistency by just switching the labels on plots afterwards.

Unfortunately, very soon after the computation nodes had been shut down and data shares erased, we discovered that the problem was not in the "ICT"/"non-ICT" classifier but rather in the status classifier that shows if a student has graduated, dropped out or was still studying on a given year. This classifier was recoded (permuted) in ETL incorrectly and this was the source of the inconsistency. Moreover, since this classifier was used in further computations (e.g. aggregating it with `max` operation), is was impossible to just relabel the output.

[2] srm – `http://manpages.ubuntu.com/manpages/trusty/man1/srm.1.html`
[3] Gnuplot – `http://www.gnuplot.info/`

		Local cluster		Production		Ratio	
Script	Description	A	B	A	Final	A	Final/B
1	Aggregation of education data	1 min 3 s	25 min	1 min 42 s	2 h	1.6	4.8
2	Aggregation of tax data (monthly income)	46 s	18 h 10 min	1 h 6 min 28 s	221 h 55 min	86.7	12.2
3	Aggregation of tax data (average yearly income)	4 min 34 s	1 h 55 min	6 min 35 s	15 h 14 min	1.4	7.9
4	Aggregation of tax data and joining the two datasets	32 s	32 min	50 s	4 h 15 min	1.6	8.0
5	Compiling the analysis table (shifting)	6 min 7 s	39 h 3 min	46 min 10 s	141 h 11 min	8.0	3.6
Sum		13 min 2 s	60 h 5 min	2 h 4 min 45 s	384 h 35 min	9.6	6.4

Table 7.2: Running times of the privacy-preserving ETL scripts on test data sets A and B (data taken from [88, Figure 6.5]) and the final imported data in production environment. The last columns show the running time ratios between production and testing environments.

Nevertheless, about half of the initially planned 52 figures were correct as they either did not use the status classifier or used a part of it that was not used for further computations and could still be relabelled with confidence. Moreover, the results from the SHAREMIND study were more accurate than the ones from the study made with traditional methods, as no records had to be removed from the data sets to provide privacy.

7.5 Best practices and lessons learned

7.5.1 Fault tolerance

One of the main problems during the analysis period was the need to restart the SHAREMIND service too often. From December 2014, Cybernetica's computation node was restarted 25 times. The main reason for restarting the service was the unexpected disconnection of the client application (RMIND) while an analysis script was running. The disconnects were most probably caused by the fact that CentAR used an internet service meant for individuals, while computation parties had a more strict service level agreement (SLA) with the ISP.

At this time, a client application disconnect during a SMC protocol run caused all of the three computation servers to enter a state where no new client nor computation party connections were accepted. Thus, all of the three servers had to be restarted and since they were hosted by different parties, it took time, sometimes more than a day. While gracefully failing the current computation upon a client disconnect would have saved the time spent on restarting the services, we feel that there is a better solution. In practical secure multi-party computation deployments, the client application should not be required to be online during the computation. Instead, it should be able to disconnect from the computation parties and leave the SMC process running on them. A client application should be able to reconnect and resume the session at any time to check if the computation has finished and retrieve the results. This requires a more complicated session management, but with practical analysis steps running two or more days, it is inevitable. A new session management with this feature was already being designed. However, as a fundamental component of both the SHAREMIND computation server and RMIND client application, we could not deliver it during the analysis without thorough testing and another code review by RMIT.

Moreover, it is possible to automatically recover from some network errors. For example, when connection to a computation party is restored, the computation can either continue from the point it was left off, or it can be rolled back to a known checkpoint and resumed from there. This kind of automatic recovery requires transactional tasks, either on RMIND script level or on individual SMC protocol level built into SHAREMIND. In terms of SMC, these are considered benign faults.

7.5.2 Performance tweaks

When testing the ETL process with test set B on our cluster, we realised that several optimisations are possible to enhance the performance. Most of these optimisations took advantage of vectorising the computation to save on the number of network communication rounds.

For example, the second step of ETL groups tax data by each individual and aggregates records in each such group into one record. Aggregating the values in each group sequentially takes a lot of time as there are many small groups. We optimised this aggregating functionality on two levels. First, for each column, the aggregations for all groups were done in parallel. Secondly, in each group, the aggregation function was applied by folding similar to a binary tree. Hence, it took $\lceil \log_2 m \rceil$ rounds of folding for a group with m records. For test set B, this optimisation saved us 40% of ETL step 2 time on our cluster.

Similarly, we enhanced RMIND with support for matrices so operations on several matrix columns could be carried out in parallel. Previously, matrices had to be used by defining and using each of its columns as independent vectors.

The tax data consists of floating point numbers that represent currency. As such, we are interested in no more than two decimal places after the comma for this data. Hence, instead of carrying out costly aggregation operations on secret-shared floating point values, we can multiply tax data with large enough constant and convert it to an integer value. Later, we can convert the aggregation results back to floating point numbers and divide them with the same constant.

Lastly, we optimised the RMIND scripts written by the analysts. As analysts are used to work on local data on their decently powerful computers, they tend to write inefficient code. For example, a special statistical analysis with access to the underlying data may be able to automatically cache results of identical expressions used in several places, whereas RMIND evaluates the expression again every time. We introduced temporary variables for several such expressions. For the same reason, the aggregated floating point tax data was rounded and cached in one go so it could be reused on several plots.

Nevertheless, as can be seen from Table 7.2 the ETL running time does not scale linearly when switching from local cluster to production environment deployed over the internet. Still, assessing the running time of the computation is a valuable input for planning in such projects. Hence, running the whole computation on a production-sized test data set in a production environment should be included in the project timeline as early as possible. This would also bring out the bottlenecks in the ETL scripts themselves – which operations are network bandwidth and which ones are latency bound. However, by the time the computation nodes were deployed and ETL scripts written, there was no time to run test set B in PRIST any more. Alternatively, there is a possibility to run the analysis process

in simulator that replaces all multi-party protocol calls with run-time estimates. This kind of simulator is being developed for the SHAREMIND platform, but currently the regression models are environment-specific, i.e. a model obtained from a local cluster cannot be used to predict the running time in a production environment with enough precision. More general regression models with high accuracy remain a future goal.

7.5.3 RMIND recode function

The incorrect recoding in the ETL that invalidated the student status classifier was introduced due to a human error. However, it could have been avoided by introducing a new feature in RMIND that got postponed due to other, higher-priority tasks.

The classifiers are generated by the data import tool by automatically assigning numerical identifiers to columns that are said to contain discrete values. For all such classifiers, the data import tool outputs the mapping to its log file. For example, for gender it may output: "1 – male, 2 – female". This information is then shared with the analyst who can use the correct classifiers in his RMIND scripts.

However, it is important to notice here that as classifiers are generated automatically, a classifier for the same column may have different output depending on which values are encountered first. For example, if the first row in the data set belongs to a female, the above classifier would be "1 – female, 2 – male" instead. To avoid changing all of the RMIND scripts using this classifier after the data import, a `recode` operation was introduced in RMIND. This operation allowed the analyst to map the classifier values generated by the data import tool to the values used in the scripts. As this was to be done only once, it was considered to be less error-prone. Unfortunately, as we had many different data sets (test sets A, B and the production data set), each having its own classifier mapping for the student status, a wrong mapping for the `recode` operation was used on production data set.

To avoid such human error in the future, the data import tool should save the generated classifiers as metadata together with the data values so RMIND could apply the correct mapping automatically. RMIND scripts could then use the human-readable labels instead of the generated classifier value, for example `table$gender=="male"` instead of `table$gender==1`.

7.6 Conclusion

PRIST is the first large-scale registry-based statistical study on linked data sets using secure multi-party computation technology. Although we were unable to

fulfil all of the stated analysis objectives due to both technical and human errors, we were able to carry out a full analysis process, where each involved party deployed and maintained their part of the required components. Each party – not just the computation parties, but also input and result parties – was in sole control of its environment and the trust relationships were securely formed by using an existing national PKI.

Moreover, in terms of data set sizes and running time, this study demonstrates that privacy-preserving studies using SMC technology on large databases are feasible.

CHAPTER 8

PRIVACY-PRESERVING DATA INTEGRATION ON FEDERATED DATABASES

8.1 Motivation

Studies like PRIST provide valuable information to steer government decisions. More so, if these studies are carried out on a regular basis. However, even though each subsequent study is a little bit easier, forming a consortium of data owners, computation parties and analysts can be cumbersome. Ideally, as in highly digitalised governments the data is already there, such studies should be semi-automatic.

On the other hand, people expect the government to protect their privacy, so it cannot let all researchers or analysts to freely combine and use the data sets. This requirement is also in line with Hippocratic databases – the idea that a database system should provide data security and keep privacy of the individuals, while still providing value to their user [68].

What the government needs is a data usage policy that is bundled with a specific data set and contains information who is allowed to access the data and what can be done with it. These policies should be "sticky" in a way that the derivatives of the corresponding data also are also covered by the same policy. To automate data sharing with usage policies, a technical solution to enforce such policies is needed. From a technical point of view, it is easy to see that the data usage policy has to be enforced by someone other than the end user. Given access to the data itself, an end user can just ignore its usage policy. For the same reason any usage policy that allows to access the original data is not enforceable in the presence of unauthorised access.

In an ideal world, we could think of a trusted third party (TTP) that has ac-

cess to the original data $\mathcal{D}$ together with its usage policy and responds to queries allowed by that policy. For example, if user A is allowed to perform function f on the data $\mathcal{D}$, the TTP responds with the function result. At the same time, running function g is disallowed for user A, as it is not explicitly white-listed (see Figure 8.1). In real world, the TTP is replaced by a cryptographic solution, e.g. secure multi-party computation.

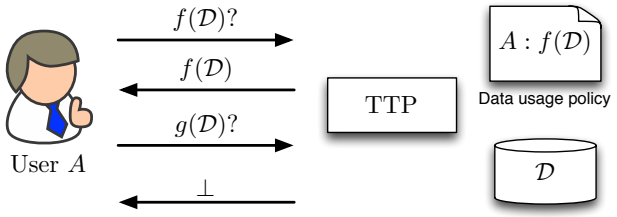


Figure 8.1: Enforcing a data usage policy with a trusted third party.

In this chapter, we discuss how secure multi-party computation can be used to enforce a data usage policy on federated databases. We will model the solution after two practical solutions: the SHAREMIND SMC framework and Cybernetica's Unified eXchange Platform. We extend the ideas initially proposed in [23].

8.2 The Unified eXchange Platform

8.2.1 Requirements for SMC

As discussed in Chapter 3, there are two main requirements for deploying secure multi-party computation – non-collusion of computation parties and secure authenticated communication channels. In Estonia, the latter requirement is fulfilled by a nation-wide data exchange layer X-Road that connects state databases. X-Road is a distributed service bus that allows information systems to securely use each other's services [127]. The communication is carried out over a secure mutually authenticated channel and all exchanged messages are digitally signed, providing a legally-binding long-term evidence value. In the context of SHAREMIND, we do not consider all SMC protocol messages exchanged between computation parties to be digitally signed as it would introduce a performance penalty. We propose to sign only messages between computation nodes and client applications (input and result parties).

In addition to government databases, many private organisations have also joined the X-Road. It was initially deployed in 2001 and by the end of 2015, Estonian X-Road connected 939 organisations and 219 databases. Over 485 million

requests were serviced by X-Road in 2015[1]. Estonian X-Road is based on the Unified eXchange Platform (UXP) developed by Cybernetica.

The other requirement, non-collusion of computation parties, can be enforced with either contractual obligations or a data owner assuming the role of one of the computation parties itself. The fact that UXP is distributed by design, suits well with this SMC requirement.

8.2.2 Status of privacy protection on UXP

Since 2011, Estonian X-Road (and UXP) incorporates a pseudonymisation feature that protects identities of individuals while still allowing to connect records from different databases belonging to the same individual [128]. Technically, it is based on the same idea as the privacy-preserving equi-join described in Chapter 5 – the key column values (e.g. personal ID codes) are encrypted with AES block cipher used as a pseudo-random permutation function. The common cipher key is generated by one of the database owners and distributed to others by encrypting it with their public key. Each party already has a key pair as the communication is done over TLS channels. However, this encryption does not protect from data owners as they have access to the AES cipher key.

In [128], Willemson also argues that pseudonymisation cannot be considered a strong security mechanism. It is provided as a convenient and simple measure to protect against trivial identification. Next, we will propose an architecture that combines secure multi-party computation with the aforementioned privacy-preserving join method to allow more secure database linking and analysis on UXP.

8.2.3 UXP components

In a usual service oriented architecture, in addition to service descriptions, parties must pairwise also decide on the security measures taken to protect the data transmission. As parties might have different IT capabilities, this might result in many different incompatible security configurations.

In UXP, this is covered by providing a standardised *security server* that takes care of the communication channel security – message encryption, authentication and time-stamping to provide long-term proof value. As shown on Figure 8.2, each participating party, whether a service provider or a consumer, has its own security server deployed in front of its service (on provider side) or information system (on consumer side). The security servers are almost transparent to the participants. The service and its consumer still exchange their protocol-specific

[1]Estonian X-Road statistics, `https://www.ria.ee/ee/x-tee-statistika.html`, accessed January 26, 2016.

messages. However, these messages are transferred over a TLS channel between the security servers for secure transmission over the internet. Moreover, security servers also take care of the inter-organisation authentication and service-level authorisation so the service itself does not have to know about external parties.

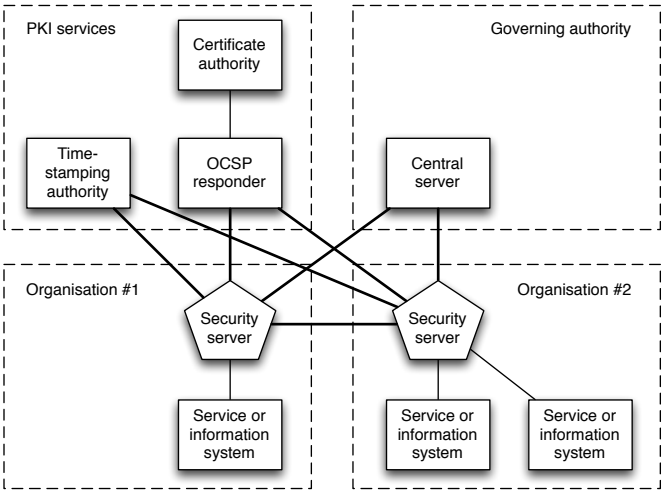


Figure 8.2: An Unified eXchange Platform (UXP) deployment with two organisations.

The security server software and its updates are provided by a separate *governing authority*. This allows to use a high-level security configuration throughout the setup. Moreover, the governing authority is responsible for registering new organisations to the UXP network, maintaining the list of current members (i.e. an address book) and monitoring the security servers. However, this does not mean that the governing authority is a communication bottleneck. Each security server caches contact information about relevant partner organisations locally and all of the service calls are made directly between the participating parties (provider and consumer). Such decentralised setup allows organisations to exchange messages securely even if other organisation servers are down or some more central services (the governing authority or trust services) are temporarily inaccessible.

For authentication and authorisation, UXP uses public key infrastructure (PKI). After registering at the governing authority, an organisation can request the certification authority (CA) to sign their keys. To obtain long-term proof value for the UXP service calls, every message exchanged between security servers is written to an audit log that is periodically time-stamped by an authorised time-stamping service. The audit log messages are chained together so it is not possible to insert or remove any intermediate messages once the log is time-stamped. The trust service providers (CA and time-stamping service) used by UXP organisations are authorised by the governing authority.

8.3 SHAREMIND as a UXP service

8.3.1 Roles and data flow

UXP security servers communicate using digitally signed XML-formatted messages that add a considerable amount of overhead to the actual payload. As such, this message format is not suitable for use in bandwidth-bound SMC protocols. Hence, as also mentioned above, we only consider SHAREMIND client application messages to be carried over UXP. This gives us traceable and signed data sharing requests as well as queries made on the secret-shared data. SHAREMIND computation nodes continue to use mutually authenticated encrypted TLS channels for communication between themselves.

We introduce a new role, *Customer*, the user who requests the data owners to import their data to a SHAREMIND deployment. From the SMC perspective, *Customer* is not the same as input party as it does not access the original data. The *Customer* may be the same person or organisation as the result party.

The process (see Figure 8.3) is started by the *Customer* who knows what kind of input data is needed. *Customer* assembles a data model description containing information about which data records from which input parties. This data model description together with the identity (public key) of the result party is sent to any of the relevant input parties via UXP.

If the input party receiving the request accepts it[2], it generates a (random) identifier for this request and forwards the signed request together with the generated ID to all other involved input parties. First, they collaboratively choose a suitable online SHAREMIND deployment to use. This can be implemented by using preference lists at each security server. On secret sharing, the involved input parties mutually agree on the first common SHAREMIND deployment from their lists. The *Customer* making the input request may also suggest a SHAREMIND deployment, but ultimately the decision should be up to the input parties that own and are responsible for the data. If the data usage policy of an input party requires tighter control over the computation process, it may request to be one of the computation nodes. It can then halt the computation process at any time if it sees any suspicious activity.

Each input party's security server uses one of its services to retrieve the relevant data set from its database and, using the provided data model description, secret shares each value in this data set. The shares are distributed among the computation nodes of the chosen SHAREMIND deployment under a common data store identified by the generated request ID. Each input party may also attach a data usage policy with their uploaded shares. This policy contains which oper-

[2]We recognise, that in some cases, depending on the requested data, accepting the request may involve manual review by an authorised official.

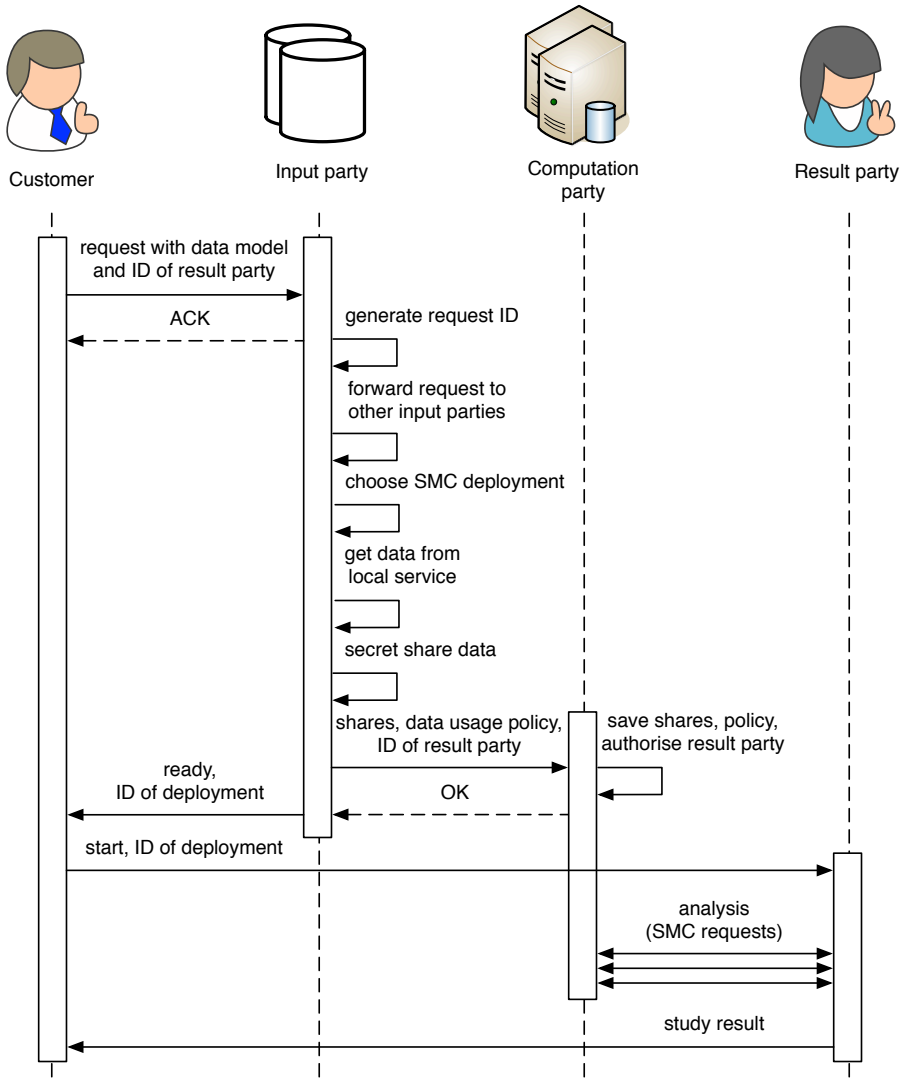


Figure 8.3: Sequence diagram of SHAREMIND SMC system as a service on the Unified eXchange Platform. All messages exchanged between different parties are carried over UXP. For messages between result party and computation parties, this is optional.

ations and by whom are allowed on this data set. As long as at least one of the computation parties honours the policy, no unauthorised data access can occur. This provides a cryptographic enforcement of data usage policies. Moreover, the identity of the authorised result party is also forwarded to the computation nodes so they can grant access to the corresponding certificate holder. Each input party notifies the *Customer* of the completion together with the contact information of the chosen SHAREMIND deployment.

A result party can now use a SHAREMIND client application to connect to the computation nodes through its UXP security server and issue commands. For example, the result party can use RMIND to securely link together uploaded data sets and perform statistical analysis on them. However, only operations allowed by the attached data usage policies are executed by the computation nodes. Auditing capability is added by the security servers in front of computation nodes logging and time-stamping all incoming queries from the result party.

8.3.2 A hybrid setup

Consider a scenario, where the result party is not able to deploy its own UXP security server, e.g. for the lack of technical capability. For simple use cases where specialised analysis software like RMIND is not needed, the result party could use a common UXP security server with a web interface that provides access to the services provided by other organisations on UXP. This web interface, called *portal* in the UXP terminology, takes care of user authentication and forwards his identity to the requested service. In fact, an UXP portal is also used by the *Customer* for the initial data import request.

However, a common security server may not be an option if the requests done by the result party contain private parameters or the secret-shared computation result contains sensitive data. In this case, we may also consider a scenario, where the result party communicates directly with the computation parties using SHAREMIND protocol without the UXP security servers on either ends. Since there are no security servers in front of computation parties, the security servers of input parties also communicate with computation parties directly using the SHAREMIND protocol. Note that we do not consider a scenario where we omit security servers of the input parties, as we only consider existing UXP organisations to be in this role.

With the UXP security servers only at the input parties and the *Customer*, only the initial data import requests are signed and provide long-term proof value as provided by the UXP. The rest of the communication is also secure, as SHAREMIND uses mutually authenticated TLS channels for exchanging messages with the client applications. All incoming messages, including data imports and query requests with parameter values, are independently logged by each computation

party and may be later compared to solve any disputes. Moreover, if this type of hybrid deployment proves practical, it is possible to enforce message signing by the client applications and introduce time-stamping of computation party log entries for long-term proof value.

8.3.3 PRIST as a service

As an example, let us consider how the PRIST study described in Chapter 7 would be conducted if SHAREMIND is available as a service on Estonian X-Road. The Estonian Association of Information Technology and Telecommunications (ITL) is in the role of the *Customer*, as it is the party interested in the results of the study. As ITL itself does not have the required competence to perform statistical studies, the role of result party is delegated to statisticians at CentAR.

First, CentAR compiles a data model concerning necessary fields from the input databases – the Education Information System managed by the Ministry of Education and Research (HTM), and tax records held by the Tax and Customs Board (EMTA). This data model is given to ITL who then uses its UXP security server or a web portal to submit a database linking request to one of the input parties, e.g. the Tax and Customs Board, together with the identity (certificate containing a public key) of the authorised result party CentAR.

EMTA forwards the request to the other input party, HTM, and together they choose a suitable SHAREMIND deployment. Most probably, EMTA is interested in hosting one of the computation nodes itself as it requires tighter control over its data and can then review the analysis requests made by the result party in real time. The Information System Authority (RIA) would be a good candidate for hosting a second computation node. RIA coordinates the development and administration of the national information system and is, among other roles, the governing authority for the X-Road. As such it has the necessary IT-capability to host a generic SHAREMIND node that can be used in most of the deployments. The third computation node could be hosted by any of the other involved organisations or even a private company like Cybernetica.

Note that no distribution of application binaries or code review is needed at this point as the necessary software is already deployed. We envision that the X-Road governing authority is responsible for distributing SHAREMIND Application Server software to the computation party hosts. This would provide maximal compatibility as the X-Road governing authority also distributes and updates the UXP security server software. Moreover, if the result party and computation parties communicate through UXP security servers then no separate key exchange is necessary as the key pairs of security servers may be used. Otherwise, the computation nodes receive the key of the result party with the shares and the result party can use an UXP portal to obtain keys of computation parties.

As the next step, the security servers of both input parties load the requested data set and secret share it among the computation nodes of the chosen SHAREMIND deployment. Each computation party also registers CentAR's certificate to grant them access to the given secret-shared data store. EMTA and HTM both notify the ITL (and CentAR) that the requested data is ready for analysis.

CentAR uses RMIND to connect to the computation parties and carry out the analysis. Each request is logged by each computation party and if UXP security servers are used then each request is also signed and time-stamped providing long-term proof value.

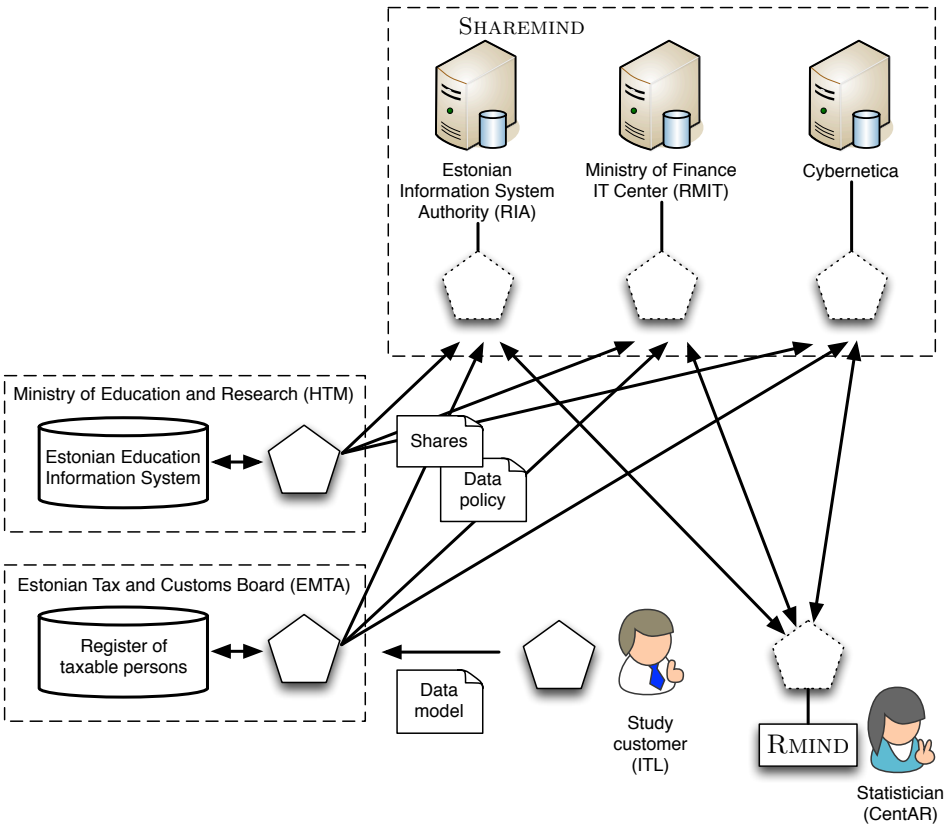


Figure 8.4: PRIST study using SHAREMIND as a service on Unified eXchange Platform. Pentagrams are UXP security servers, dotted lines denote security servers that are omitted in the hybrid deployment.

It is easy to see that once the necessary data models are prepared by CentAR, it is possible to carry out such analysis every year without requesting the input parties to export their data set and use a separate application to secret share it. Moreover, as SHAREMIND computation nodes are provided as service, they can

be requested dynamically. Finally, if the analysis process is more-or-less the same each year, the whole process can be automated so ITL just receives the report every year.

8.3.4 Final considerations

In general, deploying secure multi-party technology on a federated database infrastructure like UXP provides us with means to link data sets while cryptographically enforcing data usage policies. Providing SHAREMIND computation node as a service on UXP removes the necessity to search for application-specific set of computation parties and sign a contract between them and the input parties. Moreover, there is no need for a key exchange or it can be bootstrapped on existing UXP trust network. Simpler or recurring data analysis adhering to the predefined data usage policies can be carried out automatically without the need for separate data exports and manual intervention by the officials at input parties.

However, note that this solution does not protect against an analyst who cleverly combines different combinations of query parameters allowed by the data usage policy to extract individual values from the data sets. Such behaviour can be discouraged by reviewing the request logs at the computation nodes, or mitigated by using differential privacy if inexact results are acceptable.

CONCLUSION

This thesis studies technical and trust issues related to applying secure multi-party computation technology in practice. We start by examining the deployments of two practical real-world secure multi-party computation applications – the Danish sugar beet auction from 2008 and the Estonian ITL financial benchmarking application from 2011. We bring out shortcomings that may impede using the technology in other, more complex, deployments. For example, at the time we were lacking an oblivious sorting implementation and a capability to link existing data sets together in a privacy-preserving way. The thesis includes previously published work by the author that addresses those problems.

As a separate topic, we examine the challenges specific to deploying secure multi-party computation in web-based applications. As more and more applications are moving to the cloud, web-based user interfaces for interacting with the service become important. We take the ITL financial benchmarking application as a baseline and describe how secret sharing and interacting with computation nodes is possible from a web browser. A short overview of how the web-based technologies have evolved reveals that with HTML5 and its supporting technologies, web-based frontends for SMC applications for inputting data and receiving results are now easier to implement.

The main result of this thesis is a life cycle overview of the world's first large-scale registry-based statistical study conducted using secure multi-party computation technology. We give an overview of both the technical and organisational side and bring out issues that are related to deploying SMC for large organisations like governmental institutions. For example, to the best of our knowledge, the PRIST study is the first SMC application, where trust relations and responsibilities between the non-colluding computation parties were formally established by contracts. We also report on the technical shortcomings and human errors that occurred during this study, and give pointers on how to mitigate similar problems in future deployments.

Finally, we propose to deploy secure multi-party computation as a service on a federated data exchange infrastructure. This allows institutions to share their data with others with the guarantee that attached usage policies are cryptographically

enforced. Such a deployment provides a way to make data-oriented decisions quicker and leads to a more informed and reactive government. As an example, we show how the SHAREMIND secure multi-party computation system can be integrated with X-Road, the Estonian government information exchange backbone.

Bibliography

[1] Isikuandmete kaitse seadus. Passed 15.02.2007 - RT I 2007, 24, 127; RT I, 12.07.2014, 51, Personal Data Protection Act, English translation available at https://www.riigiteataja.ee/en/eli/509072014018/consolide

[2] Maksukorralduse seadus. Passed 20.02.2002 - RT I 2002, 26, 150; RT I, 11.07.2014, 11, Taxation Act, English translation available at https://www.riigiteataja.ee/en/eli/501092014002/consolide

[3] Agrawal, R., Evfimievski, A., Srikant, R.: Information sharing across private databases. In: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data. pp. 86–97. SIGMOD '03, ACM, New York, NY, USA (2003)

[4] Agrawal, R., Kiernan, J., Srikant, R., Xu, Y.: Order preserving encryption for numeric data. In: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data. pp. 563–574. SIGMOD '04, ACM, New York, NY, USA (2004)

[5] Andmekaitse Inspektsioon: Taotluse läbi vaatamata jätmise teade. 2.2.-7/13/557r (in Estonian). Available at http://adr.rik.ee/aki/dokument/3679385 (January 2014)

[6] Archer, D.W., Rohloff, K.: Computing with Data Privacy: Steps toward Realization. IEEE Security & Privacy 13(1), 22–29 (2015)

[7] Aumasson, J.P., Fischer, S., Khazaei, S., Meier, W., Rechberger, C.: New Features of Latin Dances: Analysis of Salsa, ChaCha, and Rumba. In: Fast Software Encryption, Lecture Notes in Computer Science, vol. 5086, pp. 470–488. Springer (2008)

[8] Babbage, S., Catalano, D., Cid, C., de Weger, B., Dunkelman, O., Gehrmann, C., Granboulan, L., Güneysu, T., Hermans, J., Lange, T., Lenstra, A., Mitchell, C., Näslund, M., Nguyen, P., Paar, C., Paterson, K.,

Pelzl, J., Pornin, T., Preneel, B., Rechberger, C., Rijmen, V., Robshaw, M., Rupp, A., Schläffer, M., Vaudenay, S., Vercauteren, F., Ward, M.: ECRYPT II Yearly Report on Algorithms and Keysizes (2011–2012). Tech. rep., European Network of Excellence in Cryptology II (Sep 2012), http://www.ecrypt.eu.org/ecrypt2/documents/D.SPA.20.pdf

[9] Barbaro, M., Zeller, T.: A face is exposed for AOL searcher no. 4417749. New York Times (August 2006)

[10] Barker, E., Kelsey, J.: Recommendation for Random Number Generation Using Deterministic Random Bit Generators. NIST Special Publication 800-90A (2012)

[11] Batcher, K.E.: Sorting networks and their applications. In: Proceedings of the April 30–May 2, 1968, Spring joint computer conference. pp. 307–314. AFIPS '68 (Spring), ACM, New York, NY, USA (1968)

[12] Beaver, D., Micali, S., Rogaway, P.: The round complexity of secure protocols. In: Proceedings of the Twenty-second Annual ACM Symposium on Theory of Computing. pp. 503–513. STOC '90, ACM, New York, NY, USA (1990)

[13] Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Advances in Cryptology - CRYPTO '91, Lecture Notes in Computer Science, vol. 576, pp. 420–432. Springer (1992)

[14] Bellare, M., Hoang, V.T., Rogaway, P.: Foundations of garbled circuits. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security. pp. 784–796. CCS '12, ACM, New York, NY, USA (2012)

[15] Ben-David, A., Nisan, N., Pinkas, B.: Fairplaymp: A system for secure multi-party computation. In: Proceedings of the 15th ACM Conference on Computer and Communications Security. pp. 257–266. CCS '08, ACM, New York, NY, USA (2008)

[16] Bernstein, D.: ChaCha, a variant of Salsa20. http://cr.yp.to/chacha.html (2008)

[17] Blakley, G.: Safeguarding cryptographic keys. In: Proceedings of the 1979 AFIPS National Computer Conference. pp. 313–317. AFIPS Press, Monval, NJ, USA (1979)

[18] Bogdanov, D.: How to securely perform computations on secret-shared data. Master's thesis, University of Tartu (2007)

[19] Bogdanov, D.: Sharemind: programmable secure computations with practical applications. Ph.D. thesis, University of Tartu (2013), http://hdl.handle.net/10062/29041

[20] Bogdanov, D., Jõemets, M., Siim, S., Vaht, M.: How the Estonian Tax and Customs Board Evaluated a Tax Fraud Detection System Based on Secure Multi-party Computation. In: Financial Cryptography and Data Security - 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers. Lecture Notes in Computer Science, Springer (2015)

[21] Bogdanov, D., Kamm, L.: Constructing Privacy-Preserving Information Systems Using Secure Multiparty Computation. Tech. Rep. T-4-13, Cybernetica, http://cyber.ee/en/research/publications/. (2011)

[22] Bogdanov, D., Kamm, L., Kubo, B., Rebane, R., Sokk, V., Talviste, R.: Students and Taxes: a Privacy-Preserving Social Study Using Secure Computation. Proceedings on Privacy Enhancing Technologies (PoPETs) 2016(3) (2016), (to appear)

[23] Bogdanov, D., Kamm, L., Laur, S., Pruulmann-Vengerfeldt, P., Talviste, R., Willemson, J.: Privacy-preserving statistical data analysis on federated databases. In: Proceedings of the Annual Privacy Forum. APF'14. Lecture Notes in Computer Science, vol. 8450, pp. 30–55. Springer (2014)

[24] Bogdanov, D., Kamm, L., Laur, S., Sokk, V.: Rmind: a tool for cryptographically secure statistical analysis. Cryptology ePrint Archive, Report 2014/512 (2014), http://eprint.iacr.org/

[25] Bogdanov, D., Laud, P., Laur, S., Pullonen, P.: From input private to universally composable secure multi-party computation primitives. In: IEEE 27th Computer Security Foundations Symposium, CSF 2014. pp. 184–198. IEEE (July 2014)

[26] Bogdanov, D., Laud, P., Randmets, J.: Domain-Polymorphic Programming of Privacy-Preserving Applications. In: Proceedings of the First ACM Workshop on Language Support for Privacy-enhancing Technologies, PETShop '13. pp. 23–26. ACM Digital Library, ACM (2013)

[27] Bogdanov, D., Laur, S., Talviste, R.: Oblivious Sorting of Secret-Shared Data. Tech. Rep. T-4-19, Cybernetica, http://cyber.ee/en/research/publications/. (2013)

[28] Bogdanov, D., Laur, S., Talviste, R.: A Practical Analysis of Oblivious Sorting Algorithms for Secure Multi-party Computation. In: Proceedings of the 19th Nordic Conference on Secure IT Systems. NordSec'14, Lecture Notes in Computer Science, vol. 8788, pp. 59–74. Springer (2014)

[29] Bogdanov, D., Laur, S., Willemson, J.: Sharemind: A Framework for Fast Privacy-Preserving Computations. In: Proceedings of the 13th European Symposium on Research in Computer Security - ESORICS'08. Lecture Notes in Computer Science, vol. 5283, pp. 192–206. Springer (2008)

[30] Bogdanov, D., Niitsoo, M., Toft, T., Willemson, J.: High-performance secure multi-party computation for data mining applications. International Journal of Information Security 11(6), 403–418 (2012)

[31] Bogdanov, D., Talviste, R., Willemson, J.: Deploying secure multi-party computation for financial data analysis (short paper). In: Proceedings of the 16th International Conference on Financial Cryptography and Data Security. FC'12. Lecture Notes in Computer Science, vol. 7397, pp. 57–64. Springer (2012)

[32] Bogetoft, P., Christensen, D., Damgård, I., Geisler, M., Jakobsen, T., Krøigaard, M., Nielsen, J., Nielsen, J., Nielsen, K., Pagter, J., Schwartzbach, M., Toft, T.: Secure multiparty computation goes live. In: Financial Cryptography and Data Security, Lecture Notes in Computer Science, vol. 5628, pp. 325–343. Springer (2009)

[33] Boldyreva, A., Chenette, N., Lee, Y., O'Neill, A.: Order-preserving symmetric encryption. In: Advances in Cryptology - EUROCRYPT 2009, Lecture Notes in Computer Science, vol. 5479, pp. 224–241. Springer (2009)

[34] Boneh, D., Goh, E.J., Nissim, K.: Evaluating 2-dnf formulas on ciphertexts. In: Theory of Cryptography, Lecture Notes in Computer Science, vol. 3378, pp. 325–341. Springer (2005)

[35] Boyar, J., Peralta, R.: A New Combinational Logic Minimization Technique with Applications to Cryptology. In: Experimental Algorithms, Lecture Notes in Computer Science, vol. 6049, pp. 178–189. Springer (2010)

[36] Boyar, J., Peralta, R.: A Small Depth-16 Circuit for the AES S-Box. In: Information Security and Privacy Research, SEC. IFIP Advances in Information and Communication Technology, vol. 376, pp. 287–298. Springer (2012)

[37] Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) Fully Homomorphic Encryption Without Bootstrapping. In: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference. pp. 309–325. ITCS '12, ACM, New York, NY, USA (2012)

[38] Burkhart, M., Strasser, M., Many, D., Dimitropoulos, X.: SEPIA: privacy-preserving aggregation of multi-domain network events and statistics. In: Proc. of USENIX conference on Security. USENIX Association (2010)

[39] Canetti, R.: Security and composition of multiparty cryptographic protocols. Journal of Cryptology 13(1), 143–202 (2000)

[40] Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067 (2000), http://eprint.iacr.org/

[41] Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: Proceedings of 42nd IEEE Symposium on Foundations of Computer Science, FOCS. pp. 136–145 (2001)

[42] Carter, L., Wegman, M.N.: Universal Classes of Hash Functions. J. Comput. Syst. Sci. 18(2), 143–154 (1979)

[43] Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols (abstract). In: Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings. Lecture Notes in Computer Science, vol. 293, p. 462. Springer (1987)

[44] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, chap. 8.2 Counting Sort, pp. 168–170. MIT Press and McGraw-Hill, 2nd edn. (2001)

[45] Cramer, R., Damgård, I.B., Nielsen, J.B.: Secure Multiparty Computation and Secret Sharing. Cambridge University Press (2015)

[46] Damgård, I., Damgård, K., Nielsen, K., Nordholt, P.S., Toft, T.: Confidential benchmarking based on multiparty computation. Cryptology ePrint Archive, Report 2015/1006 (2015), http://eprint.iacr.org/

[47] Damgård, I., Fitzi, M., Kiltz, E., Nielsen, J.B., Toft, T.: Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In: Third Theory of Cryptography Conference, TCC. Lecture Notes in Computer Science, vol. 3876, pp. 285–304. Springer (2006)

[48] Damgård, I., Geisler, M., Krøigaard, M., Nielsen, J.B.: Asynchronous multiparty computation: Theory and implementation. In: Public Key Cryptography – PKC 2009, Lecture Notes in Computer Science, vol. 5443, pp. 160–179. Springer (2009)

[49] Damgård, I., Keller, M.: Secure multiparty AES. In: Financial Cryptography and Data Security. Lecture Notes in Computer Science, vol. 6052, pp. 367–374. Springer (2010)

[50] Damgård, I., Keller, M., Larraia, E., Miles, C., Smart, N.P.: Implementing AES via an Actively/Covertly Secure Dishonest-Majority MPC Protocol. In: Security and Cryptography for Networks, Lecture Notes in Computer Science, vol. 7485, pp. 241–263. Springer (2012)

[51] Damgård, I., Pastro, V., Smart, N., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Advances in Cryptology – CRYPTO 2012, Lecture Notes in Computer Science, vol. 7417, pp. 643–662. Springer (2012)

[52] Demmler, D., Schneider, T., Zohner, M.: ABY - A framework for efficient mixed-protocol secure two-party computation. In: 22nd Annual Network and Distributed System Security Symposium, NDSS 2015. The Internet Society (2015)

[53] Dwork, C.: Differential privacy. In: Proceedings of ICALP 2006. Lecture Notes in Computer Science, vol. 4052, pp. 1–12. Springer (2006)

[54] Edmonds, J.: How to Think about Algorithms, chap. 5.2 Counting Sort (a Stable Sort), pp. 72–75. Cambridge University Press (2008)

[55] Fette, I., Melnikov, A.: The WebSocket Protocol. RFC 6455 (Proposed Standard) (Dec 2011), `http://www.ietf.org/rfc/rfc6455.txt`

[56] Forster, F.: libsortnetwork. Published online at `http://verplant.org/libsortnetwork/` (June 2011), accessed December 16, 2015

[57] Frederiksen, T.K., Jakobsen, T.P., Nielsen, J.B.: Faster Maliciously Secure Two-Party Computation Using the GPU. In: Security and Cryptography for Networks, Lecture Notes in Computer Science, vol. 8642, pp. 358–379. Springer International Publishing (2014)

[58] Freedman, M., Nissim, K., Pinkas, B.: Efficient Private Matching and Set Intersection. In: Advances in Cryptology – EUROCRYPT 2004, Lecture Notes in Computer Science, vol. 3027, pp. 1–19. Springer (2004)

[59] Gentry, C.: A fully homomorphic encryption scheme. Ph.D. thesis, Stanford University (2009), https://crypto.stanford.edu/craig

[60] Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing. pp. 169–178. STOC '09, ACM, New York, NY, USA (2009)

[61] Gentry, C., Halevi, S., Smart, N.P.: Better bootstrapping in fully homomorphic encryption. In: Public Key Cryptography – PKC 2012, Lecture Notes in Computer Science, vol. 7293, pp. 1–16. Springer (2012)

[62] Gentry, C., Halevi, S., Smart, N.P.: Fully homomorphic encryption with polylog overhead. In: Advances in Cryptology – EUROCRYPT 2012, Lecture Notes in Computer Science, vol. 7237, pp. 465–482. Springer (2012)

[63] Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the AES circuit. In: Advances in Cryptology – CRYPTO 2012, Lecture Notes in Computer Science, vol. 7417, pp. 850–867. Springer (2012)

[64] Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the AES circuit (updated implementation). Cryptology ePrint Archive, Report 2012/099 (2012), http://eprint.iacr.org/

[65] Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: Advances in Cryptology - CRYPTO 2013, Lecture Notes in Computer Science, vol. 8042, pp. 75–92. Springer (2013)

[66] Goh, E.J.: Secure indexes. Cryptology ePrint Archive, Report 2003/216 (2003), http://eprint.iacr.org/

[67] Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game. In: Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing. pp. 218–229. STOC '87, ACM, New York, NY, USA (1987)

[68] Grandison, T., Johnson, C., Kiernan, J.: Hippocratic databases: Current capabilities and future trends. In: Handbook of Database Security, pp. 409–429. Springer US (2008)

[69] Grofig, P., Härterich, M., Hang, I., Kerschbaum, F., Kohler, M., Schaad, A., Schröpfer, A., Tighzert, W.: Experiences and observations on the industrial implementation of a system to search over outsourced encrypted data. In: Sicherheit 2014: Sicherheit, Schutz und Zuverlässigkeit, Beiträge der 7. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft

für Informatik e.V. (GI), 19.-21. März 2014, Wien, Österreich. LNI, vol. 228, pp. 115–125. GI (2014), http://subs.emis.de/LNI/Proceedings/Proceedings228/article7.html

[70] Hadavi, M.A., Jalili, R., Damiani, E., Cimato, S.: Security and searchability in secret sharing-based data outsourcing. International Journal of Information Security 14(6), 513–529 (2015)

[71] Hamada, K., Ikarashi, D., Chida, K., Takahashi, K.: Oblivious radix sort: An efficient sorting algorithm for practical secure multi-party computation. Cryptology ePrint Archive, Report 2014/121 (2014), http://eprint.iacr.org/

[72] Hamada, K., Kikuchi, R., Ikarashi, D., Chida, K., Takahashi, K.: Practically Efficient Multi-party Sorting Protocols from Comparison Sort Algorithms. In: Proc. of ICISC'12, Lecture Notes in Computer Science, vol. 7839, pp. 202–216. Springer (2013)

[73] Hazay, C., Lindell, Y.: Constructions of truly practical secure protocols using standardsmartcards. In: ACM Conference on Computer and Communications Security. pp. 491–500 (2008)

[74] The transaction information system for 1000-euro purchases will be completed in November. *Ärileht*, Oct 22, 2013. http://arileht.delfi.ee/news/uudised?id=66955998 (in Estonian). Last accessed: Sept 5, 2014.

[75] Henecka, W., K ögl, S., Sadeghi, A.R., Schneider, T., Wehrenberg, I.: TASTY: Tool for Automating Secure Two-party Computations. In: Proceedings of the 17th ACM Conference on Computer and Communications Security. pp. 451–462. CCS '10, ACM, New York, NY, USA (2010)

[76] Hollerith, H.: US395781 (A) - ART OF COMPILING STATISTICS. European Patent Office (1889), http://worldwide.espacenet.com/publicationDetails/biblio?CC=US&NR=395781

[77] Huang, Y., Evans, D., Katz, J., Malka, L.: Faster Secure Two-party Computation Using Garbled Circuits. In: Proceedings of the 20th USENIX Conference on Security. pp. 35–35. SEC'11, USENIX Association, Berkeley, CA, USA (2011)

[78] Huang, Y., Katz, J., Evans, D.: Quid-pro-quo-tocols: Strengthening semi-honest protocols with dual execution. In: Security and Privacy (SP), 2012 IEEE Symposium on. pp. 272–284 (May 2012)

[79] Ippolito, B.: Remote JSON – JSONP. Published online at http://bob.ippoli.to/archives/2005/12/05/remote-json-jsonp/ (Dec 2005), accessed December 16, 2015

[80] Ishai, Y., Paskin, A.: Evaluating branching programs on encrypted data. In: 4th Theory of Cryptography Conference, TCC. Lecture Notes in Computer Science, vol. 4392, pp. 575–594. Springer (2007)

[81] Ishiguro, T., Kiyomoto, S., Miyake, Y.: Latin Dances Revisited: New Analytic Results of Salsa20 and ChaCha. In: Information and Communications Security, Lecture Notes in Computer Science, vol. 7043, pp. 255–266. Springer (2011)

[82] ISO: ISO/IEC 14882:2011 Information technology — Programming languages — C++. International Organization for Standardization (Feb 2012), http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=50372

[83] Jagomägi, T.: Private communication with an ITL board member (2013)

[84] Jagomägis, R.: SecreC: a Privacy-Aware Programming Language with Applications in Data Mining. Master's thesis, Institute of Computer Science, University of Tartu (2010)

[85] Jagomägis, R., Jensen, J.U., Jõemets, M., Nielsen, K., Nordholt, P.S., Rebane, R., Vaht, M.: D23.1 – Secure Survey Prototype. Confidential deliverable of the PRACTICE (Privacy-Preserving Computation in the Cloud) project (2015)

[86] Jawurek, M., Johns, M., Rieck, K.: Smart metering de-pseudonymization. In: Proceedings of the 27th Annual Computer Security Applications Conference. pp. 227–236. ACSAC '11, ACM, New York, NY, USA (2011)

[87] Jónsson, K.V., Kreitz, G., Uddin, M.: Secure Multi-Party Sorting and Applications. Cryptology ePrint Archive, Report 2011/122 (2011), http://eprint.iacr.org/

[88] Kamm, L.: Privacy-preserving statistical analysis using secure multi-party computation. Ph.D. thesis, University of Tartu (2015), http://hdl.handle.net/10062/45343

[89] Kamm, L., Willemson, J.: Secure Floating-Point Arithmetic and Private Satellite Collision Analysis. International Journal of Information Security pp. 1–18 (2014)

[90] Keller, M., Scholl, P., Smart, N.P.: An architecture for practical actively secure mpc with dishonest majority. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security. pp. 549–560. CCS '13, ACM, New York, NY, USA (2013)

[91] Knuth, D.E.: The art of computer programming, volume 3: (2nd ed.) sorting and searching. Addison Wesley Longman Publishing Co., Inc., USA (1998)

[92] Kreuter, B., Shelat, A., Shen, C.H.: Billion-gate secure computation with malicious adversaries. In: Proceedings of the 21st USENIX Conference on Security Symposium. pp. 14–14. Security'12, USENIX Association, Berkeley, CA, USA (2012), http://dl.acm.org/citation.cfm?id=2362793.2362807

[93] Krips, T., Willemson, J.: Hybrid model of fixed and floating point numbers in secure multiparty computations. In: Proceedings of the 17th International Information Security Conference, ISC 2014, Lecture Notes in Computer Science, vol. 8783, pp. 179–197. Springer (2014)

[94] Lane, J., Heus, P., Mulcahy, T.: Data Access in a Cyber World: Making Use of Cyberinfrastructure. Transactions on Data Privacy 1(1), 2–16 (2008)

[95] Launchbury, J., Diatchki, I.S., DuBuisson, T., Adams-Moran, A.: Efficient lookup-table protocol in secure multiparty computation. In: ACM SIGPLAN International Conference on Functional Programming, ICFP'12. pp. 189–200. ACM (2012)

[96] Laur, S., Talviste, R., Willemson, J.: From Oblivious AES to Efficient and Secure Database Join in the Multiparty Setting. In: Applied Cryptography and Network Security. ACNS'13, Lecture Notes in Computer Science, vol. 7954, pp. 84–101. Springer (2013)

[97] Laur, S., Talviste, R., Willemson, J.: From oblivious aes to efficient and secure database join in the multiparty setting. Cryptology ePrint Archive, Report 2013/203 (2013), http://eprint.iacr.org/

[98] Laur, S., Willemson, J., Zhang, B.: Round-Efficient Oblivious Database Manipulation. In: Information Security, Lecture Notes in Computer Science, vol. 7001, pp. 262–277. Springer (2011)

[99] Li, N., Li, T., Venkatasubramanian, S.: t-closeness: Privacy Beyond k-anonymity and ℓ-diversity. In: Proceedings of ICDE 2007 (2007)

[100] Lindell, Y., Pinkas, B.: A Proof of Security of Yao's Protocol for Two-Party Computation. Journal of Cryptology 22(2), 161–188 (2009)

[101] Machanavajjhala, A., Kifer, D., Gehrke, J., Venkitasubramaniam, M.: L-diversity: Privacy beyond k-anonymity. ACM Transactions on Knowledge Discovery from Data (TKDD) 1(1) (Mar 2007)

[102] Malka, L.: Vmcrypt: Modular software architecture for scalable secure computation. In: Proceedings of the 18th ACM Conference on Computer and Communications Security. pp. 715–724. CCS '11, ACM, New York, NY, USA (2011)

[103] Malkhi, D., Nisan, N., Pinkas, B., Sella, Y.: Fairplay—a secure two-party computation system. In: Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13. pp. 20–20. SSYM'04, USENIX Association, Berkeley, CA, USA (2004), http://dl.acm.org/citation.cfm?id=1251375.1251395

[104] Naor, M., Pinkas, B., Sumner, R.: Privacy preserving auctions and mechanism design. In: Proceedings of the 1st ACM Conference on Electronic Commerce. pp. 129–139. EC '99, ACM, New York, NY, USA (1999)

[105] Narayanan, A., Shmatikov, V.: Robust de-anonymization of large sparse datasets. In: Proceedings of IEEE S&P '08. pp. 111–125 (2008)

[106] National Institute of Standards and Technology (NIST): Advanced Encryption Standard (AES). Federal Information Processing Standards Publications FIPS-197 (2001)

[107] Nielsen, J., Nordholt, P., Orlandi, C., Burra, S.: A new approach to practical active-secure two-party computation. In: Advances in Cryptology - CRYPTO 2012, Lecture Notes in Computer Science, vol. 7417, pp. 681–700. Springer (2012)

[108] Nielsen, K.: Private communication with the authors of the Danish sugar beet auction (2015)

[109] Nir, Y., Langley, A.: ChaCha20 and Poly1305 for IETF Protocols. RFC 7539 (Informational) (May 2015), http://www.ietf.org/rfc/rfc7539.txt

[110] Pfitzmann, B., Waidner, M.: Composition and integrity preservation of secure reactive systems. In: Proceedings of the 7th ACM Conference on Computer and Communications Security, CCS. pp. 245–254. ACM (2000)

[111] Pikma, T.: Auditing of Secure Multiparty Computations. Master's thesis, Institute of Computer Science, University of Tartu (2014)

[112] Pinkas, B., Schneider, T., Smart, N., Williams, S.: Secure two-party computation is practical. In: Advances in Cryptology – ASIACRYPT 2009, Lecture Notes in Computer Science, vol. 5912, pp. 250–267. Springer (2009)

[113] Popa, R.A., Redfield, C.M.S., Zeldovich, N., Balakrishnan, H.: CryptDB: Protecting Confidentiality with Encrypted Query Processing. In: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles. pp. 85–100. SOSP '11, ACM, New York, NY, USA (2011)

[114] Pullonen, P., Siim, S.: Combining Secret Sharing and Garbled Circuits for Efficient Private IEEE 754 Floating-Point Computations. In: Financial Cryptography and Data Security - FC 2015 Workshops, BITCOIN, WAHC and Wearable 2015, San Juan, Puerto Rico, January 30, 2015, Revised Selected Papers, Lecture Notes in Computer Science, vol. 8976, pp. 172–183. Springer (2015)

[115] R Core Team: R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria (2014), `http://www.R-project.org/`

[116] Shamir, A.: How to share a secret. Communications of the ACM 22, 612–613 (1979)

[117] Shi, Z., Zhang, B., Feng, D., Wu, W.: Improved Key Recovery Attacks on Reduced-Round Salsa20 and ChaCha. In: Information Security and Cryptology – ICISC 2012, Lecture Notes in Computer Science, vol. 7839, pp. 337–351. Springer (2013)

[118] Smart, N., Vercauteren, F.: Fully homomorphic SIMD operations. Designs, Codes and Cryptography 71(1), 57–81 (2014)

[119] Song, D.X., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on. pp. 44–55 (2000)

[120] Sweeney, L.: K-anonymity: A model for protecting privacy. Int. J. Uncertain. Fuzziness Knowl.-Based Syst. 10(5), 557–570 (Oct 2002)

[121] Talviste, R.: Deploying secure multiparty computation for joint data analysis—a case study. Master's thesis, Institute of Computer Science, University of Tartu (2011)

[122] Thompson, K.: Reflections on trusting trust. Communications of the ACM 27(8), 761–763 (1984)

[123] Vabariigi President: "Käibemaksuseaduse ja raamatupidamise seaduse muutmise seaduse" väljakuulutamata jätmine. RT III, 21.12.2013, 1, https://www.riigiteataja.ee/akt/321122013001 (in Estonian)

[124] Vaht, M.: The Analysis and Design of a Privacy-Preserving Survey System. Master's thesis, Institute of Computer Science, University of Tartu (2015)

[125] Violino, B.: Security technique protects multi-party computation. Communications of the ACM (January 22 2010), published online at http://cacm.acm.org/news/68288

[126] Wang, G., Luo, T., Goodrich, M.T., Du, W., Zhu, Z.: Bureaucratic protocols for secure two-party sorting, selection, and permuting. In: Proc. of ASIACCS'10. pp. 226–237. ACM (2010)

[127] Willemson, J., Ansper, A.: A secure and scalable infrastructure for inter-organizational data exchange and egovernment applications. In: Availability, Reliability and Security, 2008. ARES 08. Third International Conference on. pp. 572–577 (March 2008)

[128] Willemson, J.: Pseudonymization Service for X-Road eGovernment Data Exchange Layer. In: Proceedings of EGOVIS 2011. Lecture Notes in Computer Science, vol. 6866, pp. 135–145. Springer (2011)

[129] Yao, A.C.C.: Protocols for secure computations. In: Foundations of Computer Science, 1982. SFCS '08. 23rd Annual Symposium on. pp. 160–164 (Nov 1982)

[130] Yao, A.C.C.: How to generate and exchange secrets. In: Foundations of Computer Science, 1986., 27th Annual Symposium on. pp. 162–167 (Oct 1986)

[131] Zhang, B.: Generic constant-round oblivious sorting algorithm for mpc. In: Provable Security, Lecture Notes in Computer Science, vol. 6980, pp. 240–256. Springer (2011)

ACKNOWLEDGMENTS

Throughout my studies, I have received support from the European Social Fund Doctoral Studies and Internationalisation Programme DoRa that has given me the opportunity to combine my studies with the work I love. During my studies, my research has been mainly supported by the following projects and grants: the European Union Seventh Framework Programme (FP7/2007- 2013) under grant agreement no. 284731 (UaESMC), the Implementing Agency Archimedes Foundation for the Privacy-preserving statistical studies on linked databases (PRIST) project, and the European Union Horison 2020 innovation and cooperation project SUNFISH under grant agreement no. 644666. I also recognise the support by the European Regional Development Fund through the Estonian Center of Excellence in Computer Science (EXCS) and the European Social Fund through the Estonian Doctoral School in Information and Communication Technology (IKTDK).

I'd like to thank both academic and supporting staff in the Institute of Computer Science at the University of Tartu and my colleagues at Cybernetica AS – the two organisations that have become a large part of my identity and without which it is difficult to imagine my everyday life. My sincere gratitude goes out to the whole SHAREMIND team and especially to all of the colleagues at Cybernetica who have supervised me, either officially or unofficially: Dan Bogdanov, Peeter Laud, Jan Willemson and Liina Kamm.

However, most of all, I am grateful to my supervisor, colleague, friend and mentor Dan Bogdanov who has guided me starting from my bachelor studies. Without his support, I would certainly not be where I am now, both professionally and academically. As the challenges of this thesis were both practical and theoretical, I am most grateful for the insights of my academical supervisor Sven Laur. He ensured that I always go beyond solving the immediate problem and give the issue a full scientific treatment. There is nothing more heart-warming than a supervisor finally saying *"Looks like a thesis."*

Finally, I would like to give a warm hug to my wife Karin who has stood beside me throughout my studies and career, supporting and expressing great interest in what I do. I know that I am not an easy person to be with, especially when there is a deadline near by that I have to meet. Thank you.

KOKKUVÕTE (SUMMARY IN ESTONIAN)

TURVALISE ÜHISARVUTUSE RAKENDAMINE

Andmetest on kasu vaid siis kui neid kasutada. Üha suurenevate andmemahtudega tekib organisatsioonidel uus probleem – neil ei ole ressursse, et kogutud andmeid talletada ja analüüsida. Pilveteenuste kasutamine andmete hoiustamiseks ja arvutusteks on mitmel puhul majanduslikult mõistlik lahendus, kuid tihti ei luba konfidentsiaalsusnõuded andmeid pilves hoida. Andmete krüpteerimine enne pilve laadimist lahendab selle probleemi, kuid sel juhul ei saa nende andmetega pilves midagi peale hakata. Pilves hoiustatud krüpteeritud andmete pidev allalaadimine ja dekrüpteerimine ei ole andmemahte arvestades tavaliselt sobiv lahendus.

Isegi, kui selline lähenemisviis organisatsioonile sobib, piirab see tugevalt kogutud andmete potentsiaalset väärtust. Kõige enam saab andmetest õppida, kui ühendada erinevad andmekogud. Nii on võimalik avastada seoseid ja trende või saada jälile sotsiaalsete probleemide põhjustele. Näiteks saab riik haridus- ja maksuandmeid ühendades läbi viia kõrghariduse erialade tasuvusanalüüse. Sarnaseid näiteid leiab ka erasektorist – ühendades erinevate pankade maksekohustuste andmed, on võimalik väljastada väiksema krediidiriskiga laene.

Pahatihti on aga selliste andmekogude liitmine konfidentsiaalsus- või privaatsusnõuete tõttu keelatud. Õigusega, sest taoline kombineeritud andmekogu oleks atraktiivne sihtmärk nii häkkeritele kui ka lihtsalt uudishimulikele töötajatele ja ametnikele, kes oma positsiooni kuritarvitada võivad.

Turvaline ühisarvutus on tehnoloogia, mis võimaldab mitmel sõltumatul osapoolel koos andmeid töödelda, ilma et ükski neist pääseks ligi konkreetsetele väärtustele. Nimetatud tehnoloogia on krüptograafia teoorias tuntud juba enam kui 30 aastat, kuid esimese päris andmete peal töötava turvalise ühisarvutuse rakenduse-

ni jõuti alles 2008. aastal. Peale seda on turvalist arvutust kasutatud ka hajusates lahendustes, mis töötavad üle interneti ning seda tehnoloogiat pakutakse isegi teenusena konkreetsete usaldus- ja privaatsusprobleemide lahendamiseks.

Selles töös vaatlemegi usaldusküsimusi ja tehnilisi probleeme, mis kaasnevad turvalise ühisarvutuse tehnoloogia juurutamisega praktilistes rakendustes. Esmalt anname ülevaate kahest esimesest praktilisest turvalise ühisarvutuse juurutusest, milleks olid turvaline suhkrupeedi oksjon Taanis 2008. aastal ning Eesti Infotehnoloogia ja Telekommunikatsiooni Liidu (ITL) liikmete finantsandmete analüüs 2011. aastal. Toome välja nende rakenduste kitsaskohad, mis võiks turvalise ühisarvutuse edasist levikut piirata, ning kirjeldame lahendusi.

Käesoleva töö autor on teostanud ja testinud turvalist andmete sorteerimist ning erinevate andmekogude liitmist. Olulise tulemusena kirjeldame ka AES plokkšifri teostust, kus ei võti, andmed ega ka krüpteeritud tulemus pole ühelegi arvutavale osapoolele teada.

Lähtudes ITL-i liikmete finantsandmete analüüsirakenduse näitest, pöörame erilist tähelepanu turvalist ühisarvutust kasutavatele veebirakendustele. Kuna üha enam rakendusi ja teenuseid muutuvad pilvepõhisteks, muutuvad ka veebipõhised kasutajaliidesed järjest olulisemaks. Anname ülevaate, kuidas veebitehnoloogiate areng võimaldab üha mugavamalt veebilehitsejal teostada krüptograafilisi operatsioone ja suhelda turvalise ühisarvutuse arvutusosapooltega.

Käesoleva töö põhitulemus on ülevaade esimesest turvalise ühisarvutusega läbiviidud uuringust PRIST, milles kasutati suuremahulisi registriandmeid. Autor keskendub eelkõige uuringu tehnilistele, juurutus- ja usaldusküsimustele, mis tulenevad turvalise ühisarvutuse rakendamisest suurtes organisatsioonides nagu riigiasutused. Kirjeldame protsessi, mis on vaja läbida, et tagada sellise uuringu vastavus seadusandlusega. Näiteks on PRIST uuring esimene turvalise ühisarvutuse rakendus, kus arvutusosapoolte sõltumatus ja sellest tulenevad kohustused on ka lepinguliselt tagatud. Anname ülevaate projekti käigus tõstatunud tehnilistest, inimlikest ja jõudlusprobleemidest, kirjeldame kasutatud lahendusi ning pakume välja ideid, kuidas taolisi probleeme tulevikus vältida.

Lõpetuseks pakume töös välja arhitektuuri, mis ühendab endas födereeritud andmevahetusplatvormi ja turvalise ühisarvutuse tehnoloogiat. Selline lahendus võimaldab läbi viia erinevaid andmekogusid hõlmavaid uuringuid, tagades samal ajal iga andmeomaniku seatud andmete kasutamise piirangu range järgimise. Praktilise näitena pakub töö autor välja Eesti riigi andmevahetuskihi X-tee täiustamise turvalise ühisarvutuse teenusega, kasutades selleks turvalise ühisarvutuse platvormi SHAREMIND. Selline arhitektuur muudab PRIST-i laadsete mitut andmekogu hõlmavate uuringute läbiviimise oluliselt kiiremaks ja mugavamaks. See omakorda võimaldab avalikul võimul olla paremini kursis riigis toimuvaga ning teostada efektiivsemat juhtimist.

LIST OF ORIGINAL PUBLICATIONS

1. Bogdanov, D., Talviste, R., Willemson, J.: Deploying secure multi-party computation for financial data analysis (short paper). In: Proceedings of the 16th International Conference on Financial Cryptography and Data Security. FC'12. Lecture Notes in Computer Science, vol. 7397, pp. 57–64. Springer (2012).

2. Laur, S., Talviste, R., Willemson, J.: From Oblivious AES to Efficient and Secure Database Join in the Multiparty Setting. In: Applied Cryptography and Network Security. ACNS'13, Lecture Notes in Computer Science, vol. 7954, pp. 84–101. Springer (2013).

3. Bogdanov, D., Kamm, L., Laur, S., Pruulmann-Vengerfeldt, P., Talviste, R., Willemson, J.: Privacy-preserving statistical data analysis on federated databases. In: Proceedings of the Annual Privacy Forum. APF'14. Lecture Notes in Computer Science, vol. 8450, pp. 30–55. Springer (2014).

4. Bogdanov, D., Laur, S., Talviste, R.: A Practical Analysis of Oblivious Sorting Algorithms for Secure Multi-party Computation. In: Proceedings of the 19th Nordic Conference on Secure IT Systems. NordSec'14, Lecture Notes in Computer Science, vol. 8788, pp. 59–74. Springer (2014).

5. Bogdanov, D., Kamm, L., Kubo, B., Rebane, R., Sokk, V., Talviste, R.: Students and Taxes: a Privacy-Preserving Social Study Using Secure Computation. Proceedings on Privacy Enhancing Technologies (PoPETs) 2016(3) (2016), (to appear).

CURRICULUM VITAE

Personal data

Name	Riivo Talviste
Birth	April 2nd, 1987 Pärnu, Estonia
Citizenship	Estonian
Languages	Estonian, English
E-mail	riivo.talviste@cyber.ee

Education

2011–	University of Tartu, Ph.D. candidate in Computer Science
2009–2011	University of Tartu, M.Sc. in Information Technology
2006–2009	University of Tartu, B.Sc. in Information Technology
2003–2006	Pärnu Koidula Gymnasium, secondary education
1994–2003	Pärnu Old Town Secondary School, primary education

Employment

2015–	University of Tartu, visiting lecturer in IT Law
2012–	Cybernetica AS, junior researcher
2011–2012	Cybernetica AS, programmer
2008–2010	Swedbank AS, software developer

ELULOOKIRJELDUS

Isikuandmed

Nimi	Riivo Talviste
Sünniaeg ja -koht	2. aprill 1987 Pärnu, Eesti
Kodakondsus	eestlane
Keelteoskus	eesti, inglise
E-post	riivo.talviste@cyber.ee

Haridustee

2011–	Tartu Ülikool, informaatika doktorant
2009–2011	Tartu Ülikool, MSc infotehnoloogias
2006–2009	Tartu Ülikool, BSc infotehnoloogias
2003–2006	Pärnu Koidula Gümnaasium, keskharidus
1994–2003	Pärnu Vanalinna Põhikool, põhiharidus

Teenistuskäik

2015–	Tartu Ülikool, IT-õiguse külalislektor
2012–	Cybernetica AS, nooremteadur
2011–2012	Cybernetica AS, programmeerija
2008–2010	Swedbank AS, tarkvaraarendaja

DISSERTATIONES MATHEMATICAE UNIVERSITATIS TARTUENSIS

1. **Mati Heinloo.** The design of nonhomogeneous spherical vessels, cylindrical tubes and circular discs. Tartu, 1991, 23 p.
2. **Boris Komrakov.** Primitive actions and the Sophus Lie problem. Tartu, 1991, 14 p.
3. **Jaak Heinloo.** Phenomenological (continuum) theory of turbulence. Tartu, 1992, 47 p.
4. **Ants Tauts.** Infinite formulae in intuitionistic logic of higher order. Tartu, 1992, 15 p.
5. **Tarmo Soomere.** Kinetic theory of Rossby waves. Tartu, 1992, 32 p.
6. **Jüri Majak.** Optimization of plastic axisymmetric plates and shells in the case of Von Mises yield condition. Tartu, 1992, 32 p.
7. **Ants Aasma.** Matrix transformations of summability and absolute summability fields of matrix methods. Tartu, 1993, 32 p.
8. **Helle Hein.** Optimization of plastic axisymmetric plates and shells with piece-wise constant thickness. Tartu, 1993, 28 p.
9. **Toomas Kiho.** Study of optimality of iterated Lavrentiev method and its generalizations. Tartu, 1994, 23 p.
10. **Arne Kokk.** Joint spectral theory and extension of non-trivial multiplicative linear functionals. Tartu, 1995, 165 p.
11. **Toomas Lepikult.** Automated calculation of dynamically loaded rigid-plastic structures. Tartu, 1995, 93 p, (in Russian).
12. **Sander Hannus.** Parametrical optimization of the plastic cylindrical shells by taking into account geometrical and physical nonlinearities. Tartu, 1995, 74 p, (in Russian).
13. **Sergei Tupailo.** Hilbert's epsilon-symbol in predicative subsystems of analysis. Tartu, 1996, 134 p.
14. **Enno Saks.** Analysis and optimization of elastic-plastic shafts in torsion. Tartu, 1996, 96 p.
15. **Valdis Laan.** Pullbacks and flatness properties of acts. Tartu, 1999, 90 p.
16. **Märt Põldvere.** Subspaces of Banach spaces having Phelps' uniqueness property. Tartu, 1999, 74 p.
17. **Jelena Ausekle.** Compactness of operators in Lorentz and Orlicz sequence spaces. Tartu, 1999, 72 p.
18. **Krista Fischer.** Structural mean models for analyzing the effect of compliance in clinical trials. Tartu, 1999, 124 p.

19. **Helger Lipmaa.** Secure and efficient time-stamping systems. Tartu, 1999, 56 p.
20. **Jüri Lember.** Consistency of empirical k-centres. Tartu, 1999, 148 p.
21. **Ella Puman.** Optimization of plastic conical shells. Tartu, 2000, 102 p.
22. **Kaili Müürisep.** Eesti keele arvutigrammatika: süntaks. Tartu, 2000, 107 lk.
23. **Varmo Vene.** Categorical programming with inductive and coinductive types. Tartu, 2000, 116 p.
24. **Olga Sokratova.** Ω-rings, their flat and projective acts with some applications. Tartu, 2000, 120 p.
25. **Maria Zeltser.** Investigation of double sequence spaces by soft and hard analitical methods. Tartu, 2001, 154 p.
26. **Ernst Tungel.** Optimization of plastic spherical shells. Tartu, 2001, 90 p.
27. **Tiina Puolakainen.** Eesti keele arvutigrammatika: morfoloogiline ühestamine. Tartu, 2001, 138 p.
28. **Rainis Haller.** *M(r,s)*-inequalities. Tartu, 2002, 78 p.
29. **Jan Villemson.** Size-efficient interval time stamps. Tartu, 2002, 82 p.
30. **Eno Tõnisson.** Solving of expession manipulation exercises in computer algebra systems. Tartu, 2002, 92 p.
31. **Mart Abel.** Structure of Gelfand-Mazur algebras. Tartu, 2003. 94 p.
32. **Vladimir Kuchmei.** Affine completeness of some ockham algebras. Tartu, 2003. 100 p.
33. **Olga Dunajeva.** Asymptotic matrix methods in statistical inference problems. Tartu 2003. 78 p.
34. **Mare Tarang.** Stability of the spline collocation method for volterra integro-differential equations. Tartu 2004. 90 p.
35. **Tatjana Nahtman.** Permutation invariance and reparameterizations in linear models. Tartu 2004. 91 p.
36. **Märt Möls.** Linear mixed models with equivalent predictors. Tartu 2004. 70 p.
37. **Kristiina Hakk.** Approximation methods for weakly singular integral equations with discontinuous coefficients. Tartu 2004, 137 p.
38. **Meelis Käärik.** Fitting sets to probability distributions. Tartu 2005, 90 p.
39. **Inga Parts.** Piecewise polynomial collocation methods for solving weakly singular integro-differential equations. Tartu 2005, 140 p.
40. **Natalia Saealle.** Convergence and summability with speed of functional series. Tartu 2005, 91 p.
41. **Tanel Kaart.** The reliability of linear mixed models in genetic studies. Tartu 2006, 124 p.
42. **Kadre Torn.** Shear and bending response of inelastic structures to dynamic load. Tartu 2006, 142 p.

43. **Kristel Mikkor.** Uniform factorisation for compact subsets of Banach spaces of operators. Tartu 2006, 72 p.
44. **Darja Saveljeva.** Quadratic and cubic spline collocation for Volterra integral equations. Tartu 2006, 117 p.
45. **Kristo Heero.** Path planning and learning strategies for mobile robots in dynamic partially unknown environments. Tartu 2006, 123 p.
46. **Annely Mürk.** Optimization of inelastic plates with cracks. Tartu 2006. 137 p.
47. **Annemai Raidjõe.** Sequence spaces defined by modulus functions and superposition operators. Tartu 2006, 97 p.
48. **Olga Panova.** Real Gelfand-Mazur algebras. Tartu 2006, 82 p.
49. **Härmel Nestra.** Iteratively defined transfinite trace semantics and program slicing with respect to them. Tartu 2006, 116 p.
50. **Margus Pihlak.** Approximation of multivariate distribution functions. Tartu 2007, 82 p.
51. **Ene Käärik.** Handling dropouts in repeated measurements using copulas. Tartu 2007, 99 p.
52. **Artur Sepp.** Affine models in mathematical finance: an analytical approach. Tartu 2007, 147 p.
53. **Marina Issakova.** Solving of linear equations, linear inequalities and systems of linear equations in interactive learning environment. Tartu 2007, 170 p.
54. **Kaja Sõstra.** Restriction estimator for domains. Tartu 2007, 104 p.
55. **Kaarel Kaljurand.** Attempto controlled English as a Semantic Web language. Tartu 2007, 162 p.
56. **Mart Anton.** Mechanical modeling of IPMC actuators at large deformations. Tartu 2008, 123 p.
57. **Evely Leetma.** Solution of smoothing problems with obstacles. Tartu 2009, 81 p.
58. **Ants Kaasik.** Estimating ruin probabilities in the Cramér-Lundberg model with heavy-tailed claims. Tartu 2009, 139 p.
59. **Reimo Palm.** Numerical Comparison of Regularization Algorithms for Solving Ill-Posed Problems. Tartu 2010, 105 p.
60. **Indrek Zolk.** The commuting bounded approximation property of Banach spaces. Tartu 2010, 107 p.
61. **Jüri Reimand.** Functional analysis of gene lists, networks and regulatory systems. Tartu 2010, 153 p.
62. **Ahti Peder.** Superpositional Graphs and Finding the Description of Structure by Counting Method. Tartu 2010, 87 p.
63. **Marek Kolk.** Piecewise Polynomial Collocation for Volterra Integral Equations with Singularities. Tartu 2010, 134 p.

64. **Vesal Vojdani.** Static Data Race Analysis of Heap-Manipulating C Programs. Tartu 2010, 137 p.
65. **Larissa Roots.** Free vibrations of stepped cylindrical shells containing cracks. Tartu 2010, 94 p.
66. **Mark Fišel.** Optimizing Statistical Machine Translation via Input Modification. Tartu 2011, 104 p.
67. **Margus Niitsoo**. Black-box Oracle Separation Techniques with Applications in Time-stamping. Tartu 2011, 174 p.
68. **Olga Liivapuu.** Graded q-differential algebras and algebraic models in noncommutative geometry. Tartu 2011, 112 p.
69. **Aleksei Lissitsin.** Convex approximation properties of Banach spaces. Tartu 2011, 107 p.
70. **Lauri Tart.** Morita equivalence of partially ordered semigroups. Tartu 2011, 101 p.
71. **Siim Karus.** Maintainability of XML Transformations. Tartu 2011, 142 p.
72. **Margus Treumuth.** A Framework for Asynchronous Dialogue Systems: Concepts, Issues and Design Aspects. Tartu 2011, 95 p.
73. **Dmitri Lepp.** Solving simplification problems in the domain of exponents, monomials and polynomials in interactive learning environment T-algebra. Tartu 2011, 202 p.
74. **Meelis Kull.** Statistical enrichment analysis in algorithms for studying gene regulation. Tartu 2011, 151 p.
75. **Nadežda Bazunova.** Differential calculus $d^3 = 0$ on binary and ternary associative algebras. Tartu 2011, 99 p.
76. **Natalja Lepik.** Estimation of domains under restrictions built upon generalized regression and synthetic estimators. Tartu 2011, 133 p.
77. **Bingsheng Zhang.** Efficient cryptographic protocols for secure and private remote databases. Tartu 2011, 206 p.
78. **Reina Uba.** Merging business process models. Tartu 2011, 166 p.
79. **Uuno Puus.** Structural performance as a success factor in software development projects – Estonian experience. Tartu 2012, 106 p.
80. **Marje Johanson.** $M(r, s)$-ideals of compact operators. Tartu 2012, 103 p.
81. **Georg Singer.** Web search engines and complex information needs. Tartu 2012, 218 p.
82. **Vitali Retšnoi.** Vector fields and Lie group representations. Tartu 2012, 108 p.
83. **Dan Bogdanov.** Sharemind: programmable secure computations with practical applications. Tartu 2013, 191 p.
84. **Jevgeni Kabanov.** Towards a more productive Java EE ecosystem. Tartu 2013, 151 p.
85. **Erge Ideon**. Rational spline collocation for boundary value problems. Tartu, 2013, 111 p.
86. **Esta Kägo**. Natural vibrations of elastic stepped plates with cracks. Tartu, 2013, 114 p.

87. **Margus Freudenthal.** Simpl: A toolkit for Domain-Specific Language development in enterprise information systems. Tartu, 2013, 151 p.
88. **Boriss Vlassov.** Optimization of stepped plates in the case of smooth yield surfaces. Tartu, 2013, 104 p.
89. **Elina Safiulina.** Parallel and semiparallel space-like submanifolds of low dimension in pseudo-Euclidean space. Tartu, 2013, 85 p.
90. **Raivo Kolde.** Methods for re-using public gene expression data. Tartu, 2014, 121 p.
91. **Vladimir Šor.** Statistical Approach for Memory Leak Detection in Java Applications. Tartu, 2014, 155 p.
92. **Naved Ahmed.** Deriving Security Requirements from Business Process Models. Tartu, 2014, 171 p.
93. **Kerli Orav-Puurand.** Central Part Interpolation Schemes for Weakly Singular Integral Equations. Tartu, 2014, 109 p.
94. **Liina Kamm.** Privacy-preserving statistical analysis using secure multi-party computation. Tartu, 2015, 201 p.
95. **Kaido Lätt.** Singular fractional differential equations and cordial Volterra integral operators. Tartu, 2015, 93 p.
96. **Oleg Košik.** Categorical equivalence in algebra. Tartu, 2015, 84 p.
97. **Kati Ain.** Compactness and null sequences defined by ℓ_p spaces. Tartu, 2015, 90 p.
98. **Helle Hallik.** Rational spline histopolation. Tartu, 2015, 100 p.
99. **Johann Langemets.** Geometrical structure in diameter 2 Banach spaces. Tartu, 2015, 132 p.
100. **Abel Armas Cervantes.** Diagnosing Behavioral Differences between Business Process Models. Tartu, 2015, 193 p.
101. **Fredrik Milani.** On Sub-Processes, Process Variation and their Interplay: An Integrated Divide-and-Conquer Method for Modeling Business Processes with Variation. Tartu, 2015, 164 p.
102. **Huber Raul Flores Macario.** Service-Oriented and Evidence-aware Mobile Cloud Computing. Tartu, 2015, 163 p.
103. **Tauno Metsalu.** Statistical analysis of multivariate data in bioinformatics. Tartu, 2016, 197 p.