Chair of Network Architectures and Services
Department of Informatics
Technical University of Munich

Secure and Privacy-Preserving Services based on Secure Multiparty Computation

Marcel von Maltitz

Dissertation

Network Architectures and Services NET-2019-07-2

TECHNISCHE UNIVERSITÄT MÜNCHEN
Institut für Informatik
Lehrstuhl für Netzarchitekturen und Netzdienste

Secure and Privacy-Preserving Services based on Secure Multiparty Computation

Marcel Léon von Maltitz

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:		Prof. Dr. Jens Großklags
Prüfer der Dissertation:	1.	Prof. Dr.-Ing. Georg Carle
	2.	Assoc. Prof. Dr. Florian Kerschbaum

Die Dissertation wurde am 23.04.2019 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 17.07.2019 angenommen.

Cataloging-in-Publication Data
Marcel von Maltitz
Secure and Privacy-Preserving Services based on Secure Multiparty Computation
Dissertation, 2019
Network Architectures and Services, Department of Computer Science
Technische Universität München

ISBN 978-3-937201-67-2
ISSN 1868-2634(print)
ISSN 1868-2642(electronic)
DOI 10.2313/NET-2019-07-02
Network Architectures and Services NET
Series Editor: Georg Carle, Technische Universität München, Germany

Abstract

The last decades show that ubiquitous application of information technology often entails a loss of privacy for the individual. Users seemingly agree in giving away their data in order to use a corresponding technology. However, the underlying reason is that the current state of technology often does not allow to opt for privacy protection. In order to give users a real choice, technologies have to be made available which particularly take the preservation of privacy into account.

For realizing privacy-preserving services and systems, three challenges have to be considered: a clear meaning of privacy for a certain context has to be established. Technological means have to be identified which are able to support this intention and it must be investigated how these means can be applied effectively.

First, we study several concepts of privacy. Among them, we select a contemporary understanding which also provides a useful foundation for building privacy-preserving technology. We then choose Secure Multiparty Computation (SMC) as a specific cryptographic approach since it has become a promising method for privacy in data processing in the last years. Ultimately aiming for a privacy-preserving service we first identify the potential of SMC by understanding its performance and privacy characteristics, its infrastructural requirements and premises for its application. We establish a baseline of performance, scalability, and its resource consumption. We varied the network and host properties, and analyzed their individual influence on several performance and resource aspects. We complement these experiments with comparative measurements in emulated real world scenarios, e.g., intranet, Internet and mobile Internet. From that we derive its suitability for infrastructurally different settings of application. Our results confirm the current understanding that secure computation is communication-bound. Latency on the network has the highest influence on the computation time while the influence of host parameters is considerably lower. In our last step of the assessment of SMC, we choose a real-world algorithm for statistical analysis from the domain of medical studies. We reimplemented the algorithm as a secure computation and applied it in two settings: Firstly, we measured it with synthetic data in a testbed. Secondly, we executed the secure computation over the Internet between two hosts of different research institutions which are 500 km apart from each other. From our assessments, we derive several insights regarding infrastructure and privacy implications as well as its performance properties. In particular, we identify which privacy goals SMC is able to fulfill and conclude from our infrastructural and performance findings that SMC has a potential of practical application which is currently best leveraged in the intranet setting.

Previous applications of SMC have often been deployed in an ad-hoc manner with large manual overhead. To enable widespread adaptation of SMC, an architecture is needed which allows administratively easy service-like application. For that, we create *SMC as a Service* which is characterized by making securely computed data available to outside clients by traditional queries while hiding the secure computations from them. For the latter, we develop a management architecture that is dedicated to make SMC self-managing, robust and fit for dynamic contexts. Data access is made possible by a gateway component which accepts queries and orchestrates corresponding secure computations. From our identified notion of privacy we see that SMC alone only fulfills some privacy requirements. We cover the remaining requirements by providing means for data access control which make access transparent for the data owning components and allow them stay in control of how their data is used. In the end, we achieve a fully privacy-preserving service with SMC at the heart of its architecture.

Kurzfassung

Die letzten Jahrzehnte zeigen, dass der allgegenwärtige Einsatz von Informationstechnologie oft einen Verlust von Privatheit jedes Einzelnen mit sich gebracht hat. Nutzer scheinen einverstanden, ihre Daten preiszugeben, um die entsprechenden Technologien verwenden zu können. Dies ist jedoch der Tatsache geschuldet, dass heutige Informationstechnologien es häufig nicht erlauben, sich gezielt für den Schutz der eigenen Daten zu entscheiden. Um Nutzern eine echte Wahl zu geben, müssen Technologien bereitgestellt werden, welche die Erhaltung von Privatheit gezielt berücksichtigen.

Um privatheitserhaltende Dienste und Systeme zu schaffen, müssen drei Herausforderungen angegangen werden: Eine klares Verständnis von Privatheit für einen gewählten Kontext muss gewonnen werden. Technologische Ansätze müssen identifiziert werden, welche diese Auffassung unterstützen und es muss untersucht werden, wie diese Ansätze effektiv eingesetzt werden können.

Zunächst befassen wir uns mit verschiedenen Konzepten von Privatheit. Aus diesen wählen wir eine zeitgemäße Auffassung, die ebenso eine nützliche Grundlage für privatheitsschützende Technologien darstellt. Wir ziehen dann Secure Multiparty Computation (SMC) als konkreten kryptographischen Ansatz heran, da er sich in den letzten Jahren zu einer vielversprechenden Methode für die Erzielung von Privatheit in der Datenverarbeitung entwickelt hat. Wir untersuchen zunächst das Potential von SMC, indem wir ihre Leistungs- und Privatheitscharakteristika identifizieren und ihre infrastrukturellen Anforderungen und Prämissen für ihren Einsatz ermitteln. Wir erstellen eine Basislinie der Leistung von SMC, ihrer Skalierbarkeit und des Ressourcenverbrauchs. Hierbei veränderten wir die Netzwerk- und Hosteigenschaften und analysierten deren Einfluss. Wir ergänzen diese Experimente mit vergleichenden Messungen in emulierten Szenarien, das heißt, der Einsatz im Intranet, Internet wie auch im mobilen Internet. Daraus ermitteln wir ihre Eignung für den Einsatz in infrastrukturell verschiedenen Bereichen. Unsere Ergebnisse bestätigen das aktuelle Verständnis, dass die Leistung von SMC durch den Kommunikationskanal bestimmt wird. Die Latenz als Parameter hat den größten Einfluss auf die Berechnungszeit während der der Hostparameter deutlich geringer ist. Für unseren letzten Beitrag zur Bewertung von SMC ziehen wir einen gängigen Algorithmus für statistische Analysen aus der Domäne medizinischer Studien heran. Wir schrieben den Algorithmus in eine sichere Berechnung um und wandten sie in zwei Umgebungen an: Zunächst maßen wir ihr Leistungsverhalten mit synthetischen Daten in einem Testbett. Ferner führten wir sie über das Internet zwischen zwei Hosts verschiedener Forschungseinrichtungen aus, welche circa 500 km weit voneinander entfernt liegen. Unsere Untersuchungen geben uns nicht nur ein Verständnis für die Leistungsmerkmale von SMC, sondern erlauben auch die Ableitung von vielfältigen Implikationen für ihre Infrastruktur- und Privatheitseigenschaften. Insbesondere identifizieren wir diejenigen Schutzziele von Privatheit, welche SMC in der Lage ist, zu erfüllen und schließen ferner, dass das festgestellte Potential zur praktischen Anwendung von SMC zur Zeit am besten in Intranet-Umgebungen ausgenutzt werden kann.

Der bisher übliche Einsatz von SMC geschieht meist einzelfallbasiert mit großem manuellem Mehraufwand. Um eine weite Verbreitung von SMC zu erzielen, ist jedoch eine Architektur notwendig, welche eine administrativ einfache und service-artige Anwendung ermöglicht. Dafür entwickeln wir *SMC as a Service*. Dessen Hauptmerkmal ist, dass sicher berechnete Daten mithilfe von klassischen Anfragen für Dritte zugänglich gemacht werden, während die eigentlichen Berechnungen vor diesen verschatten bleiben. Für letzteres entwickeln wir eine Management-Architektur, welche SMC autonom, robust und bereit für den Einsatz in dynamischen Umgebungen macht. Der Datenzugriff wird durch eine Gateway-Komponente ermöglicht, welche Anfragen annimmt und korrespondierende Berechnungen orchestriert. Ausgehend von der zuvor gewählten Auffassung von Privatheit sehen wir, dass

SMC allein nur manche Privatheitsanforderungen erfüllt. Die verbleibenden Anforderungen decken wir durch Methoden der Datenzugriffskontrolle ab: Diese machen Datenzugriff transparent für jene Komponenten, welche die ursprünglichen Daten besitzen und ermöglichen ihnen, die konkrete Nutzung ihrer Daten zu kontrollieren. Dadurch erhalten wir einen vollständig privatheitserhaltenden Dienst mit SMC im Herzen seiner Architektur.

Acknowledgments

This thesis would have not been possible without the support by many individuals. I will not be able to name all of them but I want to highlight some.

I would like to thank Prof. Dr.-Ing. Georg Carle for giving me the opportunity to join the Chair for Network Architectures and Services and for supervising the dissertation. I highly appreciate your initial sparks of inspiration giving me a direction to head in. Likewise, you gave me the freedom to follow my own interests and to shape the topic to my liking. Thank you for your guidance and regular corrections of the course. I also would like to thank Prof. Dr. Florian Kerschbaum for being my second assessor and Prof. Dr. Jens Großklags for chairing the examination committee. Thank you for your interest, your time and your support for finishing my PhD.

My feelings about working at the Chair are mainly influenced by my colleagues who shared a considerable amount of time with me during my PhD. I would like to thank all of them for accompanying me during my way. Above that, I want to express my special gratitude to some of them: thank you, Holger, for being such a pleasant office mate. You facilitated my start at the Chair tremendously and cooperating with you—be it in research projects or conducting student theses—was always easy, inspiring and fun. Thank you, Florian and Dominik, and the whole testbed team. You made my measurements possible. You supported me during the process and helped me to fulfill the rather special requirements I sometimes had for the test infrastructure.

I would like to thank all my students whose thesis I was allowed to supervise. While you hopefully learned something about computer science, I learned a lot about education and applied didactics. A special thanks goes to two of my master students, Stefan and Dominik. Thank you for your cooperation in my research and in particular for building the prototype of my privacy-preserving service architecture.

Thank you, Dr. rer. nat. Hendrik Ballhausen, for suggesting a research collaboration on secure evaluation of medical studies and for making it possible. All credit belongs to you that we could use real medical data, had the necessary infrastructure available and the support of many more colleagues, without whom we would not have been able to conduct our evaluations. It was great working with you.

I would also like to thank my proof readers, Lukas, Max, and Daniel. Thank you for your feedback.

University and academia only covers a part of my life. I would like to thank my friends for making the other parts enjoyable and worth living. Thank you for your support and being with me. You know who you are. In particular, I want to thank Lukas for being at my side. There are no words which can adequately express my gratitude.

Lastly, I am deeply thankful to my parents Gisela and Axel. Always trusting in my choice, you gave me all necessary space for development and seemingly effortless guidance. You had the biggest impact on who I am today.

Contents

I Performance and Application of SMC

1. Introduction

Digitization has fundamentally changed our lives. Through the ubiquity of mobile phones and computers, we experience an interconnectedness unparalleled in the history of civilization. Every day, we consume tremendous amounts of information and most of us are also generating tons of information, especially by the use of social media and technically mediated communication.

The end-devices connecting us to the digital world are only the top layer of the technology stack making this possible. Underneath, additional innovations were crucial: the Internet provides the infrastructure for connectivity which enables information exchange. Cloud computing provides data availability through data storage and processing facilities.

However, the practice of remotely centralizing data leads to a loss of privacy for data owners and users. Third parties are able to access the users' data and attacks on these infrastructures increase in terms of likeliness and impact. Similarly, users lose control over uploaded data and become unable to track how and for which purposes their data is actually used.

During the last years, the awareness of this problem has increased in some parts of the world as society witnessed the revelation of global Internet surveillance. A public discourse emerged, renegotiating the value of privacy and the necessity of its protection.

In order to achieve a handling of privacy which is compatible with a pluralistic society, it is important that this discourse can develop freely. However, the current default of technological systems is undermining, or at least disregarding privacy. Merely using technology then seemingly implies that we acquiesce in or even agree to our loss of privacy. If we fail to have a real choice on the technical level, the discourse becomes biased.

Using technology on the one hand and protecting one's data on the other hand is a false dichotomy. Privacy-enhancing technologies demonstrate that the two are not fundamentally in conflict; instead, it is actually possible to achieve both at the same time.

In the 1990s, public key cryptography was seen as a holy grail for privacy in the digital world [Tim94]. The corresponding technical understanding of privacy was strongly dominated by the idea that certain information should be confidential and therefore not available to a third party at all. From this viewpoint, encryption provides privacy by guaranteeing that some information is just random data for those who are not eligible for accessing it.

This is an all-or-nothing approach to data access. Being so coarse, it is not satisfyingly applicable to domains where data *should* be available to third parties and allow *some* usage of the data while prohibiting others.

The realization of privacy-preserving systems which achieve this property is not a trivial task. Typically, specific protection strongly depends on the semantics, i.e., the meaning and the structure of the data in question. In general, the more one can process, compute with or transform the data, the more possibilities of privacy protection become available. This is especially true for numerical data: in the overwhelming majority of use cases, it is not the raw data points which are *finally* important, but metrics and aggregations derived from it. The latter are of much higher relevance for decision making while they are actually less privacy-critical than the original data points.

Processing critical data into uncritical aggregates becomes difficult in particular if private data from several sources should be merged. This normally requires a trusted third party which gets access to all this data and performs the processing. However, it is quite common that authorizing such a third party is not possible or at least entails high administrative overhead like contractual regulations.

Cryptographic approaches like Secure Multiparty Computation (SMC) solve this problem on a technical level. SMC enables the processing and combining of data from several stakeholders while ensuring that each stakeholder's data is not made available to any other party. Only the result becomes accessible for the participants. We can use this approach to generate uncritical data from private data of several sources, allowing to exploit the data's utility without violating the privacy requirements of their owners.

In this thesis, we examine SMC as a foundation of privacy-preserving services for data processing. Our contributions are twofold: SMC comes with certain assumptions and premises about the environment of application and the initial state of the raw data. It also has implications for computation performance and resource consumption. To understand the premises and implications of applying SMC we first assess a state-of-the-art SMC framework and investigate the computing performance one can achieve with it. We do this in a testbed and in a real-world setting.

To enable widespread adoption of SMC, its application must be possible in an automated, self-dependent and robust manner. We hence develop a service architecture for data processing with SMC at its core. We provide measures of self-management and auto-configuration as well as data querying and access control. We will see that SMC only fulfills some privacy protection goals. Therefore, we extend our architecture to address additional privacy requirements: While allowing clients to request securely computed data, we enable data owners to stay informed about computation requests and to exert control over their data during its full lifetime.

1.1 Research Questions

The foundations for SMC were laid in the 1980s. Since then, this field of research flourished both in theory and in practice. In the 2000s, a milestone [BCD+09] was reached by performing one of the first practical field applications of SMC. Until now, security and performance of the technical foundations of SMC are still improving.

Most results were achieved in settings that have been setup a single time for conducting the desired computations. Infrastructural problems have been excluded, rendering the setting more controlled but also more artificial. We are missing an architecture that enables application of SMC to data processing problems in an automated way. We advance the state of the art by providing such a privacy-preserving architecture. It uses SMC as its core technology, but also takes infrastructural constraints of the environment into account.

We conduct this task guided by the following research questions:

Q1: Which understanding of privacy can be used to create privacy-preserving technology?

In the literature, there are multiple concepts of privacy. They range from legal notions over social norms to technical understandings. We examine which foundation can be used to derive requirements for a privacy-preserving technology in general and for service architectures in particular.

Q2: What are the performance characteristics of SMC and which environments for application do they suggest?

Usage of SMC comes at a cost: the protection of data during processing is achieved by mathematical transformations. They add a computational overhead to the processing. Furthermore, processing has to be carried out as interactive protocol between several cooperating nodes. That means, also communication overhead is added. It is vital to understand the implications on performance since we have to take them into account when designing a service architecture in the later chapters.

Q3: Which infrastructural requirements must be addressed when applying SMC in domains with critical data and which privacy requirements are then fulfilled?

Our architecture has to make SMC possible in an automated and self-dependent manner. For that, we have to identify which infrastructural problems our architecture actually has to solve and which premises of SMC it has to fulfill. Moreover, we have to build a fully privacy-preserving service based on the notion identified in Q1. Consequently, we have to assess to which degree this is achieved by employing SMC and which privacy properties have to be realized by other means.

Q4: How must SMC be managed to work reliably in dynamic environments?

From the insights of the previous questions we will see that our architecture needs two parts: To realize SMC as a Service, the internals of the system must be hidden from the clients. This also implies that setup, configuration, handling of computations and occurring problems should not be observable to the outside. To achieve this, the system must be able to cope with these challenges in a self-dependent and autonomous manner.

Q5: How can our architecture be extended in order to realize a fully privacy-preserving service?

The other part addresses making the functionality of SMC available to outside clients without involving them in the computations. This includes stating requests for computation and retrieving the computation result without contact to SMC itself. Furthermore, Q3 will show that SMC can only fulfill some of the desired privacy protection goals. Our architecture must hence be extended by further protection functionality to achieve a fully privacy-preserving service.

1.2 Structure of this Thesis

Chapter 1 This Chapter 1 introduces the topic and presents the research questions to be answered. We give an overview of the following chapters and list the publications which have been created in the context of this thesis.

Chapter 2 In Chapter 2, we provide an overview of the development of the notion of privacy and select a current understanding which enables design and assessment of technological solutions with respect to privacy (**Q1**).

Chapter 3 In Chapter 3, we elaborate on the background and state of the art of SMC. We examine the theoretic background on SMC. We elaborate on the objective, possible adversary models and build up a taxonomy of practical approaches. Lastly, we present the state of the art of available implementations. This part provides the foundation for the following chapters.

Chapters 4 & 5 We perform thorough baseline measurements (**Q2**) of a state-of-the-art implementation of SMC in Chapters 4 and 5. We use a simple use case in order to identify the best case performance. Several host and network parameters are varied in order to understand their influence on the resource consumption and performance of the applied SMC solution. Chapter 4 evaluates these environmental parameters separately, Chapter 5 examines common usage settings represented as a combination of parameters.

Chapter 6 In Chapter 6, we consider medical research as relevant real-world setting in which SMC offers promising advantages. For this setting, we develop a secure implementation of an often used statistical evaluation (**Q3**). We also assess the solution in a testbed and over the Internet (**Q2**). Again, several parameters are varied in order to understand the environmental influences on the computation. These measurements complement the baseline established in the previous chapters.

Chapter 7 In the next chapters, we address the challenge to realize SMC as a service and develop the corresponding architecture of such a privacy-preserving system. To specifically address Questions **Q4** and **Q5**, we first refine them in Chapter 7. We discuss the findings of the previous chapters and derive a use case to be addressed. We present the related work for our architecture and therefore the following chapters. We develop a sketch of our solution which we use to conduct further analyses of the solution space. From this, we obtain a rich set of requirements that guides us in the subsequent chapters.

Chapter 8 Then, we focus on the creation of the first part of the anticipated architecture: the development of an orchestration layer for SMC which enables setup and execution of SMC sessions in dynamic environments (**Q4**) in Chapter 8. The central idea is providing management nodes, called *Gateways*. These are capable of orchestrating a group of SMC peers: The Gateways stay informed about the data the peers individually provide and the computations they offer. They hold an active connection to the peers. This allows to send control messages for initiating and conducting SMC computations. During the computations, Gateways monitor ongoing sessions and perform recovery activities in case a computation failed due to host or network errors. As a last point, we discuss how our approach is generic instead of being tied to a specific SMC implementation.

Chapter 9 We extend our solution in Chapter 9. The data made available via SMC becomes accessible to outside third parties. For that, these do not have to be enabled for secure computations themselves. A traditional data query is sufficient. This method of data retrieval is extended by two layers of access control. On the one hand, the Gateway is able to check the legitimacy of incoming queries. On the other hand, the query is forwarded to the actual SMC peers in an authenticated and integrity-protected fashion. This enables the peers themselves to control how their data should be used and permits them to refrain from cooperation if their privacy rules are violated. By doing so, we also fulfill two additional privacy protection goals of our notion of privacy from Chapter 2, ultimately obtaining a fully privacy-preserving service (**Q5**).

Chapter 10 Chapter 10 performs a roundup of the gained insights by comparing traditional architectures against SMC, and in particular our contributions. We examine several productively deployed systems which fulfill a similar goal of distributed data collection and data processing as our approach. We generalize them to find a common abstract architecture and define several categories for assessment. Namely, we examine differences and commonalities in terms of architectural and infrastructural characteristics, data protection properties, performance and resource consumption. This enables us to assess in which cases SMC is a helpful approach and fit for application.

Chapter 11 In Chapter 11, we conclude the thesis. We provide answers to the research questions and suggest further directions for research on SMC and its application.

1.3 Publications in the Context of this Thesis

The following papers have been published in the context of this thesis. We always indicate at the beginning of a section if its content has been published in one of these papers.

Chapter 4 M. von Maltitz and G. Carle. A Performance and Resource Consumption Assessment of Secret Sharing based Secure Multiparty Computation. In J. Garcia-Alfaro, J. Herrera-Joancomarti, G. Livraga, and R. Rios, editors, *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 357–372. Springer International Publishing, Barcelona, Spain, 2018

Chapter 7 M. von Maltitz and G. Carle. Leveraging Secure Multiparty Computation in the Internet of Things. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, pages 508–510, New York, New York, USA, 2018. ACM Press

Chapter 8 M. von Maltitz, S. Smarzly, H. Kinkelin, and G. Carle. A Management Framework for Secure Multiparty Computation in Dynamic Environments. In *Proceedings of 30th IEEE/IFIP Network Operations and Management Symposium*, Taipei, Taiwan, 2018. IEEE

Chapter 9 M. von Maltitz, D. Bitzer, and G. Carle. Data Querying and Access Control for Secure Multiparty Computation. In *Proceedings of the 16th IFIP/IEEE International Symposium on Integrated Network Management*, Washington, DC, USA, 2019. IEEE

2. Notions of Privacy

In this thesis, we want to contribute to technology which honors, preserves and protects the privacy of data and their owners. In order to achieve this, it is vital to have a clear understanding of the concept *privacy*, providing a goal to aim for. So, what does privacy mean?

In order to identify a concise and coherent concept which finally allows to be concretely addressed by technology, we have to narrow down the scope in which we want to understand privacy. Hence, we ask: *what does privacy mean in the context of technology handling information of individuals?*

In this chapter, we discuss a selection of proposals how to understand privacy. While they provide different perspectives on the same topic, they should be especially understood as an evolutionary process: the concept *privacy* was developed and diversified in recent decades in order to keep up with societal changes and especially with technological innovation.

We start with a glance at the Latin etymology of the term privacy. Then, we consider some social and legal approaches to understanding it during the 19th and 20th century. Afterwards, we proceed to current attempts of grasping privacy in social and also in technical contexts. This also considers the domain of data protection which is highly related to privacy.

Our review of concepts finishes with a description of *privacy* using structured protection goals. With this step, *privacy* reaches the level of conceptual maturity which *security* possesses since it was dissected into *confidentiality*, *integrity*, and *availability* (CIA).

2.1 Etymology

Two words can be identified as the root of the term *privacy*: *privus* and *privatus*. Consulting a Latin dictionary [LS79] their meaning is as follows:

privus is a) *single* b) *each, every individual* c) *one's own, private, peculiar, particular* d) *deprived of, without.*
Privatus is understood as a) *apart from the state, peculiar to one's self, of or belonging to an individual, private (as opposite to public)* b) *for persons: not in public or official life, private, deprived of office, a private individual, one who is not a magistrate, or in any public office* c) *for objects: isolated, apart from the villages, a private life, withdrawn from state affairs* c) *in private, in private use, from one's private property* d) *a man in private*

life, citizen (as opposite to magistratus) e) *in the time of the emperors: private, i.e. not imperial, not belonging to the emperor or to the imperial family, not given by the emperor.*

For our considerations it is interesting to see that the terms *privus* and *privatus* already depict the idea of an individual—as opposed to society or a group of persons—and his or her own personal affairs. These affairs are contrasted to the individual's public life and official function as well as the emperor and the state. This contrast contributes to the basic concept that there is a public sphere and a private sphere. Every action and information of individuals seem to fall either in the one or the other category. The next section shows that this interpretation of *privacy* is still current even millennia later.

2.2 The Right to Privacy

A very early treatise which argues for recognizing a new kind of personal protection, called *The Right to Privacy* [WB90], was published in 1890 by Samuel D. Warren and Louis D. Brandeis. The starting point are economic progress and technological inventions: "Instantaneous photographs" [WB90, p. 195] enabled taking pictures of people without their consent.[1] This led to undesired effects like the circulation of portraits of private persons. Concomitantly, journalism and newspapers invading the private sphere of people in order to publish details about their personal life ("idle gossip") are part of a current public debate.

Warren and Brandeis argue that these developments contradict moral and ethical standards: they cause notable emotional harm and mental pain to the victims of these infringements as well as moral decay of the recipients. However, the observed behavior is not yet recognized as legal misdemeanor. They argue that it should be understood as such and show that established legislation like *slander and libel*, *the law of defamation*, the *copyright* and implicit contracts are related but do not yet address the issues they identified: personal information, like letters, is not necessarily protected by the copyright, as they cannot be considered as valuable intellectual work [WB90, p. 201]. Similarly, the consequence of privacy violations is in their terms "mental pain", "distress", "injury of feelings", which are neither covered by the law of defamation [WB90, p. 197] nor libel and slander. These in turn only address "damage to reputation", only regarding harmed relationships to others. Concluding that the identified problems cannot be addressed with legal means at that time, they demand a new understanding of personal rights including protection from infringements of the stated form.

With a multitude of examples and distinctions from other laws they establish an understanding what should be protected and why. For them, privacy is the *right to be let alone*, which is in turn only one expression of a more fundamental principle of an "inviolate personality" [WB90, p. 205], the "immunity of the person" and "the right to one's personality" [WB90, p. 207]. Every individual should be able to decide for oneself to whom and to what extend one's "thoughts, sentiments, and emotions shall be communicated to others" [WB90, p. 198]. This further extends to expressions of the individual like private letters. Concomitantly, no one else should be allowed to publish or make these kinds of recordings available.

Aiming for a legal foundation for privacy they further develop the understanding of the *protection of the person*. Their concept of privacy is that an individual should be able to withdraw from the public world and control what and how much from this private sphere is shared with others.

The treatise of Warren and Brandeis is a milestone towards privacy and constitutes an important foundation for all later attempts to conceptualize privacy.

[1] Earlier, consent of photography was implicitly given, since persons had to sit motionless for several minutes in order to create the photo.

2.3 Constitutive Privacy

Picking up the arguments and the viewpoint of Warren and Brandeis at the very beginning of his seminal book *Privacy and Freedom* [Wes70], Alan Westin provides a clear definition of privacy:

> "Privacy is the claim of individuals, groups, or institutions to determine for themselves when, how, and to what extent information about them is communicated to others. Viewed in terms of the relation of the individual to social participation, privacy is the voluntary and temporary withdrawal of a person from the general society through physical or psychological means, either in a state of solitude or small-group intimacy or, when among larger groups, in a condition of anonymity or reserve. [Wes70, p. 5]

On the background of this definition, Westin argues that perceiving privacy as a value is not a new development, nor has it taken place as late as the 19th century despite the privacy milestone of Warren and Brandeis. Instead, basic forms of desiring and achieving privacy can already be found in the animal world. [Wes70, p. 7ff]

Regarding mankind, it is obvious that different societies do not share the same understanding of how privacy can be achieved or which interaction is actually perceived as privacy-preserving or privacy-invading. However, given the previous definition, and taking the actual societal background and norms, the availability of secluded spaces, the housing situation etc. into account, Westin argues that individuals in different societies and cultures all have a fundamental desire for privacy and differently developed ways and conventions how to achieve it.

Westin identifies four refined types of privacy: *solitude*, *intimacy*, *anonymity*, and *reserve*. Solitude is understood as a fundamental kind of privacy: seclusion yields physical distance and the most complete withdrawal from other persons and society. Intimacy means that "the individual is acting as a part of a small unit that claims and is allowed to exercise corporate seclusion so that it may achieve a close, relaxed, and frank relationship between two or more individuals" [Wes70, p. 34]. Anonymity is the method of keeping the own identity unknown to the persons an individual is interacting with, instead of restricting the amount of information communicated to them. On the contrary, as Westin states, anonymity allows an individual to be surprisingly open regarding private information, since the other person will not be able to exploit the gained information due to the unknown identity of the individual. Lastly, reserve means "the creation of a psychological barrier against unwanted intrusion" [Wes70, p. 35] and results in limiting the amount of information that is communicated by the individual to the persons in the surrounding, creating a mental distance. This can happen while being in the midst of large crowd, by signalling the desire to be let alone, e.g., stopping to speak [Wes70, p. 32], closing the eyes, hiding the face [Wes70, p. 13], or facing the wall[2] [Wes70, p. 15]. We would argue that reserve can be understood as a means to privacy of last resort. If all other ways of gaining privacy are impossible at a given situation, and privacy is desired, reserve will be the method of choice. Since reserve is a first step to self-censorship regarding the expression of the self, it is desirable to enable individuals to achieve privacy with other means which do not affect the own character and behavior at such a fundamental level.

Westin not only discusses what privacy is and how it can be achieved. He also elaborates why the need of privacy exists by showing the functions it fulfills for the individual. In other words, there are more fundamental values or desires which cause the need for privacy. We will only select those functions mentioned by Westin which are relevant for our question and our context.

[2] In that spirit, looking at one's own smartphone while being in the public space occurs also to be a way of achieving privacy from one's physical surrounding and one's fellow human beings.

A first function of privacy is to ensure autonomy, "the desire to avoid being manipulated or dominated wholly by others" [Wes70, p. 36]. Staying in control of what information is communicated to whom constitutes an essential part of it. A second relevant function is social release. If individuals have the autonomy to shape how they are perceived by others, they have the opportunity to embody different roles in different contexts. The "pressure of playing social roles" however, requires some relaxation by "lay[ing] their masks aside for rest" [Wes70, p. 38].

> [The need for privacy]—that is, insulation of actions and thought from surveillance by others—is the individual counterpart to the functional requirement of social structure that some measure of exemption from full observability be provided for. Otherwise, the pressure to live up to the details of all (and often conflicting) social norms would become literally unbearable; in a complex society, schizophrenic behavior would become the rule rather than the formidable exception it already is. (Robert Merton as cited by [Wes70, p. 63f.])

Similarly, privacy softens social norms and rules. The corpus of social norms contains a multitude of minor rules which can be expected to be broken often without larger harm done to the society. Here, privacy is a relief on the side of the individual as well as on society. The individual can expect not to be prosecuted for trivial offenses and—which is easy to miss— society is released from having to punish every single deviation [Wes70, p. 38]. "The firm expectation of having privacy for permissible deviations is a distinguishing characteristic of life in a free society" [Wes70, p. 39]. Two further functions, less relevant for our goals, are self-evaluation [Wes70, p. 40] and limited and protected communication [Wes70, p. 41].

In conclusion, Westin adapts the ideas of Warren and Brandeis, but argues that the need for privacy is even more fundamental. While it is present in all societies, the understanding of what is considered private and how privacy can be achieved highly differs. Additionally, he argues that it is not only in the interest of individuals to enable privacy in a society, but it is also vital for society itself. With regard to realizing privacy, he already recognizes that privacy in the public is desirable, that it can be achieved and that there are established methods for doing so. The last point is especially interesting, since thirty years later, Helen Nissenbaum again has to defend this understanding against opposite positions.

2.4 Contextual Integrity

Helen Nissenbaum [Nis98, Nis04] observes that the idea of a private sphere to withdraw from the world suggests the existence of an opposite. There seems to be a public world where information about every individual is freely available as soon as it was shared. This enables and allows the data to be processed and finally used "from whomever collects it" [Nis98, p. 1]. With the advent of new technological developments, huge parts of communication, interaction and transactions are now mediated by electronic infrastructure. This tremendously increases the ability to perform collection and processing of information.

She notices that the discussion about privacy still focuses too much on the private sphere and the personal space likewise, but leaves out public spaces. Hence, she aims to address what she calls *privacy in public*. Westin's theory already understood this value. However, Nissenbaum reinforces it again. From her point of view, society (including companies and government) does still not live up to this value and technological advances jeopardize it more than it was possible at Westin's time.

Conceptually, her approach is to dissolve the dichotomy of a private and a public sphere. Instead, she suggests to understand bits of information in the context they were initially given. Examples for contexts are the appointment with the doctor, a meeting with the bank consultant or an evening with friends. The intuitive understanding of privacy suggests

that some information is highly adequate in one of these contexts but considered misplaced in the others. Privacy is hence preserved if information and the context in which it is communicated match in terms of social norms and the individual's understanding. This condition is termed *appropriateness*. A second condition—*distribution*—addresses the flow of information. It focuses on how information is distributed, whether the flow is uni- or bidirectional, whether forwarding of information to third parties is legitimate, and how participants can choose which information to share. Her approach manages to transition from a static label for information, being private or not, to a dynamic understanding which also considers the current context, the individuals involved and the roles they play.

With *contextual integrity*, she lays a conceptual foundation for the protection goals *unlinkability* and *purpose binding*. The former demands that no two pieces of information given separately (and most probably in different contexts) should be combined. The result of combination could provide further, context-spanning insights which have not been intended by the originating individual. Purpose binding, on the other side, directly states that information may not be moved outside of its context and be reused for another purpose. Furthermore, since social and individual norms play an important role in this concept, the necessity for *intervenability* is increased. It enables individuals to actually influence how information is shared, based on their norms and opinions.

2.5 Global Privacy Standard and Privacy by Design

Another attempt to conceptualize privacy is the *Global Privacy Standard (GPS)* [Cav06]. Comparing "leading privacy practices and codes from around the world", the 27th International Data Protection Commissioners Conference aimed for "develop[ing] a harmonized set of fair information practices".

Starting with a high-level understanding of *privacy* which "relates to personal control and freedom of choice" [Cav06, p. 2], they identified ten principles to be respected when handling personal data of individuals. These principles regard the complete lifecycle of data from collection to processing to final deletion but are more organizationally than technically oriented. This standard encompasses the principles of *consent* of the data source regarding the data collection, use or disclosure; *accountability* of privacy-related actions; *purposes* have to be specified for processed personal data; *collection limitation* of personal data; *use, retention and disclosure limitation*; *accuracy* correctness, and up-to-dateness of the collected data; guaranteeing *security* of the collected personal data; *openness* of policies and practices to enable accountability, *access* of the individuals to their personal data, *compliance* of the executed processes with their stated policies.

In 2010, Cavoukian [Cav10] developed the information management principles *Privacy by Design (PbD)*. PbD identifies principles which help considering privacy protection from the beginning of the design of a system in order to make it a default property of newly developed systems. For existing systems, the principles give guidance for manual privacy audits. Structurally, PbD is located on a higher level of abstraction than GPS. Some PbD principles subsume multiple GPS statements; further principles are introduced by PbD which were not yet covered by GPS.

Cavoukian's contribution is not genuinely a stringent conceptualization of *privacy*. While the GPS only creates a vague idea of what privacy *means*, it provides a collection of principles to protect personal data and autonomy of individuals when facing organizations and data processing technology. Here, the GPS directly addresses the (organizational) contexts in which data is processed and aims for direct applicability. Also PbD, being more high-level than GPS, still does not constitute a fundamental concept. PbD provides aims for GPS and more general and process-oriented suggestions how GPS practices can be realized.

2.6 ISO/IEC 29100: Privacy Framework

The International Organization for Standardization and the International Electrotechnical Commission created the ISO/IEC 29100 [ISO11]. It "provides a high-level framework for the protection of personally identifiable information (PII) within information and communication technology (ICT) systems". Within this framework, organizational, technical and procedural aspects are addressed.

Their method encompasses several building blocks: initially they establish a common terminology with respect to handling and processing PII. This allows to express considerations and advice in this domain specifically and unambiguously. Furthermore, with defining roles of entities interacting with PII, they are able to create abstract categories for typical interaction patterns when handling PII. This allows to address these classes of problems generically when stating requirements and advices. They show domains which can influence the identification and relevance of privacy safeguarding and ultimately name principles of privacy protection.

The latter "were derived from existing principles developed by a number of states, countries and international organizations". This catalogue contains the following principles: *Consent and choice*, *Purpose legitimacy and specification*, *Collection limitation*, *Data minimization*, *Use, retention and disclosure limitation*, *Accuracy and quality*, *Openness, transparency and notice*, *Individual participation and access*, *Accountability*, *Information security*, *Privacy compliance*.

Similar to [Cav06], they focus directly on data handling and give direct advice which caveats to consider and which goals to fulfill. Due to that reason, our assessment of ISO/IEC 29100 is similar to the one about Cavoukian's work. It constitutes a pragmatic and practical approach of addressing the problem of handling personal data. On the ideological level, they do not provide an answer to the question of what privacy can mean. Instead, they refer to legal frameworks requiring the protection of personal data and the growing complexity of ICT systems, arguing that it makes a standardized approach to the realization of data protection necessary.

2.7 Privacy and Data Protection Goals

Pfitzmann and Rost [RP09] refer to the established protection goals for security (*confidentiality*, *integrity* and *availability*), and advocate for the continued usage of protection goals for describing certain properties when creating laws, infrastructures and systems. Following a very strict method they aim for building a structure in the space of present protection goals: they notice that *availability* and *confidentiality* constitute a *dual*, i.e., a pair of opposites. In their words, *availability* is guaranteed accessibility to some information during a specified time frame while *confidentiality* is guaranteed non-accessibility to some information during a specified time frame. Starting from this observation, they first aim for finding a dual to *integrity*. Understanding *integrity* as guaranteed genuineness, they define *contingency* to be the guarantee that there is no guarantee of genuineness for some information. This constitutes a foundation for derived protection goals like plausible deniability.

For data protection and privacy the authors introduce *unlinkability*. They define it as the impossibility to link information and entities among different domains.[3] A domain is constituted by a common purpose and context. Here, the influence of Nissenbaum's *contextual integrity* becomes obvious.

[3]For a detailed formalization of unlinkability see [SK03].

In order to enable data owners or users to understand how personal and privacy-critical information is processed, a further protection goal—*transparency*—is defined. By their definition, *transparency* is given with respect to a system component if an entity is able to gain insights into the component effectively making it observable and discernible. Among others, an important application of *transparency* is linkage control [Han12]: for data owners and users it should be recognizable under which circumstances which of their information is linked or at least potentially linkable over different domains.

The authors do not yet point to *intervenability* as a protection goal, but it is already recognizable when describing measures for realizing *unlinkability*.

In [BPPB11], the focus how to constitute privacy shifts to some degree. Borcea-Pfitzmann, Pfitzmann and Berg advocate that personal privacy is constituted by data minimization, user control and contextual integrity. Their main argument for the combination of these three is founded on two considerations: historically, these three concepts of privacy preservation emerged in chronological order as a reaction to the technical possibilities of that time. Similar, in the cases of data minimization and user control, further technological advances rendered the approaches insufficient by making their full realization impossible. In their concept, the understanding of contextual integrity is further differentiated. Nissenbaum states that the context in which personal information is disclosed may not be altered. Similarly, information may not be moved out of its original context into another one. They further differentiate this concept and coin Nissenbaum's concept *disclosure context*. A disclosure context is a set of characteristical environmental properties. Contexts shall support individuals to decide which personal information are appropriate and which to disclose in each context. Besides this type of context, the authors define *integrity contexts*. These are environmental properties which are attached to an information when it is disclosed in order to ensure its correct understanding and interpretation. In other words, it enables contextualization by providing explanatory metadata. With the high granularity of their concept, they aim for immediate application in technical systems.

Bock and Rost [BR11] directly refer to [RP09] and establish a double triad of protection goals: CIA for security and *unlinkability*, *transparency*, and *intervenability* for privacy. According to them, the goal of the latter triad is to operationalize two fundamental social requirements: system provider must be able to control their systems and prove this ability. Furthermore, it must be possible for all stakeholders to use a provided system in a fair manner. They perceive data protection as a measure to address and level the asymmetry of power which exists between organizations, i.e., service providers and individuals, i.e., service users [Ros17].

They also refer to Privacy by Design and the Global Privacy Standard (cf. Section 2.5). For them, they are a combination of privacy enhancing technologies and processes for improving data protection. They honor them as a modern understanding of which components constitute effective data protection. However, in their eyes, they lack a strict structure following an established method (like the definition of protection goals) and consider them too vague and organizationally to be technically realizable principles. For this reason, they advocate for the new protection goals for privacy and data protection mentioned above. The meaning of both, Privacy by Design and the Global Privacy Standard, is completely included in the data protection goals. Furthermore, weaknesses of both approaches, as identified by Simon Davies [Dav10], are addressed.

The strict concept of a double triad is reiterated in [HJR15]. The definitions of two of the three privacy goals are rephrased here: *Transparency* is the "property that all privacy-relevant data processing—including the legal, technical, and organizational setting—can be understood and reconstructed at any time". *Intervenability* is defined as "the property that intervention is possible concerning all ongoing or planned privacy-relevant data

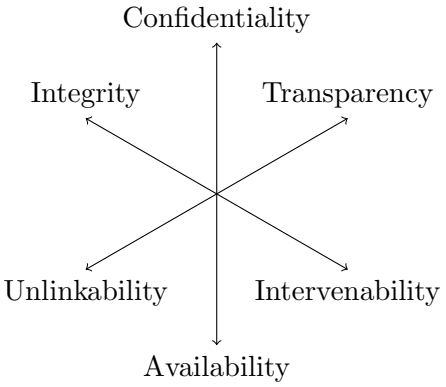

Figure 2.1: The six protection goals for privacy engineering [HJR15]

processing". They elaborate that the privacy and security goals are purposefully put into conflict by constructing the duals as explained above. Fundamentally, the duals constitute a star-like structure as shown in Figure 2.1 while the authors recognize that "several other interrelations among these six protection goals" exist. These interrelations and the impossibility to fulfill all goals simultaneously requires deliberate design decisions when aiming for privacy protection in a concrete system. These existing tradeoffs have to be reconsidered and answered anew for every design.

Working with protection goals for privacy and data protection similar to security found wide and important adaptation. The independent data protection authorities in Germany provide the *standard data protection model* [SDM15]. This acts as a unified concept for data protection consulting and auditing. Its aim is to narrow the gap between high-level legal requirements on the one hand and concrete processes and technical systems processing personal data on the other hand. Providing a common measure for assessment fosters consensus about what and how to protect and enables commensurability of different organizations or their technical realizations. They build their concept of privacy and data protection fully on the above introduced data protection goals and enhance it by the fundamental objective of data minimization. Besides that the data protection triad constitutes the basis for their model of protection planning, assessment and improvement.

The definitions of the protection goals show that they are located on two different levels which Hansen, Jensen and Rost [HJR15] call *hard privacy* and *soft privacy*. *Hard privacy* encompasses data minimization and unlinkability. They are system-oriented and their fulfillment depends mainly on how data is handled inside a system. It is independent of interaction with the stakeholders. *Soft privacy* focuses on the relationship of stakeholders (data owners, users, service providers, etc.) and the system. It considers the system's transparency for data owners and users as well as intervenability of stakeholders in the system's processes.

A second adaptation happened on the level of the European Union. The European Union Agency for Network and Information Security (ENISA) published a report on *Privacy and Data Protection by Design – from policy to engineering* [Eur14] in 2014. Among others, ENISA aims for developing "advice and recommendations on good practice in information security", assisting the implementation of relevant EU legislation and enhancing the existing expertise in EU member states in their domain of network and information security [Eur14, p. ii]. Similar to the aforementioned national data protection model, their report also contributes to "bridging the gap between the legal framework and the available technological implementation measures. In the domain of privacy principles, they refer to aforementioned approaches like the Global Privacy Standard and Privacy by Design, and to the structure of data protection goals alike. In other words, they understand data

protection goals as state-of-the-art and as a major contribution to the concept of privacy, which they seek to unify over the domain of the European Union.

2.8 Conclusion

This overview already shows that the societal conception of privacy has been subject to changes several times. It is interesting to see that these changes are often triggered when technical developments undermined the *intuitive feeling* of privacy while the *established concepts* of privacy were not able to name and grasp the breach. This led to a revision of the meaning of *privacy* and adaptations to the new technical and societal circumstances. The corresponding public discourse and theoretic reevaluation typically yielded a more specific, explicit and detailed concept.

Especially in the last decade, the discussion about privacy once more gained momentum and a lively discussion provided us with a protection goal based privacy concept which is well-aligned to the established security concept: structurally it bears the same shape, also consisting of a triad of protection goals. With respect to the content, it successfully complements *security* building the aforementioned double triad. This clear and concise concept of what privacy *constitutes* is supported by a multitude of approaches like the ISO/IEC 29100, PbD and GPS. These contribute a large amount of details to the concept and provide the means which can be used to fulfill these requirements on the level of organizations, processes and concrete technical systems.

Based on the current rich and faceted understanding of *privacy*, we also use the privacy protection goals established in [BR11] as foundation for our following considerations. In other words, when referring to *privacy*, we directly mean the protection goals *unlinkability*, *transparency* and *intervenability* if not stated otherwise. In this context, security (operationalized as CIA) is taken as premise. Especially, initial confidentiality often constitutes a requirement which is necessary but not sufficient to realize privacy.

3. Background on Secure Multiparty Computation

Secure Multiparty Computation (SMC) is a cryptographic primitive which has been devised in the 1980s and practically considered since the late 2000s. It will become clear that SMC is an excellent foundation for fulfilling our previously chosen privacy protection goals in technical systems.

In the next chapters we will assess practical applicability of SMC (Chapters 4–6) and we will investigate how SMC can be incorporated in the context of dynamic environments (Chapters 7–9). Now, we present a fundamental background on SMC to give context for the following chapters. We sketch the problem domain, explain the overall objective of SMC and present the corresponding security model. Focusing on realizations, we show how approaches can be categorized and on which base technologies SMC can be realized. Lastly, we present the state of the art by addressing the main directions of ongoing research: feasibility results, security, and performance improvements.

3.1 Problem Domain

We explain the problem domain for SMC using a selection of examples that represent a certain class of problems: in all cases, data should be processed among a group of participants without the need that any of them share their private information with any other party.

Example A dating platform provides a matching mechanism so that individuals can signal other participants whether or not they are interested. The goal is to identify matches, i.e., pairs (A, B) of individuals, where *A is interested in* $B \wedge$ *B is interested in A*. Realized naïvely, individuals are informed about an assessment as soon as it is given. However, when A is not interested in B, B's vote is of no use and gives away private information of B without any benefit. A better variant would be that A is only informed about B's vote if A also voted and is actually interested. In all other cases, A should not get any information. Commercial products like Tinder [Tin19] realize comparable mechanisms with the help of the provider acting as a Trusted Third Party (TTP). Votes are kept "secret" (i.e., only known to the TTP) except if an actual match happens. Then, both parties, A and B are informed. Instead of handing all votes to a TTP, it is desirable to achieve this functionality by mere communication between A and B (cf. [CDN15, p. 9f.]).

Example Several clinics treat patients of similar diseases. While each institution alone does not have enough data to obtain significant results from patient data, together they would. However, sharing patients' data with other parties, even clinics, is prohibited by law and would require explicit consent by each patient. Doing so would have several negative implications: Complying with legal requirements increases the effort for every clinic. All affected patients would have to be asked for permission; they could reject participation and render the amount of available data smaller or even systematically skewed. Also, security requirements increase since more data is then stored at a single place. Therefore, it is desirable to allow the clinics to perform their studies (given, the result itself is not privacy-critical) without actually sharing patient of individuals.

Example Companies of a certain sector want to understand how they are ranked among their competitors with respect to some key performance indicators. Their competitors are also interested in the ranking. However, the ranking is based on confidential values about the companies' performance, hence, no single company agrees to share that information with any competitor nor a neutral TTP. It is desirable to enable this ranking without needing any company to give away confidential information or finding a TTP which is actually trusted by all participants (cf. [BTW12]). This generalizes to all *coopetition* scenarios, where competitors want to allow some cooperation in terms of common data processing without undermining confidentiality of each party's data.

Example Today, auctions are typically performed by a TTP. It executes the process, collects the votes and announces when the highest bidder has changed. Since the only result information which must be made available is the final highest bidder and the price to be paid, all other bidding information is only necessary during the process for correct evaluation of the intermediate steps. It is desirable to realize auctions without a TTP and without making all bids available to anyone (cf. [BCD+09], [ZDT+16]).

These and similar cases are the scenarios where SMC can provide essential benefits by making data combination and processing possible without undermining confidentiality and privacy of the input data.

Notion of Trust

In the examples, we see that *trust* plays a special role. Generally, in these contexts, a TTP is an undesirable necessity. We refine this observation here by providing a definition of *trust*.

Definition 3.1 (Trust) Trust *is to faithfully believe that a component is handling data, which we deem to be private, only in the desired and expected manner, adhering only to the prescribed purpose and using the data for no other purposes nor sharing it with any other party.*

Trust also always includes the assumption that a component is actually capable to realize the protection necessary to achieve the aforementioned goals.

Trust is needed when there are no mechanisms in place which can actually enforce and ensure adhering to the privacy protection goals. Consequently, it is desirable to reduce the amount of trust which is necessary for a system to be privacy-preserving. With SMC we have an approach at hand which provides exactly this.

3.2 Objective

Against the background of the use cases provided in the last section, we can now present a formal description of SMC and its security respectively privacy properties. Our presentation follows Cramer et al. [CDN15]. They give the following definition of SMC:

Definition 3.2 (Secure Multiparty Computation [CDN15]) *[T]he parties, or* players*, that participate are called $P_1, \ldots, P_n$. Each player P_i holds a secret input x_i, and the players agree on some function f that takes n inputs. Their goal is to compute $y = f(x_1, \ldots, x_n)$ while making sure that the following two conditions are satisfied:*

- *Correctness: the correct value of y is computed; and*
- *Privacy: y is the* only *new information that is released*

Computing f such that privacy and correctness are achieved is referred to as computing f securely*.*

Such a computation is termed *multiparty* if referring to the general case of having $n > 1$ participants. In some works the special case $n = 2$ is considered and termed *two-party computation*. We will use the terms *party*, *player*, *peer* and *participant* interchangeably.

In general, two types of parties will be differentiated when considering adversary models: *honest* parties correctly provide their input and their aim is to successfully carry out the common protocol in order to obtain a correct result in the end. *Corrupted* parties are in contact with an adversary who is able to read all information of the party or is even capable to control the behavior of the party. An adversary can control more than a single party. In this case, the adversary is able to combine the information of all corrupted parties and to make decisions on this combined base of information. Protocols may have an upper bound $t < n$ of corrupted parties for which they still can be proven secure. Further information about the adversary are given in Section 3.3.1.

Privacy in the definition above states that y is the only new information released. To be more precise in what *new information* formally means, Cramer et al. define the view_j of a player P_j to be all values the player is able to see during the execution. Regarding the set of corrupted parties $C \subset P_1, \ldots, P_n$ with $|C| \leq t$, the values $\{\text{view}_j\}_{P_j \in C}$ can be termed the *leaked values*. In particular, some values are inevitably known to the corrupted parties, i.e., their input x_j and their output y_j; these are called the *allowed values*. Then—in simplified form—privacy can be defined as follows:

Definition 3.3 (Protocol Privacy) *A protocol is private if it always holds that the leaked values can be computed efficiently from the allowed values.*

This nicely encapsulates that there is a tradeoff between the protection and the utility of the information. In order to gain some value from information, it may not be completely locked up. In contrast, making information fully available would undermine protection. Hence, the desired property is *controlled leakage*, allowing the data owner to precisely adjust the tradeoff for the use case at hand. [CDN15, p. 3]

3.3 Adversary and Security Model

Like other cryptographic primitives, SMC protocols are typically evaluated using a formal security and adversary framework. It will become clear that the domain of SMC was able to adapt some established notions from other areas of cryptography but also needed some new formalisms which are specifically crafted for handling the security of computation protocols.

3.3.1 Taxonomy of Models

In this section, we give an overview of the relevant properties of adversaries which are considered in the context of SMC. In our presentation, we follow Canetti [Can00], as he comprehensively outlines the named notions. These notions are fundamental and hence, widely accepted and well-established.

Passive vs. Active

A *passive* adversary—also called *eavesdropping* adversary or *honest-but-curious* adversary—will collect all information it is able to gain control of but cannot modify the behavior of the corrupted parties. The parties conform to the prescribed protocol. The adversary's intention is to derive private information from the data it is able to access.

On the contrary, an *active*, *malicious* or *byzantine* adversary has the property of a passive adversary *and* is able to influence the corrupted parties. After corruption, these parties only follow the commands of the adversary and do not necessarily follow the protocol anymore. The two main goals of an active adversary are provoking leakage of private information of one or more players and/or influencing the final result at its own discretion.

Remark: Passive adversaries are obviously a weaker adversary notion. However, being secure in the passive security model was a first relevant goal for initial protocols to achieve. It can hence be seen as a stepping stone towards protocols which are also secure against active adversaries. Lastly, it is often argued that passive security can be sufficient in certain cases when it is reasonable to assume that players have a personal incentive to cooperate in conformance of the protocol in order to get a meaningful result [CDN15]. Furthermore, there are so called *semi-honest-to-active* compilers which can automatically convert passively secure protocols into protocols with active security (e.g., [GMW87, DOS17]).

Remark: Regarding active adversaries, it is vital to differentiate which types of malicious behavior are possible and which behavior can actually be addressed. Players are by design free to choose from the set of valid inputs for a given protocol (*choice of inputs* [CDN15, p. 11]). This is also true for the adversary, even if the choice is based on corrupt intentions. Due to this essential design decision, attacks based on the freedom of choice can also not be prevented and are said to be out of scope of SMC. The choice of inputs has to be differentiated from *deviation from the protocol*. The passive adversary has to follow the given SMC protocol and perform all steps correctly. The active adversary can behave inconsistently, i.e., pretend to have different input values when communicating with different parties, violate invariants, etc. This can actually be addressed on the level of protocol design.

Information Theoretic Security vs. Computational Security

In classical topics of cryptography like secret key and public key cryptography, the distinction between *information theoretic security* and *computational security* is already made. The former—also termed *perfect security*—is given if an adversary of even unbounded computational power and time is not able to break a scheme "due to the fact that [the adversary] simply does not have enough 'information' to succeed in its attack" [KL08, p. 47]. For example, this is the case when using a one-time pad where the plaintext is completely decoupled from the ciphertext, given the key is unknown. In opposition to this, computational security is a weaker notion where two relaxations are made [KL08, p. 49]:

1. Security is only preserved against *efficient*[1] adversaries that run in a feasible amount of time, and

[1] typically *probabilistic polynomial time*

2. Adversaries can potentially succeed with some *very small probability* (that is small enough so that we are not concerned that it will ever really happen).

Weakening the notion is actually beneficial: The tools for information-theoretic secure encryption are very limited and have some very impractical implications [KL08, p. 36]. Computational security now states requirements which are practically achievable by cryptographic methods.

Similarly, different notions of an adversary's power are established for secure protocols: the *secure channel* setting assumes absolutely secure point-to-point channels between each pair of participants and gives the adversary unlimited computational power. The *computational* setting, in analogy to the aforementioned notion, restricts the player to probabilistic polynomial time computations. In turn, he is deemed to be capable to learn all the (possibly encrypted) communication between the participants, i.e., no existence of secure channels.

Remark: Some of the later named base technologies like Linear Secret Sharing may be built upon information-theoretically secure constructs. However, it is vital to consider whether eavesdropping on all links between the participants would already allow reconstruction of the input values. In these cases, securing the point-to-point channels has yet to be fulfilled by cryptographic (and hence, most surely) *computationally secure* means.

Adaptive vs. Non-Adaptive

Typically, adversaries are not restricted to a single participant. Instead, it is assumed that they can compromise a larger number of players. Then, the following distinction is made: a *non-adaptive* or *static* adversary can choose an arbitrary but fixed set of players to corrupt. This choice has to happen a single time and cannot be adapted during the interaction. An *adaptive* or *dynamic* adversary, however, is able to first corrupt a set of players, and then proceed in corrupting further players during the interaction. Most importantly, the adversary is able to choose further victims based on the information gathered so far during execution.

Honesty of the Majority

Another fundamental distinction concerns the overall number of corrupted parties. Information-theoretically secure constructions must assume that the strict majority of the participants is honest. Otherwise, security guarantees do not hold anymore. Especially, this is also valid for the two-party case; here, realization of secure protocols for *arbitrary* ideal functionality is not possible [CLOS02].[2] Damgård et al. [DPSZ12] summarize these results as follows:

> In the case of *dishonest majority* [...] unconditionally[3] secure protocols cannot exist. Under computational assumptions, it was shown in [CLOS02] how to construct UC-secure MPC protocols that handle the case where all but one of the parties are actively corrupted.

3.3.2 Perfect Simulatability and Universal Composability

In order to prove desired properties of developed SMC protocols, an accepted formal method is needed. The aim of such a framework is to formalize the previously intuitive notion of whether certain desired properties (like correctness, privacy and security) are actually achieved in the given context. Most importantly, the formalism must correspond

[2] A simple example: computing the XOR function of two players, each providing a single bit, always leaks the input value of the other party.

[3] i.e., information-theoretically, author's note

to the real world to a sufficient degree, i.e., intuitively insecure protocols may not be deemed formally secure and vice versa.

While there already existed other notions of security (for other domains like secret and public key cryptography), it was not trivially possible to adapt them to secure protocols. Micali and Rogaway [MR92] describe it as follows: "[P]rotocols are extremely complex objects: after all, by defining security for encryption, signatures, and pseudorandom generation, one is defining properties of algorithms; but to properly define protocol security, one needs instead to define properties of the *interaction* of several algorithms, some of which may be deliberately designed to disrupt the joint computation in clever ways". In other words, proving protocol security made it necessary to reason about new cryptographic fundamentals and interactions.

There were previous attempts to formalize secure computation protocols, but from Micali's and Rogaway's point of view they "were either vague, or not sufficiently general, or considered 'secure' protocols that should have not been called such at a closer analysis" [MR92]. In their reasoning, the correctness and the (input) privacy of a given protocol should not be considered two separate properties, but proved simultaneously as being intertwined properties[4]. They strive in merging these two properties by using simulator based proofs—a notion for which they refer to [GMR85]: "[O]ur simulator plays the role of the uncorrupted players. In fact, the adversary interacts with a simulator just as though she were interacting with the network. The 'good' simulators (those which show that a protocol is secure) manage to interact with *any* adversary in a way which makes it indistinguishable to her whether it is the simulator or the network with whom she speaks."

In the following, we describe a short sketch of the proof method: secure protocols in general and SMC in particular aim for enabling interactions for which otherwise a TTP would be necessary. Hence, an *ideal world* is modeled, where an actual TTP is assumed. This entity cannot be corrupted and faithfully executes the desired functionality. For secure computation, it collects the inputs of all parties (including corrupted parties), computes the given function and announces the result to all participants. This setting is considered the ideal world, as it directly fulfills the aforementioned goal: the correct function is evaluated and none of the parties learns more than the own input and the public output. In the *real world* no such TTP exists; instead a secure protocol (possibly featuring interactions between all participants) is employed. Now, instead of proving single properties of the real world setup, one shows that the whole information collection by the (passive) adversary or all consequences by possible (malicious) interaction of the (active) adversary in the real world setting would also be achievable by the adversary in the ideal world setting. If this proof succeeds, it shows that the real world never performs worse, i.e., solely allows attacks which are also possible when a TTP is present.

Highly simplified, one can state:

Definition 3.4 (Perfect Simulatability) *A real world protocol π securely realizes an ideal functionality $\mathcal{F}$ with respect to a given adversary if there exists a simulator which achieves indistinguishability of π and an ideal protocol ϕ of $\mathcal{F}$ for this adversary.*

[4] Consider the following examples: per definition of secure protocols, we allow the result of the computation to be public and the adversary may not learn more than that and its own inputs. If this is the case, the protocol is considered private. However, the adversary may be able to legally manipulate the own inputs so that some other player's private input is directly published. The protocol would formally be considered private, although is intuitively not protecting privacy. To catch this case, we require correctness in order to ensure privacy.

Vice versa, if privacy is not given in the first place, the adversary may be able to choose his input based on the knowledge of the other peers in order to achieve a desired result. The protocol will run correctly and the output will be considered formally correct with regard to the inputs. However, intuitively the adversary was actually able to manipulate the outcome of the protocol.

The approach of simulator-based proofs for SMC protocols is now widely accepted and applied up to the present day.

Universal Composability

Micali and Rogaway [MR92] also examine a property which they desire for realizations of SMC protocols as well as the formal framework of secure protocols: they term it *reducibility* if an SMC protocol can be built of smaller subprotocols which provide a specific functionality. Additionally, the formal framework should enable the following: the correctness and privacy of a full protocol should be provable while assuming that the subprotocols are not (yet) *real-world secure protocols*, but *ideal world protocols*. The immediate benefit is composability of protocols and modularity of proofs, as subprotocols can then be proven to be correct separately.

Canetti [Can00] addresses this challenge as *modular composition* and later, in a more general context including concurrency, as *universal composability*, proving its fulfillment in [Can13].

While working with another definition of composability—based on reactive simulatability [PW01, BPW07] instead of universal composability [Can13]—Bodganov et al. [BLLP14] enhance the notion and also provide a relevant practical result: in order to achieve full security guarantees in terms of universal composability, the input shares (created by secret sharing the players' input) must be made independent from the output shares which are finally made available to all parties. Otherwise, information leakage is generally possible. Firstly, they present how this rerandomization—called *resharing*—can be practically achieved and secondly, they formally show in which compositions resharing is actually necessary. These insights allow greater flexibility in protocol design: since performing resharing imposes a performance overhead, it is beneficial to know in which cases it is not strictly necessary and hence can be omitted.

3.4 Realizations

The previous elaborations address SMC as a theoretical construct without stating how SMC can be practically realized. We turn towards this question in the following.

3.4.1 Taxonomy of Approaches

There are fundamental properties in which SMC realizations can differ. We present these alongside with their most important representatives.

Boolean Circuits vs. Arithmetic Circuits

Creation of complex SMC protocols is achieved by composition using a set of primitive secure functions [CDN15, p. 36]. These compositions are called *circuits*. A circuit in general is an acyclic directed graph where the nodes are called *gates* and the edges are called *wires*. Each gate has at most two inputs and can have an arbitrary amount of outputs. There is a set of special nodes, one for each player, which have itself no input and an arbitrary amount of outputs. These represent the players' inputs. Analogously, the final outputs are defined: each player is assigned a node which has a single input and no output wire.

Fundamentally, one differentiates between *Boolean Circuits* and *Arithmetic Circuits*. The difference is the set of primitives which constitute the gates. In the first case, these are logical operations—`AND`, `OR`, `NOT`—, in the second, arithmetic functions—addition and multiplication (including scalar multiplication).

It is vital to point out that both have equal expressiveness; neither of both is able to compute more functions than the other. The reason is that logic operations can be expressed as arithmetic operations and vice versa. However, depending on the base technology, either of the two sets can be rather considered to be the primitives, while the others are then composed operations. The decision which base to prefer has considerate implications on the performance of certain derived operations.

Base Technology

We see that complex functions can be built from small set of primitive operations. Specific technologies are then necessary which provide these primitives. We will present three of them that are well-established: garbled circuits, homomorphic encryption and secret sharing. Beforehand, we touch another direction which does not provide primitives but evades their necessity.

Single-Purpose Protocols A comparatively low-hanging fruit is the design of special-purpose protocols. Given a specific use case, a privacy-preserving protocol is developed which exactly matches this purpose. Therefore, these approaches do not provide basic primitives for modular composition.

From a formal and an academic point of view, these protocols are merely feasibility results, but not necessarily promising work to build upon. The reasons are several disadvantages: their specificity hinders reuse, universal composability is not trivially given, and for every newly designed protocol (be it just a derivation of a previous one) a new privacy and correctness proof has to be carried out. In other words, neither protocol modularity nor proof modularity is given.

Examples are [CKV+02] and subsequent work [RM09, SKM10]. The former aims for building an SMC toolkit, consisting of a set of single-purpose protocols, while stating that composability (and hence privacy of intermediate values) is not given.

However, in this context it is interesting to note what Ben Kreuter stated at the Real World Crypto Symposium 2017 [Kre17] regarding the employment of SMC at Google [BIK+17]: they are interested in using SMC in general, but do not try to use generic foundations. Instead, they also create single-purpose protocols for their use cases. The reason is mostly that real-world problems are not well-solved in the generic proposition: according to him, they do not cope well with constraints present in today's Internet infrastructure (e.g., lack of direct connectivity, NAT), for the devices in use (e.g., energy consumption on smartphones), are not robust against real-world errors (e.g., failing end-devices during the protocol) and still require too much communication for high-scale deployment while "the network is usually the costliest resource".

Garbled Circuits Yao [Yao82, Yao86] laid the foundation for SMC in the 1980s by proposing new questions to be asked. He presented the *millionaire's problem*—two millionaires want to find out who is richer without letting each other know about the exact amount of their own money—and demanding a unified framework for secure computation problems so that formal reasoning about it becomes feasible. Roughly during the same time and in context of the aforementioned publications, Yao also presented garbled circuits as generic solution for a whole class of functions during talks he gave—however, there is no publication of him proposing garbled circuits as solution.

We outline the approach of garbled circuits briefly, following [Sny12]. The idea of garbled circuits is to replace the informed evaluation of a known boolean gate with two inputs

and a single output (e. g., `AND`, `OR`, ...) with a generic computation where knowledge about the type of the gate and the actual input or output values is not necessary anymore. In order to do so, the input and output values are replaced by random strings (while the property of equality is preserved). The gate itself can be expressed as a finite mapping, using the four lines in a truth table depending on the gate type, permuting all possible combinations of the both inputs being mapped to the corresponding output. In order to hide which gate type is currently evaluated, the tuples of the four lines are substituted by a generic computation which enables calculation of the obfuscated output value using the two obfuscated input values without knowing the type of the gate.

More specifically, two players P_0 and P_1 holding their private input x_0 and x_1 respectively want to compute a function $f(x_0, x_1) = y$. P_0 creates a boolean circuit out of f and then has to garble each gate of the circuit as follows: for $i \in \{0, 1\}$ let w_i be the input for P_i. Let w_2 be the output. Then for each $v \in \{0, 1\}, i \in \{0, 1, 2\}$ a key k_i^v is created. These keys are later used as inputs and outputs of the garbled gate. Additionally, *garbled values* for each combination of values for w_0, w_1, w_2 are created which allow deriving w_2 from the two corresponding input values: $gv_{u,w,r} : H(k_0^u || k_1^w || s) \oplus k_2^r$ where H is a hash function, s is a salt, $\oplus$ is the XOR operation and r is the application of the ungarbled gate on the values u and w.

P_1 receives the garbled circuit from P_0 and P_0's values. However, P_1 does not know the corresponding key for its own input. It obtains this input privately from P_0 by using 1-out-of-2 Oblivious Transfer[5] [GMW87]. Calculating $H(k_0^i || k_1^j || s)$ and XORing it with each gv yields four possible values for k_2^r. By using syntactical constraints or employing a magic number in the result values, it can be ensured that P_1 recognizes which of the four values yielded a valid result. This is the final result of this gate's evaluation and can be used as input of the next connected gate in the circuit.

The basic version of garbled circuits is highly inefficient, having a considerable communication overhead in terms of messages and the overall amount of data transferred extends to several gigabytes [Sny12]. However, since then, an ever growing amount of papers publishes improvements regarding the security [GMW87, Gol09, MNPS04, LP07] and performance [MNPS04, GMS08, HEKM11, KSS12, KS08, KMR14, PSSW09] of garbled circuits. These improvements are not by default compatible to each other and it is a challenge on its own to apply them in combination.

Homomorphic Encryption Homomorphic encryption exploits the homomorphic property of certain encryption schemes in order to allow computation on encrypted data.

Definition 3.5 (Homomorphism (cf. [Gri07])) *A homomorphism of a group $(A, \otimes)$ into a group $(B, \odot)$ is a mapping $\varphi : A \longrightarrow B$ such that $\forall x, y \in A : \varphi(x \otimes y) = \varphi(x) \odot \varphi(y)$*

In other words, the result of a function φ on the combination of values x, y using the operation $\otimes$ can be achieved by first applying the function φ individually on x and y and then applying the operation $\odot$ afterwards.

Rivest, Adleman and Dertouzos [RAD78] initially brought up the idea that the same could be possible where φ is an encryption function. In consequence, one would be able to compute with encrypted data.

Formally, Katz and Lindell [KL08, p. 416] define homomorphic public-key encryption schemes as follows (notation adjusted):

[5] 1-out-of-2 Oblivious Transfer enables the receiver to obtain exactly one of two possible values while the sender learns nothing about which value has been selected.

Definition 3.6 (Homomorphic Public-Key Encryption Schemes) *A public-key encryption scheme (*$\mathtt{Gen}$*,* $\mathtt{Enc}$*,* $\mathtt{Dec}$*) is* homomorphic *if for all n and all [pairs of keys] (pk, sk) output by* $\mathtt{Gen}(1^n)$*, it is possible to define groups $\mathcal{M}, \mathcal{C}$ such that:*

- *The plaintext space is $\mathcal{M}$, and all ciphertexts output by Enc_{pk} are elements of $\mathcal{C}$.*
- *For any $m_1, m_2 \in \mathcal{M}$ and $c_1, c_2 \in \mathcal{C}$ with $m_1 = \mathtt{Dec}_{sk}(c_1)$ and $m_2 = \mathtt{Dec}_{sk}(c_2)$, it holds that*

$$\mathtt{Dec}_{sk}(c_1 \cdot c_2) = m_1 \cdot m_2,$$

where the group operations are carried out in $\mathcal{C}$ and $\mathcal{M}$, respectively.

Several encryption schemes possess a homomorphic property, including but not limited to RSA [RSA78], ElGamal [ElG85], the Goldwasser-Micali [GM82] and the Paillier [Pai99, DJN10] system.

Typically, the homomorphic property of each of these systems only exists with respect to a single *inner* operation (the operation performed before encryption, $\otimes$). E.g., using ElGamal and RSA, an outer multiplication yields an inner multiplication, while in the Paillier system, an outer multiplication yields an inner addition. Schemes that support only a subset of all possible circuits (cf. Section 3.4.1) are termed *Somewhat Homomorphic Encryption (SHE)* [ABC+15].

Armknecht et al. [ABC+15] consolidate the emerged terminology of homomorphic schemes and suggest a corresponding hierarchy. The next step represents *Levelled Homomorphic Schemes*. These are constrained regarding the circuits they are able to handle. They may only extend to a parametrized depth d which has to be fixed and known in advance during key generation. In turn, the ciphertext and the output of the evaluation function (which performs the encrypted computation) is *compact*, i.e., they may not grow depending on the depth of the evaluated circuit. Lastly, they define *Fully Homomorphic Encryption (FHE)* schemes to be able to compute all possible circuits. Furthermore, their length does not have to be known in advance.

In 2009—more than 30 years after the first thoughts about FHE by Rivest, Adleman and Dertouzos—Gentry [Gen09] proposed the first fully homomorphic encryption scheme and implemented it in 2011 [GH11]. According to [ABC+15] his scheme is noise-based, "which means that the plaintext is hidden by noise which can be removed by decryption. However, this noise increases with each homomorphic evaluation, and once it exceeds a certain threshold, decryption will fail". He solves this problem by making his scheme *bootstrappable* which effectively means that it can evaluate its own decryption function. As a consequence, increased noise can be repeatedly reduced by performing a homomorphically secured decryption. This enables evaluations of arbitrary length.

FHE would be a very desirable building block for SMC. Computation on encrypted data would generally allow any *untrusted* third party to perform the computation, while privacy and secrecy are preserved. This would facilitate many SMC setups as mutual and synchronized connections between all participants would not be required anymore. Instead, traditional client-server architectures—being essentially easier to realize—and asynchronously sent messages would suffice.

However, FHE is far from being practically applicable. While measurement results are only sparsely available (cf. [ABC+15]), they indicate that bootstrapping and/or evaluations usually take minutes to hours even on high-performance hardware. Nevertheless, hope for practical use should not be given up yet as SMC in general was considered impractical thirty years ago. Additionally, SHE schemes do find practical applications. An SMC protocol presented later [DPSZ12] utilizes SHE in a preprocessing phase, which enables and speeds up the actual computation.

Secret Sharing Secret Sharing addresses the problem to make a secret only accessible if a predefined number k of *share*holders cooperate while $k-1$ shareholders have no ability to recover the secret. We present a corresponding definition following Shamir:

Definition 3.7 ((k, n) threshold scheme [Sha79]) *A (k,n) threshold scheme divides a secret D into n pieces $D_1, \ldots, D_n$ in such a way that*

1. *knowledge of any k or more D_i pieces makes D easily computable;*
2. *knowledge of any $k-1$ or fewer D_i pieces leaves D completely undetermined (in the sense that all its possible values are equally likely)*[6]

Shamir's Secret Sharing The practical scheme presented by Shamir is based on polynomials in the 2-dimensional plane. A random polynomial of the degree $k-1$ is generated where the y-offset represents the secret to be shared. Then n (x, y) tuples of the polynomial—the *shares*—are computed and distributed among the n shareholders. The degree of $k-1$ ensures that an arbitrary combination of k shares allows polynomial interpolation. The reconstruction of the polynomial by interpolation also recovers the y-offset which is the desired secret value.

While secret sharing seems unrelated to the problem of SMC at first glance, it becomes applicable due to a homomorphic property of some sharing schemes: operations which should be carried out on the secret input values can instead be done simultaneously on the shares. Recombining them afterwards yields the result of the computation, as if it were executed on the secret values [BOGW88].

Given multiple secrets of different parties are transformed to shares using Shamir's method. Then, operations which should be carried out on the y-offset, i.e., the secret values, can—to some degree—be carried out on the shares, i.e., the y-values of the (x, y) tuples. Specifically, when n players aim to carry out a common function $f(s_1, \ldots, s_n)$, they proceed as follows: each player p_i for $0 < i \leq n$ creates n shares $s_{i,j} = (j, y_j)$ for $0 < j \leq n$ out of its own secret s_i. Then all shares are distributed so that player p_j receives all shares $s_{i,j}$ for $0 < i \leq n$. I.e., player one now has the first share of all players; the x-value of all these shares is 1. All other players obtain the corresponding tuples analogously. Then, the common function is carried out on the set of obtained shares yielding $s_{\text{result},j}$. Afterwards, these result shares are distributed to all players so that every player can reconstruct the result polynomial whose y-value is then $f(s_1, \ldots, s_n)$. Obtaining the plaintext of a shared value is called *opening* it.

Realizing the operations of addition and multiplication is sufficient to enable computation of arbitrary functions (cf. Section 3.4.1).

For SMC, a (k,n) threshold scheme where $k = n$ would be most desirable because this would require the shares of every participating player for reconstruction. However, multiplication on polynomials induces a notable constraint here: when two polynomials of degree $k-1$ are multiplied, the degree of the result polynomial is $2k-2$. Then, reconstruction is only possible if $2k-1$ shares are available for interpolation. This is not the case with the aforementioned scheme, as only $n = k$ shares are present. In order to allow reconstruction after multiplication, the degree must be constrained by $2k-1 \leq n$ which yields $k < \frac{n}{2}$. However, using such a construction, $\frac{n}{2}$ dishonest and colluding parties would also be able to illegally reconstruct the secrets. From these considerations, the necessity of the honest majority premise (cf. Section 3.3.1) is derived. The problem is further elaborated in [BOGW88] and the tightness of the boundary is shown there. In contrast to multiplication, addition does not pose any constraints.

[6] The scheme is hence information-theoretically secure, author's note.

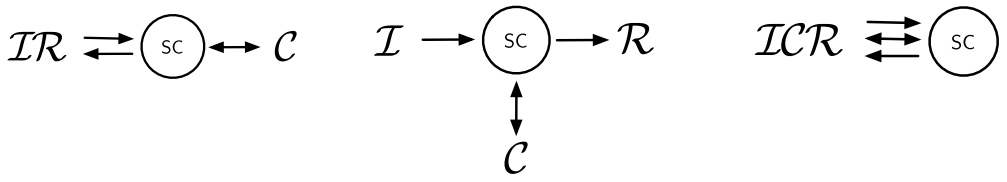

Figure 3.1: Different usage models for SMC by [ABPP16]

Additive Secret Sharing Shamir's secret sharing and polynomials are not the only method to achieve sharing based multiparty computation. Additive secret sharing is another approach which is most notably used by [BLW08, Bog13, BLLP14, DPSZ12, KOS16, DKL$^+$13, KPR18]. Here, the secret s_i of player p_i is not encoded in a polynomial, but the input value is split into random shares $s_{i,j}$ which additively constitute the secret value:

$$\sum_{j=1}^{n} s_{i,j} \equiv s_i$$

Since the sharing is built in another way, primitive operations are also constructed differently [Bog13]. Nevertheless, they again constitute the basic building blocks for arithmetic circuits which in turn allow computation of arbitrary functions. This approach also is information-theoretically secure.

Roles of Participants

Abstractly, the interaction in SMC can be modeled by specifying different roles for the participating entities [Bog13]: *input parties* are entities which provide data for the computation. Their input has to be protected, hence they already have to send their inputs as shares to the *computing parties*. The task of these parties is to perform the actual computation, already without knowing the content of the data to be computed on. Finally, *output parties* are the entities which obtain the output of the computation and are able to read it in plaintext.

Regarding security, the benefit is that there is no single entity which has to be trusted to handle the data faithfully and to be non-adversarial. The security of the system now depends essentially on the (weaker) assumption that the computing parties are *non-colluding*: since the shares of all (or even only k) computing parties combined yield the plaintext inputs, they are trusted not to cooperate in order to reconstruct the private inputs. Similarly, the channels between the computing parties must be assumed to be secure: either by the SMC realization itself or other cryptographic means. Otherwise, a *global passive observer*, listening to all channels, would also be able to reconstruct the inputs.

In general, it is possible that a single entity plays multiple roles. It is often the case that the computing parties are also the parties which finally *open* (i.e., derive the plaintext of) the final result. The reason is that in many use cases the result is deemed non-critical and non-private. Hence, it is appropriate to omit dedicated parties for deriving the plaintext output. Similarly and if appropriate, the input parties can be used as computing parties. If an approach is used which is secure even in case of $n-1$ malicious parties, this directly achieves security for each honest participant without further assumptions.

Varying the distribution of the roles to the available nodes makes different usage models (cf. Figure 3.1) possible. The choice of the model depends on the specific use case. For more information and several examples see Archer et al. [ABPP16].

Computational Model

The computational model specifies which assumptions are necessary in order to prove the security of a given system. Stronger adversaries with fewer constraints or assumptions allow the derivation of stronger security guarantees.

Standard Model The *standard model* or *plain model* is the most general and constraint-free model for computational security. The only assumption is that the adversary is constrained regarding its computational resources and time. The resource constraint is specified by only allowing *efficient*, i.e., polynomial-time algorithms to be performed by the adversary.

Preprocessing Model Ishai introduces the term *linear preprocessing model*. It dissects a protocol in two phases. Firstly, one assumes that "there is a trusted setup phase where a dealer can provide clients and servers with linearly-correlated resources, e.g., Shamir-shares of random secrets" [Ish05]. Based on this assumption a second, provably secure phase of the protocol can be realized. Afterwards, the trusted dealer of the setup phase is realized "by a secure protocol using public-key techniques" [Ish05].

By doing so, it is even possible to prove different levels of security for both phases. In the examples of [DPSZ12, KOS16], which are discussed later, the second phase is information-theoretically secure under the given assumption, while the first phase is based encryption schemes and hence cryptographically/computationally secure in the standard model.

Due to this dissection, subsequent research can work on both aspects independently. [DPSZ12] present a highly efficient second computation phase, whereas the first phase is based on SHE. As this takes several hours of preprocessing, further research in [KOS16, KPR18] strongly focuses on improving the preprocessing phase.

Other models exist but they are not relevant for our context.

3.4.2 State of the Art

Over the years, three main research directions emerged in the field of SMC. The first one is the achievement of feasibility results. They fundamentally show for selected domains that practical application of SMC is actually possible. Many of them emerged when SMC was newly found to be practically usable at all, but there are still feasibility results published today for newer applications like machine learning [BIK+17]. The second direction focuses on strengthening the adversary model and the security of SMC. Many base technologies assume non-deviation from the protocol and only honest-but-curious adversaries. While there are real world applications for which this adversary model is sufficient, in many cases no trust between participants can be assumed. In order to make use of SMC in these cases, active security is aspired which protects the participants' input data also in the presence of malicious adversaries. The third direction are performance improvements. Execution of SMC often implies a high degree of communication between all participants. Additionally, some basic technologies rely on expensive cryptography. This leads to performance bottlenecks either on the side of the computing hosts or of the network connection between them. Improving performance is also a vital direction to widen the field of application for SMC. For many real world scenarios, the performance penalty of SMC is still prohibitive.

3.4.2.1 Feasibility Results

The first practical and large-scale application of SMC happened in 2008 [BCD+09]. Its purpose was to perform an auction with multiple sellers and multiple bidders for computing

a market clearing price[7]. The data was collected from 1200 users without special technical experience by using a Java applet in a web browser. It was separated in shares locally and each share was encrypted for one of the three computing parties. Data was then sent to a single central server collecting the shares. At the time of computation, three laptops were used as computing parties being connected in a local network. Every owner obtained his shares from the collection server, decrypted his shares locally by entering the password and then started the computation manually. The whole computation, working with 1229 bids encompassing 9 million individual numbers took 30 minutes in a 100 Mbps intranet setting. The authors do not give insights how the results were distributed to the initial data input parties.

Martin Burkhart applied SEPIA—developed by him during his PhD [Bur11]—to event correlation for network data [BSMD10] in 2010. The overall goal is generation of network traffic statistics and anomaly detection. This was realized by computing histograms, entropies and performing distinct counts of input values. The collected data originates from 140 input parties while varying the number of computation parties between 3 and 9. They used Shamir's Secret Sharing and enhanced it by protocols realizing bit operations. Working with 65000 inputs per input node in a 100 Mbps intranet setting, all statistical computations took around 1 to 2 minutes.

Bogdanov et al. [BTW12] set up SMC in 2012 for computing statistical financial indicators of information and communication technology companies. These give helpful insights and the ability for self-assessment, but are based on confidential company information. Sharemind [Bog13] was used for realizing an Oblivious Batcher's odd-even merge sorting network besides other functionality for performing ranking operations. Their protocols are based on additive secret sharing, the number of computation parties is 3. The number of input parties is not documented, but a questionnaire performed along their work implies that there are around 15–30 data input parties. They provided a web-based solution which created the shares locally in the browser via JavaScript before submitting them to three computational nodes. These were located in three participating companies having the necessary knowledge to maintain such an instance.

Djatmiko et al. [DSD+13] reused SEPIA in 2013 to perform collaborative outage detection. They used the existing *Flow-based Approach for Connectivity Tracking*—which typically works with a single input source without privacy considerations—and extended it to work privately with multiple inputs. The core operation is a multiset union operation which they realize by *counting bloom filters (CBFs)*. Their CBF works on an integer array of length 32.768, which has to be generated from equally-shaped, local arrays. The necessary addition of all arrays is then performed via SMC. In other words, the main application of SMC in their context is the summation of integers. Using 90 input parties and 9 computation parties, they show that this is possible in under one second.

In 2016, Zanin et al. [ZDT+16] applied SMC to an auction method implementing the EU Emission Trading Scheme. They made it possible to transmit CO_2 emission allowances between airlines. Based on SEPIA, they implemented a comparison protocol by Nishide and Ohta [NO07] to obtain the necessary auction scheme. While varying the input parties and the computation parties from 3 to 10, the overall computation time stays between 20 and 90 seconds. The measurements indicate that while clear trends of the computation time can be derived from both named parameters in the local environment, influences in the cloud environment are notably more complex. This results in durations which do not follow a clear trend.

[7] Given multiple prices where sellers are willing to sell a certain amount and where bidders are willing to buy a certain amount of a commodity, then the market clearing price is the price where the total demand equals the total supply.

In 2017, Bonawitz et al. [BIK+17] from Google published a solution for SMC-based privacy-preserving training of Neural Networks (NNs). The trained model of a NN consists of the adjusted weights of the nodes of the network. This can be modeled as an integer vector where the length is the number of nodes of the NN. Training such a model *privately* means that a global vector is build from a multitude of local input vectors without making these available to anyone. They explicitly stress that privacy is more important in their scenario than completeness of input data or its actual correctness. The reason is that the input data is highly critical regarding privacy and data protection, whereas the NN is robust against slight errors during the training. Abstractly, they address *Secure Aggregation*, facilitating their use case: they do not need several computing parties which are able to carry out arbitrary computations. Instead, they only need a single server which finally holds the global model while being oblivious to the input data. This also means that only summation must be supported. Notwithstanding this functional simplification, they address a multitude of real-world problems simultaneously, like interrupted connections, vanishing participants, and NAT-shielded devices. They hence require that their protocol works asynchronously, a single faulty participant cannot jeopardize the full computation and that a centralized architecture is sufficient which only requires communication of each input party with a single server. Their approach is similar to additive secret sharing, where the input is recoverably masked by addition or subtraction of random numbers. This masking is *pair-wise*, i.e., all random numbers add up to zero so that they single out each other when the result is combined. For handling dropped out clients, they introduce a mechanism to reconstruct the exact mask of the failed clients. In order to further protect their privacy[8], another layer of masking is added. These masks are entangled in the protocol so that only two cases becomes possible: either a privacy-protected input vector is correctly incorporated, or the input vector including its otherwise distorting mask can be prevented from being incorporated in the model. Their performance measurements indicate that the computation with hundreds of clients and hundred thousands of vector elements only costs a low amount of seconds per client and one to several minutes for the server. These results, however, do not incorporate communication latency.

3.4.2.2 Active Security

Information theoretic secure approaches like BGW [BOGW88] initially assume the honest-but-curious-model (cf. Section 3.3.1). This is not considered to be realistic especially when cooperation with unknown entities is necessary. Hence, a vital direction of further research is strengthening SMC approaches to be secure against active adversaries.

Ben-Or et al. [BOGW88] already presented an initial method of addressing malicious adversaries. However, their proposition was only robust against $t < n/3$ malicious parties. Currently, malicious adversaries are addressed by adding an information-theoretic message authentication code per share [BOZ11] or secret user input [DPSZ12, KOS16] using a global secret-shared key. The MAC scheme must also be homomorphic. This allows obtaining valid MACs for computed results when performing corresponding calculations on the MACs of the computation inputs. The key for validating the MACs is also shared secretly. This prevents malicious parties to forge valid MACs for manipulated data. Only in the end stage of the computation, when data cannot be manipulated anymore, the MAC key is recovered so that validation becomes possible. This approach enables security against $n - 1$, i.e., the maximum number of malicious adversaries. However, the authors of [DPSZ12] state that it is still an open problem to identify a cheating party without worsening the asymptotic complexity of their protocol to become superlinear. Recently, further improvements on the approach have been made by Keller et al. [KPR18]. In

[8] If a seemingly dropped out client comes back later and sends its data, the previous exact reconstruction of its mask would allow deriving the private input data.

[KOS16] the preprocessing in the offline phase has been carried out with the technique oblivious transfer. Here, they replaced this against somewhat homomorphic encryption. By that, they achieved a performance improvememt by roughly one order of magnitude.

3.4.2.3 Performance

From a performance view point, secure computation comes with two types of costs: computational costs and communication overhead. The communication overhead typically outweighs the computational cost. E.g., in [BOGW88], with n parties, the distribution of input shares is inherently of complexity n^2, as every party has to distribute its shares to every other party. Similarly, multiplication includes a resharing step which exhibits the same structure and hence the same complexity.

In 1991, Beaver [Bea92] proposed a method which allows to replace multiplication of two shared input values with a more efficient linear combination if a so-called *multiplication triplet* (a, b, c) is already known, where $c = a * b$ and a, b are both random. This allows protocols whose communication complexity is linear in n. Similarly, the initial sharing of secret inputs is also reduced to linear complexity if already shared random numbers exist among the participants.

These benefits, notably improving the performance of the computation, require the introduction of a preprocessing phase (cf. Section 3.4.1) which generates the random values and the multiplication triplets and distributes them securely among the participants. Improving the performance of the preprocessing phase is still an active field of research [KOS16, KPR18].

3.4.3 Frameworks

In this subsection, we elaborate on some practical implementations of SMC. Here, we focus on frameworks allowing an arbitrary number of input parties while not considering the special case of two parties.

Virtual Ideal Functionality Framework

During his PhD [Gei10], Martin Geisler developed the *Virtual Ideal Functionality Framework (VIFF)*. It is strongly influenced by the software used for the market clearing price auction described in [BCD+09] and in Section 3.4.2.1. VIFF is a reimplementation in Python featuring the Paillier cryptosystem [Pai99] as primitive for two-party computations and BGW [BOGW88] for multiparty computations. A focus was laid on real-world performance, which was addressed by allowing asynchronous and parallel circuit evaluation. The project is considered to be an academic prototype. It has been discontinued; the last changes have been made in 2014. The homepage [Gei] explicitly suggests SPDZ [DPSZ12] and MASCOT [KOS16] to be used instead.

Sharemind

Sharemind [BLW08] was mainly developed by Dan Bodganov [Bog13] during his PhD. It is implemented in C++ and uses an additive secret sharing scheme in the ring $\mathbb{Z}_{2^{32}}$. Its focus was also to provide a real-world suitable framework with appropriate performance. They therefore opted to prevent only passive corruption—which is less computationally expensive—and to only use three computation parties, which are typically different from an arbitrary number of input parties. The argument is that further computation parties increase the communication overhead. Sharemind is now part of the services of Cybernetica [Cyb17] provides. It is under active development but partially closed-source.

Security through Private Information Aggregation

Sepia [BSMD10] was developed by Martin Burkhart [Bur11] during his PhD. It is implemented in Java and uses the BGW protocol. Although he realized a general purpose framework, his focus was on enabling collaborative network analysis. Therefore, he provided specialized protocols for entropy computation, distinct counting, event correlation and top-k queries. His performance measurements with Sharemind show that Sepia is comparable regarding operations per second. Being an academic prototype, Sepia has been discontinued.

Framework for Efficient Secure Computation

Fresco [Fre18] is is written in Java and developed by the Alexandra Institute in Denmark, a non-governmental organization for IT innovation and IT research. It aims for being a non-prototypical, productively applicable generic SMC framework. Fresco provides an abstraction from concrete SMC primitives so that protocol specification can be performed independently. This allows to switch primitives afterwards while keeping the specified protocol unchanged. This especially enables simple incorporation of newest research results on SMC. Former versions of Fresco supported the BGW [BOGW88] protocol, while the current version provides the computation ("online") phase of SPDZ [DPSZ12]. A full support of SPDZ is currently ongoing. Our contributions use Fresco as the reference for SMC frameworks. Since the development of Fresco also took place while conducting this thesis, our earlier measurements use the BGW implementation while later measurements employ SPDZ.

FairplayMP

Malkhi et al. [MNPS04] developed FairPlay, a framework for secure two-party computation based on garbled circuits. Ben-David et al. [BDNP08] extended this framework for the multiparty case. They modified the underlying Beaver-Micali-Rogaway protocol [BMR90] and employed BGW for the generation of the gate tables. The code has been made available on GitHub in 2012 but has not been further developed since then.

SPDZ-2/MASCOT

SPDZ-2 [DPSZ12] [KOS16] is currently developed by the University of Bristol. The software is mainly written in C++ while the protocols can be written in Python. It features the SPDZ protocol. This protocol is based on additive secret sharing and computationally secure against active adversaries corrupting up to $n-1$ of n players. It follows the preprocessing model (cf. Section 3.4.1) being split into an online and an offline phase. The former performs highly efficient execution of the actual computation by doing a computationally expensive but input independent preprocessing step during the latter phase. In this offline phase, multiplication triplets (cf. Section 3.4.2.3) are created which reduce the communication complexity of multiplications in the online phase. MASCOT [KOS16] exclusively addresses the offline phase and changes its technological foundation from somewhat homomorphic encryption to oblivious transfer. This yields further performance improvements of the preprocessing step. Later, further improvements were obtained [KPR18] by switching back to SHE .

3.5 Summary

SMC became a vital cryptographic primitive which addresses a genuine security and privacy problem: multiple parties want to correctly compute a common function without sharing their individual input values with any other party. The theory of SMC flourished

beginning in the 1980s and provided a clear definition, precise descriptions of desirable properties and a suitable method for proving validity of SMC solutions. Over time, multiple approaches emerged which allowed first theoretical, then practical consideration of SMC. The most important ones are *Garbled Circuits*, *(Fully) Homomorphic Encryption* and *Secret Sharing Schemes*. The fundamental difference is the encoding of the computation as boolean or arithmetic circuits which strongly influences how secure evaluation can be carried out. Consequently, it also affects which type of computations can be performed efficiently and which ones imply high performance penalties. It is not yet foreseeable whether these approaches will coexist in the future or if one will replace the others completely.

Research in the field of SMC is still ongoing, proposing new fields of application for SMC and especially focusing on improving the security and the performance properties of SMC solutions. Under the assumption of computational security, some of today's approaches are secure against up to $n-1$ corrupted parties controlled by an active adversary. This is typically achieved in the preprocessing model, where an information-theoretically secure computation phase is preceded by a computationally secure preprocessing phase. This preprocessing phase provides precomputed auxiliary information which enable the actual computation. Concomitantly, expensive computations are shifted into the first phase, enabling a highly efficient computation phase.

Since its first practical implementation in the late 2000s, some general purpose frameworks have been built which enable the application of SMC in arbitrary contexts. Many of them have been research projects that have been abandoned afterwards. However, some remain under active development: Sharemind has been migrated into a closed-source product of the company Cybernetica and is now offered as a service. FRESCO is further developed by the non-profit organization *Alexandra Institute* in Denmark and freely available; SPDZ-2 remains a research prototype which is still further developed and extended frequently.

Part I

Performance and Application of SMC

4. Performance Assessment of Secure Multiparty Computation

In the previous chapter we gave an overview of the background of Secure Multiparty Computation. In the first time, SMC was only of theoretical interests; practical execution was infeasible. It was until the late 2000s that first practical attempts were performed.

Today, as hardware performance generally makes its execution possible, implementations of SMC become a promising building block of secure and privacy-preserving systems. Still, real-world applicability highly depends on the setting in which SMC is carried out. Especially due to the high communication overhead of SMC sessions, their performance is strongly influenced by the networking setting.

In the Chapters 7–9, we design an architecture for SMC to allow automated and self-dependent application. Our design decisions will be influenced by the following investigatios. They provide the insight in which settings SMC can be realistically applied today. This knowledge allows us to focus on the promising settings, keep their properties in mind and address their specific challenges.

Objective

This chapter addresses Research Question Q2: we investigate the performance properties of a state-of-the-art SMC implementation. Hereby, we examine how duration and resource consumption of SMC sessions depend on host and network parameters. Furthermore, we identify the bottlenecks of SMC sessions and find out which networking parameters influence SMC most.

4.1 Selection of Framework

Current SMC frameworks vary in the degree of completeness and maturity. Hence, as a first step in the evaluation, it is necessary to choose a framework among the candidates which is worth the more detailed examination. Our reasoning about the candidates is inspired by [ZDT+16]:

Universality requires the framework to be of general purpose, i.e. no specialized solutions for single use cases are desired. Due to our use case (see Section 4.2), we focus on frameworks which are applicable in *multiparty environments*. Mere two-party solutions are not

Framework	VIFF	FairplayMP	SPDZ-2/ MASCOT	Sharemind	SEPIA	Fresco
Universality	✓	✓	✓	✓	✓	✓
Multiparty	✓	✓	✓	✗	✓	✓
Open Source License	✓	✓	✓	✗	✓	✓
Active Development	✗	✗	✓	✓	✗	✓

Table 4.1: Assessment of SMC candidate frameworks for our performance evaluation

sufficient. Furthermore, regarding roles of parties one can differentiate between input parties and computing parties. Some frameworks only allow arbitrary scaling of the first type, but fix the second one to, e.g., three parties. Out of privacy considerations, we want that the input parties also take part in the computations themselves instead of outsourcing the computation. Hence, the framework must support computations with a theoretically arbitrary number of participants. The framework must be *open source* so that we can work with it as a whitebox and retain the ability to make changes to the code. Lastly, for making a relevant contribution which is able to influence the development of the framework, it is vital that the framework is still under *active development*. Some solutions from academia were used as a proof of concept and for research, but have been abandoned afterwards.

We summarize our assessment in Table 4.1. Due to the last requirement, we decide against Viff, FairplayMP, and Sepia. Sharemind does not fullfil our multiparty requirement and its source code is not publicly available. The remaining candidates are Fresco and SPDZ-2. From them we choose the former since it already reached a higher level of maturity, leaving the latter for future research when it becomes more stable and ready-to-use.

4.2 Use Case

In this section, we present the functionality which was implemented to perform our performance measurements. From the MeasrDroid system [Cha16] (also cf. [vMDC16, vMDC17]) we adapt a fundamental aggregation functionality. The average travelling distance of a set of moving client devices (e.g., smartphones) is computed by a central server.

Scenario

We assume a set of moving devices which know their own location using a GPS module. One property of interest is the average travel distance over the set of measured devices. Via a client application, these devices can connect to a central server, which has the purpose to collect statistics about them. Each time a client connects to the server, it transmits the distance it travelled since its last connection. At any time, the average distance travelled can be requested from the statistics server.

Logic

The stream of distances can be computed by every client locally and individually by inputting the current and the previous location to the algorithm [GPS] in Listing 4.1.

```
    const double PK = 180 / 3.14169;
    const double RADIUS = 6366000;

    double gps2m(lat1, long1, lat2, long2) {
        double a1 = lat1 / PK;
        double a2 = long1 / PK;
        double b1 = lat2 / PK;
        double b2 = long2 / PK;

        double t1 = cos(a1) * cos(a2) * cos(b1) * cos(b2);
        double t2 = cos(a1) * sin(a2) * cos(b1) * sin(b2);
        double t3 = sin(a1) * sin(b1);
        double tt = acos(t1 + t2 + t3);

        return RADIUS * tt;
    }
```

Listing 4.1: Calculating the distance between two GPS coordinates

The server collects these individual distances and calculates a current average from it. Such a running average can be collected insecurely with the code shown in Listing 4.2.

```
class Server{
    int sum = 0;
    int count = 0;

    void addValue(int value){
      sum += value;
      count += 1;
    }
    void getAverage(){
      return sum/count;
    }
}
```

Listing 4.2: Streaming Interface for Running Average

The advantage of this approach is that all input data does not have to be present at the same time, but can be gathered as stream with delay between the individual inputs. Additionally, the space and time complexity is $\mathcal{O}(1)$ as only a constant number of values has to be stored and only a single division has to be performed for calculating the average.

In order to apply Fresco to this problem, the input has to be organized in synchronous sessions. In every session, each device contributes its distance since the last session[1], whereas the statistics server inputs the current value of the running sum (starting with 0). By doing so, the server fulfills the technical requirement of also contributing a value while it does not semantically change the computed result.

The summation protocol for this step is shown in Listing 4.3.

```
    private SCEConfiguration sceConf = ...;

    public ProtocolProducer prepareApplication(ProtocolFactory factory) {
        final int numPeers = sceConf.getParties().size();

        BasicNumericFactory fac = (BasicNumericFactory) factory;
        NumericIOBuilder ioBuilder = new NumericIOBuilder(fac);
        NumericProtocolBuilder npb = new NumericProtocolBuilder(fac);

```

[1]The device can either measure its GPS position at the beginning of each session or poll its location at an arbitrary frequency and sum up the determined distance. The latter approach is more exact than the former.

```
10          SInt[] inputSharings = new SInt[numPeers];
11          ioBuilder.beginParScope(); // run in _par_allel
12          for (int p = 1; p <= numPeers; p++) {
13              inputSharings[p - 1] = ioBuilder.input(this.myInput, p);
14          }
15          ioBuilder.endCurScope();
16          ProtocolProducer closeInputProtocol = ioBuilder.getProtocol();
17          ioBuilder.reset();
18
19          SInt ssum = npb.sum(inputSharings);
20          ProtocolProducer sumProtocol = npb.getProtocol();
21          this.outputs = new OInt[] { ioBuilder.output(ssum) };
22          ProtocolProducer openProtocol = ioBuilder.getProtocol();
23          ProtocolProducer gp = new SequentialProtocolProducer(
24                  closeInputProtocol, sumProtocol, openProtocol);
25          return gp;
26      }
```

Listing 4.3: Secure Summation Protocol in Fresco v0.2

The result of each round is in turn saved by the statistics server. It serves as input for the next round and as intermediate result for calculating the average. With regard to the privacy of the system, it is uncritical that only the total sum is calculated privately while the division step to retrieve the average is performed without SMC. Under the premise that the number of participants is known (which is necessary to perform the sharing correctly) the total sum would always be derivable from the average value. Consequently, the total sum does not leak any more information than the average does.

Input Data

The test data we use was retrieved from the original MeasrDroid system. The data collected by the system stems from volunteering users who were willing to donate the sensor information of their smartphones including location data. GPS information was collected in intervals of 15 to 20 minutes (depending on the individual configuration per device).

In order to accomplish the aforementioned sessions, it is necessary to provide the same length of input data for every participating node. Working with 6 test nodes (one being the statistics server), we calculated the largest number of GPS information we can retrieve from the 5 most active donators. This yielded 20000 GPS tuples for each test node.

4.3 Preliminary Execution Time Considerations

The content of this section has also been published as Section 4 of [vMC18a]. The text was written completely by the author of this thesis.

A computational model for secret sharing based SMC protocol foundations like BGW "is a complete synchronous network of n processors" [BOGW88]. The protocol itself is dissected into rounds. "In one round of computation each of the players can do an arbitrary amount of local computation, send a message to each of the players, and read all messages that were sent to it at this round" [BOGW88]. A message typically contains a share of a private local value—e.g., a polynomial in the BGW protocol—held by the sender.

From this point of view, the protocol becomes an alternating sequence of local computation and network communication:

$$comp_1, comm_1, \ldots, comp_{m-1}, comm_{m-1}, comp_m \tag{4.1}$$

We consider recombining the shares to be the last step $comp_m$. Hence, there are only $m - 1$ communication steps.

The communication steps are also synchronization points for the players. I. e., no player P_i can already perform a computation $comp_{k+2}$, while another player P_j still computes $comp_k$. Being a single step ahead is however possible for a single player when it locally possesses a polynomial while all other players still wait for a share of it in order to proceed. We denote the time costs for a step $comp_i$ as $cost_{comp_i}$. The message sent from player P_k sent to P_l during $comm_i$ is referred to as $msg_{i,k\to l}$. Two phases are typical for all SMC protocols: during the *input phase*—a single round—the own private input is transformed into shares and distributed among the players. In the *output phase* the shares of the computed result are exchanged among all players. Their recombination yields the plaintext result. In Fresco this also takes a round[2]. The round complexity of the basic arithmetic operations in the BGW protocol varies. Addition does not need any communication. Multiplication requires rerandomization of the polynomial and the reduction of its degree [BOGW88]. This requires a step of communication, and hence, a round.

The communication cost of the ith round $cost_{comm_i}$ depends on the number of messages sent. As every player sends an individual share to every other player, the overall number of shares sent is $\mathcal{O}(n^2)$. Every player p_i typically contributes its own input v_i for the computation. Hence, a single multiplication step normally means that the product of all input values $\prod_{i=1}^{n} v_i$ should be computed. In such a case, $n-1$ single multiplications are necessary; so the costs for such an array multiplication are $\mathcal{O}(n^3)$. However, analysis of Fresco shows that sending and receiving for every player can happen in parallel[3]: sending is a non-blocking action for the computation layer which hands over the messages to be sent to the communication layer of Fresco. Receiving is blocking on the computation layer, however, the communication layer is able to receive all messages simultaneously.

When a host has sent out every share and it has received all other players' shares, the next computation step can be performed. So, in spite of the theoretical complexity and due to parallelization, the communication cost per round mainly depends on the slowest pair of hosts:

$$cost_{comm_i} = \max_{1\le k,l\le n} cost_{msg_{i,k\to l}} \tag{4.2}$$

One dependency to the number of players remains: while every round is practically performed in constant time, the number of rounds per array multiplication increases linearly. A further approximative simplification of the communication costs can be made: communication between two peers is always identically structured and of comparable length. Hence, we can simplify to

$$\forall i \in \{1,\ldots,m-1\} : cost_{comm_i} = cost_{comm}. \tag{4.3}$$

Note that Equation 4.3 does not hold for computation steps. The input phase and the randomization/resharing step (e. g., during multiplication) encompasses the generation of a new polynomial and the computation of single elements of it (which become the shares). The output phase mainly includes the calculation of the Lagrange interpolation for reconstructing final computation result from the obtained shares. Every single addition and multiplication of shares includes a single addition and multiplication of large integers[4]. Combining Equations 4.1 and 4.3, the overall costs of time can be estimated by

$$cost_{overall} = \sum_{i=1}^{m} cost_{comp_i} + (m-1) * cost_{comm}. \tag{4.4}$$

[2]Some solution perform a resharing in order to make the final shares independent from the shares obtained in the computation. This is, e.g., necessary when the shares should be reused to perform further calculation. Then, another round becomes necessary during this phase.

[3]One exception is the initial input sharing phase. Here, sending of shares is only performed by a single host at a time.

[4]Represented as `BigIntegers` in Fresco.

	SMC		TTP	
Phase	Computation per host	Communication (overall)	Computation on TTP	Communication (overall)
Close	Generation of polynomial, calculation of n shares	$n^2 - n$	—	n
Addition	$n - 1$ additions	—	$n - 1$ additions	—
Multiplication	$n - 1$ multiplications, Comp_{Close}, Comp_{Open}	$n^2 - n$	$n - 1$ multiplications	—
Open	Lagrange interpolation	$n^2 - n$	—	n

Table 4.2: Performance comparison SMC vs. TTP. Computations are counted in basic (arithmetic) operations, communication in number of messages.

Applying the model of the alternating sequence, the influences on the duration are twofold: the computation performance depends in the properties of the players, the communication performance depends on the properties of the network links between them. Due to the synchronizing behavior of rounds, the costs of both sides add up to the overall costs.

Performance Comparison

Conceptually, SMC replaces a Trusted Third Party (TTP) for collaboratively computing a common function by providing a secure protocol. Canetti [Can00] used this understanding to propose a now well-established method—the real/ideal paradigm—to prove secrecy and correctness of SMC protocol designs. The method is to show an isomorphism between the *real world* where SMC is applied and an *ideal world* where a TTP handles the computation.

We can apply this understanding not only to prove secrecy and correctness, but also to assess the performance penalty that SMC introduces. Using a TTP for computation is an alternative solution for the problem SMC solves, hence it can be used as a performance baseline here. In fact, in today's productively used systems, TTP solutions are the established standard; hence the comparison with a TTP is also practically relevant. In order to do so, we align the necessary actions when using a TTP with the phases of an SMC computation. In a TTP setting, the input phase can be understood as providing the input data to the TTP. The output phase comprises sending the result from the TTP to the participants. Computation steps can be directly adapted. The whole comparison applied to the BGW protocol is shown in Table 4.2.[5]

On the technical level, another distinction has to be made: choosing SMC as solution does not only change the number of steps to be performed and the structure of interaction but also the processed data: in our use case, we initially handle input values with the type `double`. A single primitive value has the length of 8 Bytes. On the other hand, the primitive value of the BGW protocol suite is a *share*. This consists of a (x, y) tuple denoting a point on the local polynomial. In Fresco the x is stored as a `byte` and y as `BigInteger`. By inspection, we could find out that y typically has a length of 8 to 12 Bytes.

4.4 Hardware Setting

We use six homogeneous non-virtualized bare metal hosts. These are each equipped with a Intel Xeon E3-1265L V2 CPU, having eight cores at 2.50 GHz and a cache size of 8192 Kbytes. Each host possesses 15,780 MB of RAM and a 1 Gbit networking interface.

[5] Common computations are omitted: E.g., the running sum has to be turned into a current average by a single division. As both solutions have to do the same step, it is not reflected in the table.

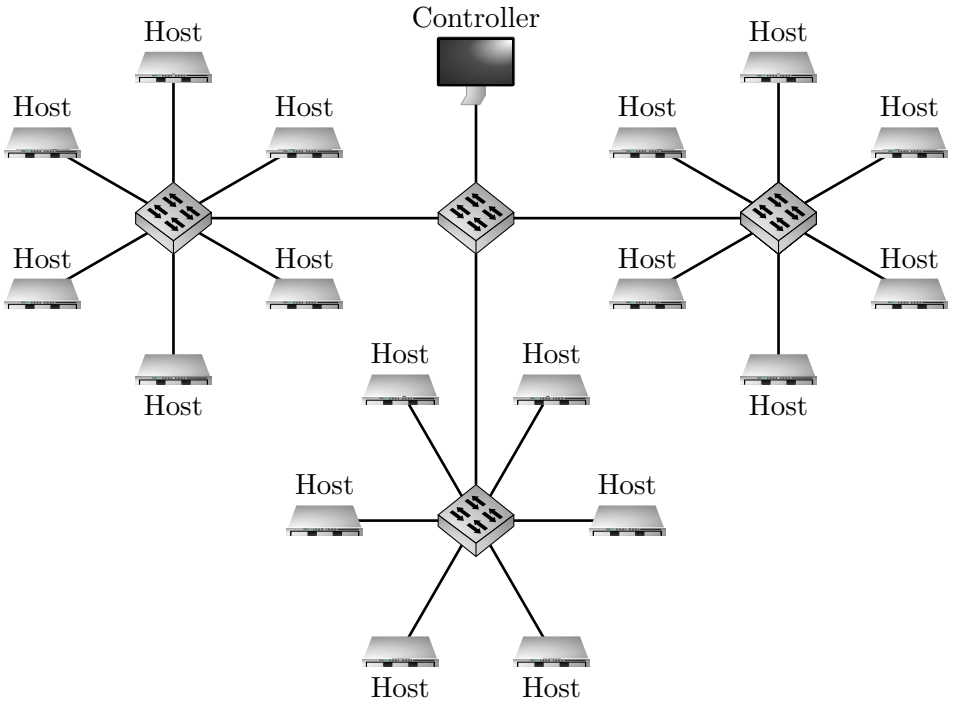

Figure 4.1: Topology of the test setup

All six hosts are connected in the shape of a star topology via a single switch. In conformance to the use cases, one host will be assigned the role of the operator. The operator participates in the collaborative computations, but does not provide own sensor values. Instead, it provides a neutral element (e.g., 0 in an addition) or the value of the previous computation, if a running sum or average is computed.

4.5 Measurement Setup

In this section, the software aspects of our test setup are described. This includes the operating systems of the host, and the measured software and the measurement software.

4.5.1 Host System

The test hosts use a dedicated PXE server to boot from. This server provides an image of Debian Jessie (8.5) using a 3.16 Linux kernel (cf. Listing 4.4).

```
1 Linux pc1 3.16.0-4-amd64 #1 SMP Debian 3.16.7-ckt25-2+deb8u3 (2016-07-02) x86_64 GNU/Linux
```

Listing 4.4: Host System

The hosts are completely non-persistent and fully reset upon reboot. This makes the utilization of an orchestration solution like ansible necessary in order to ensure the same test configuration every time the systems are rebooted (cf. Section 4.6).

4.5.2 Measured Software

The software under test is the Fresco framework (version 0.2) [Fre18] developed by the non-profit organization *Alexandra Institute*. It is a Java framework for secure multiparty computation. We evaluate the BGW protocol suite [BOGW88] provided by Fresco. BGW works with n-out-of-t secret sharings [Sha79].

The protocol suites like BGW comprise basic arithmetic operations like additions and multiplications. The Fresco framework enables users to create protocols for individual computations by combining these protocol primitives into larger sequences.

In order to evaluate the framework we wrote a lightweight wrapper which loads input data from a local storage, performs preprocessing necessary for the computations and starts the computations using Fresco. The specification of the evaluated protocols is described in the Use Case (Sec. 4.2).

The source code is compiled to a Java application which is in turn executed by the Java Virtual Machine from the OpenJDK 1.8.0_111.

4.5.3 Measurement Software

Profiling is performed using *perf* from the linux-tools (version 3.16+63), *BTrace* [BTr16] (version 1.3.8.3 (20160926)) and *tshark* (version 2.2.4).

perf

We use perf to count CPU cycles consumed and instructions performed by the executing process under test. Perf is event-based and obtains the indicators from hardware counters provided by the performance monitoring unit (PMU) [Per16]. In our context, only the overall number of cycles and instructions is considered relevant. Perf is prepended to the Java command executing the Fresco application. Perf in turn starts the process; the results are output when this process terminates. An important difference to BTrace is that, while the latter can specifically select methods to be profiled while excluding the measurement setup code, perf can only measure the Java process as a whole.

BTrace

BTrace is used to evaluate the memory consumption and the execution time of the application under test. It performs profiling by bytecode tracing, i.e., instrumenting the bytecode. BTrace consists of two parts: a Java library enables the user to write classes which specify the profiling logic and (via annotations) the methods of the code under test to be profiled. The second part is an executable `btrace` which uses a given instrumentation class to instrument a Java process accordingly. The process to be monitored is specified by its PID upon invocation of btrace. BTrace itself is then partially executed by the JVM.

BTrace allows event-based and polling-based profiling. Both options are necessary in our context. Similar to the CPU profiling above, wall-clock time measurements can be performed in an event-based manner: the profiling starts with execution of the Java process under test and stops with its termination; time counting is started when the specified methods are entered and stopped when they are left. However, the memory consumption of the Java process changes during execution time, hence it must be polled frequently in order to gain insights into the behavior of the process.

In order to avoid a non-neglible overhead by the instrumentation, we performed a preassessment. This is described in Section 4.6 and gives insights into the performance overhead depending on the polling frequency.

tshark

Tshark is part of the wireshark application and presents a command line interface to the core features of wireshark. We use tshark to capture the TCP packet stream constituting the computation communication. In the background, tshark uses dumpcap for recording the data stream. During evaluation, tshark is again used to convert the captured pcap-files to CSV which is used for further processing. By collecting the whole data stream, it is ensured that analysis is not constrained by a priori assumptions during data collection, but the full potential of the measured information can be exploited.

4.5.4 Orchestration Software

For evaluating multiparty protocols, execution of the corresponding processes on several hosts has to happen in sync. For this purpose, another layer of software is added which orchestrates the measurements. This layer uses the python library *baltinet*. Baltinet is similar to *mininet* [LHM10], but manages real instead of virtual networks. With its help, we built an orchestration script which conducts the actual measurements. The whole process is described in the next section.

4.6 Measurement Process

In this section, we describe the process in which the measurements are performed. We describe its steps on different levels of detail: the components and steps which are synthetic measurement setup are only described regarding their function so that reproducability is possible. During the steps performed by the Fresco framework, however, we also focus on the behavior and establish a first understanding of how Fresco is working on the network level. We use this basic information later to describe some effects which the measurements uncover (cf. Section 4.7).

System Setup Phase

The setup phase is only executed at the beginning of the tests and each time a restart of the hardware setup is necessary. Via PXE, the test hosts start using an image provided by the controller. This image contains the base system as described in Section 4.5.1. Afterwards Ansible [Ans18] is used to install the necessary software, including the aforementioned measurement software and the Java Runtime Environment in order to execute Fresco. Lastly, the repository holding the source code of the test application is copied to each test host.

Orchestration Phase

The baltinet orchestration script is executed on the controller (see Figure 4.1) of the hardware setup. In the beginning, it establishes SSH connections to all selected test hosts. After this initialization phase, it computes all parameter combinations to be tested and iterates over the set of parameter permutations. For each permutation, the following tasks are performed: 1) Hardware parameters are set. This includes the configuration for the network interfaces, especially setting the transmission rate and network latency, and for the CPU, setting the number of cores to be used and their frequency. 2) Then the state of the host is cleaned, killing all processes possibly dangling from a previous permutation. This includes all Java, tshark (and dumpcap), btrace and perf instances. 3) Furthermore, a cleanup of former test result files is performed. 4) Then, the software under test and the profiling tools are started. In the beginning, tshark is started using a wrapper script. It then listens for packets on the utilized port 9000. Perf stat is started wrapping the actual Java test application. Btrace is wrapped by a script which first polls for the PID until the Java application is started and its PID is found, then the PID is handed to btrace upon its invocation.

The profiling tools generate and save result files. After the current round is finished, all result files are uploaded to the controller and saved in a folder dedicated for the current measurement. The instances of the measurement tools are then terminated.

Then, the next iteration begins.

Fresco Phases

In this and the following paragraph, we focus on the phases which occur at a deeper level, inside the Fresco framework when the Java test application is started.

```
# perf stat java -XX:+PrintCompilation
    -cp .:../fresco-0.2-SNAPSHOT-jar-with-dependencies.jar AverageApplication -i2
    -p1:localhost:9001 -p2:localhost:9002 -p3:localhost:9003
    -s bgw -Dbgw.threshold=1 -l OFF -ic 1000 -pl 1
```

Listing 4.5: Starting the Fresco Application

The execution of a Fresco application performed by the command in Listing 4.5 consists of three phases which are currently of interest. The first phase contains the connection establishment between the participating nodes. Its behavior is of vital importance for the explanation of some of the following test results.

Setup Phase The invocation of `sce.runApplication()` internally calls `this.setup()` which performs the connection setup. The aim is to establish a channel in both directions between every pair of all participating hosts. Each host is informed about the set of prospective participants during application startup via command line parameters.

First, the application itself listens for incoming connections. In parallel, it performs own connection attempts driven by busy waiting: it issues an overall amount of around 2000 TCP SYN packets per second[6]. Due to our host orchestration, this phase exhibits a specific pattern: the Java applications on the test hosts are started sequentially with a certain delay between each host. Hence, the first application starts to poll for all other hosts which are not yet listening for incoming connections. When a connection is attempted to a peer which is not yet ready, this peers answers with a `TCP Reset (RST)` packet. Hence, a multitude of connections attempts to the beginning of each measurement noticeably increase the overall count of packets before the computation has even been started. When the second application comes up, it immediately connects to the first host due to one of its connection attempts. From this point in time, both hosts poll in order to connect to all other hosts. Note that this results in the first host issuing a huge amount of SYN packets depending on the duration until the application on the last host is started. Complementarily, the last host is only able to issue a comparatively small amount of SYN attempts before all other hosts connected to it.

In the following, we refer to this phase as polling phase.

As soon as the nodes build a clique, each being connected to every other, the actual computation starts.

Computation Phase During the computation, the defined application specific protocol is executed on each host. This causes a continuous flow of TCP packets between all hosts over the established connections. That means no new SYN and SYN/ACK packets are sent. The amount of packets mainly depends on the number of input elements and the number of participating peers.

Tear Down Phase When the computation has finished, the connection is torn down. This is initiated by each host sending a RST packet via the connections established from other peers to *their* listening ports and a FIN packet over the connections they established themselves to *other* peers' listening ports.

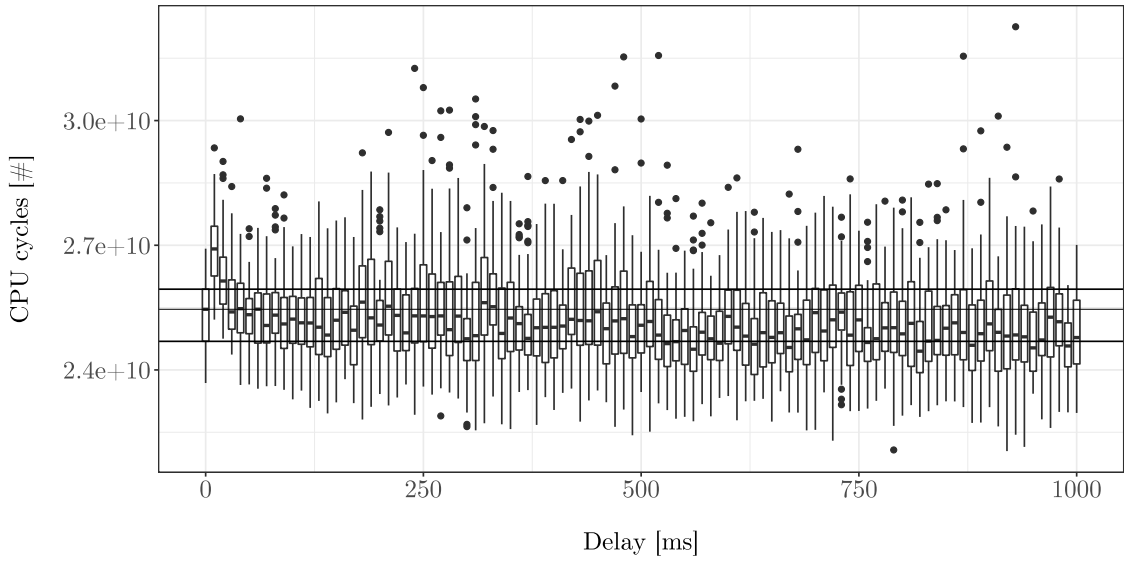

Figure 4.2: The impact of polled memory profiling on number of CPU cycles consumed by the measured process

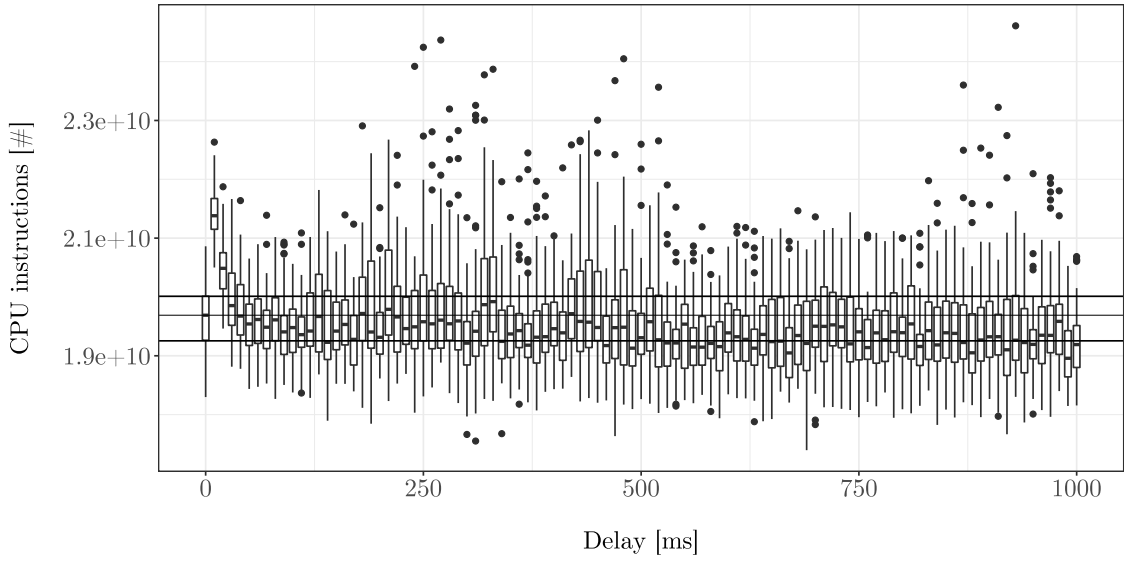

Figure 4.3: The impact of polled memory profiling on number of instructions performed by the measured process

Assessment of the influence of profiling on performance

Polling memory data from a running process introduces a performance overhead as the code under test is extended by the instrumentation code which is also executed. Hence, the question becomes important how often memory stats may be polled without introducing an overhead which considerably influences other measurement results.

In order to answer this question we perform a pre-measurement. As parameter, we vary the delay between retrieving memory stats by btrace between 10 and 1000 milliseconds. As variable of interest, we measure the number of CPU cycles consumed by the process under test and the corresponding number of instructions performed by the CPU. The variables are measured using perf. Each measurement is done 50 times, each time completing a full iteration from 10 ms to 1000 ms before restarting at 10 ms.

[6] Assuming an unmodified configuration of the CPU and the network interface.

We compare the results to a baseline, where we measured the named variables without instrumenting the process at all. This baseline is denoted as *delay = 0* in the figures 4.2 and 4.3.

The results show that the measurements are only influenced considerably when memory polling happens every 10 to 20 milliseconds. Greater intervals already show a distribution of cycles and instructions similar to uninstrumented code.

In conclusion, a lower limit of the memory polling delay to avoid non-neglible performance overhead is 30 ms. In other words, when memory polling is performed less frequently than every 30 ms, CPU measurements and memory measurements can be performed simultaneously, drastically decreasing the amount of measurements and their overall duration. In order to obtain a safety margin, we perform memory polling every 100 milliseconds.

4.7 Results

The evaluation analyzes the influence of the parameters *number of input elements*, *number of nodes*, *number of CPU cores*, *frequency of CPU*, *transmission rate*, *packet loss*, *network latency*, *parallelization of computation*, on the *execution time*, the *memory consumption* (stack & heap), the used *CPU cycles* and *instructions* and the *transferred packets* of the Fresco framework when used for secure computation of the above-mentioned use case.

The measurement points by btrace for the execution time and both memory types in the code are set to only evaluate the method which actually calls Fresco's `#runApplication` method. In other words, the Java VM initialization and the setup code for the tests are not included in the results, and hence do not introduce interferences. The term *execution time* refers to the duration of the execution of this very method.

4.7.1 Number of Points

Parameters	Values
Nodes	3
CPU Cores	8
CPU Frequency	2500 MHz
Parallelization	1
Transmission rate	1 Gbit
Added network latency	0 ms
Packet loss	0 %
Number of input elements	1–5000
Number of repetitions	50

As the first measurement, we took the number of input elements as parameter and investigated the scaling behavior of Fresco. This shows the order of magnitude in which a SMC session is carried out and provides a baseline against which the following tests can be compared.

Execution Time

Figure 4.4 shows a linear relation between the number of points and the execution time. The reason is that all computations are performed independently from each other and the execution time scales with a constant time factor per computation. Linear regression yields the following approximation for fifteen nodes:

$$0.01292s * |input\ elements| + 15.717s = execution\ time$$

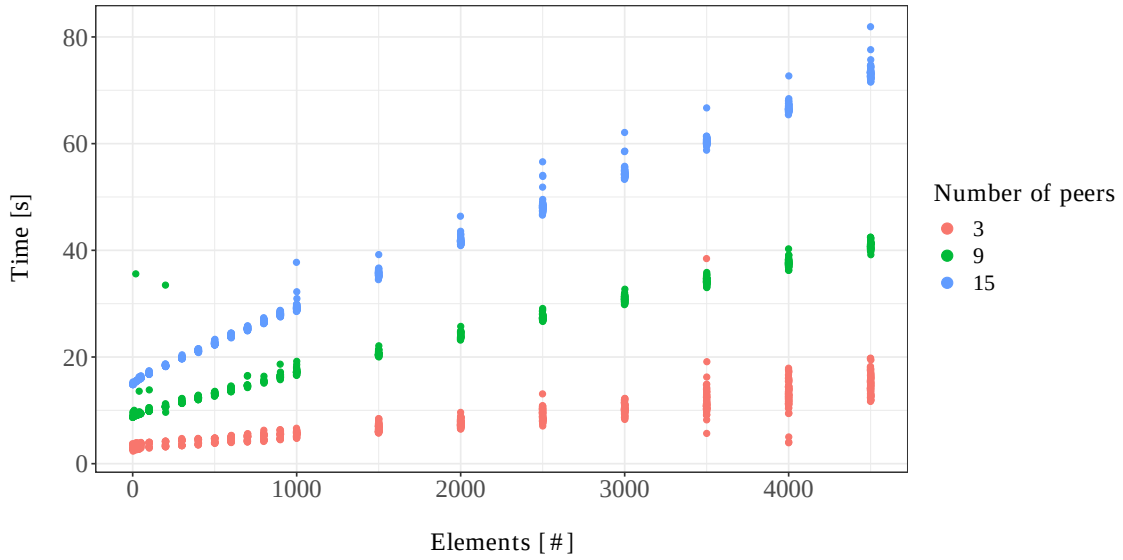

Figure 4.4: The impact of the number of input elements on the execution time

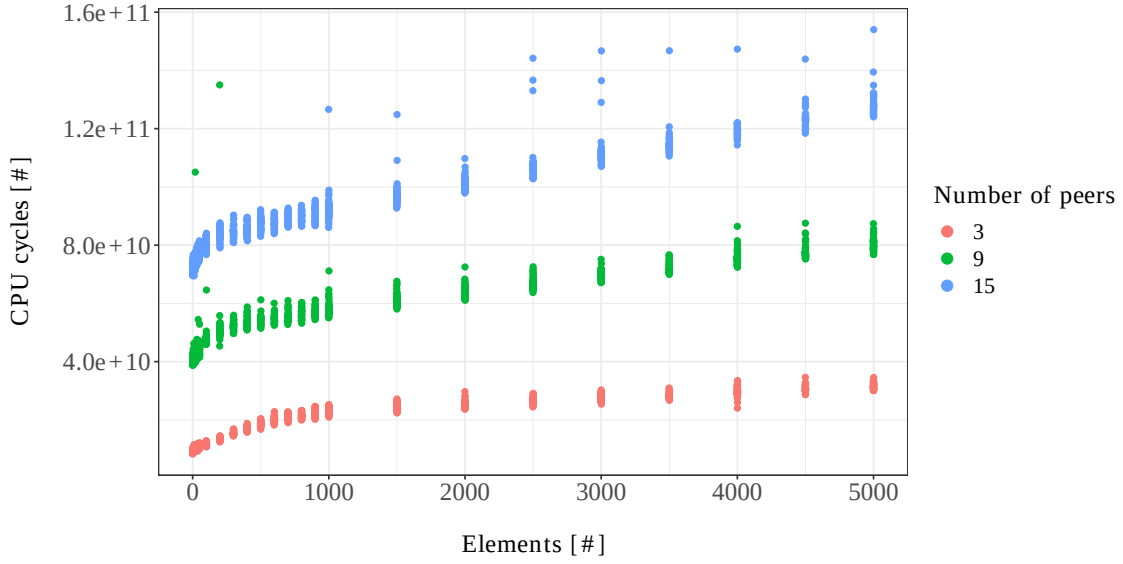

Figure 4.5: The impact of the number of input elements on the number of consumed CPU cycles

CPU Utilization

The overall development of the number of CPU cycles[7] is mainly linear with respect to an increasing number of points (Figure 4.5). At the beginning of each measurement series the increase is higher than in later measurements. We assume that the JIT (just-in-time) compiler of Java which compiles often-used byte code to machine code, starts to optimize essential parts of the application between an input length of 100 and 300 elements. This explains why the curve becomes less steep in this area. The fact that the steepness of the curve with three nodes changes later also supports this theory. Here, generally less code is executed and, in consequence, the JIT compiler reacts later.

The correlated change in the previous time diagram is, however, very small and hardly visible.

[7]We also measured the CPU instructions. We will only elaborate on them if they provide additionl insights. In most cases instructions and cycles exhibited the same behavior.

Note that the effect of the setup phase, is not removed from these diagrams. We elaborate on this effect in Section 4.7.2 on page 53 and also make its intensity visible there.

Memory Allocation

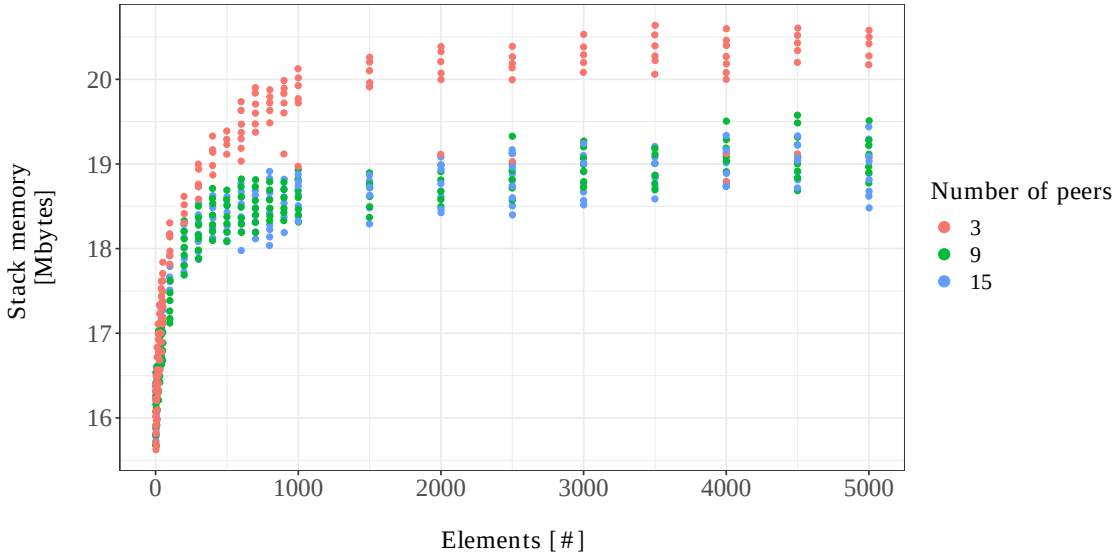

Figure 4.6: The impact of the number of input elements on the maximum allocated stack memory

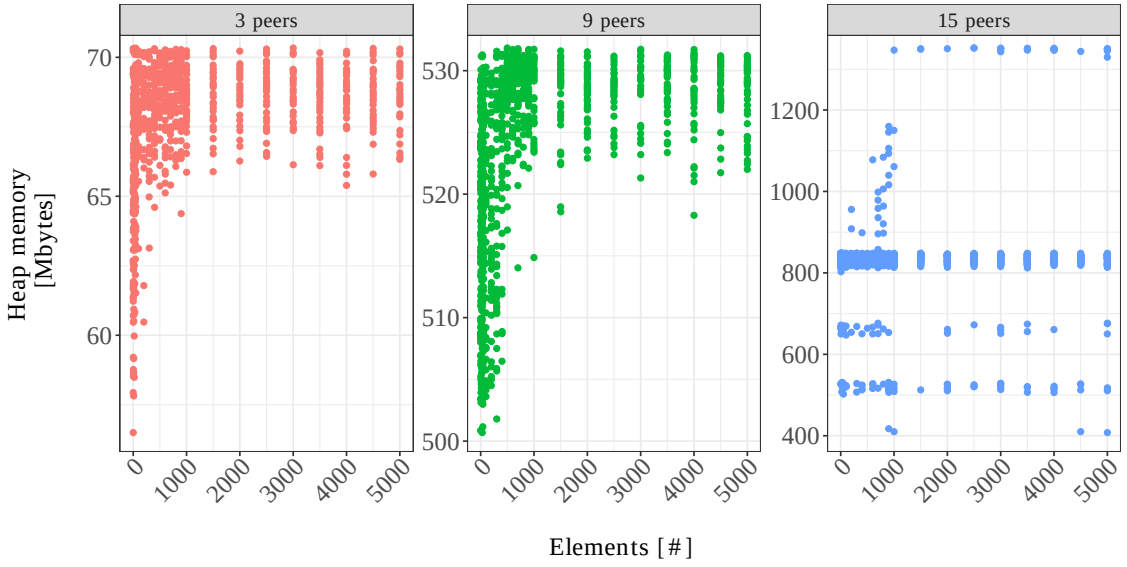

Figure 4.7: The impact of the number of input elements on the maximum allocated heap memory

In Figure 4.6 we see that stack memory allocation starts around 16 Megabytes. It increases slightly upon increasing the number of points. We expect that memory allocation stops increasing at a certain point beyond 5000 input elements. The reason is that, as long as no significant amount of memory is requested by the application in a short time frame, the garbage collector keeps an implicit, static limit of the allocated memory. Every time the application requests another memory allocation which would surpass this given limit, the garbage collector runs and resets the allocated memory back to the currently possible minimum.

During our measurements, we noticed that the stack memory was never strongly influenced by any examined parameter. We will hence omit the stack memory from now on.

Figure 4.7 shows the heap space allocation during the measurement.[8] With fewer points, the allocated memory exhibits a greater variability which is gradually reduced during input element increase. However, in the measured interval, we see that the allocated memory has a common maximum over the whole span. Its actual value depends on the number of participating nodes. This behavior regarding the heap memory shows that there is no continued creation of long term objects per computation which is retained over the duration of each single computation itself. In other words, the amount of needed heap space converges irrespective of the amount of performed computations. The test cases with 15 nodes show discrete levels of allocated heap memory (around 836 Mbytes, 662 Mbytes, 520 Mbytes). This is caused by the behavior of Java's garbage collection and described in greater detail in Section 4.7.2 and Figure 4.13.

As for the stack memory, we will only elaborate on the heap memory from now on if it deviates from the established baseline of around 70 Mbytes.

Transmitted Packets

Similar to the amount of elapsed execution time, the number of packets to be transmitted scales linearly with the number of input elements (see Figure 4.8). The reason is the same as stated above.

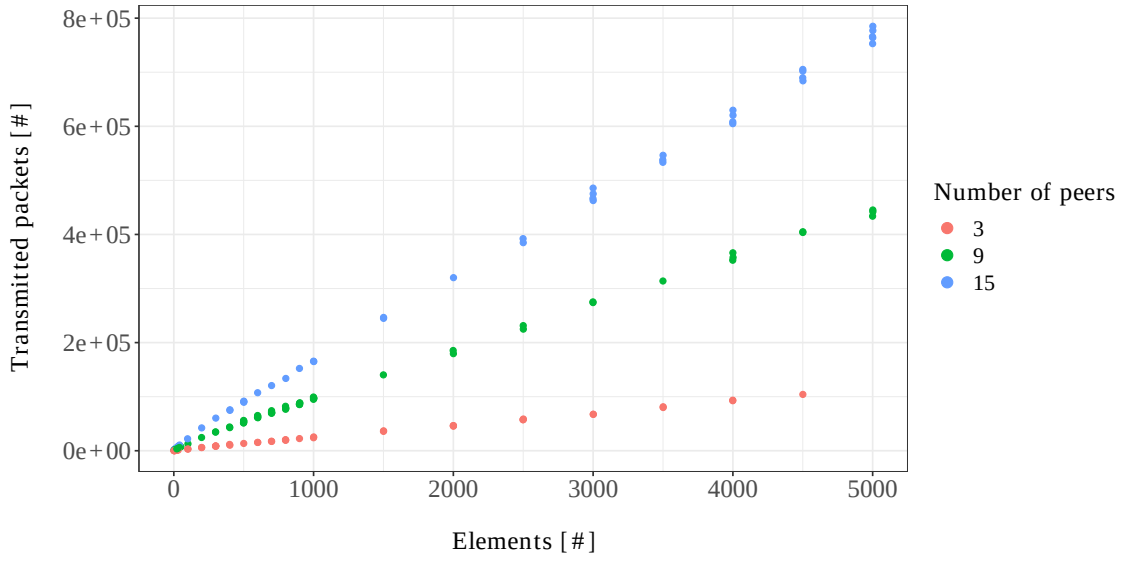

Figure 4.8: The impact of the number of input elements on the amount of transmitted packets

We investigated the impact of the setup phase on the packet transmissions in detail. Figure 4.9 shows the transmitted packets during a characteristic session. Packets carrying the SYN or RST flag are specially colored in order to make the differences between the setup phase and the computation phase visible. The setup phase is split into two sections. Both sections starts with a gradually increasing amount of SYN packets until about 2000 SYN packets per second are issued. This level is kept for about a second, then the transmission stops abruptly.

The behavior can be explained as follows: upon invocation, the Java application receives the list of the computation peers (cf. Listing 4.5). The first peer tries to *only* establish

[8] In order to retain details in the case of three nodes, the diagram has been split and the y-axes have been individually adjusted.

a connection to the next in the given list. This yields the first increase of SYN packets. Given, the peer is found—otherwise the application would fail at this point—a connection is established. The diagram reflects this by the abrupt drop of syn packets per second and the very small amount of non-SYN, non-RST packets transmitted which constitute the TCP handshake. Afterwards, both the first and the second peer try to establish a connection to the third nodes. This causes the second, similarly shaped section.

The rest of the transmission is the actual communication necessary for the computation.

The influence of the setup phase upon the number of transmitted packets is quite strong. The reason is inefficient busy waiting, which can, in a productive environment, be replaced by a more efficient solution. As the SYN and RST flagged packets only occur during the setup phase and additionally a single RST packet per node pair during the tear down phase, we can focus on the packets constituting the actual computation by excluding these packets in any of the following considerations and diagrams. So, in the following, the SYN and RST packets are filtered out beforehand.

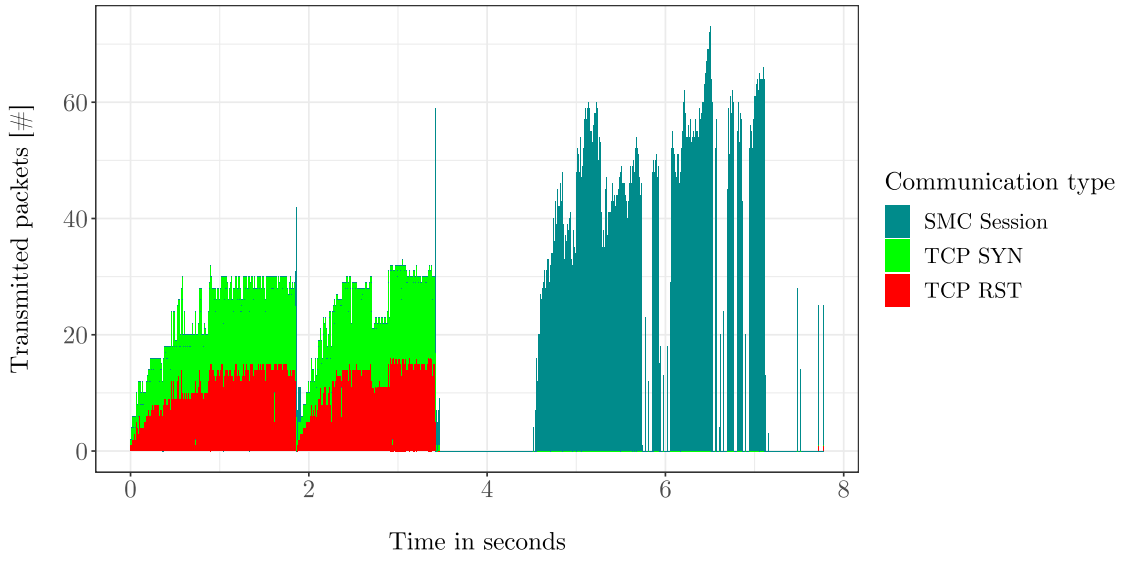

Figure 4.9: Amount of packets transferred over the time of a SMC session with 1000 input points. Vantage point is the second peer in a set of three.

4.7.2 Number of Peers

In this section, we consider one of the most important parameters: the number of participating nodes reflects the question how the overall system is able to scale in a productive environment. In our setup, we where able to examine the interval of 3 to 15 nodes. This yielded results that already show a definitive trend and allow extrapolation for greater numbers of collaborating peers. The modeled setting is that of an intranet.

The utilization of more nodes made it necessary to provide more input data to be used for the computation. The data itself does in no way influence the performance of the system. Hence, without loss of application and closeness to reality, we took our data from the original 5 donators and duplicated the inputs until every used node had an own list of GPS input tuples.

Parameters	Values
Nodes	3–15
CPU Cores	8
CPU Frequency	2500 MHz
Parallelization	1
Transmission rate	1 Gbit
Added network latency	0 ms
Packet loss	0 %
Number of input elements	1000
Number of repetitions	50

Execution Time

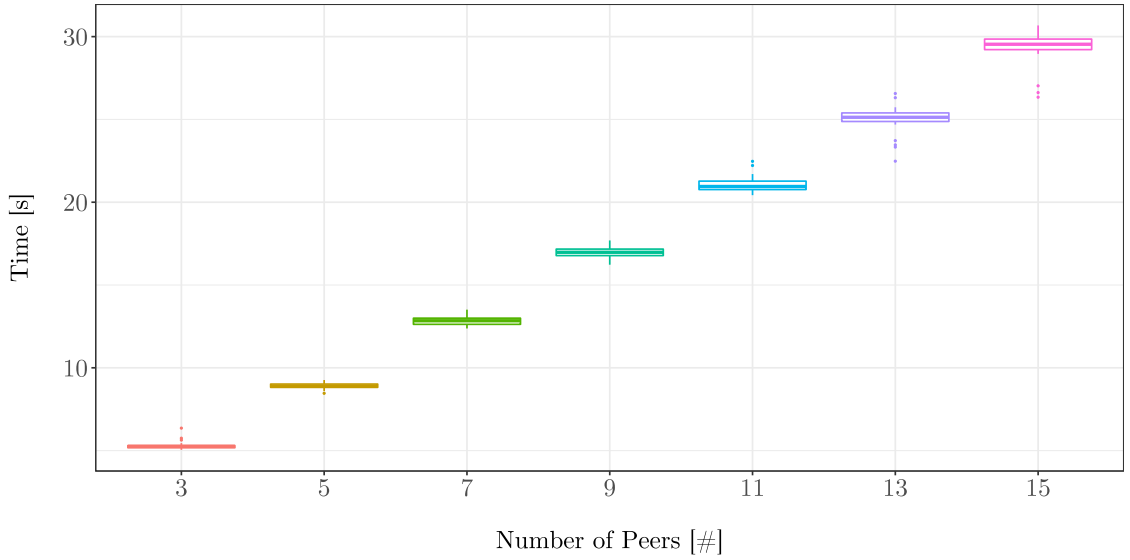

Figure 4.10: The impact of the number of peers on the execution time

Figure 4.10 shows that there is a linear relation between the number of nodes and the execution time. At first glance, this might surprise, as the utilized BGW protocol exchanges $O(n^2)$ messages upon invocation with n nodes.

In our case, the concrete protocol consists of an input, computation and finally an output phase. All of these phases require communication between every pair of hosts. However, the communication of each phase can be performed simultaneously and in parallel as there exists no dependency between the communications in the same phase.

Concretely, each node has to perform a constant number k of steps for each of the remaining $|n-1|$ nodes. Hence, $|n| * k * |n-1|$ overall steps have to be performed. Nevertheless, the $k * |n-1|$ steps of each separate node can be executed in parallel, reducing the necessary time by the factor n, and consequently yielding linear time.

The only existing sequential dependency is between the phases itself which requires that the previous phase has been fully completed before the next can start.

CPU Utilization

In correlation with the linear increase of logical steps, the number of CPU cycles also scales linearly with the number of peers (Figure 4.11). This supports our interpretation from the previous section.

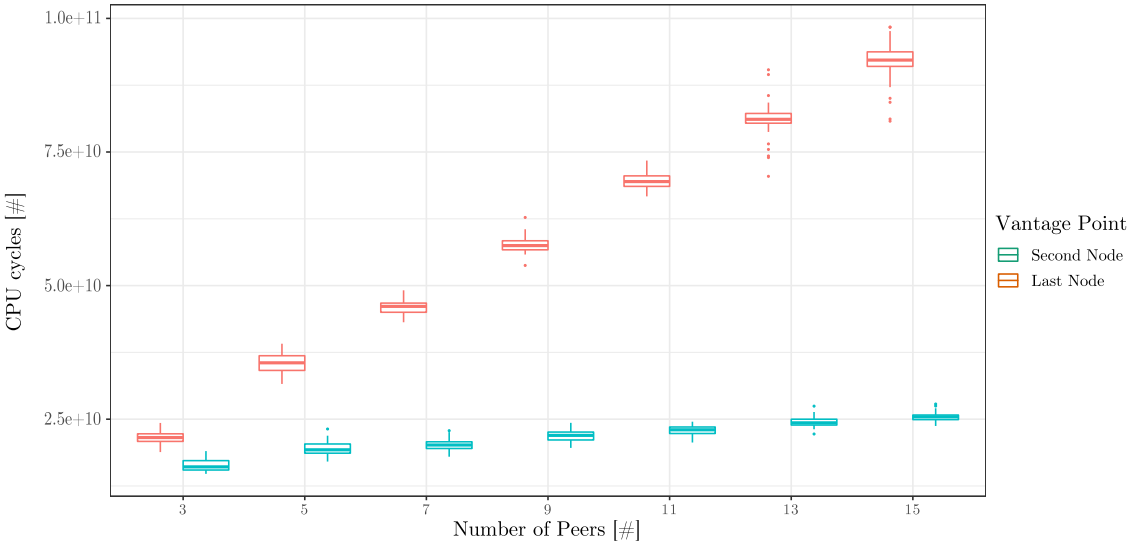

Figure 4.11: The impact of the number of peers on the number of consumed CPU cycles

However, there is a confounding variable which has to be investigated: as described earlier (Section 4.6), before the actual computation, each host performs polling of all other hosts in order to establish mutual connections between all participants. Each host here performs busy waiting, issuing a vast amount of connection attempts per second. This does not only affect the amount of transmitted packets, but also the consumed CPU cycles.

We differentiated between the CPU consumed during the polling phase and the actual computation phase by employing the following method: the test orchestration software starts the Fresco framework sequentially on all participating hosts. This means the first host has to poll the longest time before the actual computation starts while the last hosts has to perform very little to no polling. We exploited this fact by comparing the CPU usage of the first real participant (besides the role of the organising gateway being peer 1) and the last participant in the computation.

The evident difference is shown in the previously mentioned diagrams. The upper linear development depicts the CPU usage including the polling phase at the beginning. In the lower development only the CPU used for the computation alone is shown.

The measurement results show that the polling phase at the beginning contributes essentially to the amount of consumed CPU. In contrast, the increased CPU usage caused by the computation itself is relatively low.

Memory Allocation

Figure 4.12 shows the memory allocation behavior when increasing the number of participating peers.

Plateaus become visible which discretely heighten when the number of peers is increased. We investigated this behavior in greater detail and found out that it is also attributable to the garbage collector of the Java VM. Figure 4.13 shows the memory allocation over time during three exemplary rounds. There exist two individual processes which influence the depicted curve: memory allocation is performed by the provided SMC code which exhibits the same pattern in all three cases: in the first 15 seconds, allocation happens relatively fast, indicated by the steep jags in the beginning. After that time, further allocation is performed comparatively slower. The second process is the garbage collector which repeatedly removes unreferenced objects to regain allocatable memory. Here, differences between the three cases occur: from run 26 to 28, the garbage collector runs zero, one

or more times during the phase of slower memory allocation. This highly influences the maximum peak of the used amount of memory. In other words, the memory allocation does not behave differently depending on the number of peers, but the JVM handles the execution in different ways.

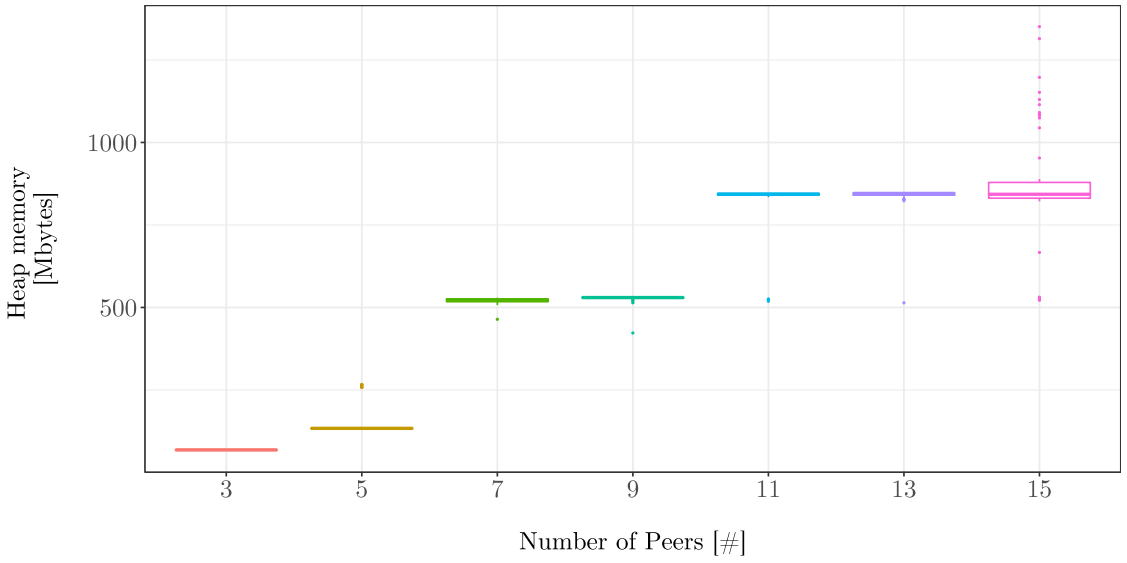

Figure 4.12: The impact of the number of peers on the maximum allocated heap memory

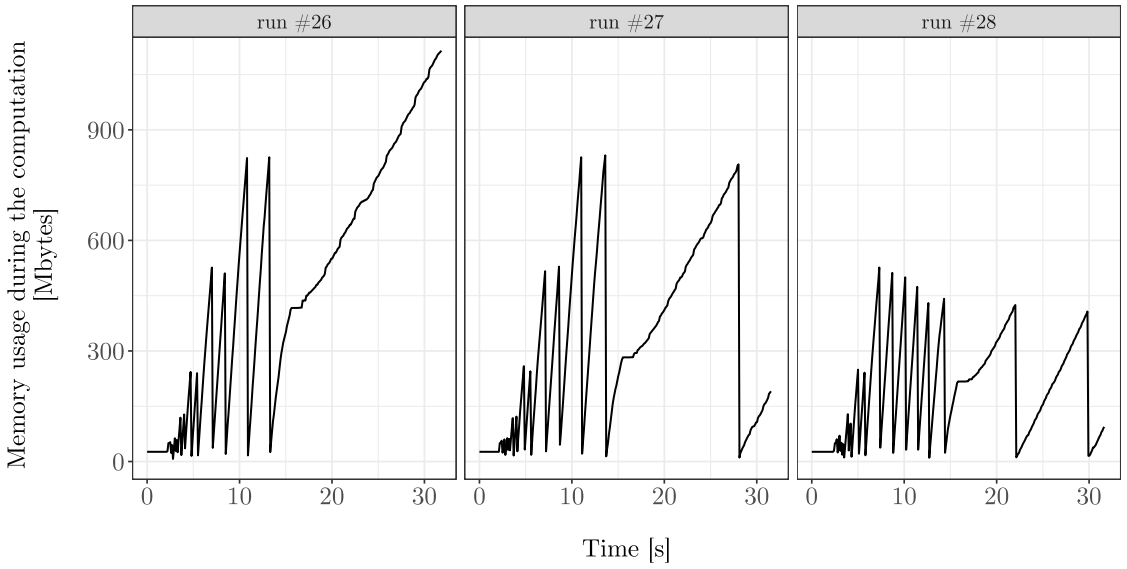

Figure 4.13: The behavior of the garbage collector regarding heap memory during a computation (detailed view of run 26, 27, and 28 with 15 nodes)

Transmitted Packets

The overall amount of packets to be sent increases quadratically with the number of participating peers. From the perspective of a single peers, this means that the number of messages sent and received individually scales linearly as depicted in Section 4.7.2.

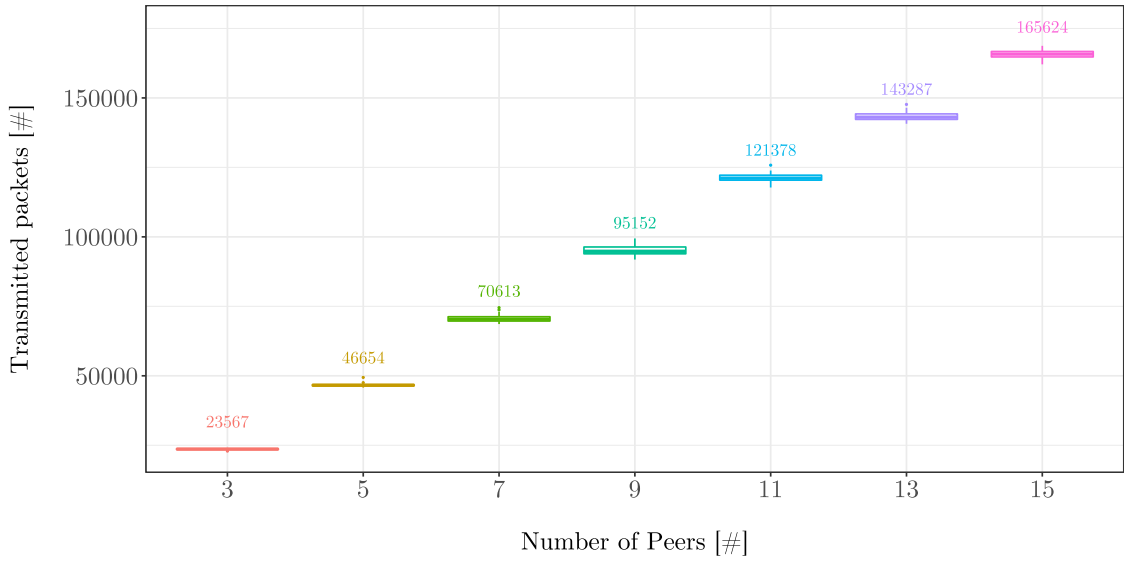

Figure 4.14: The impact of the number of peers on the amount of transmitted packets

4.7.3 Cores and CPU Frequency

Parameters	Values
Nodes	3
CPU Cores	1, 2, 4, 8
CPU Frequency	1600, 1800, 2000, 2200, 2500 MHz
Parallelization	1
Transmission rate	1 Gbit
Added network latency	0 ms
Packet loss	0 %
Number of input elements	1000
Number of repetitions	50

In order to determine the necessary resources single nodes have to provide for realizing a satisfying overall performance, we measured how the number of cores and their frequency of every node influences the performance of an SMC session.

Both parameters where manipulated via the */sys* interface (cf. Listing 4.6).

```
1   cores=...
2   freq=...
3   for i in $(seq 1 7); do
4       echo 1 > /sys/devices/system/cpu/cpu$i/online
5   done
6   for i in $(seq $cores 7); do
7       echo 0 > /sys/devices/system/cpu/cpu$i/online
8   done
9
10  # set the frequency of the online cores
11  for i in $(seq 0 $((cores-1))); do
12      echo ${freq}000 > /sys/devices/system/cpu/cpu$i/cpufreq/scaling_min_freq
13      echo ${freq}000 > /sys/devices/system/cpu/cpu$i/cpufreq/scaling_max_freq
14  done
```

Listing 4.6: Setting up artificial CPU core and frequency restrictions using the /sys interface

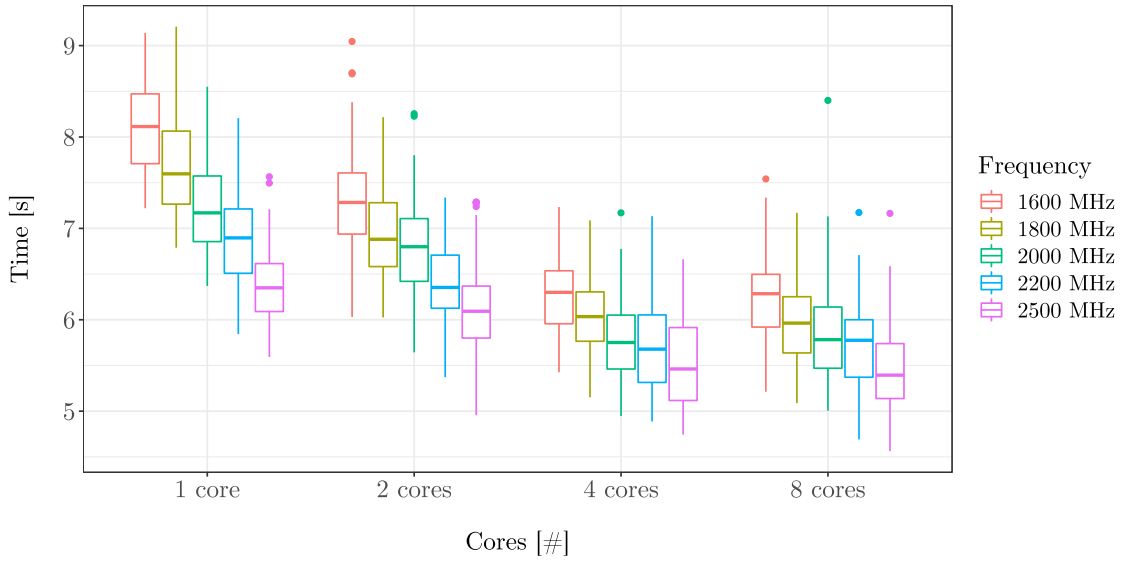

Figure 4.15: The impact of the number of CPU cores and their frequency on the execution time

Execution Time

Figure 4.15 shows that additional cores reduce the execution time. This effect is notable when comparing a single against two cores, as well as two against four cores. However, the improvement of eight cores against four cores is minimal.

This can be explained as the amount of parallelizable computations is limited in the given setting. Basic parallelization of computations is possible in lines 12 to 14 of Listing 4.3, but the computations for each input are performed in a completely sequential manner. Hence, further parallelization is impossible. Due to this reason, no improvements from four to eight cores can be achieved. In Section 4.7.7, we investigate how the potential of further parallelization can be leveraged.

CPU Utilization

We consider Figure 4.5 as reference point. We see that for 1000 input elements and 3 nodes having 8 cores with 2500 MHz, around $2.5 * 10^{10}$ cycles are consumed. This corresponds to the last column of boxes in Figure 4.16. From there, decreasing the CPU frequency or the number of cores effectively reduces the amount of CPU cycles (Figure 4.16). Here, the influence of the frequency is considerably smaller than the influence of the cores.

In order to explain the effect we depict not only the measurement data of our typical measurement host, but of all peers involved in the computation. The diagrams show that the selection of the host influences the number of consumed cycles to a greater extent than the CPU frequency and the number of cores. The only logical difference between the nodes is the order in which they are orchestrated and, in consequence, the length of the individual setup phase. Later nodes (higher ID) perform a significantly shorter polling phase and correspondingly less connection attempts. This results in a notably lower amount of consumed CPU cycles. We consider the whole influence of the number of cores and the CPU frequency on the CPU cycles to be an artifact of the setup phase. With the current measurement setup, it is unfortunately not possible to exclude the setup phase from the CPU measurements in order to eliminate this effect completely.

Memory Allocation

There is no notable influence of the number of cores and the CPU frequency on memory consumption.

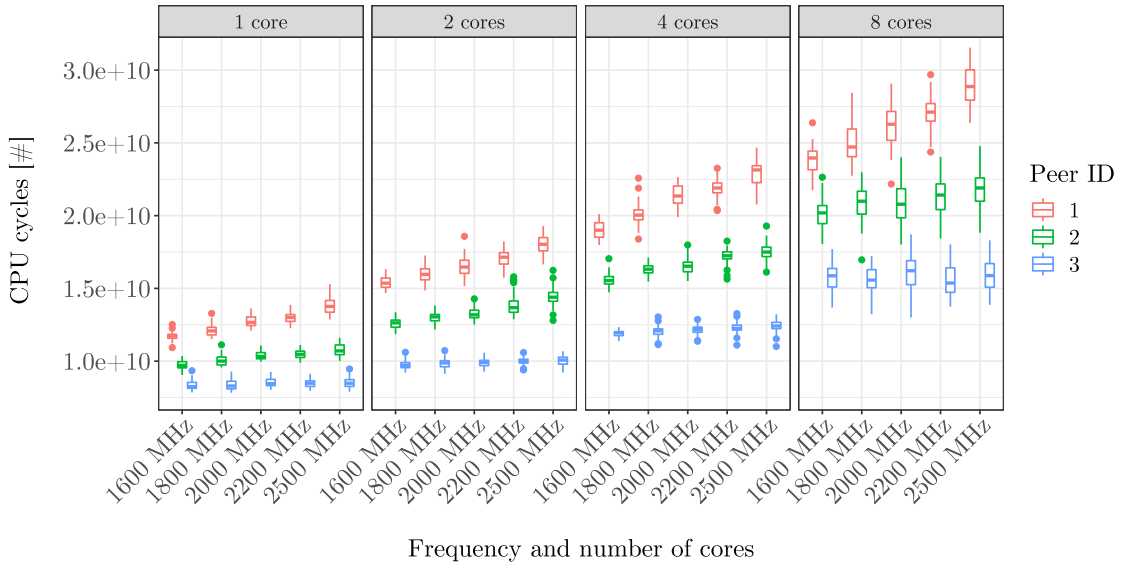

Figure 4.16: The impact of the number of CPU cores and their frequency on the number of consumed CPU cycles

Transmitted Packets

Similarly, no influence on the number of transmitted packets could be identified.

4.7.4 Transmission Rate

Parameters	Values
Nodes	3
CPU Cores	8
CPU Frequency	2500 MHz
Parallelization	1
Transmission rate	1 Mbit, 10 Mbit, 100 Mbit, 1 Gbit
Added network latency	0 ms
Packet loss	0 %
Number of input elements	1000
Number of repetitions	50

In order to assess the performance of networks with different speed, we analyzed the impact of the transmission rate on the SMC computations. Initially, the network has 1 Gbit links between all test nodes. We varied the transmission rate of these links between 1 Mbit to the original 1 Gbit using the steps of typical network speeds of 1 Mbit, 10 Mbit, 100 Mbit and 1 Gbit.

```
# rate=...
# lat=...
# loss=...
# interface=...
# tc qdisc del dev $interface root;
# tc qdisc add dev $interface root handle 1: htb default 1;
# tc class add dev $interface parent 1: classid 0:1 htb rate ${rate}kbit;
# tc qdisc add dev $interface parent 1:1 handle 10: netem delay ${lat}ms loss ${loss}%;
```

Listing 4.7: Setting up artificial bandwidth and network latency restrictions using tc

Execution Time

Figure 4.17 shows that an increase of the transmission rate from 1 Mbit to 10 Mbit considerably reduces the execution time. A further increase yields only slight improvements.

We see that a transmission rate of 1 Mbit makes the connection speed the bottleneck which is already nearly released by a connection of 10 Mbit. As the change from 100 Mbit to 1 Gbit does not yield any further improvement, we assume that the restraining factor shifts somewhere between 10 Mbit and 100 Mbit.

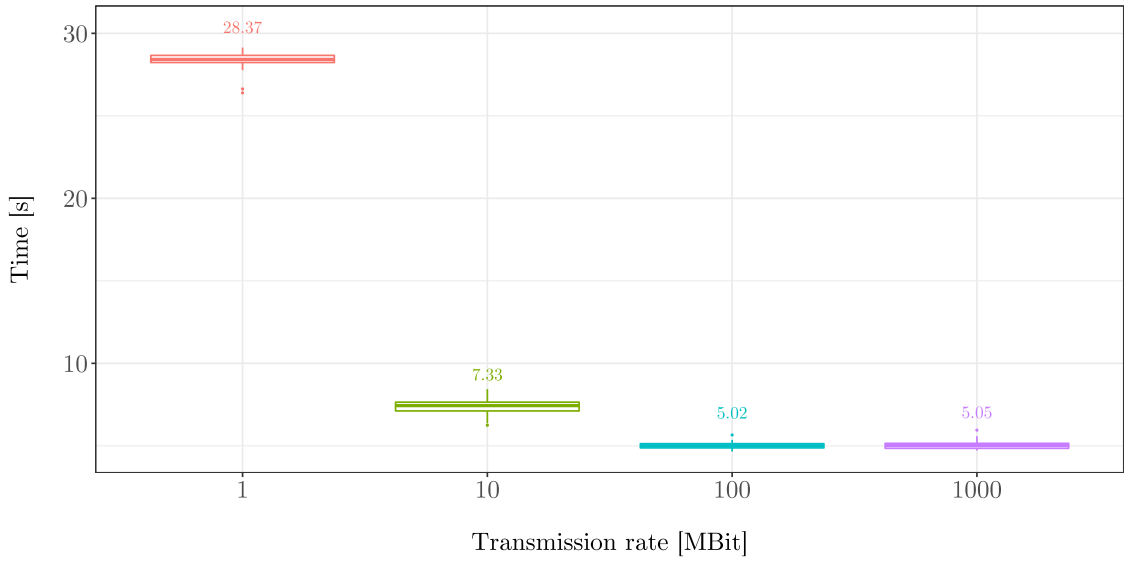

Figure 4.17: The impact of transmission rate on the execution time

CPU Utilization

The CPU consumption (Figure 4.18) with a transmission rate of 1 Mbit is notably lower than the three other cases with 10 Mbit, 100 Mbit and 1 Gbit.

Although this corresponds to the execution time measurements, we doubt that there is a relation between these two results. We assume that network parameters like transmission rate and network latency, which do by no means alter the semantic of the protocol itself and the packets transmitted and have no unavoidable implications for the application layer, do not influence the amount of consumed cycles more than slightly. We could confirm this since the number of CPU instructions behaved likewise.

However, we again see a connection between the setup phase and the behavior of the CPU. A highly restricted transmission rate constrains the amount of packets being sent during the connection establishment. The rate of connection attempts is then bounded by the transmission and not by the CPU itself, issuing the connection attempts. In consequence, less CPU cycles are consumed during the setup phase.

Memory Allocation

There is no notable influence of the transmission rate on memory consumption.

Transmitted Packets

With a transmission rate of 1 Mbit per second, the amount of transmitted packets is significantly lower than in the other three cases (cf. Figure 4.19). These, however, do not considerably differ from each other.

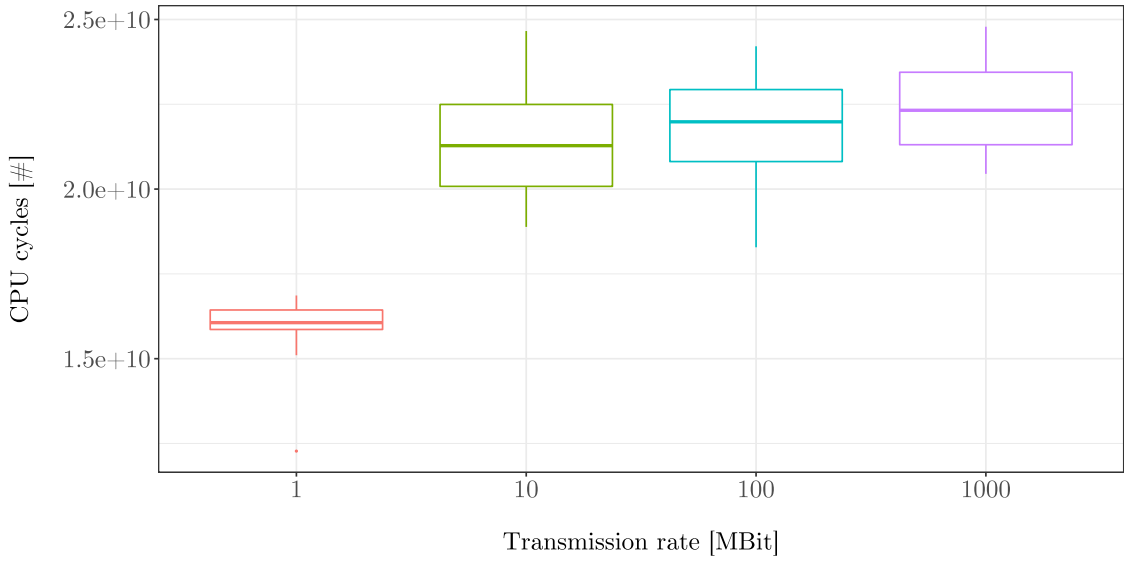

Figure 4.18: The impact of transmission rate on the number of consumed CPU cycles

We can exclude that this difference is an effect of the setup phase as we have excluded SYN and RST packets from the evaluation. In fact, including these packets would even increase the noticed gap.

During the analysis of this effect, we found a correlating phenomenon: in the case of 1 Mbit there is a notably higher amount of larger packets (see Figure 4.20). These constitute about 1.6 Mbytes of the communication which altogether consists of 5.1 Mbytes; all other communications with higher transmission rate transfer around 5.4 Mbytes.

When sending is inhibited, packets ready to be sent stall in the network layer of FRESCO. During the waiting time it is possible that another transmission to the same destination becomes necessary. In this case, both messages to be sent are combined into a single bigger packet. Each saved packet then reduces the overall amount of sent data by its headers which it would have contributed as overhead.

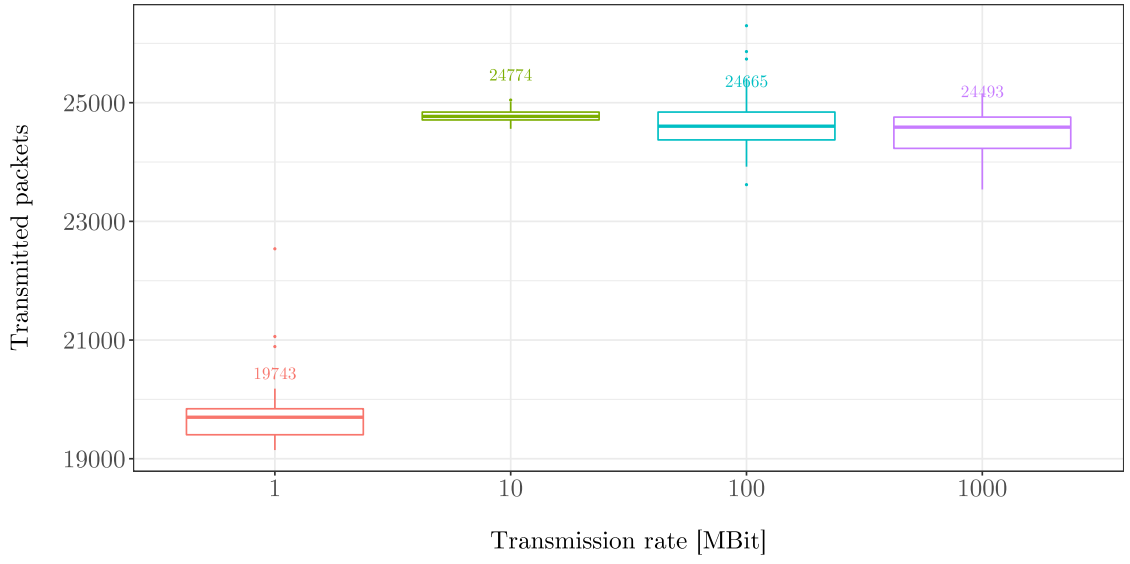

Figure 4.19: The impact of transmission rate on the amount of transmitted packets

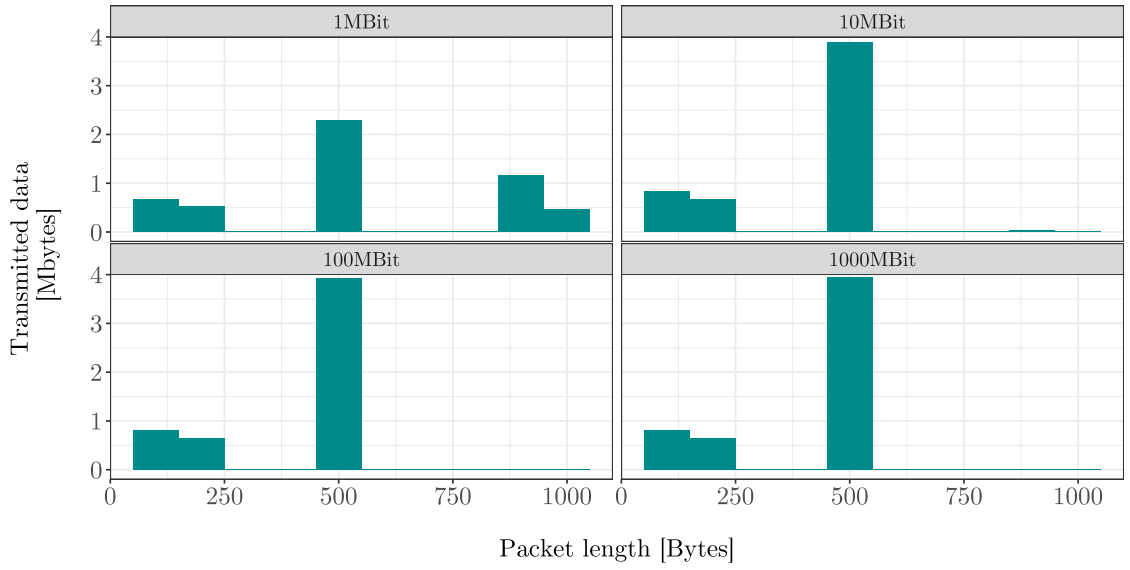

Figure 4.20: The impact of transmission rate on the length distribution of the transmitted packets.

4.7.5 Packet Loss

Parameters	Values
Nodes	3
CPU Cores	8
CPU Frequency	2500 MHz
Parallelization	1
Transmission rate	1 Gbit
Added network latency	0 ms
Packet loss	0–10 %
Number of input elements	1000
Number of repetitions	50

As the next network parameter, we examine how packet loss negatively influences SMC. This analysis is important for the deployment of SMC solutions in wireless networks and its usage via the mobile Internet. The communications is based on TCP, hence, lost packets are recovered by retransmissions. This, however, will result in an increasing amount of transmitted packets and most likely in longer sessions due to a loss of speed.

Starting at a packet loss rate of 6.0 %, we reduce the amount of repetitions due to the increased length of the measurements.

Execution Time

The execution time (Figure 4.21) constitutes a geometric series when packet loss occurs: having a fixed percentage p_{ploss} of packet loss, $p_{ploss} * |packets|_{p_{ploss}==0}$ have to be repeated. From this part, a percentage p_{ploss} has to be repeated again, as it got lost during retransmission. This continues until all packets have been sent.

Effectively, this constitutes the geometric series, which has as analytical equivalent (as $p_{ploss} < 1$):

$$\sum_{k=0}^{\infty} p_{ploss}^{k} = \frac{1}{1 - p_{ploss}}$$

This function increases hyperbolically in the interval $[0, 1[$. Consequently, one would also expect that the execution time exhibits the same characteristics. However, the steep increase only happens very late when p_{ploss} is near 1. The analyzed interval from 0 % to 10 % is rather close to zero where only a linear increase becomes visible.

We could not increase the packet loss further in order to show the hyperbolical increase as the sessions started failing at a packet loss rate of 10 %.

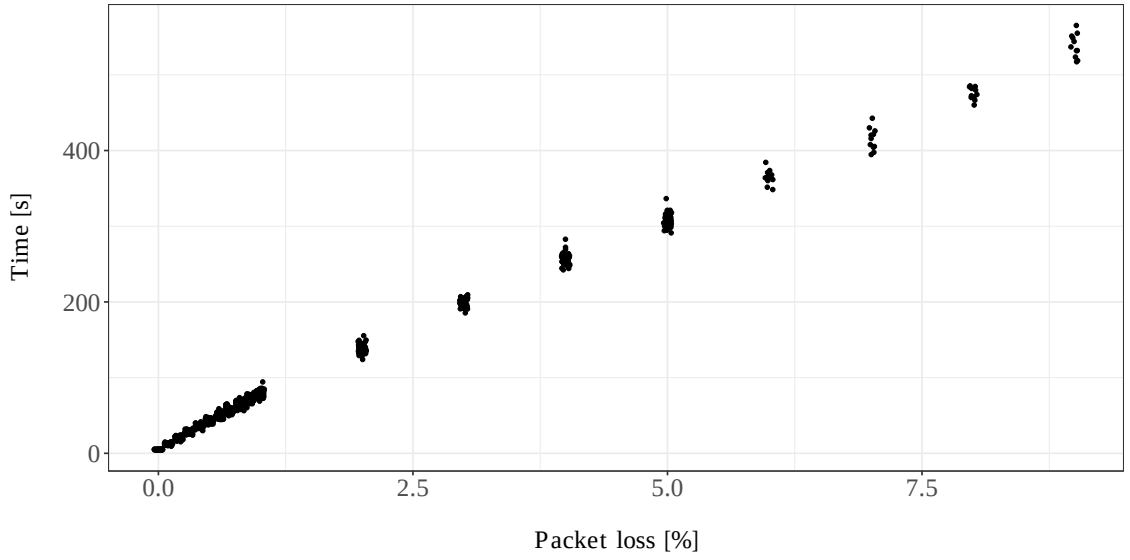

Figure 4.21: The impact of packet loss on the execution time

CPU Utilization

An increasing packet loss rate results in the reduction of CPU cycles (Figure 4.22). The reduction happens hyperbolically with decreasing variance. However, all measurements seem to have a (soft) common lower limit around $1.10 * 10^{10}$.

Our interpretation is in line with our findings regarding transmission rate manipulation. The setup phase is gradually slowed down with increasing packet loss. This results in a smaller amount of connection attempts to be made and hence less CPU usage. Most interestingly, here, the decrease happens gradually in the analyzed interval. In the case of different transmission rates, the decrease occurs between 1 Mbit and 10 Mbit, which becomes only visible as a single jump in our measurements.

Memory Allocation

There is no notable influence of the packet loss on memory consumption.

Transmitted Packets

Figure 4.23 shows that the number of transmitted packets first increases to a packet loss rate of about 5 %. Then, the amount decreases again. This does not meet the first expectations that packet loss should increase the amount of packets necessary to finish the same computation task.

Here two effects come into play which overlay in this measurement. For clarification, we depict the amount of transmitted Bytes in Figure 4.24. In line with the expectations, the amount of Bytes steadily rises with increasing packet loss rate[9]. This is uniquely caused by increased packet loss.

[9] We consider the steep reduction between 1 % and 2 % to be an artifact of the measurement setup. The measurements were not conducted directly afterwards but had to be stopped at this very place and continued at later time.

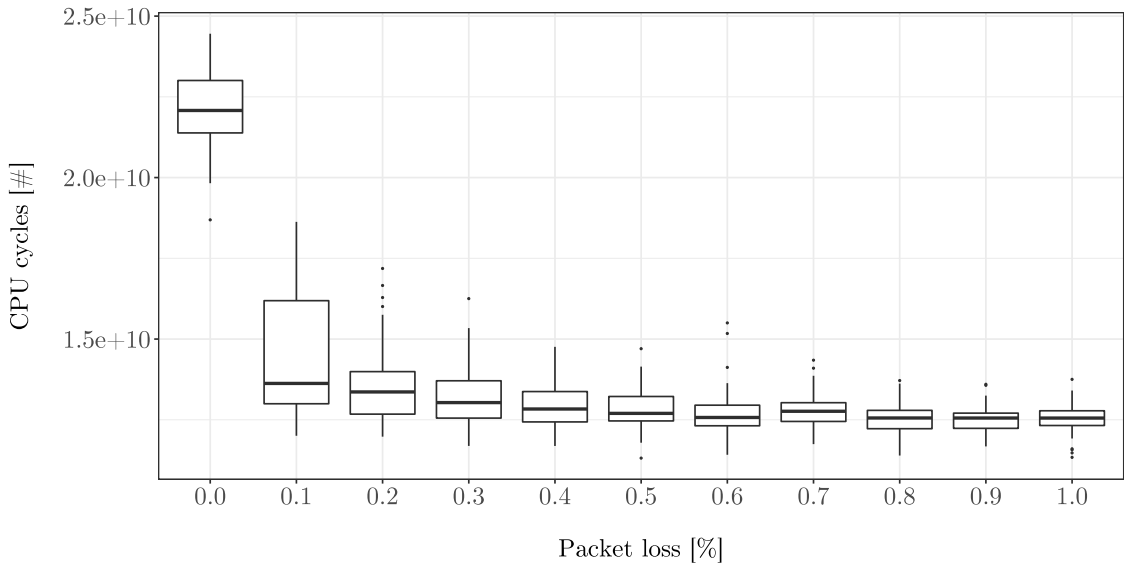

Figure 4.22: The impact of packet loss on the number of consumed CPU cycles

However, we attribute the decrease of transmitted packets to the adaptiveness of Fresco's communication layer as described in Section 4.7.4 on page 59. Encountering hindrances in the exchange of data between the peers, subsequent packets are merged, yielding larger but fewer packets.

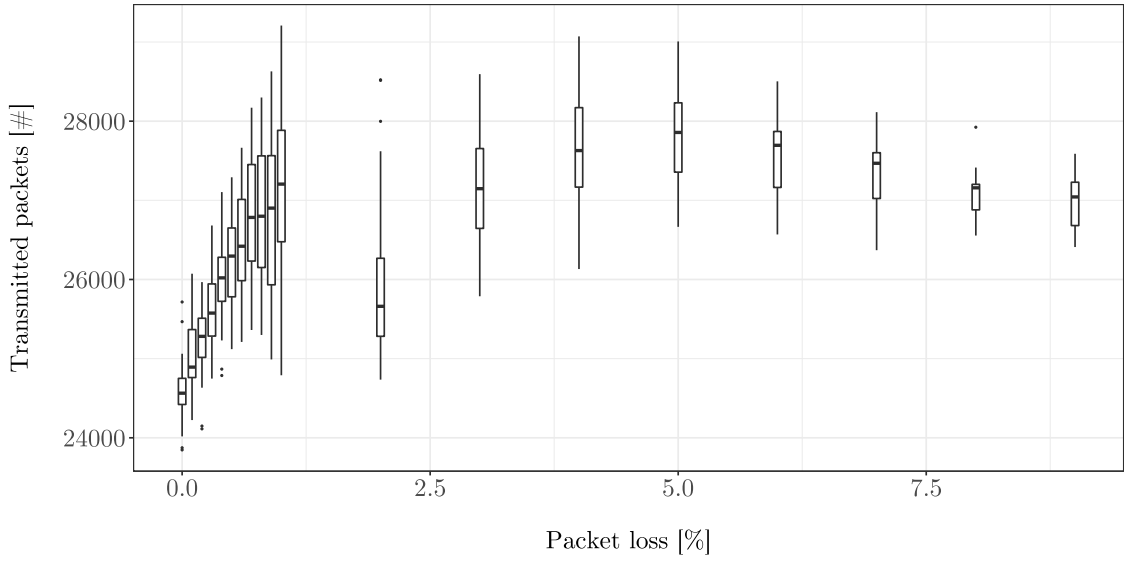

Figure 4.23: The impact of packet loss on the amount of transmitted packets

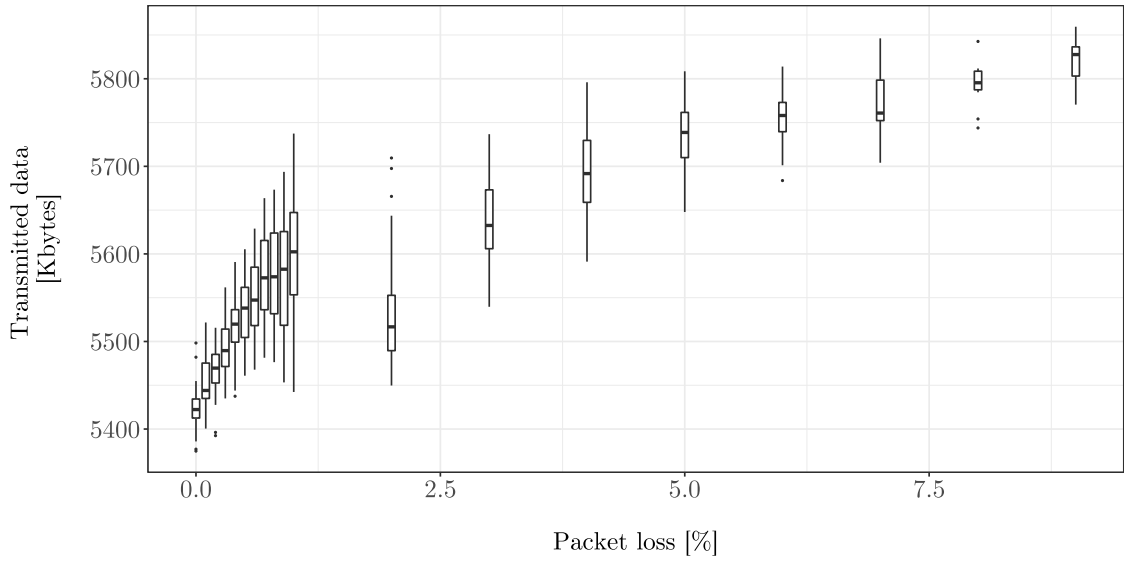

Figure 4.24: The impact of packet loss on the amount of transmitted Kbytes

4.7.6 Network Latency

Parameters	Values
Nodes	3
CPU Cores	1, 8
CPU Frequency	2500 MHz
Parallelization	1
Transmission rate	1 Gbit
Added network latency	0, 16, 50, 200, 500 ms
Packet loss	0 %
Number of input elements	1000
Number of repetitions	50

We performed experiments where we investigated different levels of network latency. The connections between the test nodes have a default round-trip time of ~0.18 ms. With this default configuration we reflect the use case of communication between nodes inside an intranet. We then added an artificial delay of 16 ms, 50 ms, 200 ms and 500 ms to the communication roundtrip, equally distributing the delay to the sending and the receiving network interface using the command in Listing 4.7.

These cases reflect communication between nodes via the Internet or mobile communication.

Execution Time

Figure 4.25 shows a regression line based on the means of the corresponding test results. It becomes visible that the execution time scales linearly with increasing network latency. Execution inside an intranet approximately takes around 4 seconds for the sequential computation of 1000 elements. Using nodes which communicate via the Internet (50 ms to 300 ms), the computation already costs 5 to 25 minutes. In a setting of 8 cores with 2500 MHz, the execution time can be approximated using linear regression, yielding the formula

$$4.63s * \frac{network\ latency}{1ms} + 56.61s = execution\ time$$

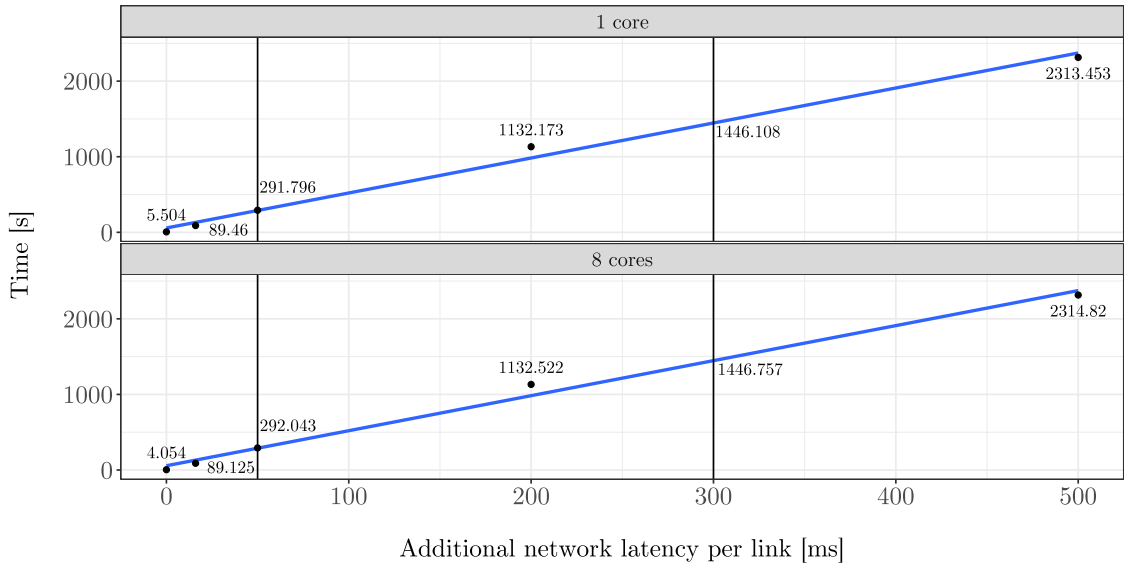

Figure 4.25: The impact of network latency on the execution time

This behavior is expected, as the number of sent messages stays constant during the experiment. Every message is delayed by an additional amount of time d. As a result, the whole communication without any parallelization is delayed $|messages| * d$.

Due to the intentionally constrained network, different amounts of cores do not cause a significant effect.

CPU Utilization

For a single core, the amount of consumed CPU cycles (Figure 4.26) and instructions (Figure 4.27) varies only slightly with increasing network latency.[10] Regarding the cycles, it increases a small amount when the network latency is 500 ms. For eight cores, the behavior is the same, with one exception: in the case when the network latency is zero, the amount of instructions and number of cycles notably surpass the corresponding single-core case and even all other cases with increased network latencies. In other words, added network latency decreases both measured variables compared to the case with no additional network latency.

As a result, two effects have to be explained: firstly, the small increase of CPU cycles with increasing network latency, and secondly the initial decrease in the 8 core case when increasing network latency from 0 to 16 ms.

Regarding the first effect, we point out that the number of instructions does not increase. Hence, only the metric of instructions per cycles degrades. This is reasonable, as with increased latency the CPU has to wait more often for data until the computations upon it can be carried out. Hence, the actual computation process itself is not changed, but its execution is delayed.

The second effect is caused by the setup phase: with no latency, 8 cores allow a very high rate of initial polling, which in turn raises the amount of CPU instructions and the consumed cycles. Adding latency reduces the possibility to poll at this rate, consequently reducing the value of these two variables.

[10]Note: These results have been generated while btrace was deactivated. Despite our considerations in Section 4.6, these measurements would be skewed by the CPU activity of btrace, which is itself also executed by the JVM. The reason is that this measurement is very long-running and without any high computation related CPU activity. In consequence, the btrace activity would outweigh the actual calculation.

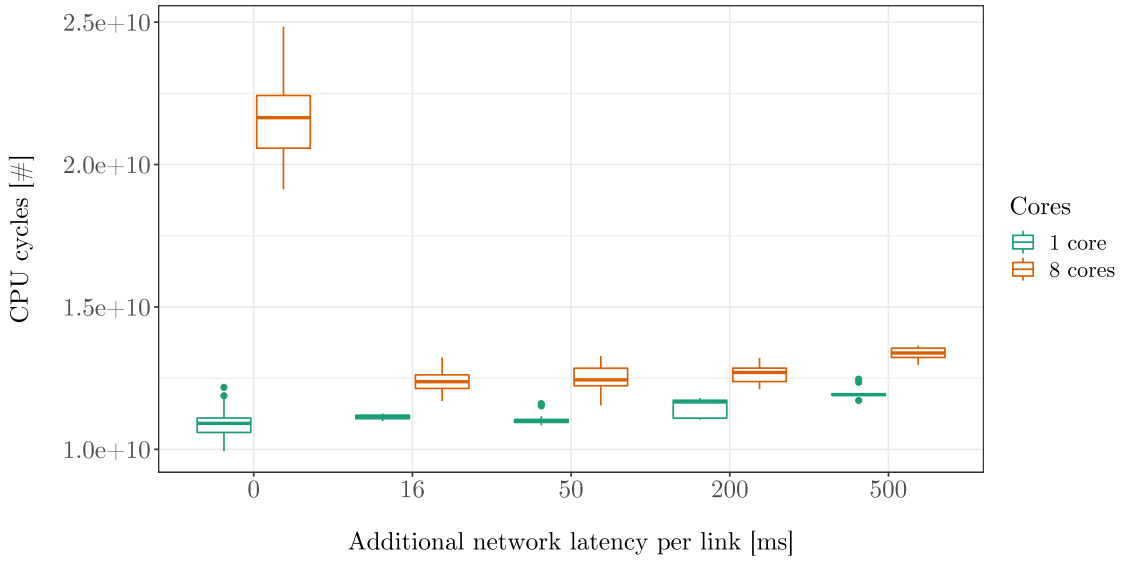

Figure 4.26: The impact of network latency on the number of consumed CPU cycles

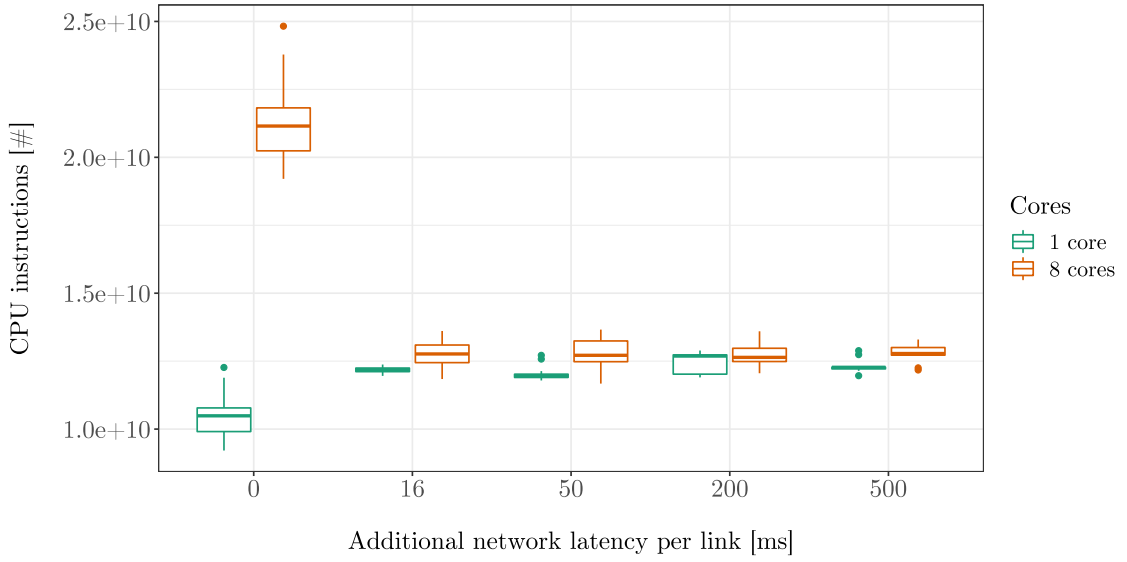

Figure 4.27: The impact of network latency on the number of CPU instructions

Memory Allocation

There is no notable influence of the network latency on memory consumption.

Transmitted Packets

The amount of transmitted packet exhibits notable variations (Figure 4.28): the cases of no additional network latency and an added latency of 50 ms have a similar amount of transmitted packets, the cases of 500 ms, 16 ms and 200 ms each have a lower and individual level. There is no trend which becomes visible in these measurements.

We could verify that this behavior is not limited to the *number of packets*, but also extends to the *overall number of Bytes transferred*. Hence, we attribute this to the merging behavior of the networking layer as described in Section 4.7.4.

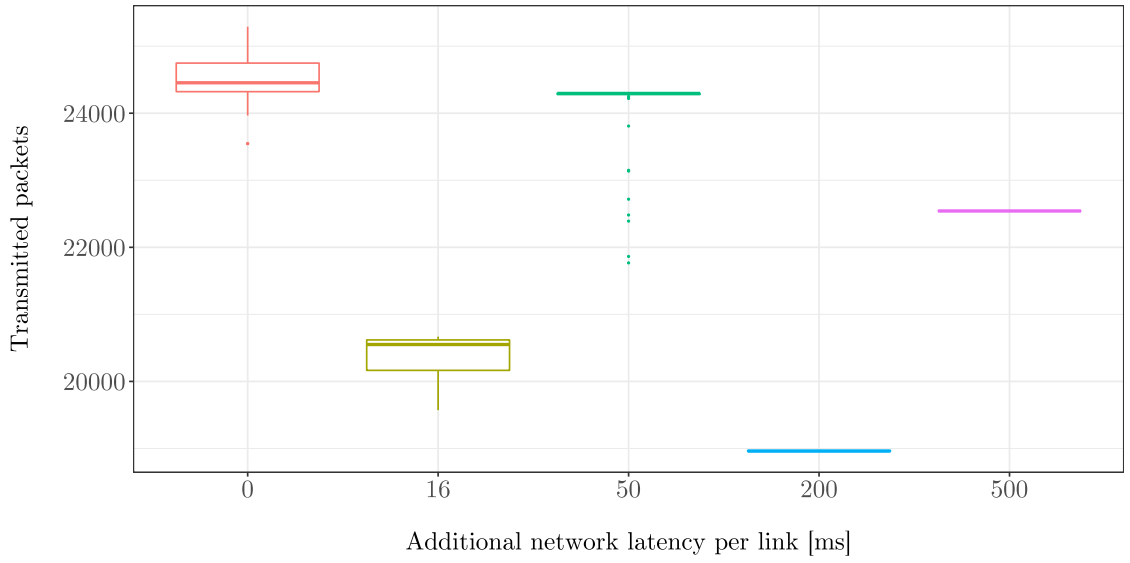

Figure 4.28: The impact of network latency on the amount of transmitted packets

4.7.7 Parallelized Protocol Invocations

Parameters	Values
Nodes	3
CPU Cores	8
CPU Frequency	2500 MHz
Parallelization	1, 5, 10, 20, 50, 100, 200, 500, 1000
Transmission rate	1 Gbit
Added network latency	0, 16, 50, 200, 500 ms
Packet loss	0 %
Number of input elements	1000
Number of repetitions	50

In order to investigate the influence of computation parallelization, we change the code as follows: we define a parallelization factor *pf*, which denotes the number of computations to be evaluated in parallel. The protocol `gp` produced in Listing 4.3 (p. 39) is not returned directly but combined with $pf - 1$ further computation protocols into a single `ParallelProtocolProducer`. This combined protocol is then returned instead of the original protocol `gp`.

Previous tests showed that the execution time is heavily influenced by the network latency. In situations when network latency is between 50 ms and 300 ms—which are usual delays for connections via the Internet—our tests already needed 5 to 25 minutes for execution.

In order make SMC applicable in such a context, it is of vital interest whether parallelization is able to compensate the higher network latency in terms of execution time.

Parallelization was parametrized with the values $\{1, 5, 10, 20, 50, 100, 200, 500, 1000\}$. Higher levels of parallelization are not possible in our setup as the overall input per node was also constrained to 1000 data points.

Execution Time

Figure 4.29 shows that parallelization has a positive impact on the elapsed execution time. In this overview, it becomes clear that except for $pf = 5$, increasing parallelization reduces

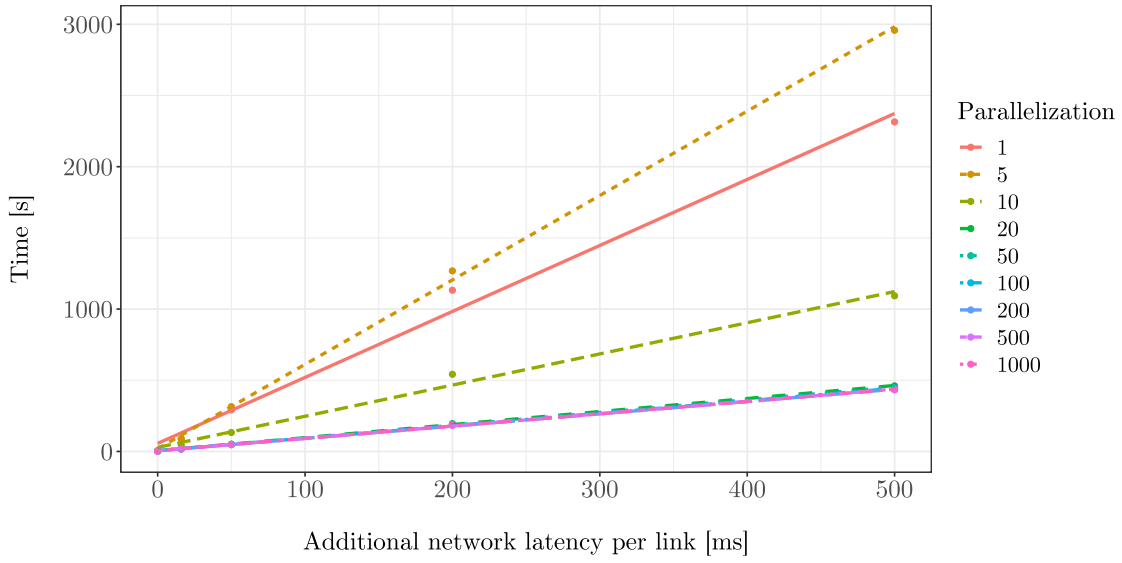

Figure 4.29: The impact of network latency on the execution time depending on parallelization by utilization of 8 cores per node.

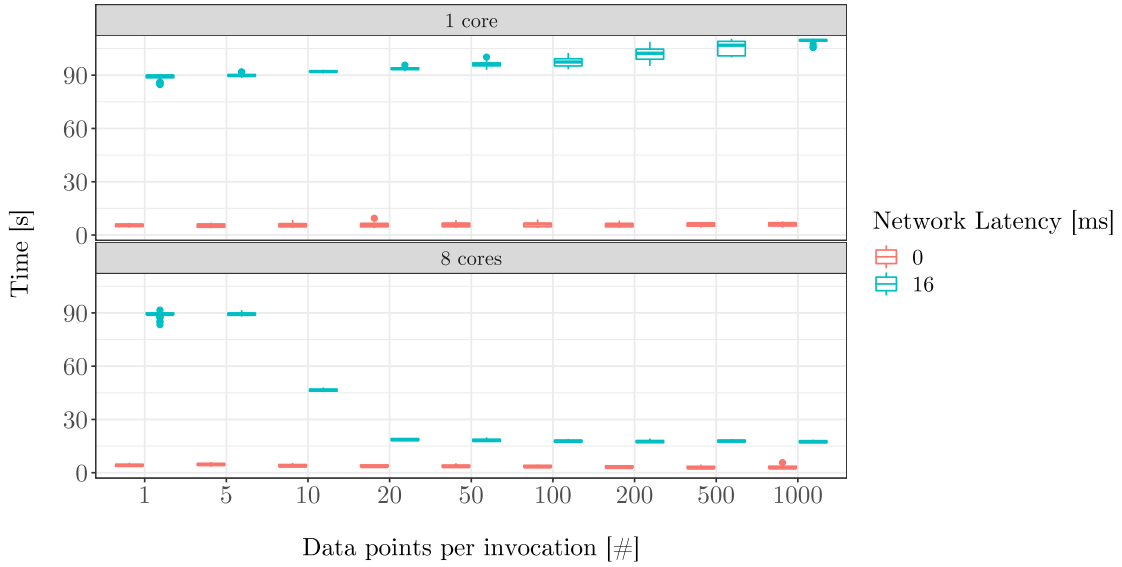

Figure 4.30: The impact of network latency on the execution time depending on parallelization compared by the number of cores.

the execution time while preserving its linear relation to network latency. For $pf = 5$ the overhead of executing protocols in parallel seems to outweigh the benefits of parallelization itself.

Figures 4.30 and 4.31 show the impact of parallelization in greater detail. We regard the case that each node has 8 cores available. Typically, the execution time first increases when parallelization is activated. I.e., parallelization itself induces a computational overhead, which also has to be compensated. Compensation typically happens with $pf \geq 10$, already providing noticeable improvements of the execution time. The execution time is further reduced with increasing pf. Only at the end, when computing the whole input in parallel, the execution time slightly increases in all cases except when the additional network latency is 16 ms where this increase already happens at $pf = 500$.[11] We consider this an anomaly.

[11] This effect is not visible in the diagrams.

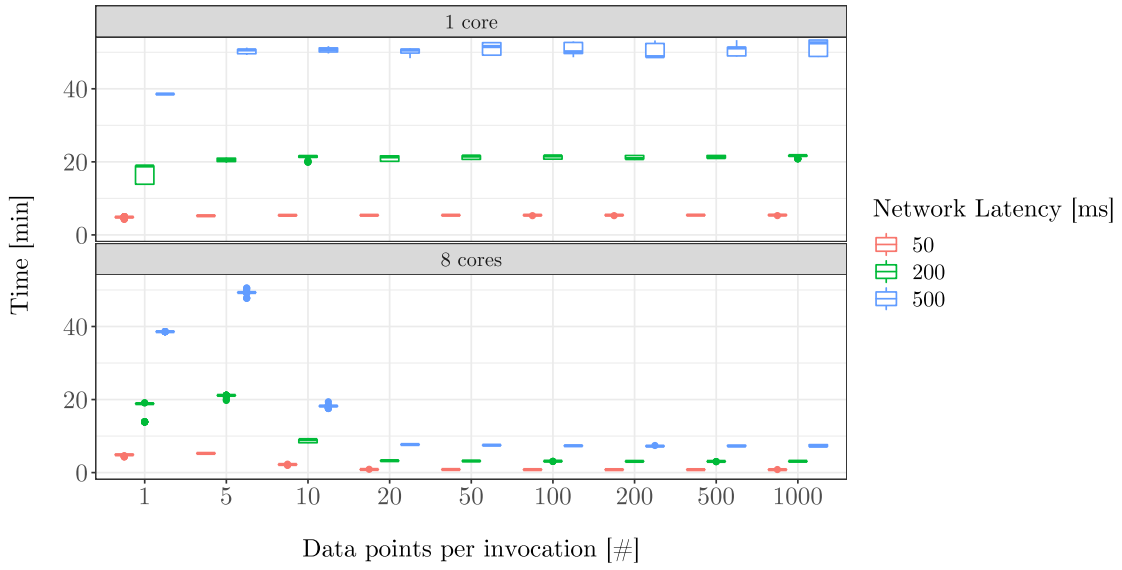

Figure 4.31: The impact of network latency on the execution time depending on parallelization compared by the number of cores.

In a real world setting, there is a delay dl between incoming data points. Hence, a higher parallelization factor induces longer waiting times between computations. From this vantage point, a low parallelization factor is more favorable. The current results show that, beginning with $pf = 20$, execution times only improve minimally. In our setting, a higher parallelization factor than 20 does not yield any advantage.

With increasing network latency, these characteristics stay the same, but scale with the overall execution time. Regarding the median of our results, it becomes visible that for an additional network latency of 0 ms, the execution time of a computation with $pf = 500$ is about 64% of the execution with $pf = 1$. With a network latency of additional 200 ms, this factor improves to ~16%, and when increasing it to 500 ms, the factor slightly degrades to ~19%. This observation can be explained as increased network latency provides "more space" for optimization: parallelization is able to improve the execution time to a much greater extent when the initial duration is considerably longer.

Regarding the case of a single core per node, it becomes clear that parallelization can only have a positive impact if cores are available which actually enable parallel computation of the provided protocols. If this premise is not given, the only effect of parallelization is the introduction of an uncompensated computational overhead which becomes visible in the top diagrams in Figures 4.30 and 4.31. The execution time slightly increases with a higher value for pf. Also in this case, this effect is stronger with higher network latency.

In general and for a fixed network latency, the processing time without parallelization of $dl + texec_1$ changes to an average delay of $\frac{pf}{2} * dl + texec_{pf}$. The first term reflects the necessity to wait for pf data points before the computation can start, the second is the (empirically measured) duration for computing a batch of pf data points in parallel. That means, parallelization reduces the overall duration if

$$\frac{pf}{2} * dl + texec_{pf} < dl + texec_1,$$

i.e.,

$$dl < \frac{texec_1 - texec_{pf}}{\frac{pf-2}{2}}.$$

For our series of measurements, the concrete resulting numbers (for 8 cores) are shown in Table 4.3. The numbered columns denote the value of the parallelization factor pf. The corresponding entries denote the above mentioned term, i.e., the maximum data latency in ms which is allowed per individual data point so that parallelization of the given factor is still more time efficient than carrying out no parallelization at all.[12]

We see, for putting parallelization effectively into use, the window of data latencies is very small. Hence, for many real-time applications where the data delay surpasses the mentioned limits, parallelization is inapplicable. However, for all use cases where data is already stored (e.g., in a data base) and can be retrieved in a batch manner, parallelization has practical use.

netlat	1	5	10	20	50	100	200	500	1000
0	0	-0.371	0.085	0.068	0.028	0.017	0.011	0.006	0.003
16	0	-0.229	10.681	7.835	2.956	1.455	0.723	0.286	0.144
50	0	-15.598	39.691	26.746	10.065	4.960	2.460	0.979	0.488
200	0	-90.606	147.530	104.101	39.252	19.285	9.574	3.809	1.896
500	0	-429.000	305.337	205.994	77.691	38.242	18.995	7.546	3.769

Table 4.3: Maximum data latency where parallelization yields a benefit

CPU Utilization

There is no notable influence of the parallelization on CPU cycles and instructions. Only the case with no additional network latency and 8 cores has significantly higher CPU activity. The explanation is found in the polling phase as described before, while parallelization has no strong influence on this fact.

Memory Allocation

There is no notable influence of the parallelization on memory consumption.

Transmitted Packets

Figure 4.32 shows the amount of transmitted packets depending on the number of CPU cores, the network latency and the parallelization factor. The changes in the amount of transmitted packets depending the aforementioned parameters are quite diverse. For the case with a single core, the following becomes apparent: with no additional network latency the amount does only fluctuate minimally while increasing the parallelization factor. In contrast, the case of 16 ms starts with a lower number of packets and then converges against the value of the 0 ms case. The packet count of the 25 ms case, however, starts a bit higher and converges from the top to the previously mentioned value. The cases of 200 ms and 500 ms exhibit a way more special behavior: they start comparatively low, then increase steeply for a $5 \leq pf \leq 20$ and then also converge.

For the other case, having 8 cores, the behavior is different to some degree: the case of 0 ms starts on the same level as the corresponding 1 core case and then increases slightly, staying roughly the same after $pf = 20$. The 16 ms case behaves similarly to its 1 core pendant, but while it increases only gradually over the increase of the parallelization factor, here, the whole increase happens between $10 \leq pf \leq 20$. The cases 50 ms, 200 ms and 500 ms exhibit a similar behavior, while starting at different heights. It is common to all but the case of 0 ms that the amount of packets decreases again slightly beginning at $pf = 20$.

[12] The negative numbers are semantically correct: no non-negative data latency would justify application of $pf = 5$. As the previous diagrams show, the reason is that $pf = 5$ takes more time than utilization of no parallelization in these cases.

In Section 4.7.4 on page 59 we already elaborated our interpretation that the communication layer of FRESCO merges packets if possible. As the amount of transmitted Kbytes shows the same behavior as the number of packets, we assume that this factor also creates the observed behavior in this case.

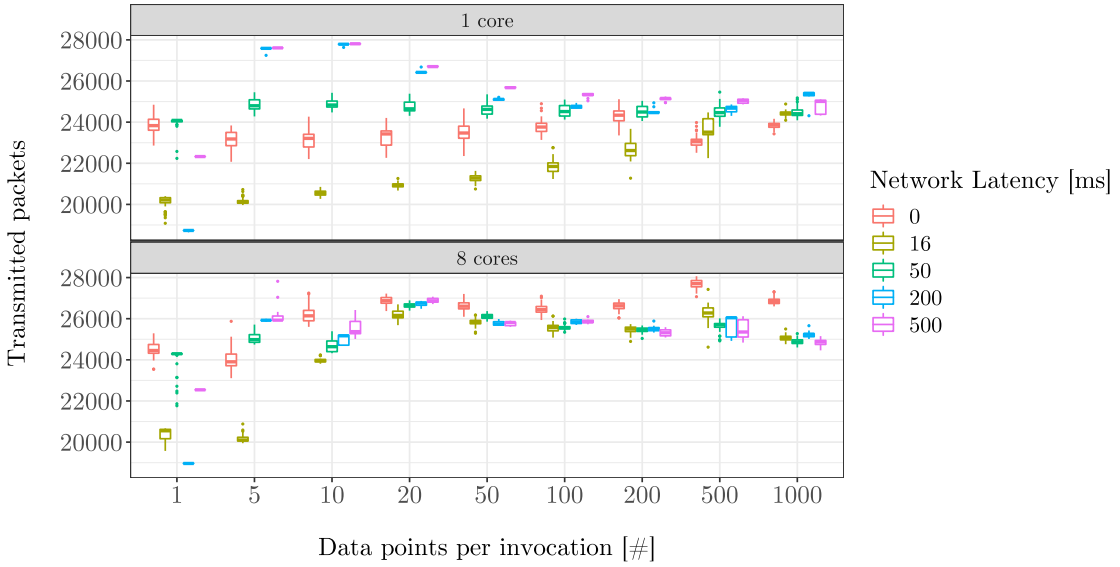

Figure 4.32: The impact of protocol parallelization on the amount of transmitted packets

4.8 Findings

In the following, we derive and aggregate overall findings from the measurement results of the previous section.

Execution Time

The measurements show that in our use case each single computation has a low duration; the overall duration increases linearly with the number of points or the number of peers. When varying the number of points, we could see that the slope of the time increase depends on the number of participating peers. More peers yield a steeper slope, i.e., a faster increase.

While these influences are comparatively small, the network parameters have the highest influence on the execution time. Considering practically relevant intervals for these parameters, we saw that the transmission rate can influence the execution time by a factor of 6, packet loss has an influence up to a factor of 110 and network latency can slow down the computation even by a factor of 450. These impediments already occur at network configurations which are realistic on the Internet or on the mobile Internet at least.

Due to this reason, we investigated whether the parallel execution of independent, subsequent computations can compensate the identified hindrances. We could show that parallelization is able to reduce the computation duration approximately by a factor of 6. Therefor, computation of 20 items in parallel is sufficient and already exploits the full parallelization potential, as long as more than a single core is available. Further increase of the parallelization factor did not yield notable improvements. However, the ability to parallelize always depends on the actual algorithm. Hence, the parallelization factor should be determined anew every time a new protocol is used.

Parallelization seems best applicable when performing batch processing of a series of input values. It guarantees that the system does not have to wait until sufficient data points for effective parallelization are available.

In contrast, when regarding real-time use cases, buffering multiple input values and waiting to perform parallel execution can easily outweigh the benefits of parallelization (cf. Table 4.3 on page 70). Hence, in these cases serial execution might be preferable.

CPU Utilization

Regarding the influence on CPU instructions and cycles, we expected that the parameters fall into two distinct categories: the number of points, the number of peers, and possibly the parallelization factor are convincing to directly influence the aforementioned variables. On contrary, the number of CPU cores and their frequency, the transmission rate, the packet loss and the network latency do not influence the executed code itself and are not assumed to influence the number of instructions and at most minimally influence the number of cycles performed by the measured Java process.

During the evaluation of our measurements we observed that these assumptions did not trivially hold without further analysis. It is correct that the number of points and peers influences the described variables. In both cases their value increases linearly. We could see that, if parameterizing the number of input elements, the slope of the instruction and cycle increase is not influenced by the number of peers.

Aditionally, the support of the just-in-time (JIT) compiler became clear when manipulating the number of input elements. Already before reaching 1000 points, the increase of instructions and cycles become flatter, implying a reduced amount of CPU instructions which is necessary to execute the code. We assume that this effect also exists when varying other parameters like the number of peers. However, this is not visible in our measurements.

When manipulating the seemingly unaffecting parameters, we could nevertheless identify changes in the value of both variables. We identified a common reason for this behavior: the setup phase as described in Section 4.6 contains an interval of intensive polling in order to establish connections to other peers. This runs as fast as the environment allows, severely increasing the number of executed instructions and used cycles. Only when the CPU itself is constrained by its number of cores or frequency, or some external factor, like the network latency or the packet loss, the artificially created amount of instructions and cycles is considerably reduced.

It was not possible during the measurements to circumvent this effect completely. However, in some experiments, we could distinctly show how much the setup phase contributed. Subtracting this identified amount, the number of instructions stayed nearly constant and the number of cycles was only minimally influenced. The latter is a consequence of the computation process having to wait longer for incoming data.

Finally, while parallelization yields a notable improvement on execution time, it does not cause any changes regarding the CPU variables. These are advantageous circumstances as it implies that no particularly strong CPUs are necessary in order to perform parallelization. Multi-core hosts, on contrary, are still necessary in order to leverage parallelization advantages at all.

Memory Allocation

In all our experiments, the maximum allocated stack memory was around 20 Mbytes. This value only varied slightly, depending on the speed of execution. Whenever impeding factors, like lower transmission rate or less CPU cores slow down the execution, the amount of stack memory is reduced to approximately 17 Mbytes. However, this is not an effect of the executed code, but of the garbage collector which starts to act differently, and in these cases frees unused memory at a lower limit.

The maximum allocated heap memory is the first variable which seemingly exhibits critical behavior. While it converges to a certain value when increasing the number of elements, it increases step-wise when scaling the amount of participants. The behavior matches the expectations: while the values of the previous computation can simply be forgotten and do not have to be saved during the whole computation process, each additional peer increases the amount of data which has to be continuously kept in memory.

This may be true, but it does not happen in the dimensions that we see in the first glance. We investigated memory consumption in detail (cf. Figure 4.13), and found out that the overall maximum allocation is not the decisive feature. Every time the garbage collector is invoked, the maximum memory allocation is reset back to 20–40 MBytes. In other words, while it seems to consume considerable amount of memory, this is another artifact of the garbage collector.

Transmitted Packets

Like the CPU variables, the amount of transmitted packets is mostly influenced by the setup phase and its high frequency polling. However, as we have saved the whole communication dump during the measurements, we are able to simply remove all packets which were exchanged during the setup phase. The packets of the setup phase consist of nearly exclusively SYN and RST packets. Additionally, they occur also nearly exclusively in the setup phase. Only in the tear down phase, a very small amount of RST packets is issued.

Our measurements show that the current implementation of the setup phase is burdensome for all participants as well as the network (depending on the performance of the communicating hosts). Hence, we highly suggest to exchange this mechanism against some less resource-intensive method of building the inital peer connections in order to keep the solution efficient. A possible solution without need to change the Fresco framework itself is to orchestrate the startup of the software by another management layer. After this layer has identified connectability of all peers in an efficient manner, Fresco is started. This should result in essentially shorter polling time.

Due to the possibility to filter out setup phase traffic, we have been able to analyze other effects without the influence of the polling. Increasing the number of points or the number of peers leads to a linear increase of the amount of transmitted packets. Both they provide a multiplicative factor to the slope.

Additionally, when manipulating the network parameters, we could see that the communication layer of Fresco performs aggregation of packets: when using a low transmission rate, high network latency or having packet loss, the histogram of packet sizes (e. g., Figure 4.20) shows a higher amount of bigger packets. The reason is as follows: the computation layer creates a message for a specified peer and hands it over to the networking layer of Fresco. There are cases where the networking layer is not able to send the corresponding packet before a new message from the same sender to the same recipient is created. When this second message is handed to the networking layer, the messages are combined into a single packet.

This reduction also decreases the number of headers which is to be sent as overhead. Hence, a smaller overall amount of data has to be transmitted.

4.9 Practical Implications

Our results show that Fresco as an implementation of SMC possesses a performance and resource utilization behavior which allows practical application: in the setting of an intranet, computations are efficiently performed. The execution time is around 2 to 3 ms

per session and peer. This allows batch processing of data and interactive use cases. Performance might, however, not be sufficient for the realization of real-time applications depending on the computation to be carried out. Regarding the hosts systems, multiple cores are necessary when parallelization can be utilized. In other cases, secure computation should also be feasible with weaker devices. Memory seemed to be a problem at first glance, but could be attributed to the garbage collector. Having a setting of memory constrained devices, a more economical programming language than Java would be more appropriate.

In wide area networks as the Internet and possibly mobile Internet, network latency is the most influential constraining factor. Execution time worsens substantially with increasing latency. In these contexts, we currently only see batch processing as a use case: if it is acceptable to wait several minutes for a computation result, SMC can be utilized. However, in this context it is more likely that parallelization can be applied, which decreases the latency penalties to some degree. Further improvement of the situation would require to reduce the amount of transmitted packets. This could be possible by stricter orchestration of computations running in parallel, where packets between different peers would be used for multiple sessions simultaneously. Given that each participant has k input values available and all protocols are executed completely simultaneously and synchronized. Then, k shares from the same source to the same destination could be combined to a single message, effectively reducing the communication duration by factor k. On the contrary, our current solution applies parallelization which does not enforce message combination, but only enabled waiting times per host to be used for further computations.

4.10 Related Work

The newly gained interest in SMC during the last year resulted in a multitude of publications which propose successive improvements of established approaches (cf. Section 3.4.1). Most of the time, these papers include a small evaluation which demonstrates the (improved) efficiency of the presented solution. Related work is mostly constituted out of this type of publication. We compare our work especially regarding the evaluation method and the variables analyzed.

Pinkas et al. [PSSW09] address garbled circuits and present several optimizations. They perform an evaluation regarding computation time and amount of transferred data.

Bogdanov et al. [BLW08] provide *Sharemind*, an SMC framework based on secret sharing. A relevant characteristic is that multiparty computations have to be reduced to three-party computations in order to fit to Shareminds system architecture. Under the assumption of non-collusion between the three parties, privacy is still preserved for an arbitrary number of input parties, which provide the data to work on. In their performance evaluation, they assess the influence of the number of input values (in terms of vector size) on the running time.

In [BNTW12], Bogdanov et al. furthermore present functional extensions to Sharemind. They improve the efficiency of existing protocol primitives and add new secure operations. Their performance analysis primarily discusses theoretic complexity of their solutions. Additionally, they perform a practical benchmark of their software in a high performance computing cluster inspecting the influence of parallelized operations on the running time.

Kerschbaum et al. [KBD09] propose single purpose comparison protocols based on homomorphic encryption and secret sharing schemes. They perform a theoretical analysis of round complexity and compare both developed solutions in a virtual setting with respect to running time.

Kerschbaum et al. [KDSB09] propose a benchmarking protocol for the evaluation of key performance indicators based on oblivious transfer and homomorphic encryption. They

furthermore specify a communication pattern—central coordination—for which they claim that it makes the communication overhead in SMC solutions practically independent from network latency. However, a weakness of their study which they state themselves is that "computation time far exceeds the delay on the network" already in the first place. With their communication pattern, they also introduce parallelized round execution on all participating clients. This bears some similarity to how Fresco is implemented. Their performance evaluation focuses on theoretic complexity and running time.

Ben-David et al. [BDNP08] present *FairplayMP*, "a generic system for secure multi-party computation". It is based on garbled circuits and in particular on the Beaver-Micali-Rogaway [BMR90] protocol. They evaluate dependency of the running time on circuit size, number of participants, circuit depth and a security parameter. Furthermore, they evaluate real world settings for voting and computing auctions regarding running time.

Henecka et al. [HKS+10] propose the SMC compiler *TASTY* which generates and optimizes SMC protocols based on homomorphic encryption and/or garbled circuits from high-level specifications. They assess their solution in terms of circuit size, amount of communicated data between the peers, and running time.

In [KOS16] Keller et al. describe a "practical protocol for secure multi-party computation of arithmetic circuits based on oblivious transfer". They assess their approach regarding throughput and duration while varying the number of participating parties, the transmission rate and the number of inputs (for an auction algorithm).

Sepia is written by Burkhart [Bur11] and has been proposed and evaluated in [BSMD10]. They assessed their solution by determining its CPU and bandwidth requirements while varying the number of inputting and computing peers as well as the running time for different types of protocols. Furthermore, Internet-wide tests and comparisons with VIFF and FairplayMP were performed.

While these publications present short evaluations of their own work, the body of research is lacking the proposition of a general performance model for SMC and thorough examination of an SMC solution with respect to more than the number of peers and points on the side of the parameters and often solely the overall running time on the side of the variables.

In order to assess feasibility of SMC in real world contexts, we additionally addressed the resource consumption on the deployed hosts and the dependency on network parameters.

4.11 Key Contributions of this Chapter

In this chapter, we examined the performance characteristics of an SMC implementation in detail. Among a selection of available frameworks, we selected the candidate Fresco which seems especially promising for practical application, feasible analysis and later extension by our work preformed in the following chapters. In detail, our contributions are:

Base-line assessment of SMC We conducted a performance analysis, focusing on the overhead SMC causes when performing computations of a basic but realistic use case. Herefor, we chose the production-ready SMC framework Fresco. We will use the same framework in other chapters to prove feasibility of SMC in dynamic contexts (cf. Chapters 7–9).

Comprehensive study of performance characteristics In comparison to related work, we essentially extended the parameter and variable space of performance measurements. Whereas mainly time measurements are performed, and parameters often focus on

the number of cooperating peers, we assess the following dimensions: *number of points*, *number of nodes*, *number of cores*, *frequency of CPU*, *transmission rate*, *packet loss*, *network latency*, and *parallelization of computation*.

Execution time Each computation takes only a small duration, in our use case it costs around 2–3 ms per session and peer. The biggest influence on the execution time is caused by the networking parameters. Here, the network latency can render computations infeasible under realistic Internet conditions. We also examined whether parallelization of independent computations can level this effect. It achieves a time reduction, but the effects are two orders of magnitude lower than the negative impact of network latency.

CPU Cycles and Instructions The CPU variables are mainly influenced by the setup phase of Fresco where all peers establish mutual connections. This is only a Fresco specific characteristic and not typical for SMC in general. As this this effect is caused by busy waiting and polling for connection establishment, CPU consumption can be reduced when adding constraints like a higher network latency or lower transmission rate.

Besides this effect, we could identify that the Just-in-Time compiler of the JVM improves efficiency of the computation and reduces the amount of cycles consumed.

Furthermore, parallelization does not yield any CPU penalties while reducing the computation time as described above.

Memory consumption Stack memory is not notably influenced and always ranges around the acceptable value of 20 MB. In contrast, it seemed that heap memory can become a critical resource, as every added peer implies higher memory consumption on all participants. For 11 to 15 peers, already 840 MB are allocated. We could, however, show that this is an artifact of the garbage collection and the actually used amount of memory is considerably lower. This effect is therefore Java specific and assume other implementations which do not build on the JVM to be more memory efficient.

Transferred Packets Initially, the amount of transferred packets also was mainly influenced by the aforementioned setup phase. However, we could remove this influence during evaluation. The remaining effects were in line with theoretical expectations. We could especially observe a linear increase of packets per peer when scaling the number of participants. In the network, this yields the expected quadratic increase of messages among all communication channels.

Furthermore, we detected an effect of the networking layer of Fresco: when sending of a message to a specified recipient is delayed by the environment (e.g., network latency), the next message to the same recipient is merged into the previous packet, if it is available before the first could be sent. This improves transmission efficiency in these cases as the header of the potential second packet can be omitted. In consequence, less data has to be transferred.

4.12 Statement on Author's Contributions

The findings of this chapter have been published in the following paper:
M. von Maltitz and G. Carle. A Performance and Resource Consumption Assessment of Secret Sharing based Secure Multiparty Computation. In J. Garcia-Alfaro, J. Herrera-Joancomarti, G. Livraga, and R. Rios, editors, *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 357–372. Springer International Publishing, Barcelona, Spain, 2018 (reference [vMC18a]). It is based on the measurements performed by the author and presented in this chapter.

The publication [vMC18a] features a reduced version of Section 4.1 where we argue for the selection of Fresco. Shortened, the use case of Section 4.2 is presented as measurement scenario. Identified related work (cf. Section 4.10)

is only presented in a highly condensed manner. As chapter of this thesis, the results are presented in a more systematic and comprehensive manner; especially each varied parameter is discussed in a dedicated subsection. Furthermore, performance assessment of computation parallelization has been omitted completely in the publication for brevity. Vice versa, the preliminary execution time considerations and the theoretical comparison of SMC with a TTP were initially written for the paper, and incorporated into this thesis as Section 4.3 with only minor textual changes.

5. Real-World Scenario Assessment of Secure Multiparty Computation

In the previous chapter, we examined the influence of relevant parameters on the performance of Fresco in a structured manner. We examined the parameters in isolation, as executing the complete set of permutations requires an amount of time which is too big to be practically feasible. However, this lead to the omission of realistic settings.

Objective

Due to this reason, this chapter examines a selection of settings where Fresco could be of interest. Each setting constitutes an individual combination of all examined parameters so that a realistic and specific environment is represented. This yields better assessment in which settings SMC can be successfully applied.

5.1 Settings

For our real world assessment, we examine a set of different settings. These consist of two intranet settings, tethered and unethered, as well as Internet-wide communication and communication via mobile Internet.

5.1.1 Intranet

The first two settings examine the environment of tethered and untethered intranet. These reflect home networks and corporate intranets inside a single building.

Tethered

This setting occurs in an environment like the following: when a company wants to employ Fresco for some of their business data, they will deliberately select hardware which yields the best performance of their setup. Hence, we do not consider the utilized host hardware to be a bottleneck. The network is that of an intranet, being considerably high-speed with a low latency and practically no packet loss.

Regarding the desired quality of the service, we cannot predict the actual requirements, but two cases seem sensible: in this setting, interactive services are conceivable, i.e., where the answer to a request should not take longer than a second as well as batch services which allow a delay of several minutes or more. We will also test whether and how much parallelization should be applied in order to support these requirements.

We model this setting as follows:

Parameters	Values
Nodes	3
CPU Cores	8
CPU Frequency	2500 MHz
Parallelization	0, 10, 20
Transmission rate	1 Gbit
Added network latency	0 ms
Packet loss	0 %
Number of input elements	100
Number of repetitions	20

Wireless

With wireless home and business networks and also the Internet of things and smart homes in mind, we consider embedded low-end devices mostly connected via wireless LAN inside a single regional domain like a building or an appartment. In terms of host hardware, we take the Raspberry Pi [Ras17] as reference. The current model is the *Raspberry Pi 3 B* and possesses a 1.2 GHz 64-bit quadcore ARMv8 CPU and a 802.11n wireless LAN module. The smaller version *Zero W* has a 1 GHz single-core CPU and a 802.11 b/g/n wireless LAN module.

Regarding the network, we assume a standard 802.11n wireless LAN with 54 Mbit/s. The latency is about 10 ms and packet loss around 2 %.

Again, we mainly expect interactive services where the delay until a response is received should not exceed some seconds.

We aim to model this environment using the following parameter values, while respecting the constraints of our measurement hardware:

Parameters	Values
Nodes	3
CPU Cores	4
CPU Frequency	1600 MHz
Parallelization	1, 10, 20
Transmission rate	54 Mbit
Added network latency	10 ms
Packet loss	2 %
Number of input elements	100
Number of repetitions	20

5.1.2 Internet

In this section we focus on another setting where the fundamental difference is that the hosts are not in the same intranet, but connected via the Internet. Several publications (e.g., [BCD+09], [ZDT+16], [BSMD10], [BTW12]) depict cases which are or at least could be performed via the Internet.

We assume, like in the first case, dedicated hardware. The network, however, is strongly constrained in this setting. We assume an Internet connection of 100 Mbit/s downstream and 5 Mbit/s upstream and a latency of 10 ms. Furthermore, we assume enough capacity in the network so that there is no notable packet loss.

Due to the differences between intranet and Internet we change our performance expectations from an interactive service to a service which is able to deliver a batch of results in a time frame of several minutes at maximum. We also assess parallelization in this context.

The parameters of this model are:

Parameters	Values
Nodes	3
CPU Cores	8
CPU Frequency	2500 MHz
Parallelization	1, 10, 20
Transmission rate	100 Mbit/5 Mbit
Added network latency	50–150 ms
Packet loss	0 %
Number of input elements	100
Number of repetitions	20

5.1.3 Mobile Internet

Smartphones are ubiquitous today and typically store a huge amount of privacy-sensitive data. Hence, this platform is a promising domain where SMC can find application.

Current state-of-the-art smartphones typically have two to four CPU cores at a speed of 1.5 to 2.5 GHz.

Regarding networking, we have two options here. The first is again a wireless LAN connection with other peers being in proximity. The parameters of this case have certain similarity to the second intranet case. Hence, we omit this case here.

The other one is a connection via the mobile Internet. We assume the usage of the standard Long Time Evolution (LTE) here. That means, the transmission rate is theoretically up to 300 Mbit/s, but practically ~50 Mbit/s today. Latency is assumed to be around 200 ms. For realistic conditions, we vary the packet loss between 0 and 5 %.

In the real world, mobile Internet connections bear additional problems which we do not address here: firstly, establishment of peer to peer connections between participants is normally not possible due to provider restrictions. Secondly, interrupts during a session require a restart. Hence, with non-negligible probability of connection failures, it will not practically be possible to carry out longer sessions.

Parameters	Values
Nodes	3
CPU Cores	4
CPU Frequency	1600 MHz
Parallelization	1, 10, 20
Transmission rate	50 Mbit
Added network latency	100 ms
Packet loss	0, 1 %
Number of input elements	100
Number of repetitions	20

5.2 Results

In the following, we interpret the results of the setting-specific measurements. While the detailed effects have been explained in the previous chapter, here, we make an overall assessment regarding the applicability of the given SMC solution in the contexts.

5.2.1 Running Time

As we are measuring the real world settings, we broadened our time measurements: while we assessed the time of the SMC execution alone in the previous chapter, we now regard the complete running time of the process i.e., the time from the start of the computation process until the final result is received.

We observe that the network parameters have the biggest influence (Figure 5.1). While the tethered intranet case is executed around 3 seconds, untethered intranet already needs 25 seconds for execution, shortly followed by the Internet measurement with less packet loss but higher latency. All other measurements have an overall duration of a minute or more.

For real-time settings, we have to consider that these measurements computed 100 elements. Consequently, the average time per element is below a second in all cases. However, a necessary premise for this result is that there is no essential startup overhead for every computation.

For settings which need to perform computation in batches, parallelization is a helpful strategy to mitigate prohibitive running times. The diagram shows that it reduces the running time approximately by a factor of four, reducing all measurements to run under 25 seconds. We consider this to be acceptable in all cases with hosts of fixed locations. In the mobile settings, this duration could already be infeasible.

Please note that the amortized view and the parallelization cannot be trivially combined. This results from the simple fact that at least pf elements have to be computed in order to perform a parallelization of a factor of pf. Furthermore, if the input elements occur sequentially after another (and not in bulk), the delay of the calculation of the first element is still the full running time of the parallelized computation and the delay of the following $pf - 1$ data points.

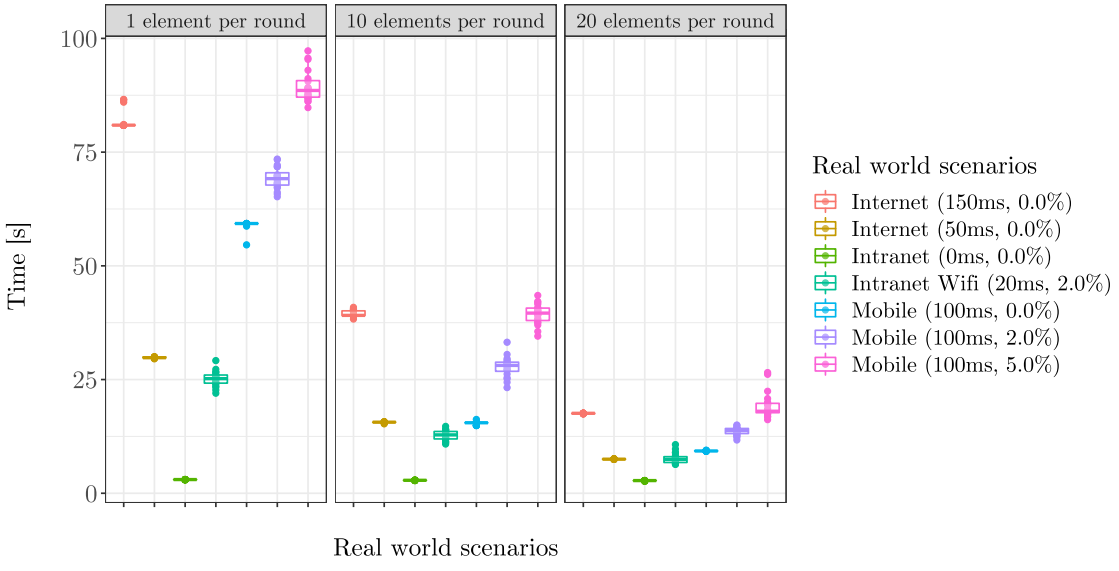

Figure 5.1: Execution time in the use cases

5.2.2 CPU Utilization

The measurement of the CPU behavior in the real world settings shows well-known effects explained in the previous chapter: only the case of tethered intranet, with no further constraints and high-end host systems has a drastically increased number of cycles. This has already been explained as an effect caused by the setup phase in Section 4.7.2 on page 53.

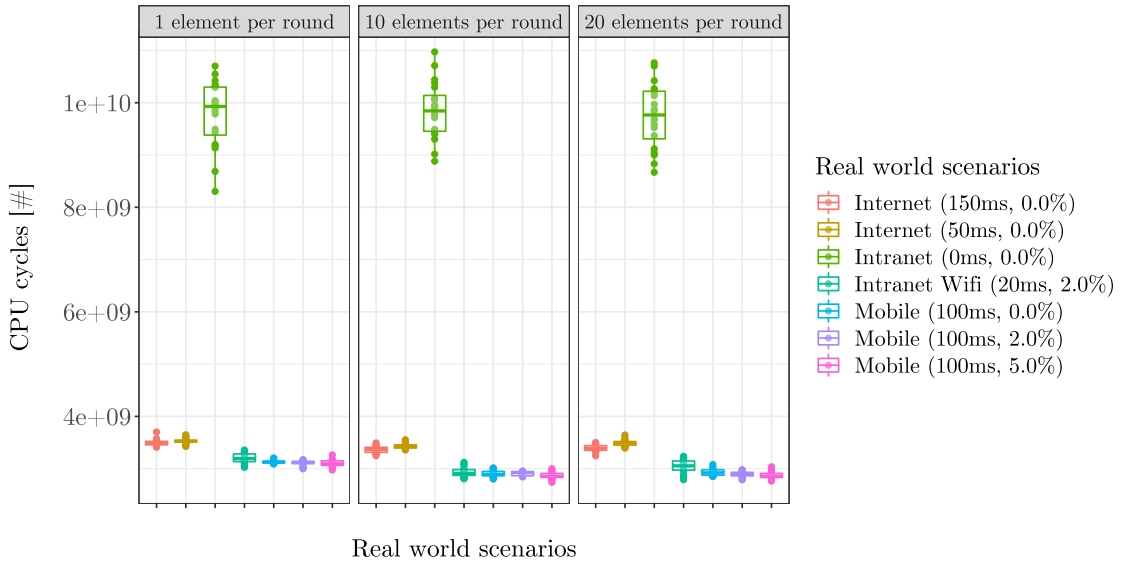

Figure 5.2: CPU cycles in the use cases

5.2.3 Transmitted Packets

The amount of transmitted packets (Figure 5.3) exhibts different behavior depending on the degree of parallelization. With no parallelization, only in the tethered intranet case considerably more packets are sent than all other settings. With parallelization, the amount is reduced by more than half in some of the cases. The tethered intranet case, however, is even increased; so are both Internet cases. In the previous chapter we identified and discussed that the network layer aggregates packet at certain occasions (cf. page 73). We assume that these measurement results are another example for this behavior.

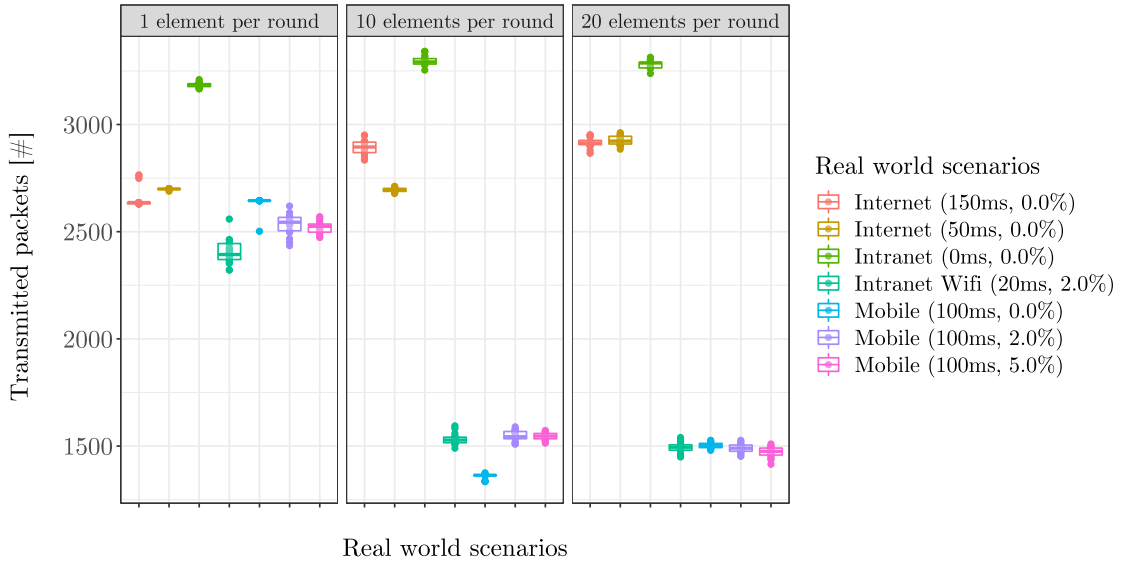

Figure 5.3: Transmitted packets in the use cases

5.3 Key Contributions of this Chapter

In the previous chapter, we performed a systematical analysis of the performance influences of several parameters on the resource consumption of Fresco. In this chapter, we created parameter combinations to form realistic scenarios in which we analyzed Fresco.

In general, only the time and the heap memory resource differentiate strongly enough to enable or prevent different ways of application as seen in the last chapter. All other variables are uncritical regarding application. However, we argued that the memory needed is actually lower than the memory allocated. The identified peaks are rather artifacts of the garbage collector. Hence, we focus on execution time here. Our insights are as follows:

Intranet setting The tethered intranet setting yields durations of a small amount of milliseconds per computation. This enables applications from soft real-time applications over interactive scenarios to batch processing. Moving to untethered connections, the duration is increased so that it becomes unpractical for real-time applications.

Since the durations are comparatively small, parallelization yields only moderate benefits.

Internet setting In the Internet setting, the speed of the computation mainly depends on the network latency. High latency renders real-time and—depending on the specific use case—interactive scenarios infeasible. Nevertheless, batch and background processing of data is still a valid application.

Mobile Internet setting On certain occasions, mobile Internet cases perform better than Internet cases. This mainly depends on the slightly improved network latency which has a higher influence compared to packet loss. However, it remains prohibitively slow for real-time and interactive app and realistic considerations show that the set of possible applications is further constrained: while batch and background use cases are theoretically feasible, we assume them to be incompatible with the mobile setting where the nodes are moving devices with occasional connection losses.

Parallelization Generally, parallelization yields vivid improvements for all use cases effectively reducing the overall time for 100 computations to under 20 seconds. While this means that every computation takes around 20 ms, the actual delay from inputting a raw value to receiving the result is still 20 seconds. In consequence, this performance improvement does not support real-time or interactive use cases.

6. Secure Evaluation of Patient Data in Medical Studies

In the last chapters, we introduced a performance baseline for SMC. We used a simple use case in order to identify the boundary of optimal SMC performance. Here, we complement these measurements with a realistic case of SMC application.

We address a domain in which application of SMC is highly promising and with a great prospect of beneficial utilization: in the medical sector, patient data and health records are highly critical, and even specially protected by data protection legislation, like the General Data Protection Regulation (GDPR) in the EU or the Health Insurance Portability and Accountability Act (HIPAA) in the United States. At the same time, it constitutes the foundation for gaining insights about the human body, diseases and effective medication. In other words, being able to use this data for research can yield essential benefits for society.

Data protection and data utilization are in a strong conflict. Here, SMC can be a technological solution to this problem: data protection and data utilization are decoupled, fulfilling legal requirements and making data evaluation possible without requiring bureaucratic overhead, e.g., data usage agreements and contracts.

For exemplifying, how SMC can be used to solve this conflict, we select the topic of *survival analysis*; a common approach to assess the effectiveness of treatments and medication.

Objective

In this chapter, we address Research Question Q3. We develop a secure algorithm which allows the execution of certain kinds of multi-centric studies without requiring the parties to share their data. Afterwards, we assess the performance and scalability of the solution in two settings, providing further answers to Research Question Q2. Firstly, it is evaluated thoroughly in a testbed using synthetic data. Secondly, we use a real-world setup between the University Hospital of Ludwig-Maximilians-Universität München and Charité Berlin for computing on real data from a medical study [NAK+18].

6.1 Survival Analysis

In life sciences, the survival of a population is a vital performance indicator. It allows to estimate the influence of diseases, illnesses, therapies or medications on patients over time.

It investigates how and when certain preconditions lead to the event of death and how high the probability of this event is at a given time.

Survival analysis is not restricted to actual deaths of individuals; it generalizes for several other contexts. In consequence, *death* is substituted with the happening of a predefined kind of *event* or *failure* which can have various meanings depending on the context. Examples for events are the infection with an illness, regression of a tumor after treatments or even the first drug abuse by teenage individuals [KM03]. However, the standard question is still the actual survival after diagnosis of an illness or a selected therapy.

From collected survival information further insights can be derived. Examples are the median of survival, which tells about the typical life time of an individual of the population or comparison of different treatments (or lifestyles) with respect to life expectancy.

Modelling

Survival in a population is defined as

$$S(t) \equiv \frac{N(t)}{N(0)}. \tag{6.1}$$

where $t \equiv 0$ is the initial starting point of observation and $N(0)$ is the initial population. $S(t)$ then is a function of time, denoting the percentage of survived individuals.

The median of survival, t_{median} is then implicitly defined as:

$$S(t_{median}) = \frac{1}{2}. \tag{6.2}$$

Correspondingly, the life expectancy can be defined as

$$\int_0^1 t \; dS \tag{6.3}$$

Real-World Problem: Censoring

The modeling of $S(t)$ and its derived characteristics is simple, however, it exhibits a fundamental problem which is rooted in the real world: the population is typically neither directly not fully observable. This problem is addressed by substituting the population against a sample, the *study population* and binning the time dimension into discrete intervals $t \in T$. With this step, survival becomes observable by discrete events of the members of the study: if we have a population of $N(t)$ and $failures_t$ at time t, then $N(t+1) = N(t) - failures_t$.

Even is this setting information loss occurs. This is called *censoring*. Firstly, participants may stop being observable starting at some time $t' > 0$ ("lost to follow up"). These participants cannot be counted for the remaining time. Alternatively, the study itself can be the reason. If it ends at time t'' and $N(t'') > 0$, there are participants whose event time remains unknown. The only information is that their time-to-event is greater than the overall duration of the study itself.

Despite these hindrances, the participants in question should not be removed from the data set since they exhibit valuable insights until their state becomes unknown. Besides the events $failures_t$ we hence define the number $censorings_t$ at time t. The abovementioned formula then becomes $N(t+1) = N(t) - failures_t - censorings_t$.

time	Risk set		Failures	
t	treatment	control	treatment	control
1	21	21	0	2
2	21	19	0	2
3	21	17	0	1
4	21	16	0	2
5	21	14	0	2
6	21	12	3	0
7	17	12	1	0
8	16	12	0	4
10	15	8	1	0
11	13	8	0	2
12	12	6	0	2
13	12	4	1	0
15	11	4	0	1
16	11	3	1	0
17	10	3	0	1
22	7	2	1	1
23	6	1	1	1

Table 6.1: Example data. For each t, the size of both risk sets and the number of events (failures) are reported. Two cases appear: in transition from $t = 1$ to $t = 2$, two failures in the control group lead to a decrease of the risk set from size 21 to size 19. In transition from $t = 8$ to $t = 10$, no failures are reported in the treatment group. Nevertheless, the size decreases from 16 to 15. Here, censoring of a participant occurred.

6.1.1 Kaplan–Meier Estimator

We then need an estimator $\hat{S}(t)$ which sufficiently approximates $S(t)$. This function should achieve maximum utilization of the available data and should be bias-free. This means that the limit of the estimator is the actual function in question, i.e., the censored data does not distort the result.

A common method in life sciences is the *Kaplan–Meier estimator* [KM58], also named *product limit estimator*. The estimator is designed as follows: the study begins at $t \equiv 0$. The number of initial participants $N(0)$ of the study is denoted as the $riskset_0$. For each time t the number of events is denoted $failures_t$. $\frac{failures_t}{riskset_t}$ then is the probability for an event at time t. We can then define

$$\hat{S}(t) \equiv \prod_{t' \leq t} (1 - \frac{failures_{t'}}{riskset_{t'}}). \tag{6.4}$$

It is helpful to see that if no events happen for a certain time t', then the corresponding factor is 1. Consequently, all time intervals with no events can be neglected in the computation.

Example An example of data which is necessary for the calculation of the Kaplan–Meier estimator is given in Table 6.1. Here, we consider two different groups, *treatment* and *control*.

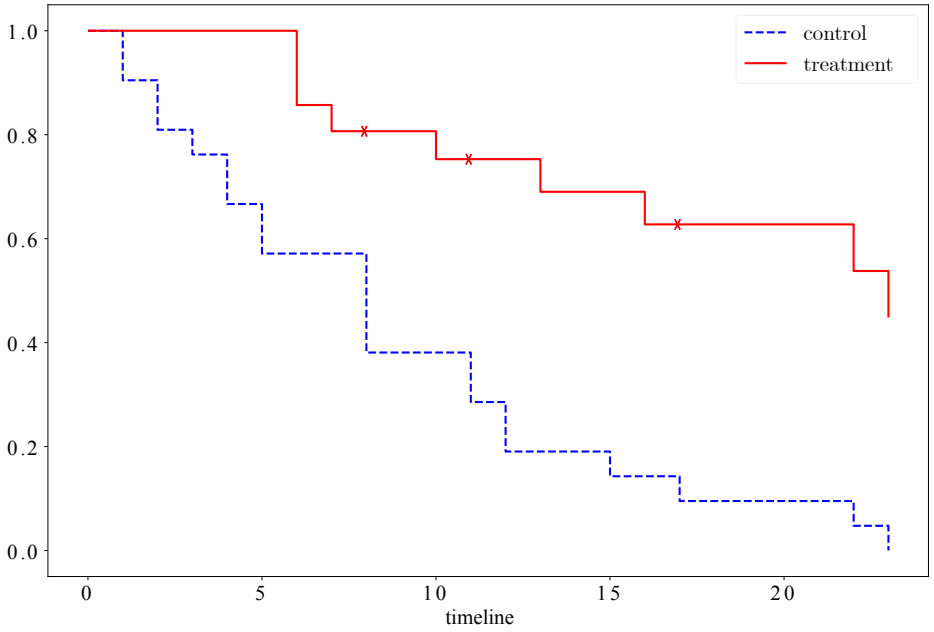

Figure 6.1: Visualization of the example data from Table 6.1. Each decrease in the line indicates the occurrence of a corresponding amount of events. Each marker on a horizontal line is the censoring of a participant. In the visualization a clear distinction between the survival of the participants of the treatment and the control group becomes clear.

Visualization as Kaplan–Meier Survival Plot

The results of the Kaplan–Meier estimator can be plotted. An example visualization of the data from Table 6.1 is shown in Figure 6.1. The obtained diagram is already a measure of communication in life sciences, especially in publications examining the difference in survival of different populations. If events occur at a given point in time, a decrease in the plot illustrates this fact. Censorings are represented as crosses on the line. A common additional feature would be confidence intervals which span around the actual curve.

Differences in survival often become tangible in the plots, giving first insights about the consequences of a disease or the performance of the examined treatment. Nevertheless, instead of relying to visual comparison, further metrics are considered which allow numerical assessment how different two or more Kaplan–Meier estimators of compared groups are.

6.1.2 Log-Rank Test

One of these measures for the significance of the difference between populations is the so-called *P value*. It can be derived by the log-rank test developed by Mantel [Man66] [PP72]. We consider two groups, which we call *treatment* (*ttmt*) and *control* (*ctrl*). $riskset_{ctrl,t}$ is then the size of the control group at time t, and $failures_{ctrl,t}$ is the number of events at time t. $riskset_{ttmt,t}$ and $failures_{ttmt,t}$ have the analogue meaning.

Let $riskset_t$ be the cardinality of the living populations of both groups, defined by Equation 6.5. Similarly, the number of the events at time t is $failures_t$ defined by Equation 6.6.

$$riskset_t \equiv riskset_{ctrl,t} + riskset_{ttmt,t} \tag{6.5}$$

$$failures_t \equiv failures_{ttmt,t} + failures_{ctrl,t} \tag{6.6}$$

We then define:

Definition 6.1 (Null hypothesis of the log-rank test) *There is no difference between two treatments with respect to event characteristics.*

The task is then to compute the probability for the obtained experiment results under this assumption. Given equal event characteristics, the number of events to be expected in the treatment group is $expected_{ttmt,t}$, defined as:

$$expected_{ttmt,t} \equiv failures_t * (\frac{riskset_{ttmt,t}}{riskset_t}). \tag{6.7}$$

Let

$$O_{grp} \equiv \sum_{t \in T} failure_{grp,t} \tag{6.8}$$

denote the total number of observed events in group grp and let

$$E_{grp} \equiv \sum_{t \in T} expected_{grp,t} \tag{6.9}$$

denote the total number of expected events in group grp.

It then follows that

$$O_{ttmt} - E_{ttmt} \equiv \sum_{t \in T} (failure_{ttmt,t} - expected_{ttmt,t}) \tag{6.10}$$

$$Var(O - E) \equiv \sum_{t \in T} failures_t * (riskset_t - failures_t) * \frac{riskset_{ttmt,t}}{riskset_t} * \frac{riskset_{ctrl,t}}{riskset_t} * \frac{1}{riskset_t - 1} \tag{6.11}$$

If the hypothesis in Definition 6.1 is correct, then

$$Z \equiv \frac{(O_{ttmt} - E_{ttmt})^2}{Var(O - E)} \tag{6.12}$$

will be χ^2-distributed with one degree of freedom [Man66, PP72].

We can therefore use Z as the log-rank statistics and obtain a P value by a lookup in a χ^2-table.

6.1.3 Basic Algorithm

In order to realize the aforementioned computations in software, a simple algorithm can be defined in a straightforward manner. We present it in Listing 6.1.

We assume that the whole data set from the executed study is available on a single host. The data possesses the structure as shown in Table 6.1 and is made available as a list of records called `entries`. For each entry of this table, $expected_{ttmt,t}$ is computed and the sums of Equations 6.8, 6.9 and 6.11 are built iteration-wise. In the last step, Equation 6.12 is calculated. The algorithm returns the log-rank value which allows the lookup in a χ^2-table.

```
expected_failure = 0
variance = 0
failures_treatment = 0
for entry in entries:
    failures = entry.failure_treatment + entry.failure_control
    risk_set = entry.riskset_treatment + entry.riskset_control

    e = entry.riskset_treatment * failures / risk_set
    v = failures * (risk_set - failures) * entry.riskset_treatment * entry.riskset_control
        / (risk_set * risk_set * (risk_set - 1))

    expected_failure += e
    variance += v
    failures_treatment += entry.failure_treatment

diff = failures_treatment - expected_failure
log_rank = diff*diff/variance
return log_rank
```

Listing 6.1: Kaplan–Meier Estimation with Log-Rank Test

6.2 Cooperative Evaluation of Partitioned Data Sets

In the previous section we showed how the Kaplan–Meier estimator in combination with the log-rank test can be applied if only having a single party, i.e., a single source of data. Data protection and privacy considerations come up when multiple parties perform a study in a separated manner, and the gained data is partitioned among them. This is the case in so-called *multi-centric studies*. These studies are seen to be especially reliable and significant due to the markedly higher sample size.

Being able to evaluate multi-centric studies without having to share the data among the conducting institutions is desirable due to several reasons: input data consists of patient information which is considered to be personally relatable and highly critical. In the data protection legislation, personal data of the health care sector is considered to have specially high need for protection. Collaborative processing then requires a solid legal foundation which is normally addressed by organizational means. Contracts between all parties are created which enable data sharing and the written informed consent of the study participants is acquired that their data may be shared with other research institutions. This imposes a high organization overhead to the research institutions since the written informed consent has to be collected individually from each participant—in many cases as a hardcopy agreement which is signed by hand.

In another common scenario, obtaining agreements is even harder: after studies have been carried out, institutions retain the original study data for several years. During that time, data could be used for further analysis as long as data protections policies are taken into account. A typical case is the execution of a meta-study or meta-analysis which aims to perform statistical analysis based on previously performed studies. While it is possible to base the meta-study on the aggregated results of previous studies, it is of higher quality and precision to perform them directly based on the Individual Participant Data (IPD). Theses studies naturally focus on the combination of data from formerly independent studies.

However, since the study was already finished arbitrary time earlier, contact to the participants might be lost or their involvement or interest in the study already declined so that another request for sharing consent would be rejected or not answered. If legislation recognizes that secure computation techniques do not share data in a classic sense and that it preserves the data protection requirements on the technical level without need for another consent, SMC is able to make these scenarios practically possible without the mentioned organizational overhead and the corresponding risks.

We model our concrete use case as follows: there is a set of institutions, called *parties* $p \in P$ with n = |P|. They have a common study object or research question. Each party carries out survival analysis individually. The obtained data is assumed to be structurally and semantically compatible with respect to the computation of the Kaplan–Meier estimator and the log-rank test. Concretely, every party collects input data which exhibits the structure of Table 6.1. For simplicity, we assume that the categories of study groups (*ttmt*, *ctrl*) are identical among all parties. Formally, each party $p \in P$ holds data as a map $entries_p$, where $\forall t \in T_p : entry_{t,p} \in entries_p \equiv t \rightarrow (riskset_{ttmt,t,p}, riskset_{ctrl,t,p}, failures_{ttmt,t,p}, failures_{ctrl,t,p})$ Here, $\rightarrow$ denotes the key-value mapping, the key set of a map is defined to be $\mathsf{keys}(entries)$, the values as $\mathsf{values}(entries)$ and the value of key t $\mathsf{value}(entries, t)$ accordingly. The set T_p denotes all times of the study where institution p recorded some events in one or more of their groups. For each T_p, we only assume the requirements for the Kaplan–Meier estimator, i.e., uniqueness. For each pair of parties p and p', however, their time values T_p and $T_{p'}$ may be completely identical, have a non-empty intersection or be disjoint.

The goal is to compute a Kaplan–Meier estimator and assess it with the log-rank test among the merged available data.

6.3 Secure Implementation of the Kaplan–Meier Log-Rank Algorithm

We showed the plain, centralized and non-privacy-preserving algorithm in Listing 6.1. The algorithm uses exclusively basic arithmetic operations, i.e., addition, subtraction, multiplication and division. While it uses loops for iteration, there are no conditionals nor any comparison operations. This facilitates the creation of a secure implementation.

However, the algorithm assumes the existence of a single data structure *entries*, which does not yet exist in our distributed setting. It is, hence, a first necessary step to securely combine the initially distributed data into a single data set without leaking any information to the parties.

Merging initially distributed data

Simply generating $entries : \cup_{p \in P} entries_p$ is not expedient, since it does not handle duplicate keys in the entries. Firstly, these have to be eliminated to fulfill the uniqueness requirement and their values have to be merged in an adequate manner. In order to create a single, combined, virtual study out of the distributed studies, it is actually sufficient to realize the merge by summing up corresponding values of matching keys.

Let

$$\mathsf{keys}(entries) \equiv \cup_{p \in P} \mathsf{keys}(entries_p) \tag{6.13}$$

Furthermore, we define an auxiliary function $\mathsf{value\_or\_default}(entries, t)$ as

$$\mathsf{value\_or\_default}(entries, t) \equiv \begin{cases} \mathsf{value}(entries, t) & \text{if } t \in \mathsf{keys}(entries) \\ (riskset_{ttmt,t_{prev}}, riskset_{ctrl,t_{prev}}, 0, 0) & \text{otherwise} \end{cases} \tag{6.14}$$

where $t_{prev} \equiv \mathsf{max}(\{t' \in \mathsf{keys}(entries) : t' < t\})$.
We use this to obtain the individual values per party and row:

$$\begin{aligned} &(riskset_{ttmt,t,p}, riskset_{ctrl,t,p}, \\ &failures_{ttmt,t,p}, failures_{ctrl,t,p}) = \mathsf{value\_or\_default}(entries_p, t) \end{aligned} \tag{6.15}$$

time	Risk set		Failures	
	treatment	control	treatment	control
t	$\sum_{p \in P} riskset_{ttmt,t,p}$	$\sum_{p \in P} riskset_{ctrl,t,p}$	$\sum_{p \in P} failures_{ttmt,t,p}$	$\sum_{p \in P} failures_{ctrl,t,p}$

Table 6.2: Merged data table. If multiple parties provide partial study results, they are merged in to a single data table like the one shown in Figure 6.1.

The variables contain the corresponding values of a single row in the data of a single party. If the corresponding key is not present in the data set, a fallback value is used. It consists of the risksets of the last present time t_{prev} without decrease, and zero occurred failures. These values can then be used to perform the actual merging step by summation. By doing so, we obtain a merged data table as depicted in Table 6.2.

Performing this merge step should not leak any unnecessary information. This means that no values of each party should be shared with any other party. Additionally, the mapping $p \rightarrow \mathsf{keys}(entries_p)$ for any $p \in P$ should remain private in the general[1] case. The only intermediate information which is made available for all parties is the set $\mathsf{keys}(entries)$. The gained knowledge of party p by this intermediate result about the presence of a key t is then reduced to

$$\mathsf{if}\ t \in \mathsf{keys}(entries_p)\ \mathsf{then}\ \bot\ \mathsf{else}\ \exists p' \in P \setminus \{p\} : t \in \mathsf{keys}(entries_{p'}) \tag{6.16}$$

The merged table *entries* itself will only be available in a secret-shared manner, not accessible in plain, and directly fed into the computation of the log-rank test.

We first address secure realization of Equation 6.13. This is effectively a basic secure union set operation for which we apply the algorithm provided by [BA12]. The complete algorithm works on shares, no opening is necessary. That means that all entry keys are exclusively processed in a secret-shared manner, the only plaintext values are the indexes when iterating over the `keys` array.

```
1 keys = sort([entry in keys(entries_p1)...keys(entries_pn)])
2 size = length(keys)
3 c = []
4 for i = 1 to size - 1 do in parallel:
5   x_i = equals(keys[i], keys[i+1])
6   c[i] = mult(keys[i], sub(1, x_i))
7 c[size] = keys[size]
8 return sort(c)
```

Listing 6.2: Secure Union Set Protocol by [BA12]

The algorithm in Listing 6.2 works as follows: in line 1 the entries of all peers are obtained and stored in a single list, then the list is sorted. It is a technical necessity to sort the list: afterwards, duplicate entries are located directly succeeding each other which enables the following approach. For all elements beginning at the first and ending at the second last one (!), in line 5 equality of the current element and its successor is checked. If they are equal, `x_i` is a secret-shared `1`, otherwise `0`. Then, in line 6, the current element `key[i]` is multiplied by the binary inverse of x_i. If equality was true, `mult(keys[i], sub(1, x_i)) == 0`, otherwise it is `keys[i]`[2]. In other words, the element stays the same if the successor is different, otherwise it is nullified. This can be done in parallel, since the computation does not happen in situ but fills the new list `c`. The last element

[1] Special cases like n = 2 will allow derivation of further information.

[2] This is the realization of a conditional in arithmetic circuits.

is guaranteed to be unique, hence it is unconditionally copied (line 7). The second sort (line 8) is not a technical necessity, strictly speaking. Instead, it is required to prevent information leakage: the location of the nullified entries is still unchanged, and it carries enough information to allow derivation of its former value. This in turn would leak the information how many parties have provided the value in question. Sorting eliminates this leakage by moving all nullified entries to the front of the list. Afterwards, only the overall number of zeros is known. This information constitutes no leakage since this number could also be derived by $|keys| - |c\backslash\{0\}|$. These two values are necessarily known to each party during the computation. The obtained sorted list `c` is then opened for each party.

Since keys(*entries*) is available in plain, Equation 6.14 and 6.15 are easily realized for each party p by a map filled with $entries_p$ and the necessary fallback values generated on the fly. Table 6.2 is then obtained by iterating over the sorted keys(*entries*), while each party provides its inputs for the four columns. Together they perform a secure addition of their values row by row. Since the rows are independent, this can be done in parallel. The result is the merged data table in a secret-shared manner. I.e., their open values do not become accessible by the parties but the application of the log-rank test on this data becomes possible.

Computation of the Log-Rank Test

The computation of the log-rank test exclusively consists of arithmetic operations. It is therefore straightforward to realize, since it does not make any bigger changes necessary to represent it as an arithmetic circuit. Since we already presented the centralized algorithm in Listing 6.1, we only point out the differences of the secure implementation shown in Listing 6.3. In this algorithm, all values from the `entry` record are represented as shares and not accessible in plain.

The structure of the algorithm shows that there are two cases of computation: if the current computation depends on previous secret-shared intermediary results, or if it does not. In the first case, sequential execution is necessary in order to ensure that intermediary results are available before usage. In the second case, parallel execution is possible. The algorithm begins in a sequential mode (line 16) to ensure correct execution of the method `log_rank`. Computations on entries are independent of each other and, per entry, also the execution of `computeExpected` and `computeVariance` are independent. Hence, they are performed for all entries in parallel (line 20). Within these methods, computations fully depend on the corresponding previous steps, hence, these are performed sequentially (lines 2 and 9). Also the summation methods for lists (line 26–28) are optimized for parallel execution. In line 12, we alternate multiplications and divisions. Otherwise, arithmetic overflows are likely to occur.

```
1  computeExpected(entry):
2    compute in sequence:
3      failures = add(entry.failure_treatment, entry.failure_control)
4      risk_set = add(entry.riskset_treatment, entry.riskset_control)
5      e = div(mult(entry.riskset_treatment, failures), risk_set)
6      return e
7
8  computeVariance(entry):
9    compute in sequence:
10     failures = add(entry.failure_treatment, entry.failure_control)
11     risk_set = add(entry.riskset_treatment, entry.riskset_control)
12     v = div(mult(div(mult(div(mult(failures, sub(risk_set, failures)), risk_set), entry.
           riskset_treatment), risk_set), entry.riskset_control), sub(risk_set, 1))
13     return v
14
15 log_rank(entries):
```

```
16  compute in sequence:
17    expected = []
18    variances = []
19    ftEntries = []
20      for entry in entries do in parallel:
21        expected << computeExpected(entry)
22        variances << computeVariance(entry)
23        ftEntries << entry.failure_treatment
24
25    compute in parallel:
26      expected_failure = sum(expected)
27      failures_treatment = sum(ftEntries)
28      variance = sum(variances)
29
30    diff = sub(expected_failure, failures_treatment)
31    logRank = div(mult(diff, diff), variance)
32    return open(logRank)
```

Listing 6.3: Secure Kaplan–Meier Estimation with Log-Rank Test

6.4 Performance Evaluation

We carry out a performance evaluation of our secure implementation and assess the scalability of such a solution. In particular, we aim to answer the following questions:

1. How does the solution depend of the input data and setting, i.e., number of entries per peer and number of peers?
2. What are the costs for the user in terms of time and resources?
3. What is the typical bottleneck?
4. Can the time costs be positively influenced by hardware improvements?

6.4.1 Measurement Setup

We describe our measurement setup in the following. Since an overview of the setup has already been given in Chapter 4, we only outline the differences to the previous setup.

Testbed

For our test measurements we use the same testbed hardware and topology as described in Section 4.4. The software is similar to the one described in Section 4.5 with the following changes: the operating system has been updated to Debian Stretch 9.5 using a kernel as shown in Listing 6.4.

```
1 Linux pc1 4.9.0-8-amd64 #1 SMP Debian 4.9.110-3+deb9u6 (2018-10-08) x86_64 GNU/Linux)
```

Listing 6.4: Host System Testbed

Java is used in version 11.0.1 2018-10-16 LTS. The version of Fresco is 1.1.2 with minor local fixes. Most importantly, this version is not using the BGW [BOGW88] protocol suite anymore, but SPDZ as presented in [DPSZ12, DKL+13]. At the time of our measurements, only a stable realization of the online phase was available in Fresco. The offline phase is simulated by a dummy preprocessing. The performance of the offline phase is hence not considered by our tests. The performance characteristics of the online phase are notwithstanding realistic as if real preprocessing had been performed. The authors of Fresco confirmed that upon our request [GHD18]. Tshark is of version 2.6.5 and perf of version 4.9.130.

Real-World Setup

In our real-world setup, we cooperated with the University Hospital of Ludwig-Maximilians-Universität München (LMU) and Charité Berlin (CB).

Within the data umbrella of DKTK[3] of which Technical University of Munich, LMU and CB are members, the radiation oncology departments of LMU and CB were able to provide glioblastoma survival data (cf. [NAK+18]). Both, the patient data of LMU and CB contained 96 input entries each.

Additionally, LMU and CB each provided a server for executing our secure protocols. The server of LMU is equipped with an Intel Xeon Silver 4112 CPU, having eight cores at 2.60 GHz and a cache size of 8,448 KB. It possesses 128,476 MB of RAM and a 1 Gbit networking interface. It provides Debian 9.6 as operating system using a 4.9 Linux kernel (cf. Listing 6.5).

```
1 Linux lmu 4.9.0-8-amd64 #1 SMP Debian 4.9.130-2 (2018-10-27) x86_64 GNU/Linux
```

Listing 6.5: Host System LMU

We used Java 11.0.2 2018-10-16, perf 4.9.130 and tshark 2.6.5.

The server of CB uses has a Intel Xeon CPU E5-2695 v3 CPU, with two cores at 2.30GHz and a cache size of 35,840 KB. It possesses 3,945 MB of RAM and a 10 Gbit networking interface. The host is a VM based on VMWare. It provides Ubuntu 18.04.2 LTS as operating system using a 4.15 Linux kernel (cf. Listing 6.6)

```
1 Linux charite 4.15.0-46-generic #49-Ubuntu SMP Wed Feb 6 09:33:07 UTC 2019 x86_64 x86_64
      x86_64 GNU/Linux
```

Listing 6.6: Host System CB

We used Java 11.0.2 2018-10-16, perf 4.15.18 and tshark 2.6.6.

6.4.2 Method

In order to assess the following results, we provide two measures of comparison: firstly, we also implemented the log-rank algorithm insecurely to be carried out on a central server, acting as a trusted third party; for the measurements we used the LMU server. Here, a standard Java implementation of the computation has been used. In this case, we only consider the computation itself without network interaction for providing the input data to the server nor for sending the result to any recipient.

Secondly, Fresco also provides a dummy protocol suite which performs the computation in plain text without execution of secure protocols. The algorithm in question is translated into a circuit representation, but computation is then carried out locally without protocol interaction and corresponding communication. This allows to discern the influence of the circuit representation from the actual execution of interactive, synchronized multiparty protocols.

We refer to these baselines where appropriate, but do not interpret their performance behavior in greater detail.

6.4.3 Results

Fresco provides the internal measure of protocol invocations which represents a number of primitive operations to be performed during algorithm execution. This number provides an estimate of the complexity of the algorithm currently carried out.

[3]Deutsches Konsortium für Translationale Krebsforschung, German Cancer Consortium

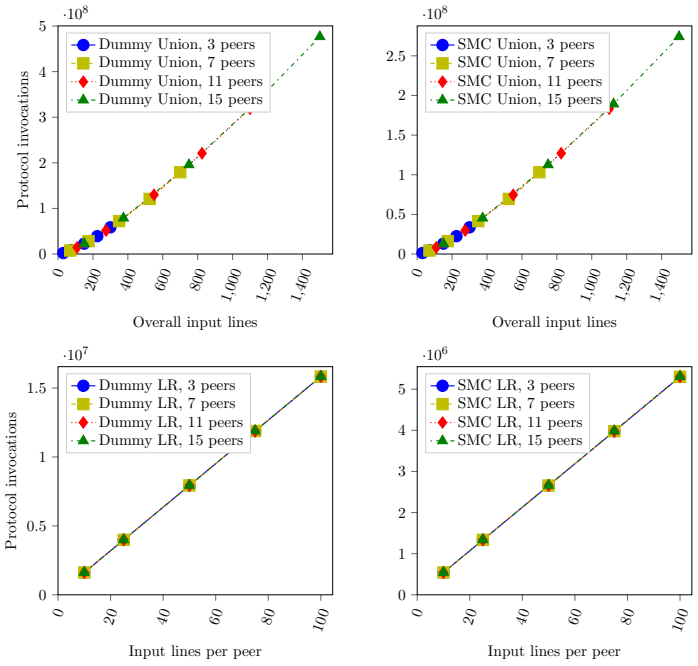

Figure 6.2: Protocol invocations depending on the lines and peers. The union algorithm is plotted depending on the overall number of input lines $n * m$. The log-rank algorithm, in contrast, is linear in m and its complexity is independent of the number of peers.

The algorithm is represented as a dependency graph of operations in Fresco. Operations can only be carried out if all previous operations have already been executed successfully. Building the dependency graph and taking care of meeting all dependencies during execution is performed manually and a task of the implementer. Exploiting independencies of operations enables Fresco to perform some computations in parallel. This improves CPU and network utilization, and in consequence leads to shorter execution times. Fresco realizes parallelization by creating *batches* of primitive operations. Each batch is executed in parallel. After all computations of a batch are carried out, the next batch is undertaken.

Closing the input values (i.e., creating shares) are the initial operations to be carried out. Being the first actions to be performed, they have no dependencies on other operations. Furthermore, they have no dependencies on each other. Due to this reason, these can always be parallelized. There are further situations in our context where we can take advantage of parallelization. We showed them in the algorithm in Listing 6.2 and 6.3, denoted by the keyword `in parallel`.

Complexity of Computation

We use the protocol invocations and batches as a first, internal indicator. Figure 6.2 compares the dummy algorithm with the secure implementation. It depicts the union algorithm and the actual log-rank computation separately. As parameters, we consider the number n of participating parties, and the number m of input values per party.

We see that the union and the log-rank algorithm seemingly differ in the order of magnitude regarding protocol invocations. However, we have to consider that only the initial merge step of the log-rank algorithm depends on $m * n$. The remaining steps only depend on n. The reason in our tests is that all participants used the same keys for their inputs, the merge then combines $n * m$ entries to m entries, eliminating the factor n. I.e., we can directly compare the number of protocol invocations of the union algorithm depending on

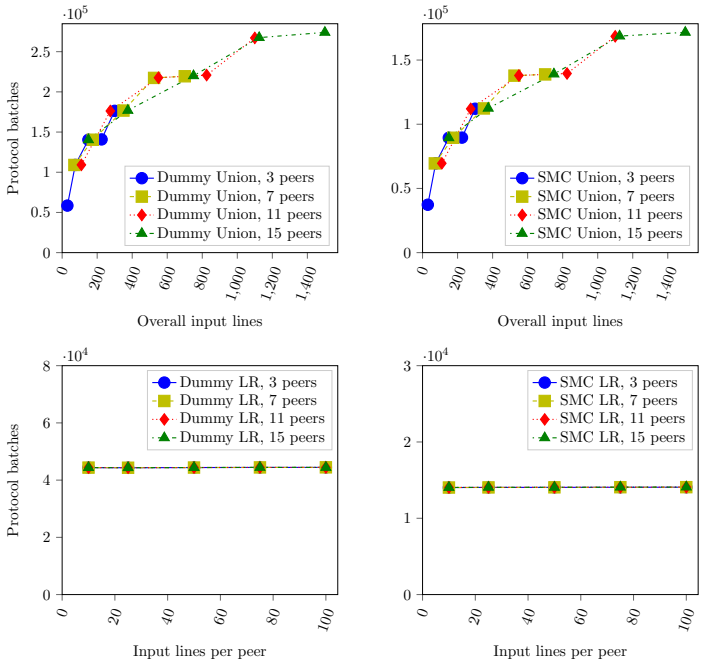

Figure 6.3: Batches depending on the lines and peers. When increasing the overall number of input lines, the sorting network becomes bigger, leading to more sequential computation batches. On contrast, the log-rank algorithm is optimally implemented so that the number of sequential steps becomes independent from the number of input lines.

$n * m$ with the protocol invocations of the log-rank algorithm depending on m. Both then range in the interval between 10^6 and 10^7 invocations.

In other words, the scaling of the union algorithm depends on the number of values it has to sort, while the log-rank algorithm scales with the number of unique keys it has after the merge step. Sorting was performed by the batcher's merge exchange sorting algorithm provided with Fresco. The sorting phases dominate the increase of invocations while the elimination is practically constant in our cases due to parallelization.

Due to this reason we plot the graphs for the union algorithm and the log-rank algorithm differently. The former is always shown with the overall number of entries $n * m$ on the x-axis while we will solely use the number m of input lines per peer for the latter if the data implies that this is the dominating dependency.

We see that protocol invocations are a helpful heuristic to approximate algorithm complexity. All composed functions like boolean operations and division likewise are already decomposed into native operations (open, close, add, multiply, ...). However, protocol invocations do not take into account different levels of (communication) complexity of the provided native operations.

Fresco builds batches from computations which are independent and, hence, can be executed in parallel. We see that this is effective (cf. Figure 6.3): the increase of the union algorithm becomes roughly linear and dependency of m is now completely removed for the log-rank computation. The absolute number from protocol invocations to batches is reduced by factor 100–1000.

Variation of Input Parameters

Comparing all three realizations of the algorithms with respect to execution time, we see that their orders of magnitude differ notably: the TTP variant costs milliseconds, the

peers	algorithm input lines	protocol invocations original	protocol invocations replaced	batches original	batches replaced	duration original	duration replaced
3	10	541802	9681	14026	362	7.452	0.131
	25	1334650	23856	14039	365	40.902	0.271
	50	2655997	47481	14043	369	41.251	0.283
	75	3977628	71106	14082	374	57.002	0.580
	100	5298722	94731	14068	375	117.215	0.611
7	10	542080	10001	14026	366	19.297	0.488
	25	1335500	24656	14038	369	45.040	0.513
	50	2657644	49081	14063	373	76.974	0.639
	75	3979801	73506	14074	378	120.186	0.854
	100	5302291	97931	14063	379	191.506	1.285

Table 6.3: Comparison of the original log-rank algorithm with a variant where all division operations have been replaced by multiplication operations. The impact on the number of protocol invocations, batches, and consequently the execution time is very strong.

dummy protocol suite is in the order of seconds and the secure variant in the order of minutes (Figure 6.4). In comparison to Figure 6.5, we see the CPU time only constitutes between 30 and 50 % of the overall execution duration. In absolute numbers, the union algorithm and the log-rank computation perform equally fast if we take the considerations of the previous paragraph into account.

Inspection of the log-rank algorithm provided further insights: it showed that the division operation has a tremendously higher impact than any other basic arithmetic operation. The source code of Fresco states that the Goldschmidt division [Gol64] is used; an approach which iteratively applies multiplications (!) until convergence of the result is reached.[4] In order to provide a quantitative statement on the impact, we performed punctual measurements for comparison. For that, we replaced all divisions in the log-rank algorithm with multiplications. Being aware that the computational results become useless, this method is nevertheless a representative substitute to assess the difference in performance. The reason is that the computations are data oblivious, i. e., the intermediate results do not influence the behavior nor the performance characteristics of the algorithm. The multiplication operation was chosen since addition and subtraction are "free", they do not cost a round of communication. The results are shown in Table 6.3. The number of protocol invocations is reduced by factor ∼55, batches by factor ∼40, and duration by factor ∼50–200. We expect that the gap gets even larger with increasing number of peers.

It is furthermore interesting to see whether the sole number of peers also has an impact on execution duration. For that, we analyzed the dependency of the algorithms on the overall number of input lines. A linear regression on the data yielded the following insights: for the union algorithm, the formula

$$0.52s + 0.28s * |\text{peers}| = \frac{\text{execution duration}}{|\text{input entry}|} \tag{6.17}$$

holds. For the log-rank algorithm we identified a slope of ≈ 0. This is in line with our observations in Figure 6.4: The spread between the different configurations in the latency diagram for the SMC implementation of the log-rank algorithm can be exclusively explained by the fact that additional peers also add further input entries.

[4] For further considerations of the division operation in SMC and Fresco in particular cf. [DDN+17]. For further explanation on the application of the Goldschmidt division for SMC cf. [BNTW12].

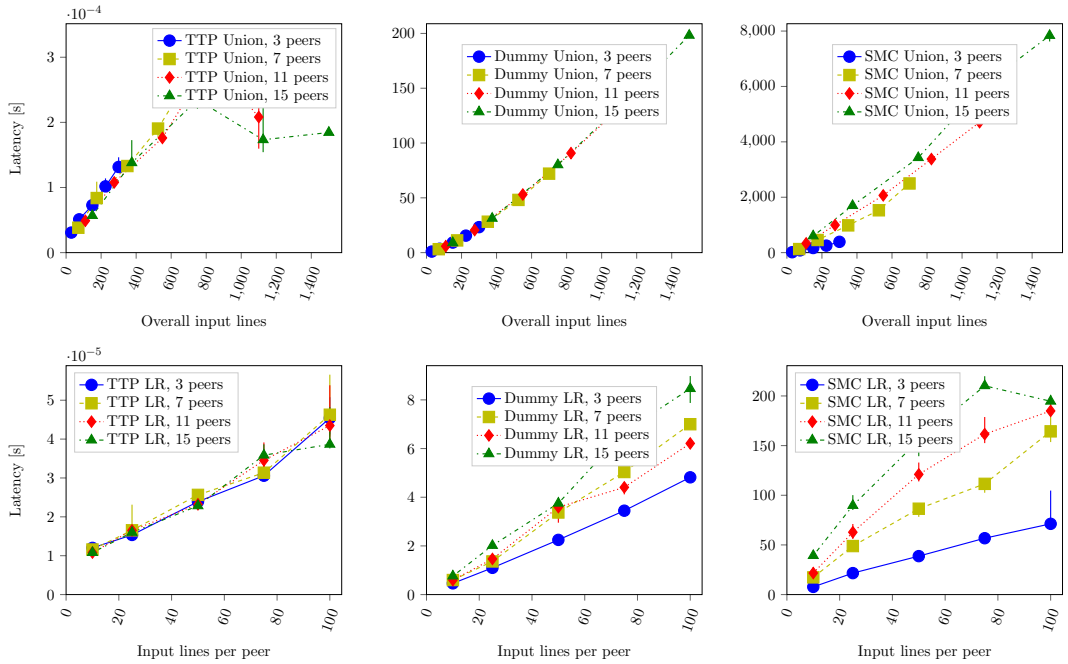

Figure 6.4: Time depending on the lines and peers. The computation time between a TTP, a dummy implementation and real SMC varies by orders of magnitudes. The reason is the increasing amount of network exchange which becomes necessary with secure computation. It becomes visible that the SMC algorithms mainly depend on the overall number of input lines. The number of peers itself has only a subordinate influence. This matches our expectations since communication to all peers can be parallelized.

In other words, time of the union algorithm is mainly influenced by the overall number of input lines $n * m$, notwithstanding whether many peers input few lines or few peers input many lines. Merely, small differences between the peer configurations are observable which have not been present when only examining the number of protocol invocations (cf. Figure 6.2). I.e., the small communication overhead of Equation 6.17 becomes visible.

In Figure 6.5, we can also see that the CPU time proportionally corresponds to the overall execution time. The explanation for different peer configurations is as said before. The merge step is performed in $\mathcal{O}(log\ n)$, hence, the lines in Figure 6.5 initially spread stronger and converge against the same slope.

In Figure 6.6, we depict the transmitted data between a single pair of hosts. Compared with the dummy protocol, the SMC implementation again differs by orders of magnitude. The reason is that computation in plain (as given in the dummy implementation) is able to do some computations (especially the basic multiplication) without any communication, while for SMC exchange is necessary every time such an operation takes place. We stress that the data shown in the graphs is the communication of a single pair. There are n^2 such pairs during each computation, the overall amount of transmitted data over the network increases accordingly. We could verify by inspection that the amount of transmitted data is equally sized for every pair. Furthermore, the majority of packets has a size of around 200 Bytes, independent from the number of peers or input lines.

We already elaborated that the log-rank algorithm is made independent of the number of peers by the initial data merging step. This is also confirmed by these measurements which show that the amount of transmitted data does not depend on the number of participating peers.

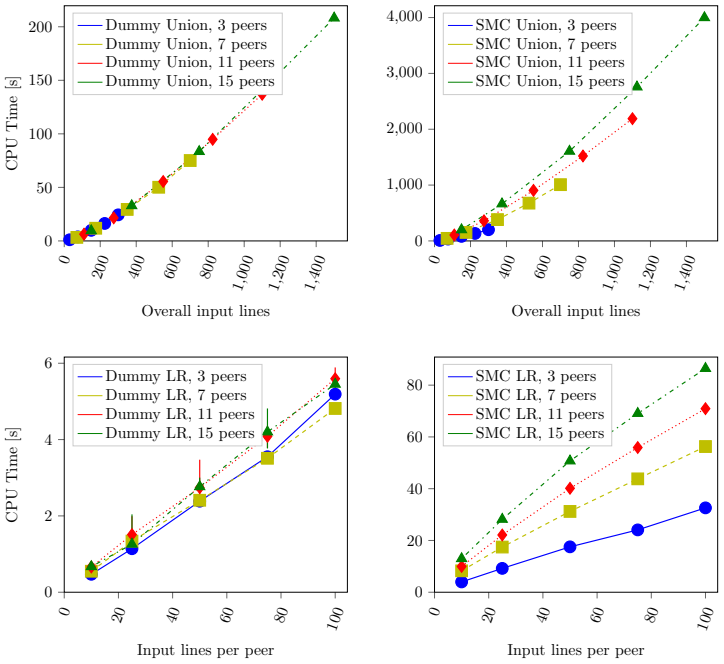

Figure 6.5: CPU time depending on the lines and peers. We can also see that the CPU is moderately more utilized when having more participating peers. The reason are the steps necessary to manage and perform communication with other peers (notwithstanding the communication delay itself).

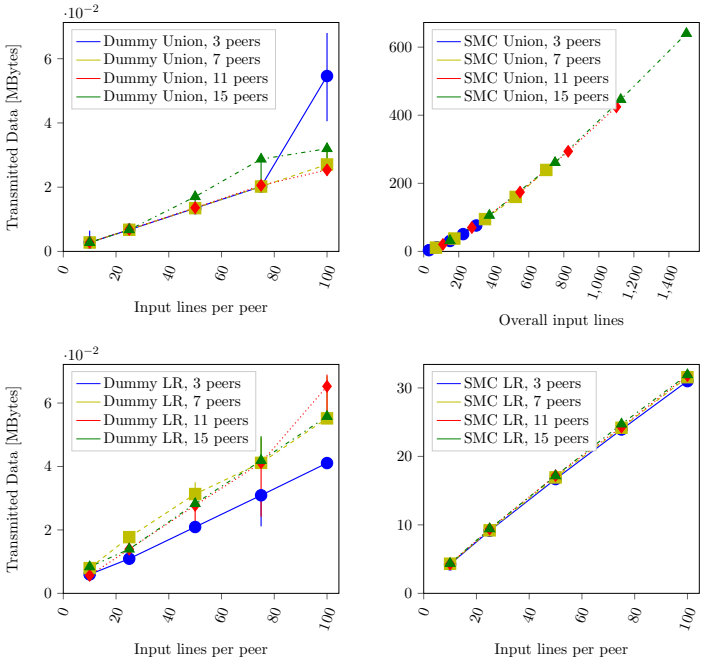

Figure 6.6: Transmitted data depending on the lines and peers. The graph depicts the amount of MBytes transferred between a single pair of hosts in the network. In the SMC case, they nearly perfectly correlate to the amount of protocol invocations.

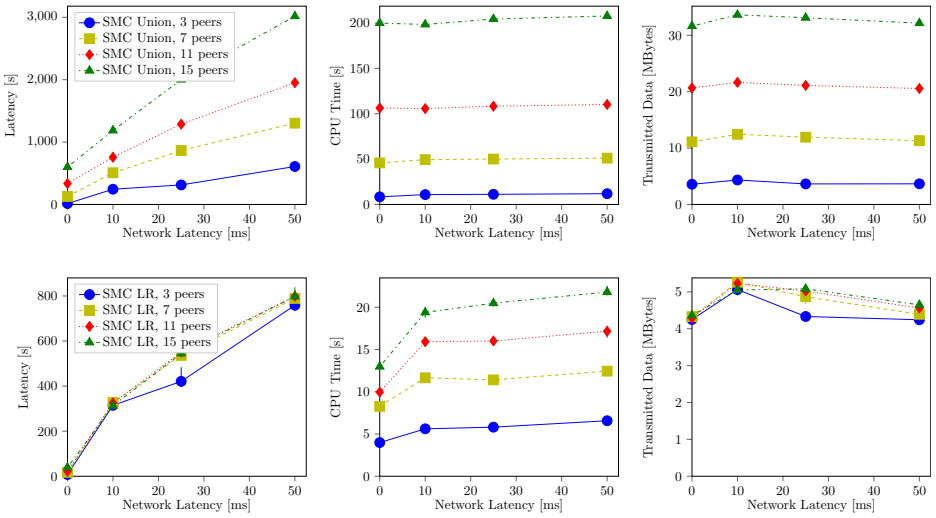

Figure 6.7: Influences of network latency manipulation. The upper row shows the union algorithm, the lower row the log-rank algorithm. It becomes clear, how network latency influences the overall execution time while neither changing CPU time nor the amount of packets transmitted.

Variation of Resource Parameters

After we analyzed the basic behavior when scaling environment parameters like the number of input lines and the number of peers, we now address the technical parameters of the setup. This encompasses the network latency, the transmission rate and the cores and frequency of the CPUs used. We still vary the number of peers and set the number of input lines per peer to 10.

Network Latency Figure 6.7 demonstrates the influence of increased packet delay on the computation. We already showed in Figure 6.6 that more data is transmitted during the union algorithm than the log-rank algorithm. Furthermore, we could find that for a majority the packet size stays roughly the same notwithstanding the variations of parameters. Hence, with a rather constant number of packets, it is expectable that packet delay influences the union algorithm correspondingly stronger than the log-rank computation. The slight variations in the amount of transmitted data over the different network latencies can be explained by variation in the average packet size. With a latency of 10 ms the packet size is roughly 80 to 100 Bytes smaller. To transport the same amount of payload, more packets are needed. This yields an increase of transferred headers which in turn causes an increase of the overall data transmitted.

The CPU time is not influenced by the packet delay; it stays completely constant for the union algorithm and only varies slightly for the log-rank computation.

Transmission Rate The influence of the transmission rate on the execution duration is similar to the findings of Chapter 4: transmission rate only inhibits the computation if it is under 10 Mbit/s. More specific inspection of the network traces shows that our use case continuously uses around 2 Mbit/s with short but high peaks during the log-rank computation. The union algorithm is characterized by a rather consistent stream of packets of the named rate.

CPU frequency and number of cores We varied the number of cores between 1 and 8, the frequency could be changed from 100 %, i.e., 2.5 GHz down to 50 %, i.e., 1.25 GHz. A lower value was not possible with out test machines. Notwithstanding a varying number of peers, the changes did not yield any significant influence on the execution duration of the algorithms. We conclude that the CPU does not constitute the bottleneck in our test setting.

6.4.4 Real-World Experiments

In the real-world experiments, we executed our protocol between two servers from different research institutions (cf. Section 6.4.1). The distance between both servers is around 500 km and the protocol was conducted via the open Internet.

The round-trip time starting at LMU is around 9 ms, from CB it is 21 ms. The transmission speed (approximated with netcat) is around 800 Mbps from LMU to CB and 100 Mbps from CB to LMU.

Due to the predetermined setup, we did not vary most of the parameters as we did in the testbed. We only changed the number of input entries per peer from 10 to 96. For the union algorithm, this caused computations duration between 80 and 577 seconds, and for the log-rank algorithm 182 to 652 seconds.

These numbers are highly influenced by the network latency between both hosts. This outweighs the small number of participants. This interpretation is also supported by the small percentage of CPU time. For the union algorithm, we range between 3 % to 9 %; for the log-rank algorithm it is even fewer, around 2 % to 5 %. The overall execution time for both algorithms lies approximately in the same range. This is in line with out previous observations, however, the effect becomes more clear in this setting. The reason is found in the different set of data used: In our synthetic data, each of the n peers had m input lines. The set of keys was identical for each peer. Therefore, the merge step (which is at the beginning of our log-rank implementation) reduced the overall number of lines by factor n and the remaining steps of the log-rank algorithm always had to compute with m lines only. On contrast, the real data used here has only a negligible amount of identical keys. This means that the log-rank algorithm always has to process roughly $m * n$ lines. This increases the time taken by the log-rank algorithm. On the other side, only having two peers reduces the interval in which we tested the union algorithm. By these two reasons, the execution times of both algorithms move into the same range.

The question arises whether the measurement results are in line with our testbed results in terms of absolute numbers: For that, we do not use the dimension of wall-clock time since we already know that it will not match due to the differences in the network latency of the used connections. Instead, the number of protocol invocations and the amount of transferred Mbytes are expedient characteristics for comparison due to their independence from time.

In order to obtain a valid comparison, we had to rescale the results: For the union algorithm, we always consider the overall number of input lines by multiplying the input per peer with the number of peers. For the log-rank algorithm, we made a case differentiation: From the testbed measurements, we chose the results by the number m of inputs per peer (since the merge step reduces all $n * m$ inputs to effectively m lines). From the real-world measurements, we directly considered the product $n * m$ since the merge step does not reduce the input here. Tables 6.4 and 6.5 list the chosen results from the testbed and the real-world setting. They represent the median values of the corresponding measurements. We can see that the real-world results fall between the results from the testbed within an expectable range of precision. This is true for the union algorithm and the log-rank algorithm likewise.

Setting	input lines	Protocol invocations	MBytes
Testbed	75	4954207	12.279469
Real-World	100	7382400	16.681843
Testbed	150	13121941	30.64034
Real-World	150	13121996	29.262298
Testbed	175	16236258	37.808268
Real-World	200	18163340	39.410625
Testbed	225	22505986	50.786938

Table 6.4: Comparison of the testbed and the real-world measurement results for the union algorithm (median values). The input lines refer to the overall number from the whole set of participants. The real-world results plausibly fall between the results of the testbed.

Setting	input lines	Protocol invocations	MBytes
Testbed	50	2659443	16.929702
Real-World	50	2497182	14.860479
Testbed	100	5302094	31.7901285
Real-World	100	5033945	28.212929

Table 6.5: Comparison of the testbed and the real-world measurement results for the log-rank algorithm (median values). The input lines refer to the number of lines the log-rank algorithm has to process after the merge step. The real-world results plausibly fall between the results of the testbed.

An overview of the most important results from the real-world measurements is given in Table A.1 in Appendix A.

6.5 Findings

From the creation of a secure implementation as described in the previous sections, we can gain certain insights about infrastructural premises, privacy implications and the performance of such a solution. This helps us to sketch fundamental requirements for an SMC service architecture as we will develop in the next chapters.

6.5.1 Infrastructure

Algorithms as SMC are carried out by protocols between several participants over the network. This yields several implications: each participant must provide an endpoint which allows other peers to connect to. When available, these endpoints must be known to the other parties and the software must be configured to correctly access them. The intermediary network must allow connections between each pair of peers. Furthermore, participants do not only have to be accessible but also identifiable. Cryptographic identities must be available for each participant and known to the other peers. An adequate mechanism of trust establishment must be in place.

Further implications exist for the input data: it must be *initially* distributed and present on all participants. The moment, data of different stakeholders is physically or logically pooled at a single place, privacy is already weakened. It must be compatible for combination. This is valid for the semantics of the data and the syntactical representation likewise. It is best achieved if individual data collection per peer is performed using the same method. Some

mechanism for data merging must exist. It must be able to transform the data coming from multiple sources in a way that it virtually could also come from a single source.

In this setting, our computation has been carried out manually. This implies several simplifications which we cannot expect for a service architecture: only a single computation session has been carried out, the protocol always stayed the same and the data did not change either. An implementation of the protocol to be executed must be available to all parties in advance. We also agreed on the exact protocol before execution. Having an SMC service, this should be flexibly chosen at runtime. Similarly, it must be clear for all participants which data to use and whether to preprocess it beforehand or not. The computation itself has to be started synchronously on all participants. Failure of the computation is possible due to the communication over the network. If an error occurs, the protocol has to be restarted. In the best case, a monitoring of the computation as an indicator of progress is available. The outcome of the computation has to be persisted and made available to the initiators of the computation. More complex computations take a considerable amount of time. Waiting for the result should be handled in a sensible fashion, e.g., by an event or a trigger which notifies of the end of the computation.

6.5.2 Privacy

We can see that the application of SMC mainly yields data minimization, input confidentiality and unlinkability. Input data is not shared with any other party but remains completely confidential during the whole computation. This also supports data minimization. Similarly, only a single type of intermediary data is created; this is the list of all entry keys. After that, the next open information already is the final result of the computation. The result of the computation is a highly condensed statement derived from comparatively large amount of input data points. Being unable to reverse the computation to obtain the input data or at least being able to estimate the contributions of the individual parties constitutes unlinkability.

On the other hand, transparency in this manual context is completely artificial. Due to the dedicated manual setup for a single computation, transparency is implicitly given. However, this does not generalize for our anticipated service architecture. There, we have a setting where multiple participants provide different types of data and rather support sets of protocols instead of a single one. Data is not requested by a single stakeholder but by a multitude of different clients and for different purposes. In these cases, transparency yet emerges to be an important requirement. The same is true for intervenability. Only when serving a variety of data and computations to different clients for several purposes, intervenability becomes vital allowing data owners to support computations or veto against them at runtime. This is not yet achieved by SMC and requires special considerations to be realized in the anticipated architecture.

6.5.3 Performance

Compared to Chapter 4, here we investigated a use case of higher complexity. This allowed us to gain further insights on performance and scaling behavior.

We divided the developed protocol into two parts: the union algorithm performs a set union operation. It eliminates duplicates in the input data and resets its order. The log-rank algorithm performs a statistical evaluation of the initially distributed data.

This division also separates the part where mainly comparison operators are used from the part which exclusively utilizes arithmetic operations. In our case, however, both performed roughly in the same time range.

With respect to arithmetic operations, we identified that the division operation is orders of magnitude more costly than the other basic arithmetic operations. The reason is the

application of an iterative algorithm which converges to the result by performing various multiplication and some comparison operations each round. This is the single most influential performance factor in the log-rank algorithm. Replacing all divisions with multiplication for performance assessment, the execution time of the log-rank computation immediately falls below a single second and has a considerably smaller slope.

Regarding influencing factors, we found that network latency is the strongest impact. The reason is that the network communication consist of a large amount of small sized packets. This is in alignment with our findings of Chapter 4. Similarly, the transmission rate does not constitute a bottleneck if at least 10 Mbit/s are guaranteed. Manipulation of the CPU did not yield any changes. We assume that the CPU would have to be constraint to a small fraction of its normal power to achieve any effects. This has not been possible in our setups.

Concluding, we answer the initially posed questions as follows:
1) *How does the solution depend of the input data and setting, i. e., number of entries per peer and number of peers?* Algorithms realized in as SMC protocols typically depend on their number of inputs as an insecure implementation of the algorithm would. Here, two restrictions apply: while the asymptotic behavior can be identical, the absolute time can be orders of magnitude higher, depending on other factors like the network. Furthermore, "basic" operations can have different performance properties than intuitively expected. We discussed this with respect to the division operation. The number of peers constitutes a management overhead, since connections between the parties must be held and during computation data exchanges must be performed between all parties.

2) *What are the costs for the user in terms of time and resources?* The time heavily depends on the complexity of the algorithm and the selection of operations used. If the set of operations is well supported the execution time can be in the realm of milliseconds. If real numbers, division or comparison operations are used, execution time quickly exceeds seconds to minutes. Also depending on the complexity of the computation, each pair of peers exchanges at least some MBytes of traffic. This also can quickly increment to hundreds of MBytes, e. g., when sorting.

3) *What is the typical bottleneck?* The bottleneck can be clearly identified. Since the transmitted data is split into small packets, typically of ≈ 200 Bytes, the amount of exchanged packets is correspondingly high. Consequently, the network latency is the factor with the highest influence.

4) *Can the time costs be positively influenced by hardware improvements?* As a consequence of the high influence of the network, it is difficult to improve performance characteristics by hardware changes. The most obvious approach of improving the participating hosts does not address the bottleneck. However, we found that CPU time at least constituted around 30 to 50 % of the computation. Here, moderate improvements by an increased CPU frequency can be expected. On contrary, every reduction of network latency would be worthwhile.

6.6 Key Contributions of this Chapter

In this chapter, investigated a real-world use case for SMC and assessed the feasibility and performance of this technology.

Real-World Use Case We identified a real-world use case of high practical relevance. Medical studies provide an essential benefit for society. Having a large basis of test subjects improves the validity and robustness of the obtained results. In so-called multi-centric studies, this is exploited by letting several institutions carry out the

same study with different participants. The gathered data is then merged. However, data protection regulations make the combination of data from different sources more difficult in certain cases and require a notable organizational overhead to fulfill the protection requirements.

In this situation, SMC is a promising remedy which allows aggregation of the study data without actually sharing it. Practically, it fulfills the data protection requirements by technical measures.

Development of Secure Algorithm Considering this use case, we developed a full algorithm which addresses the mentioned problem. We selected the widely used Kaplan–Meier estimator from the realm of survival analysis, combined with the log-rank test. The latter assesses the significance of difference between a (medicated) treatment and a control group.

We implemented the secure protocol mostly ourselves but built upon previous work about secure set operation protocols [BA12]. The implementation of the algorithm is held in a modular form, which easily allows development of other test measures for Kaplan–Meier estimators.

Performance Measurements Having obtained a secure implementation, we conducted thorough performance measurements of this solution. Following four performance questions, we investigated the impact of peers and input data on the duration, CPU time and data transmission. Furthermore, we evaluated the impact of selected network and host parameters on the computation time and resources.

Field Study We conducted the aforementioned measurements in a synthetic testbed with homogeneous hosts which were connected via an intranet. To complement our insights and get further knowledge about SMC performance in real settings, we also performed measurements in a real-world setting with heterogeneous hardware over the Internet. For that, two medical institutions provided locally distributed servers where our solution was carried out. With them, we have been able to confirm our results from the testbed.

Part II

SMC as a Service

7. From SMC to a Privacy-Preserving Service

In Part I we analyzed the performance of SMC and exemplified its application in an important real world setting. It became clear that SMC is a promising technology for supporting specific privacy protection goals. Furthermore, we could show how practical feasibility of SMC depends on the different settings.

In following three chapters, we aim for a generic architecture which fulfills the current standard of privacy protection goals using SMC as its core technology. For this purpose, we perform a requirements analysis examining the environment of deployment as well as SMC technology and their respective implications for our goal.

Objective

Focusing on Research Questions Q4 and Q5, the goal of this chapter is as follows: we discuss the findings about SMC up to this point and assess which privacy protection goals are already fulfilled by it. On this foundation we propose a category of use cases in smart environments, for which SMC is a promising secure base technology. We sketch a solution how SMC could be employed to address the identified use cases. Comparing that solution to the current state of related work, we analyze and state the requirements of such a solution. The actual realization is presented in the two subsequent chapters.

7.1 Discussion of Previous Findings

Initially, focusing on Secure Multiparty Computation itself, we examine how well it already fulfills the desired set of goals. For that purpose, we consider our insights on SMC from Chapter 3 and the general current understanding of privacy in Chapter 2. Then, we refer to the results of the Chapters 4 and 5 in order to select an environment in which application of SMC is most promising. Afterwards, we investigate how well SMC already fulfills the architectural needs that the selected environment requires. Lastly, we discuss the current state of the art in the identified setting.

Satisfaction of Protection Goals

In Chapter 2, we established the double triad of protection goals suggested by [RP09, BR11, Han12, HJR15] as the state of the art with respect to a current privacy concept.

The classical security protection goals *confidentiality*, *integrity* and *availability* are complemented by the privacy protection goals *unlinkability*, *transparency* and *intervenability*. Additionally, *data minimization* is understood to be a fundamental goal of privacy protection.

Examining the specific functionality of SMC, we see that at least two goals are well supported by it: data minimization refers to the amount of data present in a system. Normally, raw data which has been collected in a distributed fashion is gathered in a central storage. The desired result data is then created by post-processing. If applying SMC, processing is done in collaboration of all data providers. No raw data is stored centrally but only the final result of the computation. This improves fulfillment of data minimization in two dimensions: firstly, the central storage does not become *another* component which holds the raw data. Secondly, SMC avoids the generation of plain intermediate computation results as they often occur during data processing. Instead, during computation only secure shares exist which do not give any additional insights.

Furthermore, since data is not sent one-by-one by every individual data provider to the central storage, the link between the stored data and the data providers is weakened. In other words, unlinkability is achieved since the exact contribution of every data provider cannot be derived from a result. This is more fundamental than e.g., using mix networks [Cha81] which remove the link between the source and the input value, but leave the value itself unaltered: in certain settings, this is not sufficient since the input value itself can already suggest the original source. Since SMC never makes the input value accessible to any party, this attack vector is directly mitigated. In fact, unlinkability could be made even stronger by suppressing the knowledge which exact data providers contributed to a result at all.

We identify the potential of SMC to provide transparency of data usage and processing purposes for data providers: every computation has to happen in cooperation with the original data providers. This can be used as foundation for transparency mechanisms. From this, in turn, accountability can be derived.

Similarly, as data providers do not give away their data, they can exert control over it. Rudimentarily, they can reject a computation if they see any reason to do so. The aspired computation is then ultimately prevented. This basic functionality of control can be extended and developed into more elaborated and advanced mechanisms of data usage control.

Adequacy of Settings for SMC

In Chapters 4, 5 and 6 we investigated the performance of SMC. A main insight was that network latency strongly influences the overall execution time. The reason is that a multitude of small packets is sent between the participants during a computation session. It is hence way more important to have small latency than to have a high throughput. The packet size is only in the range of hundreds of Bytes.

Regarding host computers, we identified that CPU power has only moderate influence. It is advised to use multi-core systems with a frequency of at least 1 GHz. However, hosts do not have to be high-end devices.

Based on these insights we conclude that an intranet environment with (possibly mobile) low-end hosts like Raspberry Pi computers is a promising setting. This encompasses a large set of use cases, including smart homes, smart (office) buildings and smart cities to some degree. Similarly, ubiquitous computing scenarios are also reflected.

7.2 Use Case

In this section, we derive a use case from the environment and setting explained above.

We consider a smart environment. Concretely, this can be a working space, e.g., a smart office department. It consists of rooms with different sizes and different purposes: many are offices for a single or a small group of individuals. A comparatively smaller amount constitutes rooms like restrooms, kitchens, elevators and storage rooms.

We assume a basic intranet infrastructure: a local area network connects available computers of all rooms. Wi-fi is deployed so that mobile devices can also be connected to the LAN. Depending on the location and the size, smart appliances can either be connected to the network in tethered or untethered fashion. There are central services available in the network. These are either deployed in local data centers or in the cloud. Local computers can connect to them via the intranet.

We now focus on smart building appliances. These can be generally split into the following three component types:

Sensors Sensors are distributed measurement nodes. They sense a single or multiple aspects of the environment in which they are installed. We assume that one or multiple of them are installed in the rooms of the smart environment. Hereinafter, we refer to the data directly collected by them as *raw data*. The types of collected information encompass environmental (e.g., temperature, air quality, brightness), usage (e.g., occupancy, number of people, power consumption) and the state information of actuators (e.g., HVAC control, light state, window blind state).

Data Processing Units Typically, raw data is not fit for direct usage. It has to be post-processed in order to generate insights. These can be derived by statistical means or a combination of aggregation and filtering steps. Any kind of alert generation or machine learning tasks are also a specialization of this post-processing.

Actuators Actuators are components which act upon the measured and processed data. This especially encompasses the adjustment of the appliances and the building state (e.g., regulating the HVAC). For our purposes we also include informatory devices for human interaction like public displays in this category.

There are cases where computation adheres to the locality principles. Examples are the thermostats of standard heaters, motion sensors for illumination and sliding doors. Data can be post-processed in the logical and spatial proximity of the sensor (the data source) or the actuator (the data sink). In these cases, privacy is naturally supported, since local storage supports unlinkability and automatically minimizes the locations where this data is handled. Similarly, these locally bounded devices can be easier understood (transparency) and controlled (intervenability).

However, there is a class of applications where the locality principle is not trivially applicable. It may be necessary to provide inputs from multiple sensors. This raw data is then aggregated (by computations like the average, standard deviation or more complex functions) and post-processed in order to obtain an evaluation over a group of selected sub-environments like rooms of a certain type. Typical examples are the average and standard deviation of the temperature in a whole floor, the power consumption of a department or the overall number of people inside a building. These computations should be available as a typical service in the smart environment and usable by clients in need of this information. In consequence, the computation should be performed automatically, without manual intervention and on a regular basis.

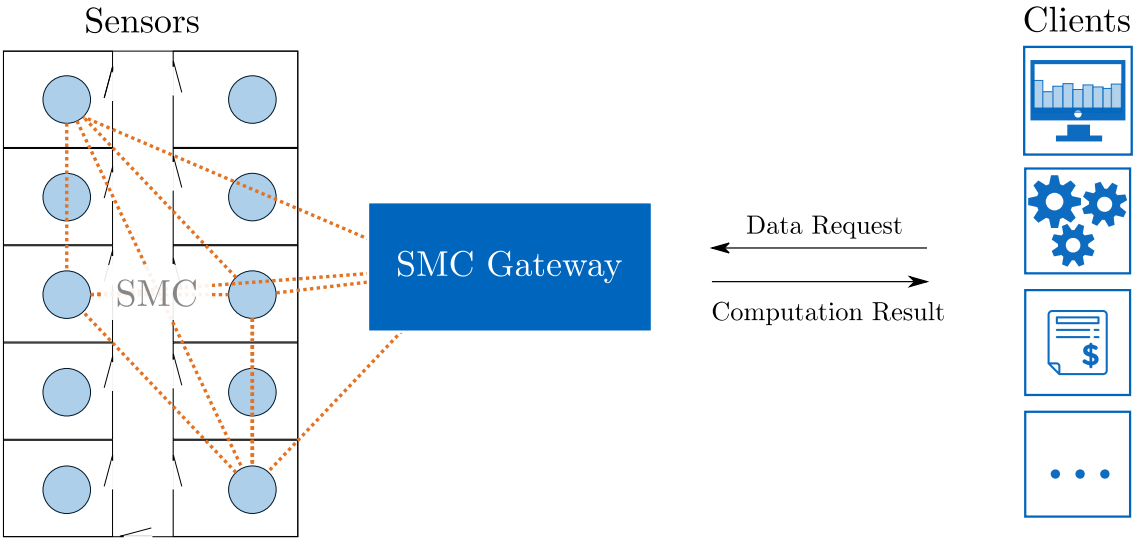

Figure 7.1: Interactions between the clients, the SMC Gateway and the SMC peers.

7.3 Solution Sketch

In this section we will shortly sketch our approach in order to facilitate later analysis (Section 7.4) and derivation of requirements (Section 7.6). The content of this section is based on the publication [vMC18b].

Current SMC frameworks mainly focus on the construction of protocols. Their goals is to show improvements of the fundamental computation primitives, the feasibility of certain computations, and to perform measurements of their performance.

In our context, we aspire a solution which carries out SMC sessions in an automated and repeatable manner and which possesses service character: data requesting clients should not have to know about SMC and how data is processed before they retrieve it. For them, it should only be necessary to post standard data requests as they are prevalent in a classical server/client architecture. The execution of SMC is completely hidden from them (cf. Figure 7.1).

We encapsulate this combination of data distribution and management centralization in the term *Virtual Centralization*, which we define as follows:

Definition 7.1 Virtual Centralization *Virtual centralization is given when there is an intermediary entity which provides centralized access to a set of data (streams) which are collected and stored at a distributed set of entities. The fact of distribution is shadowed by the intermediary.*

As depicted in the use case, data is collected by smart appliances featuring sensors. Since they will be the cooperating entities in the SMC sessions, we denote them as *peers* or *sensor platforms*. These are deployed in the environment in a distributed manner. They sense the state of the current environment and hold a history of previously created values. They are depicted on the left side of Figure 7.1. Instead of deploying a central data processing unit to which all gathered raw data is sent, we deploy a *Gateway* being capable of SMC. This Gateway is the intermediary between the peers performing SMC and the *clients* aiming to obtain the final result. In each direction—facing the peers and facing the clients, respectively—it has to provide specific features:

Client-Facing Interaction

Facing clients, the Gateway acts as a traditional service following the client-server paradigm. Depending on the environment and the current setting of deployment, a Gateway offers different types of available data. Clients can send requests for certain types of (aggregated) data. After some seemingly internal computation, the Gateway provides the requested information to the client. The clients are neither able to intervene in the computation process for the result nor able to discern how the information is obtained. The execution of SMC is completely hidden from and abstracted for the clients. Since the computation can take some time, requests can also be answered asynchronously.

The system should provide data for different goals and different clients. Therefore, access control has to be carried out in order to ensure only authorized data access.

Peer-Facing Interaction

Gateways do neither store raw data of peers nor do they compute results alone. Instead, they orchestrate a group of peers in order to make them execute SMC sessions.

Gateways hence translate incoming requests into SMC sessions which can be carried out with the corresponding peers. After the computation, the Gateway obtains the result and forwards it to the client. While delegating the actual computation, it provides availability, reliability and robustness guarantees so that these non-functional aspects are also similar to classical server systems. Likewise, exceptional cases as the loss of a collaborating data provider or networking problems are handled without the client's knowledge, if possible.

New peers should have the ability to choose from Gateways available in the same network. When a new peer connects to a Gateway providing new types of sensor data, the Gateway should reflect this information for the clients.

7.4 Analysis

After this rough sketch of a solution, we further detail and anticipate how the high-level goals can be achieved.

Client-Facing Interaction

On the side facing to clients, the main goal is to provide a service abstraction.

Service API Initially, this means that the distribution of data and data sources is hidden. Clients only have a single point of contact, the Gateway. This Gateway provides a service interface which allows posting classical data requests. Using a referencing scheme, the client specifies what data to obtain. As an answer, the Gateway provides this data while unambiguously referring to the corresponding request.

Since computations might take longer, answers should also be available in an asynchronous fashion. This implies that clients are immediately informed about the retrieval of their requests and get the means to query for the result at a later point of time. Therefore, reference to previously issued requests should be made possible for clients.

Metadata Directory In order to be informed about available data, the client must be able to query further information. Via the same API, the Gateway should provide the metadata which kind of information is currently available. The availability of information depends on the peers that are currently connected.

Request Translation The metadata must provide knowledge about the computations which can be obtained from the Gateway. This must happen in a non-SMC specific manner for the clients. Similarly, clients must be able to specify data requests in a manner which does not imply the usage of SMC; in other words the data structures must also abstract from SMC. Translation to a valid SMC session has to happen in the Gateway. This means that it must be possible to map the parameters from the request of the clients unambiguously to an SMC computation. This includes information such as the group of peers to compute with, the type of data to derive the result from and the protocol, i.e., the algorithm to be executed.

The reply from the Gateway contains the desired information. If any SMC-specific errors occur, the Gateway should also hide them from the clients and try to recover failed sessions. If failure is persistent, the Gateway should translate the error for the client abstracting away from SMC.

Peer-Facing Interaction

For a clean service abstraction, further challenges on the side facing the peers have to be addressed.

Discovery For self-management, automatic discovery is vital. It should not be necessary to statically provide address information of a Gateway to newly instantiated nodes. Instead, they should be able to discover candidates themselves. Decisions about which identified Gateway to select can then be carried out manually or rule-based. This functionality is also vital when a connection is lost or a peer changed its location. Rediscovery of Gateways should also happen in these cases.

Pairing After choosing a Gateway, an initialization process is necessary. It provides the Gateway with meta information about the peer. This must encompass the information which sensors are attached to the new peer and which data can be provided. Similarly, information about the available computation protocols must be shared. Lastly, in order to establish a peer identity and to enable secure channels, a cryptographic trust exchange must take place.

Operation The Gateway should become a controller for the peer. I.e., the Gateway must be able to initiate SMC computations on the connected peers and perform these sessions in the synchronized fashion necessary for SMC. This comprises multiple subtasks: for each computation, only a subset of the connected peers participates (e.g., selecting the temperature information only from a specific floor or room type). Hence, the Gateway must be able to specifically select peers for participation. This information must also be communicated to each of these peers since establishment of connection between them is also necessary. Similarly, these peers must also be informed about the actual computation to be carried out. This request specification initially originates from the client and it must be communicated to the peers in an appropriate form.

In order to easily obtain the result of the computation and to be aware of errors during computation, the Gateway can participate in it. However, the Gateway typically does not have sensors itself, i.e., it does not have any real input to a computation session. This is especially likely when a single Gateway covers a multitude of computations on different data types. In consequence, it must be possible to not provide any input to a computation to avoid distortion of the computed result. This can be achieved on the level of the protocols: either, it is viable not to require an input from the Gateway at all or to allow the Gateway to provide a neutral element to the computation.

Robustness and Recovery According to Virtual Centralization, the Gateway shall shadow the distributed fashion of the computation. Hence, it also has to make sure that the SMC session is executed with similar reliability guarantees as if the Gateway could compute the result itself. This is not trivially given: SMC sessions consist of a high amount of network communication via mutual connections between the participants, making them prone to connection interruptions. It is, hence, important to ensure correct execution of the shadowed SMC sessions and to provide means of recovery in cases of failure. Here, we can differentiate between the times when a session is ongoing or not. If no computation is currently carried out, the Gateway only has to preserve the ability to communicate with the peers. If a peer is lost during that state, only a cleanup on the side of the Gateway is necessary: e.g., if a peer finally left the area of the Gateway, no actions of connection reestablishment can be carried out. Otherwise, the Gateway must be ready to re-pair with the formerly lost peer when it rediscovers the Gateway.

Upon errors during a computation session, more steps have to be taken: initially, the Gateway must know if a session failed. A root cause analysis should provide the insights how the error should be handled. In simple cases, if the necessary peers are still available, a session restart can be performed, trying to carry out the computation again. If peers have been lost, the Gateway must assess whether or not the computation can be performed without the lost peer while still yielding a valid result[1]. Depending on the decision, the Gateway can either readjust the participant set and restart the session or translate the failure to a temporary or permanent failure for the requesting client.

Furthermore, every participant can delay a running computation. As this cannot be controlled by the Gateway, the request/response interaction with clients should be performed asynchronously in order to prevent open connections and timeout on the side of the client.

Availability of Computation Result Typically, either a single or all participants of a SMC session obtain the final result. In our context, the querying entity is different from the set of participants which collaboratively perform the computation. Hence it is necessary that—in a final step—the result from the computing parties is forwarded to the actual requester.

Security and Privacy Protection

As identified in Section 7.1, SMC itself already provides the fulfillment of certain protection goals. We elaborate this briefly:

Data minimization Data minimization is improved in the following manner: each peer only holds information about itself. There is no place where a third party instead of the data owner holds raw data of peers.

Unlinkability Similarly, it supports unlinkability in three ways: 1) Since there is no place where data of several different peers is stored, illegitimate linkage (i.e., without knowledge of the peers) of this data is not possible. In turn, this also improves data minimization since no new knowledge can be derived this way. 2) When SMC is applied properly, the result obtained from a computation does not allow recovery of the initial input values[2]. 3) Similarly, the result does also not allow to recover *which* peers contributed to

[1] E.g., if an average value of a dynamic set of peers should be computed, it can be of secondary interest, how many peers participated at all.

[2] This excludes special cases where only the input of a single peer is provided. Then, the input of the peer is indeed recoverable. Furthermore, the smaller the set of peers providing input, the better approximations regarding their initial input values become possible.

the result. However, this information is sometimes necessary in order to make sense of the obtained result. Hence, it should be configurable whether this information is available to the clients or not.

For other protection goals, only a rough foundation is provided by SMC. A system employing it has to appropriately make use of them in order to completely fulfill these goals in a beneficial manner.

Transparency Regarding transparency, data providers contributing to a computation definitely know about the type of computation, the data to be provided and the other cooperating peers since this information is necessary for execution. Instead of using this information only for performing the computation, this information should also be persisted to allow insights which computations have been performed in the past. This supports accountability and in turn transparency. To further enhance this measure, more metadata about the context of the request should become available to the peers. They should also be informed which client requested the computation and for which purpose; the exact time and the concrete content of the initial request. This makes data usage fully accountable for system and data providers likewise.

Intervenability The same insights can also be used to assess upcoming computations. Having a context of the requested computation, peers are in a better position to decide whether to contribute or to refrain from it. In other words, based on an informed decision, they can better exert their right to veto against computations which are deemed illegitimate. This realizes intervenability. In order to provide a stable system and ensure understandable and accountable system behavior, peers should not cancel computations in a manner as if an unexpected error happened. Instead, a clear veto process should exist where the peers' vetoes are communicated to the requesting client. The latter should especially obtain a proof of the veto performed including an understandable veto reason. Since vetoes are made transparent by these measures, accountability can then be easily added by persisting the corresponding messages in an adequate manner.

Vetoes prevent an upcoming computation from succeeding. The Gateway should assess whether an equivalent computation can be performed without the vetoing peer(s), and automatically execute these.

Access Control Access to data should be accompanied with access control. This is especially necessary since the system shall provide different kinds of data for different clients. Our method of query specification already provides fine-grained statements about the data to be accessed. Combined with environmental state (e.g., the current time), attribute-based access control can be carried out. Reusing the attributes from the queries also for authorization allows easy mapping of authorizations to actual requests and facilitates the logic necessary for permission granting and authorization verification.

Since this authorization information is now already present and long-term accountability is desirable, this information should also be logged in a trustworthy manner. In consequence, requested and granted access can then be reconstructed after the fact.

The environment in which the solution should be deployed is dynamic and churn of Gateways can occur. If two Gateways exist for the same trust domain (e.g., Public Key Infrastructure), they should mutually accept the authorizations issued by the other Gateway. Since they might only be available sequentially, communication between both Gateways should not be necessary to accomplish this.

7.5 Related Work

The feasibility results presented in the background chapter (cf. Section 3.4.2 on page 29) also represent the related work for our approach of improving practical applicability of SMC. We do not repeat the details here. We refer to Section 3.4.2 for an overview. Here, we focus on the aspect whether and how SMC was deployed as a service. Related work which has not yet been presented as feasibility result is described here in full detail.

In [BCD+09] Bogetoft et al. work with real-world data, but the setting of evaluation is partly artificial: the data was collected from real users, but the computation was carried out in an ad-hoc manner. Three laptops were used which were connected via an intranet. The computation itself was started manually. No attempts were made to enable continuous or repeated computations. The result was not distributed to the input parties during the sessions but presumably via an organizational announcement.

The main contributions of [BSMD10] are use-case-specific computation protocols for data analysis. Their evaluation was executed in a strongly controlled setting: while performing measurements in intranet and internet settings (using PlanetLab), the measurements were completely done by the researchers. There was no real interaction via SMC with the ISPs which provided the data. The authors recognize that robustness of computations in the presence of host failures is a vital challenge for future work. We also address this in our approach.

Bogdanov et al. successfully deployed an SMC solution as an application for continuous use; several computations were carried out. Although computations are only performed twice a year, they yield a real SMC service with a comparatively high amount of automation. In their publication, they stress the need for robust and practically applicable service-like SMC deployments: they "consider it an important challenge to reduce the administrative attention required for managing" a computing node to make "the technology easier to deploy in practice" [BTW12]. In their summary, the conclude that more focus should be laid on the practical problems of applying SMC: administrative real-world challenges should gain more attention and more challenging settings like cloud environments should be considered. We address exactly the former of these problems in our work.

The use case of [DSD+13] is that multiple Internet service providers (ISPs) cooperate in order to detect whether a network outage is globally or locally induced. To achieve that, they have to combine individual and confidentially kept outage data of their own networks. This approach of Djatmiko et al. was evaluated using real data from an ISP, but all computations were carried out without interaction of real stakeholders.

In [ZDT+16] Zanin et al. present an auction approach for trading emission allowances. Most notably, they realized a real system as a web service interacting with actual stakeholders. Representatives of airlines can enter their bid and an external referee, acting as auction manager, operates their auction system. While the interaction is neither fully automated nor the SMC computations are performed automatically, they provide an infrastructure which addresses a concrete use case and allows real interactions.

Bonawitz et al. [BIK+17], aspiring private training of neural networks using SMC, explicitly focus on real-world problems of their use case. They aim for a solution which is robust against host failure, enables asynchronous collection of data and considers the real Internet infrastructure (e.g., featuring NAT-shielded devices). The corresponding talk [Kre17] implies that the system will be productively used for privately creating auto-correction and auto-suggestion models for smartphone input trained by the typing history of thousands of smartphones. Their only constraint is that their approach does not yield a general purpose SMC solution but one tailored to their use case of securely aggregating private vectors.

Thoma et al. [TCF12] present a secure smart metering solution. They aim for protecting the individual, temporally fine-grained consumption data of households, providing correct and verifiable billing of each individual household and detailed (non-individual) consumption feedback for load management. They achieve this via a dual approach. Temporally fine-grained feedback data is collected by aggregating individual consumption data before sending it to the energy provider. This is performed via SMC and individual consumption data is protected. Monthly aggregates for billing are created in plain and locally for each household. These values become verifiable as every household sends individual, fine-grained but homomorphically encrypted consumption data to an untrusted *utility server*. The server can sum up the inputs and use it for secure verification of the user-side result provided at the end of the month. Like the previously described work, they also address a secure aggregation in a real-world setting and provide a framework which enables practical application. However, their solution is rather on the level of a concept. They do not elaborate how infrastructural problems like discovery of available households and interconnection between them are handled.

In conclusion, most solutions presented in related work were executed manually in highly controlled environments. Only few propose architecture for continuous real-world deployment and automated SMC computations. Among them, some do not consider realistic constraints of their environment.

We chose secure computations in dynamic environments as our scenario. Our solution directly addresses infrastructural challenges present in smart environments and provides a framework to enable automated and self-managing application of SMC.

7.6 Requirements

In the following, we state the requirements for providing an automated service for data retrieval based on SMC. The actual solution is presented in Chapters 8 and 9.

Service Abstraction

The first subgoal is to make SMC applicable in real-world contexts. In order to achieve compatibility with existing environments and infrastructures, from a client (data user) perspective, the service behaves like other state-of-the-art services.

RA.1: Single Point of Contact

Although SMC initially being a peer-to-peer solution, data obtaining clients only have to contact a single point in order to get answers for their requests. This single point of contact conceals the dynamic nature of the underlying peer-to-peer interaction. Depending on how dynamic the environment is, it is possible to statically assign the role of the Gateway to a selected node, or to allow all nodes to decide on a common Gateway for a predefined amount of time.

RA.2: Service API

Continuing the previous requirement, the single point of contact exposes a standard interface for interaction. This conforms to established standards like REST. Correspondingly, it is not necessary for the client to perform more than posting a standard request.

RA.3: Directory Service

Clients are able to get information about which data and which sources are available via the interface.

RA.4: Request-based Invocation

The Gateway is able to translate an incoming request into an SMC session which yields the desired result.

RA.5: Async Response

SMC does not guarantee a limited execution time. Hence, requests are not necessarily blocking until a response is given.

Self-Management

Concealing the complexity of an SMC-based solution as described by RA.1 especially requires the system to be self-managing. Thereby, complexity of management and configuration can be removed from the client side.

RA.6: Discovery

During setup time, nodes automatically become aware of available Gateways. Each node can choose from the list of available Gateways according to a local matching logic and have all information available which are necessary to establish an initial connection to its choice.

RA.7: Auto-Configuration

Adding and removing nodes reconfigures the environment automatically. In response, groups of nodes are dynamically formed and the directory service is adjusted. Connections between nodes enabling SMC sessions are automatically established and kept alive.

RA.8: Automatic Session Management

A group of nodes must be able to start a session upon request of their connected Gateway. This includes communicating the request which triggers the session to each node, which, in turn, selects the right data and the computation protocol. As SMC works in a synchronous fashion between participating nodes, synchronization is realized by the session management. Furthermore, nodes are informed whether a computation was successful. On failure, an automatic restart is possible.

System Stability

Due to the nature of SMC, our solution is a distributed network system. This is typically prone to network failures which have hence to be addressed.

RA.9: Fault Tolerance and Robustness

The SMC session being a highly network dependent process, connection failures or interruptions are a relevant cause of errors. Hence, we require that our system is robust against this type of failure: in unstable situations, the system eventually stabilizes itself. Similarly, the full system does not fail upon failure of single participating entities.

RA.10: Recovery

Network failures can also interrupt ongoing computations. Hence, the solution provides a resuming mechanism for computations or be able to transparently restart an interrupted computation.

Security and Privacy

Regarding security and privacy, we start with preserving the privacy features SMC itself exhibits. We then aspire to fulfill further privacy goals as discussed in Chapter 2.

RA.11: Raw Input Confidentiality

Being the main motivation for our work and the main feature of SMC, we demand that our solution is able to collaboratively compute functions upon inputs of the participants. The inputs of every participant, however, remains confidential, i.e., it must not be necessary to provide any knowledge about them but the actual final result to any other entity. As a consequence, unlinkability of computation results and inputting participants is yielded. In other words, the result of a computation does not give any insights on the individual input data.

RA.12: Confidential and Authenticated Channels for SMC

Some SMC implementations require the availability of confidential and authenticated channels. Hence, authentication has to be handled outside of the actual SMC framework and corresponding material as certificates has to be made available to it.

RA.13: Transparency

When a Gateway allows the processing of a received request, all participating nodes are also able to access the initial request and to derive the purpose of the invoked computation session.

RA.14: Intervenability

Besides the raw data remaining confidential, peers stay in control of their data. It is their decision if and how their data is used. Consequently, we require the fundamental ability for every peer to decide in which computation it opts to participate or not. When receiving the initial request, nodes become empowered to understand the context of the session invocation by RA.13 and shall furthermore be able to veto against participating the given session if it conflicts with their own access rules.

RA.15: Client-faced Access Control and Permission Transparency

As outlined in the previous point, peers are ultimately in control of their data and the success of the computation. This is a rough kind of access control which, deployed alone, has undesirable properties: For clients it is impossible to anticipate the outcome in advance, and the intentions of the uncooperative peer are not communicated explicitly. This would introduce an uncertainty on the side of the client and make it harder to understand error cases where computations did not succeed. In order to facilitate access control decisions and improve client interaction, we add another earlier layer of access control. Its aim is to anticipate the peers' decisions and to make client permissions explicit.

The Gateway is able to constrain combinations of data and computational protocols provided by the API. These combinations mirror the later peers' decisions. For explicitness, the permissions are manifested as access documents held by each client. This allows the Gateway to forward the permission statements to peers.

RA.16: Accountability

RA.13 provides insights in performed computations, their requesting client and their reason. These information should not only be used to decide about participation agreement or veto, but should also be used to create a persistent access log. This enables data owners to understand the usage of their data and potentially derive new access rules.

7.7 Statement on Author's Contributions

The findings of this chapter have been published in the following paper:
M. von Maltitz and G. Carle. Leveraging Secure Multiparty Computation in the Internet of Things. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, pages 508–510, New York, New York, USA, 2018. ACM Press (reference [vMC18b]). The publication presents the solution sketch (Section 7.3) of this chapter as a vision for privacy-preserving architectures in the Internet of Things. It establishes the segmentation of a solution into the subproblems of realizing robust and automated execution of SMC (cf. Chapter 8) and performing data requests and enforcing access control on private data (cf. Chapter 9).

The publication [vMC18b] refers to the concept of Virtual Centralization that has been developed in cooperation with Stefan Smarzly in the context of his Master's Thesis advised by the author of this thesis and by Holger Kinkelin and supervised by Georg Carle.

8. Self-Managing SMC

In the last chapter, we entered the domain of smart environments. We identified that they are a promising use case where privacy-preserving data processing is needed. We already sketched a solution architecture for this, carried out further analyses and derived a catalogue of requirements to be fulfilled by a privacy-preserving service on the foundation of SMC. Here, we will develop the first part of a corresponding solution.

Objective

In this chapter, we address Research Question Q4. We provide a Gateway which performs orchestration and management of SMC. In particular, the following features are realized: detection of Gateways by peers is made possible. Afterwards, they establish a common state so that the Gateway knows about the capabilities of the new peer. The Gateway is then able to orchestrate and manage SMC sessions. It provides the necessary runtime configuration and handles possible error cases. With the contributions of this chapter, SMC can be executed programatically in an automated, flexible and robust manner in dynamic environments.

8.1 Architecture

From a bird's eye view, both the peer component and the Gateway component can be implemented as a finite state automaton. They are depicted in Figure 8.1 and Figure 8.2. In the following sections we present and discuss the design of the most important states and interactions of both.

8.2 Gateway Discovery

The first step to a self-managing SMC infrastructure is a discovery mechanism which enables peers to detect available Gateways. We realize this by utilization of Zero Configuration Networking as specified by RFC 6762 [CK13b] and 6763 [CK13a].

The content of this and the following sections including 8.6 has also been published as [vMSKC18] in a highly condensed manner. The text was written completely by the author of this thesis.

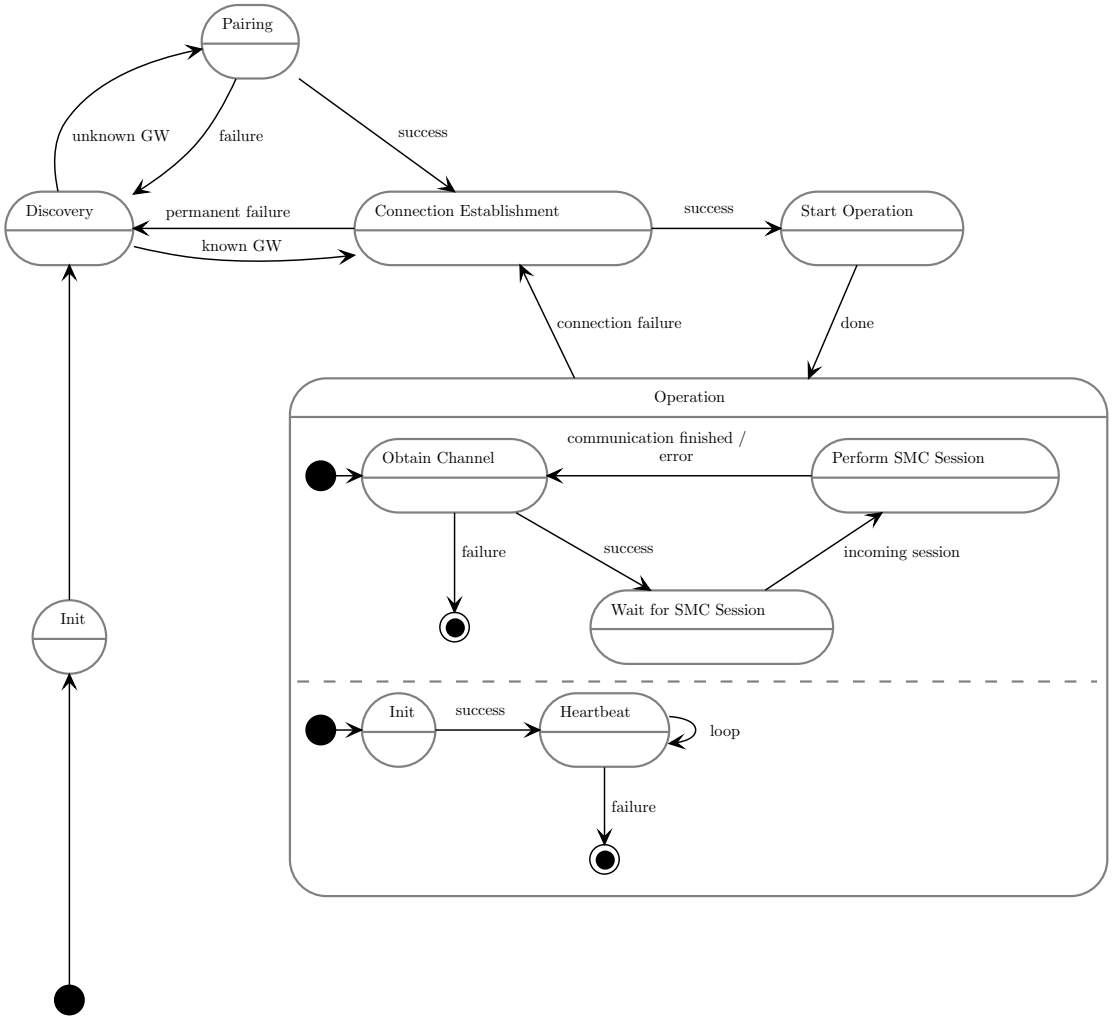

Figure 8.1: The state diagram of the peer

8.2.1 Gateway Announcement

The Gateway is equipped with a component realizing multicast DNS and DNS-based Service Discovery. It regularly sends announcements which inform the network about its presence. These consist of mDNS messages containing A, AAAA, SRV and TXT records (Listing 8.2).

The messages announce the service of a named Gateway, e.g., *gw-12345abcdef*. The A and AAAA records resolve the address of the domain name gw-12345abcdef. Its services are simultaneously announced via the SRV record shown in Listing 8.1.

```
1 gw-12345abcdef._seccomp._tcp.local <TTL> IN SRV <priority> <weight> <port> <target>.
```

Listing 8.1: SRV Record of Gateway Announcement

The SRV record informs peers about the presence of a Gateway *gw-12345abcdef* which provides a service *seccomp*, available via the port <*port*> at the host <*target*>.

```
1 gw-12345abcdef._seccomp._tcp.local <TTL> IN TXT "<KEY1>=<VALUE1>" "<KEY2>=<VALUE2>"
```

Listing 8.2: TXT record of Gateway announcement

Via the TXT record, it is possible to announce initial information about the Gateway as key/value pairs, which help peers to choose the most suitable Gateway, if several are available. Relevant attributes are shown in Table 8.1.

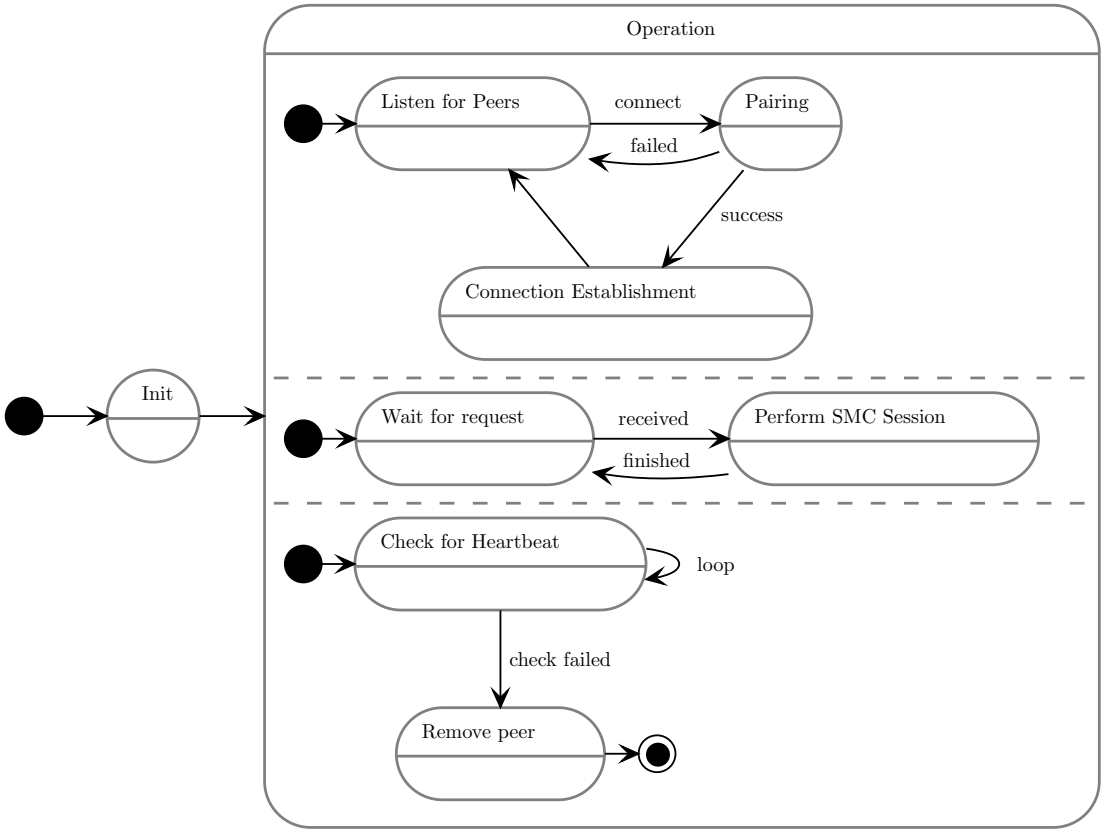

Figure 8.2: The state diagram of the Gateway

Attribute	Example
Version Number	1.0.0
Description	Gateway of the 3nd floor in Building A
Supported Sensor Types	Temperature, Humidity, CO_2
Location	Room 123

Table 8.1: TXT attributes of the Gateway announcement

The *version number* is provided to check compatibility of the software on the Gateway with the software of the peer. RFC 6763 [CK13a] recommends the attribute `protovers` for this purpose. The *human-readable description* allows manually choosing peers to identify the Gateway. *Supported sensor types* is the first attribute which helps peers to select adequate Gateways by matching their capabilities with their own. Two cases exist: in more static environments or when Gateways have pre-defined purposes (to support only a small specified number of types), peers should only connect to Gateways which already support their own sensor types. In more dynamic environments, this attribute rather indicates which types are currently supported by the Gateway, but this information can change over time. Peers providing different sensor types can nevertheless connect to the Gateway. When enough peers of a certain, not yet available sensor type have connected, the Gateway can start announcing that this sensor type is now available. Gateway selection can also depend on other attributes, for example by its *location*. If proximity is a semantically relevant feature for building computation groups and sessions, this attribute can be used to select the most suitable Gateway. More generally, when the place of deployment is not logically structured like a building, a GPS tuple can be more appropriate. Furthermore and generically, other attributes can be imagined which improve the choice of the Gateway.

Spoofing and Tampering Mitigation

By default, mDNS messages have no further security compared to plain DNS [CK13b]. Hence, they are prone to spoofing (addresses in the A/AAAA record) and tampering (e.g., contents of the TXT record).

Regarding our setting, we perform the following case differentiation:
In static environments, peers can be preconfigured to prefer certain available Gateways. In this case, it can be assumed that Gateways hold individual certificates and a corresponding private key. The certificates of these Gateways can then be provided to the peers. Being equipped with this information, peers are directly able to authenticate data coming from the Gateways. In other words, a foundation upon which mDNS messages can be authenticated is available.
In dynamic environments, we assume that formerly unknown Gateways exist. Peers can automatically connect to them upon their detection. As a consequence thereof, we cannot assume that any previous knowledge about the Gateways (including certificates) exists. Lacking this foundation, it is not possible for a peer to assess the legitimacy of a Gateway. Consequently, protecting the mDNS messages against spoofing and tampering does not yield any benefit. However, employing a trust-on-first-use (TOFU) security model, the availability of cryptographic material after a first connection would allow peers to recognize known Gateways afterwards and to authenticate their messages when a second pairing becomes necessary. Furthermore, they would be able to retrospectively verify the mDNS messages which lead to the connection with the Gateway.

In summary, there are situations in both cases where authentication of mDNS messages is possible. This can be realized by providing a signature which protects the announced domain name, its <target> address, the port and all additional information provided as TXT record using the private key of the corresponding Gateway. This signature can then be stored as a further key/value entry in the TXT record.

However, technical considerations make it inadvisable to do so:

TXT record length RFC 6763 states several constraints regarding the length of the TXT record: each key/value pair must not exceed 255 Byte. Furthermore the whole TXT record is intended to consist of 200 Bytes or less. If this is not sufficient, the record should be kept under 400 Bytes so that it fits into a single DNS message having 512 Bytes as maximum [Moc87] or under 1300 Bytes so that it fits at least into a single 1500 Byte Ethernet packet [CK13a].

Optionality of TXT records RFC 6763 states that TXT records should be considered as optional and that connecting instances should be able to work completely without them. I.e., all information provided by them should be also available during the in-band communication with the discovered service. The TXT record is merely considered as performance optimization allowing the connecting instance to make selection decisions when facing a greater set of potential service providers.

We evaluated that a signature computed using SHA-256 [EH11] as hash function and a private key based on the elliptic curve prime256v1 (X9.62/SECG curve over a 256 bit prime field) takes about 70 Bytes. As binary values are allowed in the TXT record, it would be technically possible to store the signature as entry. However, the second argument states that in-band renegotiation of the TXT content is advised, hence, verification can also be moved to the initial pairing step. Doing so does not change the aforementioned security model as it is based on the same assumptions. The only difference regarding security is that peers cannot receive the Gateway's attributes passively and verify them silently

anymore. Instead, the peer has to establish an initial connection to the Gateway and authenticate it via established means like TLS. Hence, rogue Gateways would at least get an open connection to peers and become aware of their existence.

Another solution for mDNS message authentication is the combination of mDNS/DNS-SD and DNSSEC, however, this has not yet been evaluated [CS06].

8.2.2 Peer Query

In alignment with mDNS, peers have the ability to send DNS queries polling for corresponding service type `_seccomp._tcp.local`. This triggers all reachable Gateways to send their announcement messages.

8.3 Pairing Process

Once a peer has selected one of the discovered Gateways, the next step is to perform a pairing with it. The goals are

- establishing a trust relationship between the peer and the Gateway
- providing the means for encrypted communication
- establishing a connection which allows the Gateway to initialize computations and
- providing all peer information to the Gateway which are necessary to correctly group the newly added peer with existing ones.

The pairing protocol is shown in Figure 8.3 and described as follows:

1. The first three steps reflect the discovery process. Here, the peer optionally sends a multicast query for Service Instance Enumeration [CK13a] which triggers all reachable Gateways to send out DNS-SD service announcements.
2. Due to the previous triggering or due to initial announcement, the Gateways send mDNS messages in order to inform about their presence and the provided service.
3. The peer receives these announcements (possibly from several Gateways in its proximity). Based on the service meta information contained in the TXT record the peer can decide which Gateway to connect to. With this step, the discovery phase is finished.
4. Once the peer has decided which Gateway to contact, a connection is established and wrapped via TLS. During the TLS handshake, both parties present their certificates to each other. Regarding the next step, we have to differentiate our use cases.
5. In a more static environment, we can assume that peers have been preconfigured to correctly identify and to authenticate Gateways. These Gateways possess certificates which the peer is able to validate. Based on this validation, it recognizes the Gateway as legal. This step only ensures that no rogue Gateway is contacted. It does not provide whether the contacted Gateway is the preferable one for the contacting peer.
6. Alternatively, regarding a more dynamic environment, it is not sensible to assume available validation via a public key infrastructure and signed certificates. In this case, the *trust on first use* security model can be employed. The peer accepts the currently unknown certificate and stores it for later usage. The peer is then able to

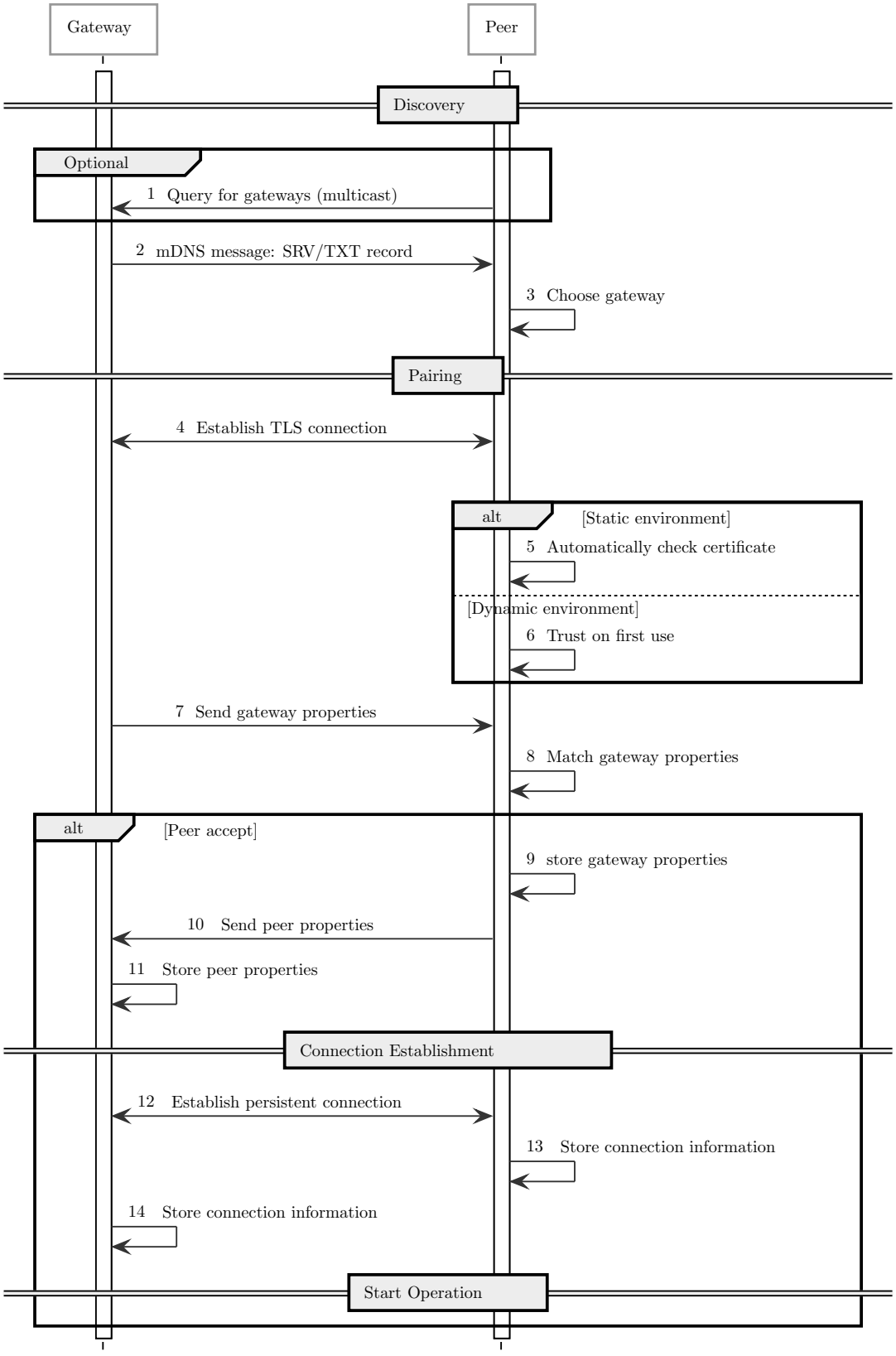

Figure 8.3: Discovery and pairing process between Gateway and peer

recognize the same Gateway at a later time. Depending on the exact use case, this concept can be improved by out-of-band checks: in dynamic environments where the peers are end-devices of users (like smartphones), this can be realized by comparison of fingerprints or similar interactions.

7. Following the best practices and suggestions in [CK13a], metadata in TXT records should not be mandatory for establishing a connection between service provider and a client[1]. In our case, the announced information is essential to select the right Gateway. Hence, this step reiterates an in-band exchange of this very metadata.

8. The peer reevaluates the obtained information and decides whether to continue the pairing process or not. For full automation this step can be rule-based. In dynamic cases where manual intervention is possible or desired, a user can make this choice.

   Besides RFC-conformity, another benefit of this renegotiation is that the provided pieces of information are bound to the identity of the Gateway via its certificate this time and hence protected against spoofing and tampering (opposed to their availability via the TXT record; cf. Section 8.2.1).

9. Given, the peer accepts the Gateway based on the provided information, the pairing process can proceed. The peer then stores this information so that the Gateway can be remembered as being formerly selected.

10. Vice versa, the Gateway obtains the same types of metadata about the peer, so that the former can include the latter into matching groups of other peers which are already known to the Gateway.

11. Subsequently, this information is stored by the Gateway.

12. As the next phase, a persistent control channel is established. This channel allows the Gateway to send orchestration messages to the peer which in turn enable the execution of SMC sessions.

13. All necessary information to hold the channel must be persisted. This happens on the side of the peer.

14. Likewise, the Gateway persists the channel information for later usage.

Afterwards, the interaction changes into operation mode. Here, the established channel is monitored by heartbeat messages.

8.4 Operation Mode

Productive interaction happens when the Gateway and the peers are in operation mode. For the sake of completeness, Figure 8.1 and Figure 8.2 show the full automaton of both component types, also including the previously described steps.

8.4.1 Peer-Side

On the peer side, two concurrent processes are started in the `Start Operation` state: the process for managing the control channel to the Gateway and the heartbeat process:

Control Channel Process This process itself performs an infinite loop: a channel to the Gateway is obtained. This channel serves for the communication of control messages for the initiation and orchestration of later SMC sessions. When the channel is obtained, the peer blocks this process, waiting for incoming messages. Upon receipt, it handles the messages accordingly. When a session finished or any error occurred, the process returns to its initial state. Here, if possible, it reuses the current channel or tries to obtain a new channel, if the previous terminated with failure.

[1] *Client* is the terminology of the RFC, meaning *peer* in our context.

If obtaining a connection to the Gateway fails, the process terminates with error. Not being able to obtain a control channel is an indicator for connection failures. Consequently, the peer terminates the heartbeat process and changes back to `Connection Establishment`.

Heartbeat Process As the control channel process can identify connection failures itself, the meaning of this process is different. Even when no SMC sessions are performed for a longer time, the heartbeat indicates to the Gateway that this very peer is still available and connected. Hence, while the previously described process is for peer-side failure identification, the heartbeat process is for Gateway-side identification.

8.4.2 Gateway-Side

After initialization, the Gateway performs three types of processes concurrently:

Listening Process Here, the Gateway continuously listens for connection attempts of new peers. It performs the pairing with newly connecting peers and—upon success—adds them to the pools of paired and active peers.

Control Channel Process This process is the counterpart of the first process of the peers. It holds the connection to a single peer and transmits control messages when available. For every connected peer, a process of this type is spawned.

Heartbeat Check Process Analogously, this process is the counterpart to the heartbeat process of the peer. It listens for heartbeat messages of the latter and terminates the connection to the peer if no heartbeat was received for a predefined time interval.

8.5 Session Orchestration

This section addresses the interaction performed via the established control channel when a computation has to be carried out. The aim of the orchestration is the planned and controlled execution of SMC sessions.

8.5.1 Orchestration Protocol

The overall sequence of interaction is depicted in Figure 8.4. The steps are as follows:

Preliminaries

We assume that a request is sent from a client to the Gateway. It contains a description which type of information should be computed and which set of data input peers should be used. For details on the client-side querying cf. Chapter 9

Preprocessing

During the *preprocessing*, the Gateway parses an incoming request and derives all necessary information to interact with the addressed participants.

1. When the Gateway receives a request from a client, it first transforms the request into a Task Description. This is a semantic change from the high-level request to a detailed executable specification of an SMC session. The Task Description Scheme and the translation are described in Section 8.5.2.
2. From this session specification, the Gateway can subsequently select the set of participants which is addressed by this session.
3. Finally, the task is marked as ready and scheduled for execution.

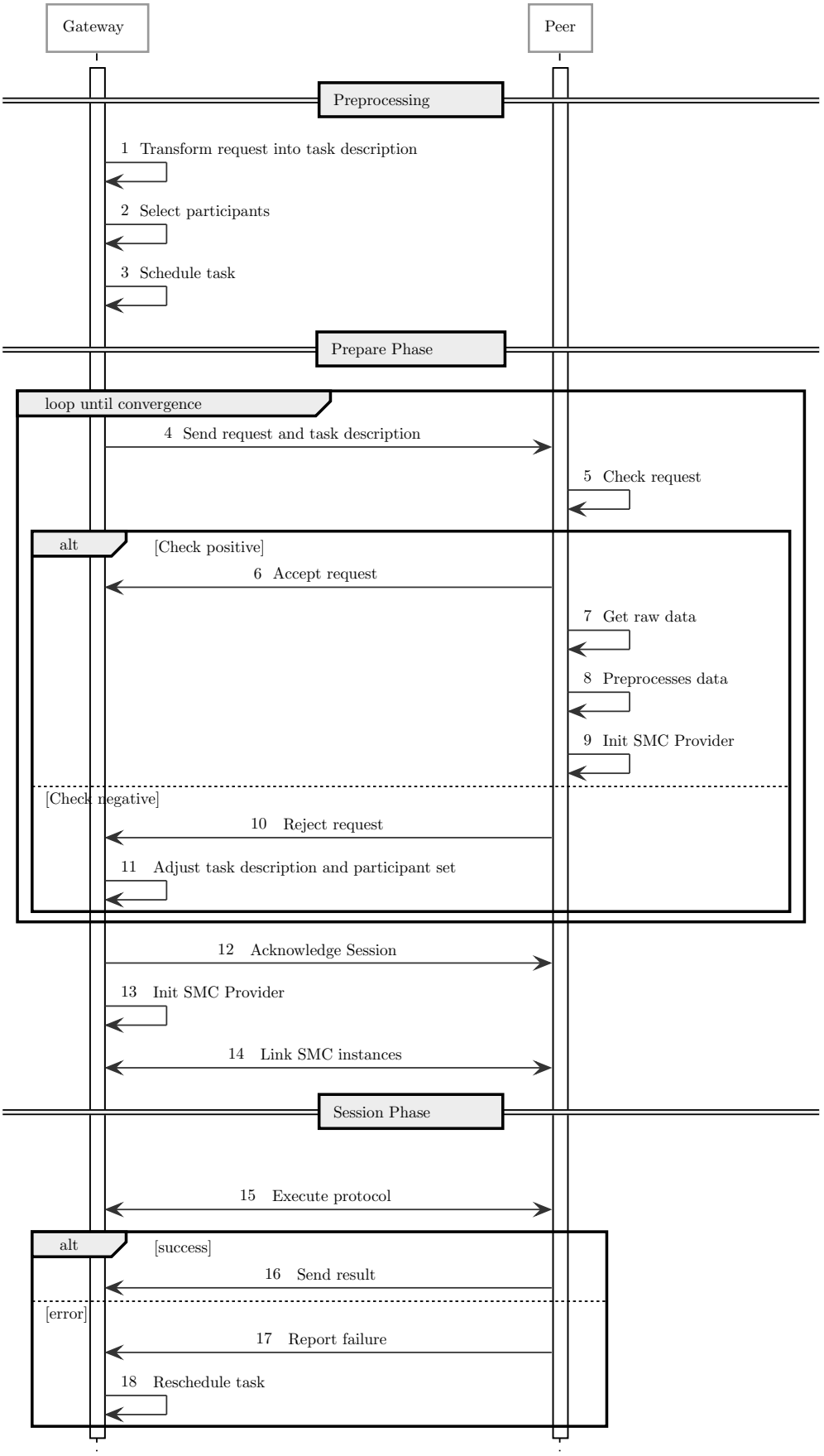

Figure 8.4: Orchestration process

Prepare Phase

During the *prepare phase*, participants are informed about the upcoming session and instructed to prepare the session on their side. Here, participants can have the specific ability to veto against incoming requests. If this is the case, the Gateway has to act accordingly: if possible, the same computation should be performed without the rejecting peer. Otherwise, it should forward the error to the requester and recover so that other client requests become possible again.

4. The Gateway sends the request and the generated Task Description to the peers. This enables the peers to act on the request and prepare all necessary steps for the planned computation in the following steps.

5. Firstly, each peer may check whether or not the posed request is in accordance to the peer's privacy and data protection rules (for examples cf. page 153).

The steps 6–9 assume that the participant has checked and accepted the request.

6. The peer sends an acceptance notification to the Gateway. This informs the Gateway as early as possible about participating and rejecting peers.

7. The Task Description includes a label that describes which type of input information should be used. The participant is able to recognize the label because it has sent the same label—indicating the availability of some information type—during its pairing. In this step, it has to gather the according information.

8. Similarly, the Task Description contains how raw data has to be preprocessed. E.g., that the last ten values should be fetched per peer and their average should be the input for the computation session.

9. Finally, on the side of the peer, the SMC framework is initialized with the preprocessed values and ready to establish connections to the other participants.

Given, the participant does not agree with the prospective computation, it acts otherwise as described in steps 10–11.

10. The peer informs the Gateway about the rejection. This rejection also includes a reason string. This enables logging and later revision. Furthermore, this error description can also be forwarded to the client if the session finally fails.

11. The Gateway acts on the rejection by regenerating the Task Description and its participant set. It is then necessary that the Gateway resends the updated information to all remaining peers. These can then reevaluate whether they agree to the new configuration.

The loop the Gateway performs is guaranteed to terminate as the choice for each participant in the finite set is either to stay or to leave. Each leaving participant is not considered again this session, hence, the participant set can only get smaller with each step. In the worst case it gradually shrinks to zero; in this case the session is finally aborted. In the other case, if no participants rejects, the set does not shrink but the break condition of a converged set is fulfilled.

Then, the sequence continues as follows:

12. The Gateway sends an acknowledgement to all remaining peers. This is necessary as it tells the peers that they do not have to expect another update of the Task Description but that the computation can now begin.

13. Lastly, the Gateway also initializes its SMC framework with the most current Task Description. Typically, it does not contribute an own value to the computation but provides a neutral element which allows participation in the session without influencing the computation itself.

14. This is the first step where the commands issued by the Gateway are forwarded by the peer to its local SMC instance. This allows the Gateway to control the execution of the SMC framework.

Session Phase

After the linking, the actual SMC computation can start. This is performed in the *session phase*.

15. The session is executed via the utilized SMC framework. We do not get into detail here, but refer to Section 8.7.

The next step assumes that the computation has been carried out successfully.

16. After the computation, typically, one or every participant is in possession of the result. Given, a single participant possesses the result, it forwards the result to the Gateway. In the other case, all participants are encouraged to sent their result as a means of error or cheating detection. In the case that the results deviate, they are discarded or a majority vote is performed.

Otherwise, when an error happened, the following is done:

17. The peers inform the Gateway about the occurred error.

18. Then, the running task is scheduled to be repeated.

This phase can also be looped if the Task Description indicated that multiple computations should be performed. This improves efficiency as a new negotiation of the computation context and the establishments of links can be skipped.

8.5.2 Task Description Scheme

We now present the Task Description Scheme (cf. Listing 8.3). It is the target format to which the Gateway has to translate incoming requests so that they become executable for the Gateway in cooperation with the SMC participants.

The *session identifier* can be generated automatically by the Gateway. Its purpose is to uniquely identify individual ongoing sessions. This enables the parallel processing of sessions with the same participants without risking interferences between ongoing sessions.

The *participant map* has to be generated from the set label which is provided by the client. If the client specifies that the group of nodes behind the label `3nd floor in Building A` should compute a given result, the Gateway must be able to resolve this label to a set of real node entities in the network using its directory information (cf. Section 9.3). This set of entities then has to be addressed in the previously explained orchestration interaction

and also be named in the Task Description. The information about the entities is enhanced with a connectable endpoint (e.g., `alpha.local:8000`) and an identifier during this session. This can also be the cryptographic identifier of this very node, e.g., the fingerprint of its certificate. This enables reidentification—also for collaborating peers—even when the endpoint information has changed.

The *sensor type* defines which type of sensor information should be used for the corresponding computation. It maps to the sensor type labels which the peers also use when transmitting their peer properties containing metadata about them (cf. Figure 8.3). The value can either be directly adapted from the client when the same labels have been reused for the client communication, or easily mapped back to the original values the nodes have communicated to the Gateway.

By default, the latest value of the queried nodes is used. This is not always desirable. Instead, it should also be possible to use multiple (preprocessed) data points as peer-side input. An example is smoothing the collected data over a sliding time window. Then, instead of the most recent value, an average of the recent n values should be used. We achieve this by using the *preselector* and the *preprocessing* attributes. The first one allows the specification of a set of input values. These values are preprocessed using the given function. The result of the preprocessing is then considered the input value of the actual SMC computation. This highly improves flexibility of the software to adapt to a higher number of use cases.

The *protocol* attribute specifies which type of SMC computation should be carried out using the specified data. It must correspond to real protocol implementations on the side of each participant addressed in this session. However, presuming that all possible participants have the same software deployed, this requirement is easily fulfilled.

Finally, the *original request* is included in the Task Description. This enables full transparency on the side of the peer, making more insights available about why processing is actually performed and which client is the requester. While the latter information is simply included in the original request, the former can be derived from e.g., the semantic labels initially used to describe which group of peers shall be queried, or deliberately added metadata in the request.

```
{
  sessionid: 3d8b3e197c7c903b7734e28224528554,
  sensor: temperature,
  preselector: last hour,
  preaggregator: avg,
  protocol: average
  participants: {
     37c178c9b4d0dd16b5f131e6ce8a8631:{
      endpoint: alpha.local:8000
    },
     0ce847e87d13729dc01f05ca4ed299d6:{
      endpoint: gamma.local:8000
    },
     fa812a395802f499ae22a2562c4bfd7a:{
      endpoint: epsilon.local:8000
    }
  }
  original_request: {
   grantid: 8b92f26d24c89c9a86225f02dffa9131e8beff81
   ...
   sig: ...
  }
}
```

Listing 8.3: Task Description Scheme

For the translation from client requests as they are suggested in Chapter 9, see Section 9.5.3.

8.6 System Stability and Recovery

In the previous section, we described the ideal case where connections are successfully established, channels are stable and communication is carried out without interruption. During operation, however, there are numerous reasons why communication could fail. Firstly, there are causes on the side of the peers: software failures, lack of resources or power management can lead interruptions of the connection to the Gateway. This is especially the case in a dynamic context with embedded or mobile devices acting as peers. Additionally, these devices can also change the location and vanish from the network. Failures can also happen in the network itself. Especially wireless connections are prone to temporary failures and interruptions.

It is hence vital to mitigate consequences of failed sessions or connections. The aim is to provide a service which is robust and stable in a dynamic context.

8.6.1 Session Recovery

We first address the case that errors occur during the orchestration process. This contains errors during the session preparation as well as during the session phase itself.

The robustness of the orchestration is founded on the fact that the whole task information is encapsulated in a single stateful object. This Task Description, as described in Section 8.5.2, contains all information necessary to perform the orchestration of the desired computation. Furthermore, in a stateful manner, it also holds information about which phases have already been performed and which should be executed next.

Based on the task object, error handling can be generalized to:

Reset Resetting the state of the current failed phase on the side of the Gateway and the peers. This especially includes termination of specifically running child processes (mainly the current SMC computation via the corresponding framework), including their dedicated network connections.

Readjustment Correcting the task information so that they reflect the new circumstances.

Rescheduling Restarting the execution of the updated task in a cleaned up environment.

Errors during Prepare Phase

During the prepare phase, networking errors are inherently considered: Peers failing before step 10 are—by a timeout—considered like peers which reject the Gateway request in step 10 in Figure 8.4. Hence, this case is handled as short-cut opposed to the abovementioned general handling routine.

Peers failing after that step are handled as if the peer failed during the computation session itself.

Errors during Session Phase

During the session phase, the SMC framework itself indicates a failing computation with the help of internal exceptions. These are forwarded as failure report via the peer software to the Gateway as shown in step 17. In this case the computation is aborted by the Gateway if necessary (typically, the SMC computation automatically fails for all other participants when a single peer fails). Afterwards, the failed state is written into the Task Description, allowing the Gateway to retry the same task a predefined number of times. During these retries, the participant set is adjusted, similar to the case where peers deliberately reject to participate.

8.6.2 System State Stabilization

Here, we address the other case, when no orchestration process is currently ongoing. This situation is comparatively easier to handle as no ongoing interactions have to be gracefully shut down.

The goals of this type of error handling are as follows:

Stable System State The Gateway is bothered with the least amount of faulty peers. Especially the detection of faulty peers should be successful before any orchestration process is carried out.

Consistency The state information about peers held by the Gateway is consistent with reality. I.e., the Gateway does not depend on peers which are already gone. Furthermore, the Gateway does not include peers in announcements which are not reachable at the point of the announcement.

Enabling Reconnection In case of temporary failures, the peer will try to reconnect. Hence, the Gateway is able to allow reconnection of formerly connected and failed peers as soon as possible.

Errors in Heart Beat Detection

During the *Start Operation* state (cf. Figure 8.1), the Heartbeat Process is started. Even when no computation is going on, the peer is instructed to send a simple heartbeat message to the Gateway in regular intervals via the established connection to the peer. On the side of the Gateway, a timer is initialized which regularly checks whether heatbeat messages have been sent. When one of the checks yields an empty result, the peer is considered to be gone or that its connection has failed. The peer is then removed from the set of available peers and unlisted from the directory.

On the side of the peer, unsuccessful attempts to send the heartbeat indicate that the connection is lost. Corresponding to Figure 8.1 the peer will then attempt to reconnect.

Permanent Failure

Furthermore, if the peer detects that further reconnection attempts in the *Connection Establishment* phase to the former Gateway yield permanent failures, the peer degrades back into *Discovery* mode and tries to find new appropriate Gateways.

8.7 Generalization: Loose Coupling of SMC

The remaining challenge is to achieve a universally applicable framework that does not depend on a *specific* implementation of SMC. Our approach towards this goal is to add a thin adapter layer between the presented framework and the SMC library of choice.

8.7.1 Premises

In order to make this approach feasible, the SMC library must fulfill some premises, which have already been presented in Section 4.1. There are three aspects also relevant for this:

Universality We aspire a framework which easily allows extensions in terms of use cases and protocols. This can only be achieved sensibly when using an SMC framework which allows composing arbitrary functions by combining protocol primitives instead of only presenting a limited and fixed set of predefined single purpose protocols. Only then, protocol extension becomes an easy task: A new computation sequence is built in a given framework and a label is incorporated which allows referring to this very protocol. Otherwise, each extension of our framework would require to implement a completely new protocol from scratch.

Multiparty Our solution is built for the cooperation of an arbitrarily sized set of peers. In consequence, also the SMC framework should support this case; otherwise it would be the limiting factor regarding applicability.

Open Source The SMC framework should allow modification of its sources or at least provide an extensive API. This is necessary as building an adapter layer must be able to seamlessly interact with the provided code.

8.7.2 Decomposition of the Peer Component

Due to several reasons, decomposition of the peer component is sensible. Firstly, splitting the peer component makes it easier to enable the generic utilization of the aforementioned features without interferences of a specific SMC implementation, secondly functional abstraction of the applied SMC Library becomes feasible. Lastly, it is more than likely that the SMC Library is written in another programming language than our solution.

In consequence, there are two types of gaps which have to be bridged upon adaptation of another SMC Library: a conceptual gap between the generic computation commands the Gateway issues and the methods available by the Library; and a technical gap between the programming language of our framework and the programming language of the actual SMC Library.

We decided to solve both problems simultaneously by defining a RPC API which is used by our framework and has to be served by an implementing adapter on the side of the SMC Library. It is then the task of the adapter to interpret the obtained messages and to trigger the according actions upon the SMC Library.

This allows flexible handling of the SMC Library code in an arbitrary programming language and ensures protocol-conformity without the necessity of adjusting more than the adapter code itself.

Concretely, we specify the following modules for our prototypical solution which supports the library Fresco.

Peer Management This module performs the above described actions and interactions of *discovery* (Sec. 8.2), *pairing* (Sec. 8.3), *orchestration* (Sec. 8.5) and entering peer-side *operation* (Sec. 8.4). These steps are generic for all kinds of SMC frameworks. Hence, they are always carried out in the same manner.

SMC Advisor In operation mode, instead of the Peer Management module, the SMC Advisor receives the messages from the Gateway and forwards them generically to the SMC Adapter.

SMC Adapter This module mediates the interaction of the Advisor or the Gateway on the one side and the SMC Library on the other side. It is written in the programming language of the SMC Library itself. This component has to be specifically crafted when a new SMC Library is employed and aligned to its API. Additionally, in this component, structurally different protocol invocations can be implemented and referenced via specific protocol labels as shown in Listing 8.3.

SMC Library This module is the third-party code which performs the actual computations.

Local Message Forwarding

In operation mode, the Peer Management module delegates message reception to the SMC Advisor. This is simply achieved by handing over the stream object, which enables message

reception. The SMC Advisor holds a local socket connection to the SMC Adapter. The obtained message is then passed along this connection and forwarded without modification. Finally, the SMC Adapter interprets the message and triggers the appropriate actions on the SMC Library.

8.7.3 Peer-Internal Interaction

The whole interaction between the aforementioned modules is shown in Figure 8.5. In alignment with Figure 8.4, we reiterate and detail some interactions which have already been described without focusing on peer-internal interaction.

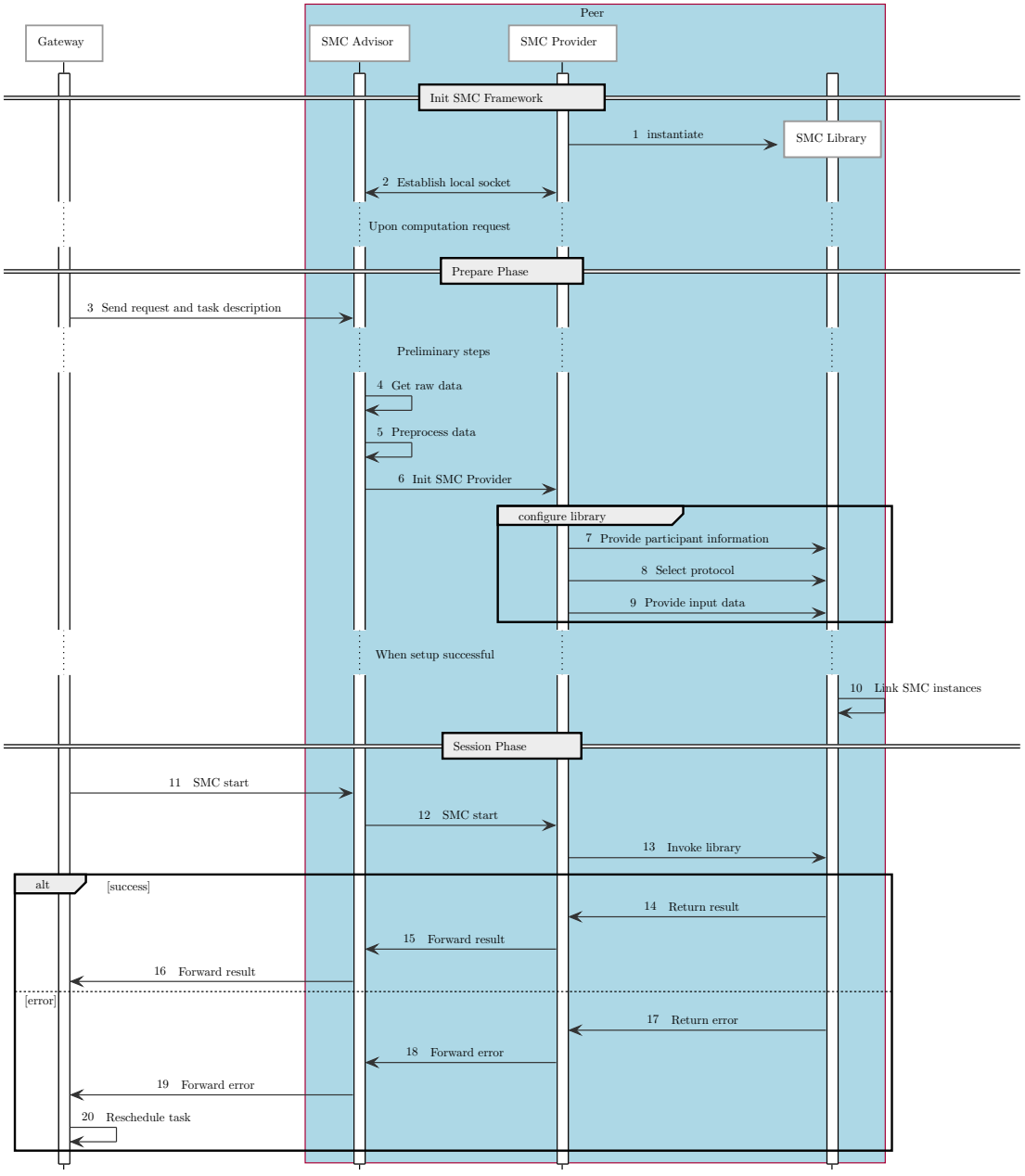

Figure 8.5: Peer-Internal Interaction

When the whole peer component is started, the SMC provider is also started in a local process. This in turn sets up the SMC Library as far as necessary (Step 1). A local socket is established which later allows message exchange (Step 2). Upon an incoming request from the Gateway during prepare phase (Step 3), the SMC Advisor decodes the Task Description, fetches the corresponding data locally (Step 4), preprocesses it (Step 5) and initializes the

SMC Provider (Step 6). Herefor, the participant set information is transmitted (Step 7), the according protocol is selected (Step 8) and the formerly fetched input data is handed over to the SMC Library (Step 9). After the session has been acknowledged by the Gateway, a communication link is set up via the SMC Library (Step 10). The participants are now ready to perform the computation. It is invoked by the Gateway (Step 11), forwarded by the SMC Advisor (Step 12) and the SMC Provider (Step 13). Then, the actual computation takes place. After it finished successfully, the response is returned by the SMC Library to the SMC Provider (Step 14), forwarded by the SMC Advisor (Step 15) and finally received by the Gateway (Step 16). Then, consistency checks can be performed by the Gateway and the whole session is finished. In case of failure, instead, an error is returned (Step 17), also forwarded (Step 18) and then received by the Gateway (Step 19). This typically invokes the rescheduling of the task (Step 20) as already shown in Figure 8.4.

8.7.4 Session Multiplexing

The previously described approach of decoupling yields another, easy to achieve benefit: Settings are imaginable where requests come in high frequency or structured in bursts. It would then be desirable that our solution is able to handle multiple computations in parallel.

Assuming that the library itself is able to handle computations in parallel[2], the following directions must be followed in order to achieve session multiplexing:

Instead of using a single instance of the SMC Library in the SMC Provider, a map *instances* : *SessionID* $\rightarrow$ *SMC Library Instance* is created. All commands to be executed on the SMC Library are then prepended with a step which fetches the instance corresponding to the current session ID; the commands are then executed using this very instance.

There are no structural changes necessary in the other components. However, the Task Descriptions (cf. Listing 8.3) must be created so that the peer endpoints described within it (cf. Listing 9.7) are mutually exclusive for all parallely handled sessions.

8.8 Key Contributions of this Chapter

This chapter addressed research question Q4. We presented a management framework which addresses the idiosyncrasies of dynamic environments and enables flexible, dynamic and robust application of SMC in such contexts.

The main challenges which have been covered are as follows:

Peer Privacy The main goals of privacy-preserving computation (RA.11) are mainly achieved by the utilization of SMC itself. It is not necessary to share individual and confidential input data with any other party. Correspondingly, the security properties of our solution mainly depend on the chosen SMC realization. Concomitantly, the computation result does not give any insights on the participants input data. Hence, unlinkability is achieved[3].

Further regarded privacy goals were accountability (RA.16), intervenability (RA.14), and request transparency (RA.13) for peers. These are supported by requiring client requests to be performed in a serialized and authenticated form which can be forwarded to the computing peers. This gives them full insights into data requests.

[2]It is possible with Fresco as long as different ports are used for different computations.

[3]Note: This depends on the exact protocols to be used. When applying non-aggregating protocols like the identification of maximum value among the inputs, raw inputs are leaked. In some cases, background knowledge then enables an attacker to link this result to the corresponding input party.

Combined with the exclusive possession of their own data (instead of central storage) they have full control over it when deciding whether to accept an incoming Computation Request (request veto).

Self-Management Discovery of Gateways (RA.6) is realized by Zero Configuration Networking. When a new peer is added to the network, it is able to detect the available Gateways and choose one of them depending on their service announcements. Afterwards they perform a pairing where metadata—enabling group creation (RA.7)—and cryptographic material—establishing trust and enabling secure channels (RA.12)—is exchanged.

Computation Orchestration When the pairing was successful, the peer is added to the list of available and computation-ready peers of the connected Gateway. The Gateway establishes a control channel for the peer which allows the former to announce incoming Computation Requests and to invoke the corresponding computation on all addressed peers (RA.8). After computation, the channel can be used to validate the result by obtaining the results of all participating peers and checking for equality.

Robustness and Recovery As SMC is not trivially resilient against host or connection failures, robustness and recovery is necessary. We achieve initial robustness of the system (RA.9) by using the control channel and a heartbeat mechanism to detect failing peers. The Gateway removes them from the list of active peers in order to prevent incoming computations which address unavailable nodes.

When failures happen during computation, it does not to finish successfully. In that case, the Gateway tries to detect the failed peer via the previously mentioned mechanisms and restarts the computation (RA.10) without that peer, if desired by the client.

Directory Service The requirement of a directory service (RA.3) is not yet completely fulfilled, but is supported by this framework, as it provides the means to update directory information during the pairing step. The actual realization of the directory service is described in Chapter 9.

The remaining requirements from Chapter 7 (RA.1, RA.2, RA.4, RA.5) address the service abstraction of our solution, including access control against clients (RA.15). They are addressed in Chapter 9.

8.9 Statement on Author's Contributions

The findings of this chapter have been published in the following paper:
M. von Maltitz, S. Smarzly, H. Kinkelin, and G. Carle. A Management Framework for Secure Multiparty Computation in Dynamic Environments. In *Proceedings of 30th IEEE/IFIP Network Operations and Management Symposium*, Taipei, Taiwan, 2018. IEEE (reference [vMSKC18]). The publication builds on results by the Master's Thesis advised by the author of this thesis and by Holger Kinkelin and supervised by Georg Carle. Sections 8.2, 8.3, 8.4, 8.5 and 8.6 provide more details compared to the publication's architecture section. The peer state machine in the thesis (Figure 8.1) has more details than the reduced variant in the publication.

The author contributed to the results in the publication and this chapter in the following ways: the requirements were proposed by the author and refined by Stefan Smarzly. The overall idea, the concept and the initial architecture were designed by the author of this thesis. The author also contributed to detailed design and implementation decisions during Stefan Smarzly's Master's Thesis. The overall service architecture provided by the author has then been detailed and implemented by Stefan Smarzly. Conceptual decisions have been made in cooperation. As an exception, the phase concept was developed solely by Stefan Smarzly while the Task Description Scheme was provided by the author. Stefan Smarzly furthermore developed the state machine for the Gateway and peer components according to the interaction requirements defined by the author of this thesis. The same is true for the approach of decoupling the framework from a specific SMC implementation (cf. SMC Advisor and SMC Provider).

9. Private and Transparent Data Querying

In the last chapter, we created a Gateway which acts as a *primus inter pares* among the computing nodes. It is able to start and orchestrate computation sessions. Correspondingly, it defines which computation is actually carried out in a given session. In this chapter, we further extend the abilities of this Gateway. We provide measures which enable it to offer SMC computations as a service to third parties, which we will call *clients*.

Objective

We address Research Question Q5. Our aim is to extend the developed Gateway to become a fully privacy-preserving service. In order to do so, the following features are provided: like peers beforehand, clients can automatically detect present Gateways in their network. Clients can request metadata that tells them which types of information are available via this Gateway. They can request access to that information from the Gateway and the Gateway handles the permissions of each client. Finally and if granted access, clients can access specified data periodically and request the execution of corresponding computations.

9.1 Architecture

We realize the querying functionality as three submodules for the Gateway: the *Directory Service (DS)* is filled with information provided by peers during their pairing phase. That is the metadata which can later inform clients about possible computations. *Client-faced access control* allows clients to request access to selected data and the Gateway handles the granting or rejection of these access requests. Clients having permission are allowed to periodically obtain certain data. This is performed by sending Computation Requests to the *Access API* of the Gateway. The Gateway then translates the request—with help of its stored metadata—into all information necessary to establish a computation session. This session is carried out by the means provided in the last chapter. When the result is calculated, the Gateway sends the result back as response to the client.

9.2 Access API

The client is given access to different functionality of the Gateway. This encompasses access to the Directory Service (DS) (Section 9.3), requesting permission to perform certain computations (Section 9.4), and actually requesting computations (Section 9.5).

Access is given via a standard REST-interface. The endpoints and the corresponding interaction are directly described in the mentioned sections.

9.3 Directory Service

Before clients can perform computations and access the information available by the peers, they need to be informed about their possibilities. For that, the DS provides metadata about input data available from the peers and their offered computation protocols. This directory is dynamically filled and updated by the peers and can be queried by the clients.

9.3.1 Purpose

Data sources are heterogeneous regarding different aspects: each node is equipped with an individual combination of sensors. This combination of sensors specifies to which computations the node can potentially contribute.

Further characteristics are the location, ownership of the node, social interconnectedness of the node owners or the logical structure of the building (e.g., departments of a company).

All these dimensions are helpful indicators which groups of nodes might be of interest for performing collaborative computations at runtime. The reason is that the values of these dimensions allow the creation of semantically relevant groups: it may be of interest how much power is consumed by each department or whether there are still people present on a given floor.

Hence, it is advantageous to enable making this meta information explicit in each participant. The anticipated benefits are as follows:

Participant-side

On the side of the participants, this meta information allows the automatic creation of relevant groups for collaboration or inclusion of new nodes into existing groups. This supports auto-configuration. New nodes should automatically become available in the according groups.

Example On a floor, a multitude of electric sockets is already equipped with sensor nodes to collect power consumption information. When a new node is added to a formerly not equipped power socket, the Gateway is informed that consumption calculations concerning the whole floor now have to include this newly added node.

Client-side

On the side of the clients, this meta information can be used to abstract and decouple from infrastructural details and to improve data selection. Instead of specifying which exact group of participants a client wants to poll, it uses higher level labels for whole groups of nodes. This facilitates configuration and simultaneously makes it more robust against changes.

On the one hand, this allows clients to stay on the abstraction level of the desired result information instead of coping with the underlying collection infrastructure. On the other hand, it specifies which information clients are allowed to obtain and prevents them to craft arbitrary computations.

Example In the previous example, a new node has been automatically added to an existing group. However, the client which requests the summative consumption data does not need any changes to adapt to the newly added node.

Developing this approach further, presets from participant groups and possible computations can be built.

9.3.2 Content Generation

In Step 10 and 11 of the peer pairing process (Figure 8.3 on page 128), we describe that the peer and the Gateway exchange and store metadata. This metadata provided by the peer can be differentiated into three types: the first type is technically necessary for performing SMC computations. It encompasses the peer's certificate, an ID derived from that certificate and endpoint information which enable connection establishment. The second type is metadata which is semantically necessary for computation: the Gateway must know which sensor types are available at a given peer and which sets of preselectors and preprocessing functions it provides. This enables selection of peers which are compatible for cooperation. The third type becomes necessary for this part of our architecture. In order to enable third parties, i.e., clients, to request computations, they must be able to query the information of interest in an appropriate manner. Therefor, peers can send further custom key/value pairs of metadata which are then stored alongside the other information. In our concrete case, we use this generic method in particular to store group labels and structured location information in order to enable derivation of semantically sensible peer sets for querying. The Gateway can create basic groups by simply combining all peers which feature the same value of a given label. Furthermore, more sophisticated logic or manual mappings can be used to create further groups from these initial labels.

Example A peer has a hierarchical location `04.03.021` given by an organizational room numbering scheme. Here, `04` is the building, `03` is the building part, and `021` is the room in the given part. Additionally, the peer possesses labels for its own type, e.g., `heater`, and the type of the room, e.g., `kitchen`. Similarly, further labels can be defined, like the `owner` of a room in the case that different rooms in a building are used by different stakeholders. The peer would then transport metadata as depicted in Listing 9.1.

```
{
  id:37c178c9b4d0dd16b5f131e6ce8a8631,
  location:04.03.021,
  endpoint: alpha.local:8000,
  preselectors: [last value, last hour, last 6 hours, ...]
  preprocessors: [avg, mean, max, min, ...]
  protocols: [avg, rank, intersection, ...]
  type: heater,
  inputs: [temperature, humidity]
  roomtype: kitchen,
  owner: ABC corp.
}
```

Listing 9.1: Peer Metadata

This metadata can be preprocessed by the Gateway before storage. Exemplarily, the location can be split into three more attributes, featuring the building, the part and the room label separately. Based on this information, semantically sensible sets provided by the Gateway to the clients are then merely logical predicates over these key/value pairs. E.g., the group of heaters in all kitchens is $\texttt{type} = \texttt{heater} \wedge \texttt{roomtype} = \texttt{kitchen}$ and all temperature data of a building part can be selected as $\texttt{building} = \texttt{04} \wedge \texttt{buildingpart} = \texttt{03} \wedge \texttt{temperature} \in \texttt{inputs}$. These can be built automatically by rules or manually by the administrator of the Gateway.

9.3.3 Metadata Queries

In the most general case, a client intends to get a full overview of the data available. When invoking `GET /directory`, it informs about all possible requests via this Gateway. The result is the post-processed metadata of the peers, now featuring sets of peers combined by

predicates and matching input/preselector/preprocessor/protocol combinations (cf. Listing 9.2). For better legibility and parsability, attributes known to be always present are also directly reflected as attribute keys in the data structure, while custom attributes are combined into the `predicate`. While this is a logical expression featuring labels provided by the peers, the Gateway can also come up with arbitrary labels. I.e., instead of specifying `type` = `hvac_sensor_station` ∧ `roomtype` = `kitchen` ∧ `building` = 04 ∧ `buildingpart` = 03 the Gateway could define the label `Environment information in the kitchens of 04.03.*`. We also applied this method in Section 8.5.2 when first considering computation requests.

```
{
 data: [
 {
   predicate: type = heater ∧ roomtype ∈ [kitchen, meetingroom]
   preselector: [last value, last hour, last 6 hours, ...]
   preprocessor: avg
   protocol: average
   input: [temperature, humidity]
 },
 {...},
 ...
 ]
}
```

Listing 9.2: GET /directory

We abstract from a concrete JSON-like serialization of the predicate. This could be trivially achieved by structures as shown in Listing 9.3.

```
    {
    ...
    predicate: and:{
                type: heater,
                 roomtype:{
                   in: [kitchen, meetingroom]
                 }
              }
    ...
    }
```

Listing 9.3: Realization of Predicates

The metadata request can be arbitrarily specified by POSTing a subset of the data structure in Listing 9.2. E.g., when defining `input: temperature`, only results are returned which fulfill that requirement.

In the extreme case, the Gateway could provide every possible combination of values of the above mentioned attributes. However, this is not desirable due to two reasons: firstly, many combinations do not make sense. It is questionable whether it is useful to compute the average of all minimums from a heterogeneous set of different room types. Similarly, for the preselector `last value`, no other preprocessor than the identity function is useful. Secondly, some combinations might be considered to be too privacy invasive: returning the average of the last value of a considerably specific set (possibly featuring only a single peer) would simply reveal the data providers private value. Performed repeatedly, this would allow tracking of certain sensor values.

These considerations show that the set of available computations must be determined with thought and consideration in order to achieve a sensible mechanism of access control as described in the next section.

9.4 Client-faced Access Control

In the following, we consider RA.15. We begin with a refinement of the security considerations, then we provide an overview of our approach followed by the technical details.

9.4.1 Refined Security and Privacy Model

The content of this section has also been published as Section 2-C of [vMBC19]. The text was written completely by the author of this thesis.

Assets

The main asset to be protected is the individual raw data of sensor platforms. We assume them to be owned each by the respective data subject, i.e., the person(s) about which the platform gathers information. This is given in use cases where a single smart building is inhabited by different parties. Examples for this are smart hotels, smart houses with individual rental apartments, and can also be the case in smart office buildings if employees have dedicated offices. The necessity for data protection is based on the possibility that sensor data gives insights about the presence and behavior of individuals [RZL13].

Protection Goals

Considering the mentioned assets, we understand security to be confidentiality of this very raw data. However, confidentiality may not hinder all processing of the data. Instead, a privacy-preserving access must be designed, meaning data access which is controllable by and accountable for the data owners. Following our privacy background in Chapter 2, we use the established privacy protection goals and given them the following meaning in our context:

Data minimization Information is only derived from raw data if it is actually needed by any client service. The purpose is known before information is created.

Unlinkability Information made available to clients does not allow recovering contributions of individual single peers. Correlations between individual peers should not be possible by client accessible data.

Transparency Peers should know what information is derived from their data and for which purpose this information is used.

Intervenability Based on this preliminary knowledge, they should remain in control of their data by deciding which computations may be carried out.

Attacks

In classical architectures, raw sensor data is pushed from sensor platforms to a centralized middleware. There, data analysis and processing is performed. The results are then made available to clients.

Consequently, previously mentioned protection goals are not or only partially fulfilled. Confidentiality is not given, since raw data is forwarded to and stored on a third-party middleware. The middleware becomes a single point of attack and a high value target. Data minimization is not enforced since arbitrary derivations of information can be performed after its collection. Concomitantly, the purpose of data is not controllable after collection, i.e., using the same data for other purposes is possible at any time in the future. In particular, new information can be generated by correlating raw data of different

sources. All this is possible without the individuals knowing and being able to intervene in the processes.

In our architecture, these attacks are mitigated, since raw data stays on the peers, clients only obtain the post-processed information they have been granted access beforehand. That information is generated by SMC in a privacy-preserving manner and the Gateway only orchestrates data processing while not having access to raw data as a classical middleware would.

The main goal of our approach is that only permitted requests of clients are answered and only the requesting clients obtain the corresponding result. This implies the following set of premises, from which we derive each attacker by the attempt to circumvent one of them. Access requires permission (A.1). Given permissions are legitimate (A.2). If a permission is valid, it was given by an accepted authority (A.3). Only the authorized client can use a given permission (A.4). Validity of permissions is correctly checked (A.5). Data contribution by peers only happens if the validity check was positive (A.6). Results of valid requests do not leak to third parties while being transmitted to the authorized clients (A.7). The results have not been modified on the way to the clients (A.8).

Further attacks are possible but out of scope of this work: malicious peers can try to obtain information from other peers or provide wrong results. We exclude the former since it has to be addressed on the level of the SMC protocols and the latter since the correctness of input values provided by peers is out of scope of realizing secure computation (cf. [CDN15, p. 11]). Lastly, clients can try to correlate information they were able to obtain. This is excluded since it depends on the exact choice of available computation queries.

Trust

An ultimate design goal is to reduce the amount of components which have to be trusted to handle private raw data faithfully. Our architecture has been designed to avoid single points of attack and high value targets holding private raw data from several parties. The remaining trust is diversified: we associate each sensor platform with individual users. These users trust their respective platform to faithfully collect and store their data. This assumption does not strongly differ from assumptions in classical architectures: in any case, by generating it, sensors have access to privacy-critical data. Furthermore, all trust requirements for the used SMC protocol realization apply.

9.4.2 Overview

The overall requirement for access control is that peers should stay in control of their data. I.e., they should be able to decide when and how it is used (RA.14). In the last chapter we already provided a veto mechanism which implements a final decision performed by every peer which participates in a computation. This decision, however, is implicit access control at runtime. Effectively, clients cannot know in advance which requests will be allowed or not. More permission transparency should be provided for the clients.

We hence introduce Permission Grants which clearly and explicitly state the permissions of a client. They are manifested as authorization certificates bound to and held by individual clients. The Permission Grants are created by the Gateway, aiming to anticipate and mirror the decisions the peers would make. This enables clients to know about their abilities and also provides more insights for logging or accountability purposes. We model the capability of the Gateway to assess whether a client should be given permission to request a certain query as function $\Phi : (query, client, context) \rightarrow \{true, false\}$. It represents an access control structure which takes a tuple of a query, a requesting client and a context and returns whether or not access is permitted.

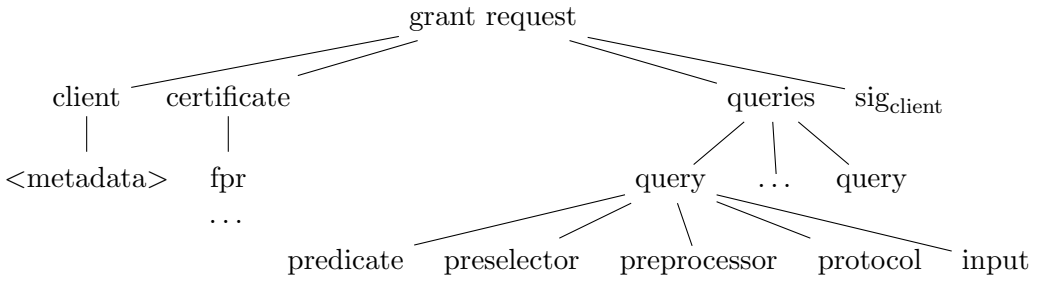

Figure 9.1: Structure of a Permission Grant Request

Permission Grants should also mimic the granularity of the peers' veto ability. I.e., they should be located on the level of individual request types. This is achieved by honoring the metadata available about the peers and creating Permission Grants for specific metadata combinations. These consider the semantically necessary attributes *protocol*, *preselector*, *preprocessor* and the custom key/value pairs as described in Section 9.3.

Besides permission transparency for clients, the approach also yields several runtime benefits: when permissions are granted and then persisted in advance, there is no need for complex decision mechanisms during the actual Computation Request. In other words, complex authorization decisions and permission granting on the one hand and permission checking during Computation Requests on the other hand can be separated. In particular, permission checking does not need an online connection to some authorization system. This is especially useful if the authorization system in turn has to negotiate the permissions to be allowed with the corresponding peers.

9.4.3 Permission Grant Request

The client has to request a Permission Grant from the Gateway[1] stating that it is allowed to request certain types of information. For that purpose, the client first retrieves the metadata of currently available data sources from the Gateway as described in the previous section. Based on this information the client decides (automatically or with manual intervention) which information access should be requested. Thereof, it derives a Permission Grant Request with a structure as shown in Figure 9.1.

The cryptographic identity of the client is the `fingerprint` of its certificate. `Client` is a hash containing further metadata like a `purpose` for accountability reasons. The `queries` array contains all requested grants following the metadata structure shown in Listing 9.1. Finally, the whole request is `sig`ned in order to guarantee authenticity.

The Permission Grant Request is then used to retrieve the Permission Grant using the protocol in Figure 9.2. The Permission Grant is bound to the cryptographic identity of the client.

1. The client sends the Permission Grant Request to the client.
2. The Gateway first validates the request itself. Given a request $r_{c,grant}$ of client c, two facts are checked: the validity of the client certificate (Equation 9.1) and the client's possession of corresponding private key.[2] Similarly, validity of the request signature

[1] We state above that issuance of Permission Grants and their validation during requests can be separately deployed on different systems. In the following we assume for simplicity that the functionality is also located on the Gateway. If necessary, we differentiate both possible cases.

[2] When using TLS, this is already handled during TLS session establishment.

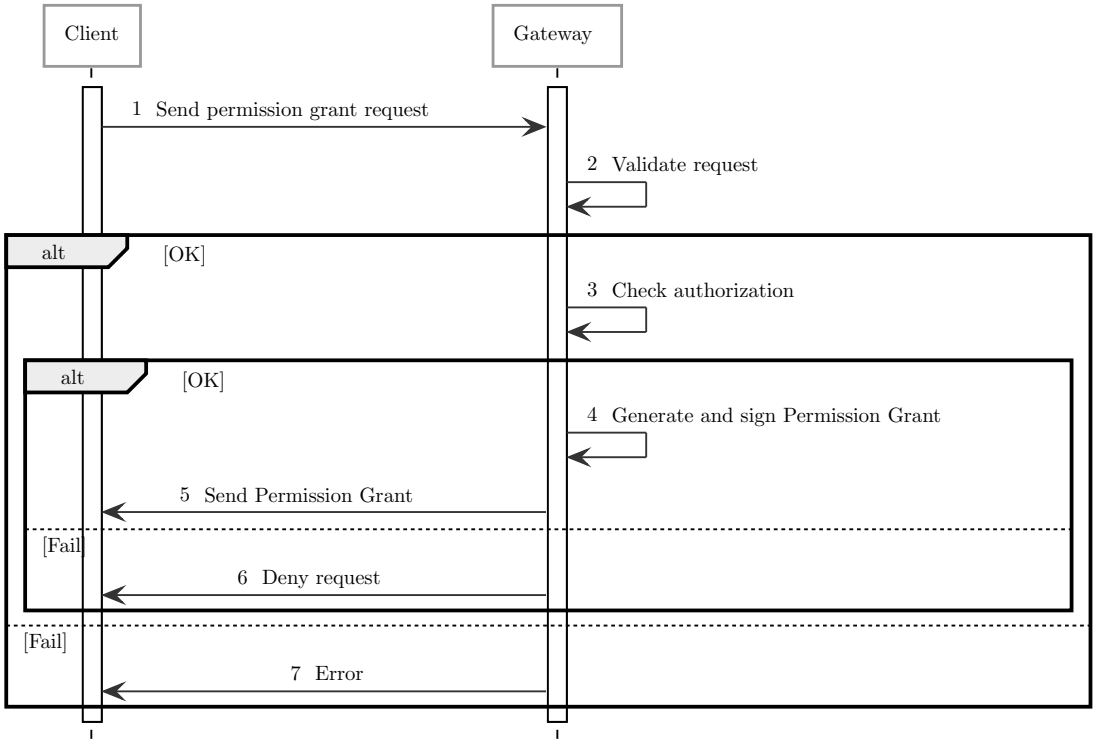

Figure 9.2: Request of a Permission Grant

(Equation 9.2) is verified. This can also include arbitrary additional checks which can further constrain the combinations a client is allowed to request. Exemplarily, the type of preselectors or preprocessors can be constrained depending on the type of data or the actual SMC protocol.

3. Then the semantics of the queries are checked (Equation 9.3) using Φ. The parameters are set as follows: the *query* reflects the demanded data in a form as described above. The *client* is represented by its certificate. The *context* is the current state of the Gateway, this encompasses the information about currently connected peers and environment information like the current time. Φ can be arbitrarily complex and can especially be based on interaction with the peers and their decisions. Alternatively, given the peers' decision and veto logic stems from a third-party component, the authorization system can also interact with this instance.

4. If the checks are positive, the Gateway generates a Permission Grant containing the legitimate queries and approves it by its signature.

5. The Permission Grant is then sent to the client.

6. If a check fails, the request is denied and the client is informed about the decision.

7. If the initial validation step fails, the client is informed about its invalid request.

$$verify(r_{c,grant}.cert) \tag{9.1}$$

$$verify(r_{c,grant}.sig_{client}, r_{c,grant}.cert) \tag{9.2}$$

$$\forall q \in r_{c,grant}.queries : \Phi(q, r_{c,grant}.cert, context) \tag{9.3}$$

Given, the protocols has been carried out successfully, the client retrieves a Permission Grant with a structure as in Figure 9.3.

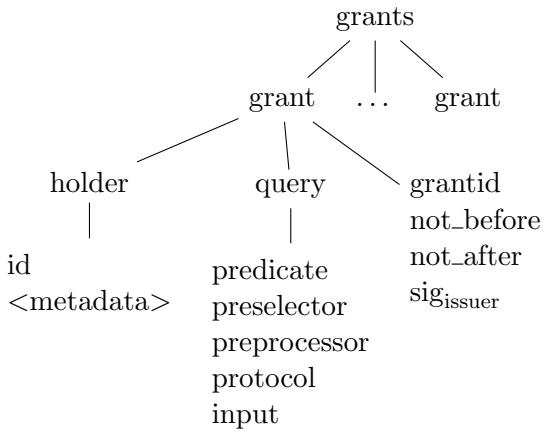

Figure 9.3: Structure of a Permission Grant

The `grants` array contains all accepted grants. The `grant_id` is an auto-generated random string which allows to refer to a grant without sending the whole structure. The attributes `not_before` and `not_after` define the time interval of validity. All other attributes are as described in Figure 9.1. By featuring the fingerprint of the client's identity, the Permission Grant is bound to the client. In consequence, reuse and sharing of permissions is rendered infeasible. The signature per grant is performed by the Gateway asserting that the request type in question is indeed granted and allowed. Several of the provided attributes per grant are redundant over the set of permitted grants. Each grant is then signed individually. This is intentional. It allows a client to only send the corresponding grant when performing a Computation Request and permits different expiration dates per grant.

Since the clients stay in control of their own permissions by storing the Permission Grant themselves, revocation of permissions becomes non-trivial for the Gateway: permissions are not represented as information which is easily changeable by the Gateway. Instead, they are documents held by the clients.

Contrary to the Permission Grants, the Gateway's authorization logic can adapt when the set of available peers or their very decisions change. Hence, we address this problem by adding an expiration date to the Permission Grant. Each permission is not valid indefinitely but only during a constrained time window. A Permission Grant can be renewed by the owning client, but when it is, adaptations in the authorization logic are also reflected in the grants. This can lead to changing permissions in the Permission Grants over time. The duration of validity should depend on the stability of the topology (i.e., how often new peers have to be incorporated into the authorization logic) and the effort to create or renew certificates (i.e., whether or not all peers have to be contacted at authorization time).

In fact, privacy-preservation of raw data does not depend on the duration of validity: in any case, peers remain able to veto against announced computations, notwithstanding the former decision of the Gateway based on the Permission Grant.

Section 9.5.2 details how the Permission Grant is used during a Computation Request.

Example The public display in building part 04.03.* requests permission to access the average temperature in the building part 04.02.* (and further permissions omitted here). This is performed with a request as shown in Listing 9.4.

```
{
  holder: {
    id: aadbbc22238fc401a127bb38ddf41ddb089ef012,
    purpose: "Public display in building 04.03.*"
  },
  certificate: ...,
  queries:
  [
   {
     predicate: building = 04 ∧ buildingpart = 02,
     preselector: last hour,
     preprocessor: avg,
     protocol: average,
     input: temperature
   },
   { ... },
  ]
  sig_client: ...
}
```

Listing 9.4: Permission Grant Request – Example

The Gateway answers with a Permission Grant as shown in Listing 9.5. The original query and the holder are copied into the grant. A time interval of validity is added. The signature is replaced by the issuer's signature.

```
{
  grants:
  [
   {
     holder: {
       id: aadbbc22238fc401a127bb38ddf41ddb089ef012,
       purpose: "Public display in building 04.03.*"
     },
     query: {
       predicate: building = 04 ∧ buildingpart = 02,
       preselector: last hour,
       preprocessor: avg,
       protocol: average,
       input: temperature
     },
     grantid: 8b92f26d24c89c9a86225f02dffa9131e8beff81,
     not_before: 1516696572,
     not_after: 1546210800,
     sig_issuer: ...
   },
   { ... },
  ]
}
```

Listing 9.5: Permission Grant – Example

9.5 Computation Request

The Computation Request is the ultimate aim all previously presented components support. It allows the client to request aggregated sensor data provided by our distributed system from the Gateway. This means triggering an SMC computation among a set of peers and forwarding the reply to the client via the Gateway. For that, a clients creates a signed request stating what to compute and its permission to request this data. The Gateway verifies whether the request is legitimate, transforms it into Task Description and proceeds as described in Figure 8.4 on page 131 and Section 8.5.1.

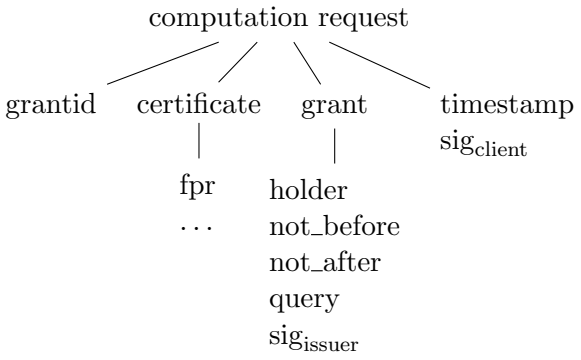

Figure 9.4: Structure of a Computation Request

9.5.1 Request Creation

With the preliminary work done by the DS and the previously described access control, creation of a request merely comprises the selection of the corresponding grant and the statement that exactly this information is currently desired. Technically, the client creates a structure as described in Figure 9.4.

The `request` states the `grantid` of the desired query. The `timestamp` has two goals: firstly, it acts as the reference point for relative preselectors like `last hour`. Secondly, it prevents the possibility to replay formerly valid requests. The `grant` attribute reflects the whole request. It states all necessary details to perform the desired computation. It only needs to be sent if the current Gateway cannot know about issued grants. This is the case when they were issued on a separate system and the grant has not been used before. Otherwise, we suggest a caching solution which makes resending the grant at request time superfluous. The Gateway then has to cache grants which it either issued itself or already received during a previously sent request. The final `sig`nature states that the client whose identity is bound inside the sent grant, indeed intended the stated request at the given time.

Example Given the grant of Listing 9.5, the public display of building part 04.03.* can now issue a computation request to obtain the current value. Such a request is exemplified in Listing 9.6.

```
1  {
2   request: 8b92f26d24c89c9a86225f02dffa9131e8beff81
3   timestamp: 1516839284
4   grant: {
5     holder: {
6       id: aadbbc22238fc401a127bb38ddf41ddb089ef012,
7       purpose: "Public display in building 04.03.*"
8     },
9     query: {
10      predicate: building = 04 ∧ buildingpart = 02,
11      preselector: last hour,
12      preprocessor: avg,
13      protocol: average,
14      input: temperature
15    },
16    grantid: 8b92f26d24c89c9a86225f02dffa9131e8beff81,
17    not_before: 1516696572,
18    not_after: 1546210800,
19    sig_issuer: ...
20  },
```

```
21    sig_client: ...
22    }
```

Listing 9.6: Computation Request

9.5.2 Access Verification

We describe the basic validation process of request received by the Gateway. The whole protocol is shown in Figure 9.5. The client sends a computation request providing a

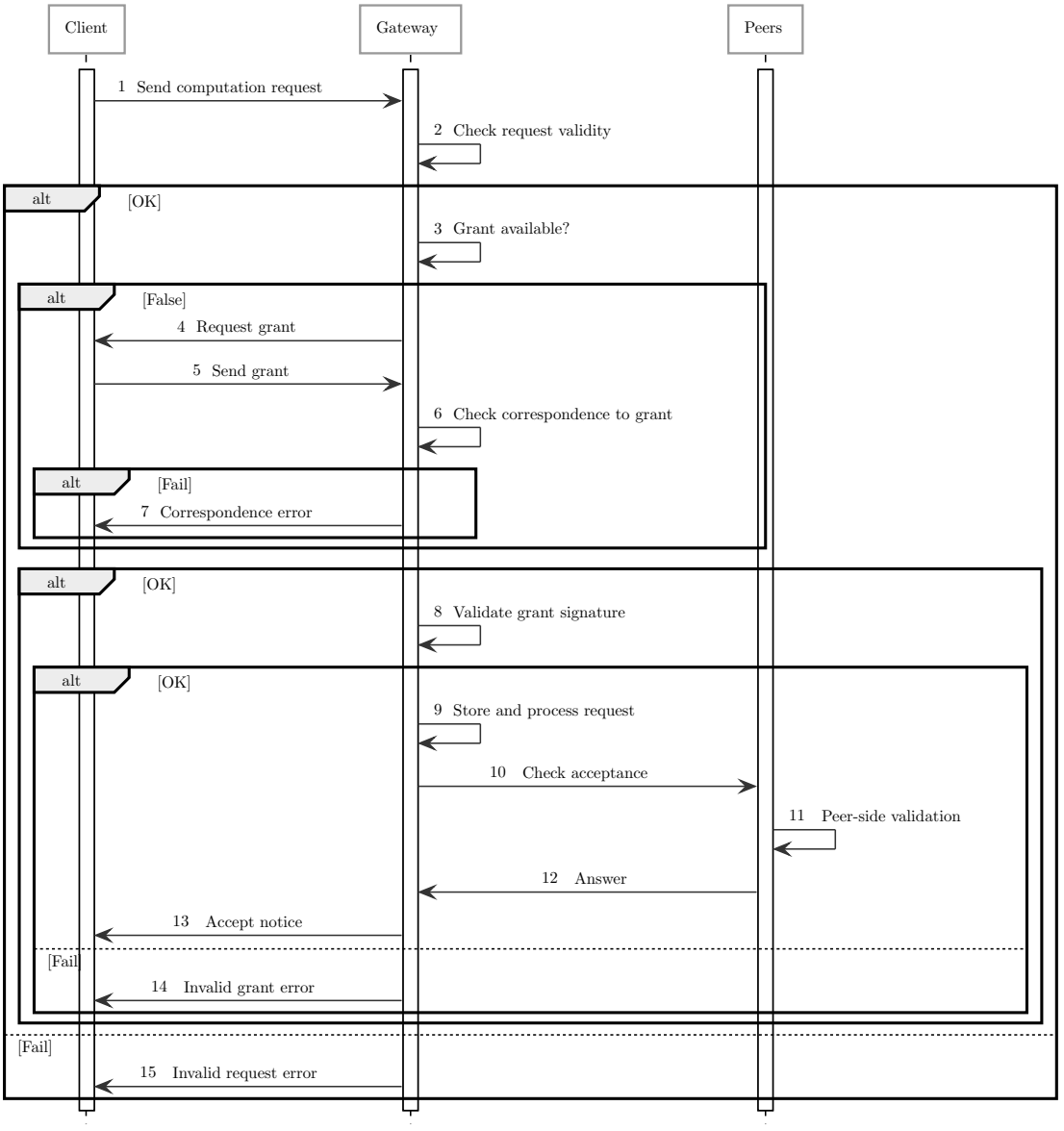

Figure 9.5: Validation of an incoming request

request data structure as described in Section 9.5.1 (Step 1). Given a request $r_{c,comp}$ of client c, the Gateway first checks the syntactical validity and whether the holder matches the requesting client and the client certificate is valid (Equation 9.4). Then it checks the authenticity of the request (Equation 9.5). (Step 2). If this is successful, the Gateway checks whether the referenced grant is sent in the request or already cached (Step 3). If this is not the case, the grant is requested from the client (Step 4) and sent back by it to the Gateway (Step 5). If correspondence of the `request` attribute with the `grantid` is not given (Step 6), the Gateway rejects the request (Step 7). If the grant is accepted syntactically, its

validity is checked by signature verification (Equation 9.6) and checking the expiration dates (Equation 9.7 – 9.8) (Step 8).

On success, the request is accepted by the Gateway, stored and processed (Step 9). The Gateway then announces the request to the peers (Step 10). Besides revalidating abovementioned checks, each peer $p \in P$ can have a set Φ_p of local policies which define how their data may be used and the corresponding privacy constraints. Satisfaction of these policies can also be checked (Equation 9.9) (Step 11). Afterwards, they answer to the Gateway accordingly (Step 12) (these steps sketch the prepare phase depicted in Figure 8.4 on page 131).

Assuming that all involved peers accepted the request, the client is informed about the acceptance and provided a request identification number which allows later result retrieval (Step 13). Lastly, in the error cases if the grant is expired, the signature is invalid, (Step 14) or the request was syntactically invalid (Step 15), it is rejected.

$$r_{c,comp}.grant.holder = c.cert.fpr \wedge verify(c.cert) \tag{9.4}$$

$$verify(sig_{client}, c.cert) \tag{9.5}$$

$$verify(issuer.cert) \wedge verify(sig_{issuer}, issuer.cert) \tag{9.6}$$

$$r_{c,comp}.grant.not\_before \leq now \tag{9.7}$$

$$now \leq r_{c,comp}.grant.not\_after \tag{9.8}$$

$$\Phi_p \vdash (c, r_{c,comp}.query) \tag{9.9}$$

Examples for rules from Φ_p are:

- Trustworthiness of the requesting client.
- Appropriateness of the requesting client for the requested data.
- Appropriateness of the frequency of a certain request. Fine-granular information can also lead to data leaks by tracking of the results.
- Legitimacy and plausibility of the set of participating peer. Sets are illegitimate when the private raw data may leak to the client or the Gateway. This is the case, e.g., when the set only consists of this single peer or the peer suspects collusion of all other participants.

After the validation by the Gateway, the request is transformed into an adequate data structure for the peers in Step 9. We describe this process in greater detail in the next section.

9.5.3 Request Translation

The generation encompasses three steps: initially, a new session identifier is created for later referencing. Multiple attributes are simply copied from the original request: these are `input`, `preselector`, `preprocessors`, `protocol`. Similarly, the `original request` is included. We consider this optional, since it is not strictly necessary for performing the computation, but it is a vital contribution to improving request transparency for all peers. The original request provides further insights which can be used to perform peer-side assessment of the validity of the proposed computation session.

Lastly the `participants` map has to be created. Therefor, the predicate from the original request is taken and resolved using the data from the Directory Service. This can be

simply performed by iterating over all currently available peers and evaluating the provided predicate. These peers are selected and their endpoint data is fetched. This information is encoded in the participant map providing the endpoints for all other peers enabling mutual connection establishment.

The same information is held in memory by the Gateway since the Task Description is then sent out to all selected peers in order to inform them about the upcoming session. The further process is described in Figure 8.4 and Section 8.5.1, where peers is given the possibility to veto against the request or to process it.

Example A full and complete Task Description is shown in Listing 9.7.

```
{
  sessionid: 3d8b3e197c7c903b7734e28224528554,
  input: temperature,
  preselector: last hour,
  preprocessor: avg,
  protocol: average
  participants: {
    37c178c9b4d0dd16b5f131e6ce8a8631:{
      endpoint: alpha.local:8000
    },
    0ce847e87d13729dc01f05ca4ed299d6:{
      endpoint: gamma.local:8000
    },
    fa812a395802f499ae22a2562c4bfd7a:{
      endpoint: epsilon.local:8000
    }
  original_request: {
   request: 8b92f26d24c89c9a86225f02dffa9131e8beff81
   timestamp: 1516839284
   grant: {
     grantid: 8b92f26d24c89c9a86225f02dffa9131e8beff81,
     holder: {
       id: aadbbc22238fc401a127bb38ddf41ddb089ef012,
       purpose: "Public display in building 04.03.*"
     },
     not_before: 1516696572,
     not_after: 1546210800,
     preselector: last hour,
     preprocessor: avg,
     protocol: average,
     input: temperature,
     predicate: building = 04 ∧ buildingpart = 02
     sig: ...
   },
   sig: ...
   }
}
```

Listing 9.7: Full Task Description

9.5.4 Result Receipt

We have shown in Chapter 4 that an SMC computation can take considerable time under certain circumstances. Performing a Computation Request should hence not be blocking for the client. Instead, request answers should be received asynchronously. Two possible options exist. The clients provide a callback during their request. This is an URI which is called later by the Gateway and where the result data is POSTed to. The other option is allowing clients to poll the Gateway regularly for the result.

The callback approach would excel regarding performance: without any preliminary knowledge about the duration of the computation, the result would nevertheless be sent to the client as soon as it is available. However, the process would become more complex: clients would have to expose a reachable endpoint to the Gateway, the network would have to allow connections to the clients (i.e., firewall), and the Gateway would need a connection and retry logic which handles the result forwarding including possible error cases.

Considering the polling approach, setup is simpler: when a Computation Request is sent to the Gateway, the Gateway generates a `sessionid`. This ID can also be sent to the client as a reference. The client is then able to poll the Gateway using the reference until the result is available. This implies only that the client is able to perform asynchronous requests, store an ID and know another message type to send. The Gateway needs the ability to store available results. This is easily feasible using the existing DS.

Since the technical details of these approaches are trivial, we omit them.

A further aspect to consider is result encryption: since the peers are in possession of the client certificate for verification purposes, they can also use the included public key for result encryption. This prevents leakage of the result to any third party including the Gateway.

9.6 Evaluation

The content of this section has also been published as Section 6 of [vMBC19]. The text was written completely by the author of this thesis.

9.6.1 Security and Privacy

We leave data processing of the underlying SMC intact without modification. I.e., the corresponding security properties still apply without constraints: the state of the art [DPSZ12, DKL+13, KOS16, KPR18] is already secure against $n-1$ maliciously colluding peers. Since the data owners' device always participates in the computation when using their data, security of the own raw data is already achieved by guaranteeing that the own device is not compromized. The computation request and the corresponding protocol allow third parties to query for data computed by SMC (RA.1 and RA.2). By doing so, securely computed information becomes accessible to outside clients. Access on the computed results can be controlled by authorization grants required for computation requests (RA.15). This enables controlled data flow when serving a heterogeneous set of deployed services. Making the request verification independent from the access control structure Φ and its *context* parameter, i.e., not relying on complex state of the Gateway for access control but on transferable documents, allows to use the same data structures to fulfill desired privacy protection goals: peers are informed about upcoming computations and their context if they are involved, since the requests include the query and the corresponding authorization grant of the requesting client (RA.13). An authenticated history of data access and usage can be built (RA.16), since authenticated requests can also be persisted by each peer. The signatures of the client, the Gateway and the peer ensure that integrity of request and corresponding grant is protected. Giving peers the ability to verify this request using their own local policies Φ_p and allowing them to veto against requested computations enables intervenability (RA.14). This is based on the fundamental necessity of SMC that computation cannot happen without cooperation of the data possessing peers. As a consequence, data minimization is supported: data sources can make sure that there are not more or different computations performed than they expect to happen.

Regarding possible attacks from Section 9.4.1, our solution performs as follows: the grant represents obtained permissions. Consequently, A.1 is mitigated since the Gateway checks

for the presence of a grant, and the peers are also able to perform this check. Without a grant, a request is not accepted. A.2 is addressed by having the access authority as trust anchor. It is assumed to only issue legitimate grants. This is complemented by enabling the peers to perform semantic checks on the requests. Forging permissions (A.3) is prevented by requiring a valid signature of the issuer on the grant (cf. Figure 9.4). Similarly, an interval of validity is included in the grants to ensure their currency. Furthermore, the possessor of the grant is included in the grant itself, ensuring that only this very client can use the given permission, mitigating A.4. Permission checks performed by the Gateway and the peers include all necessary steps to ensure these abovementioned assumptions. Additionally, they verify whether the query of the current request is permitted by the attached grant. This renders A.5 infeasible. The request is only forwarded to the peers if the grant is found valid in the abovementioned sense by the Gateway. To avoid requiring trust of the peers in the Gateway regarding these checks, all information are also forwarded to the peers. They are able to recheck them themselves. This ensures that data is only made available if the peers had sufficient proof of the validity of the request, mitigating A.6. A.7 is prevented, since the obtained result is encrypted for the receiving client. Similarly, the result cannot be changed by the Gateway (A.8), if the result is signed by the peers.

9.6.2 Performance

In order to perform an assessment of the performance of our solution, we conducted measurements in a testbed using a prototype implemented in python.

As use case we consider a single floor in a smart building with around 10 to 30 peers connected to a single Gateway. The Gateway is assumed to be decent commodity hardware with several CPUs and a sufficient amount of main memory. The network is an intranet with low latency, a typical throughput and no packet loss.

A client is present which performs requests directed to the Gateway. The performance of the Gateway is evaluated.

9.6.2.1 Setup

In the following the describe the setup in which the measurements have been executed. We provide an overview of the hardware used as well as the deployed operating system and all used software. Afterwards, we describe our own implementation and the method of the measurements.

Hardware

We use the same hardware as in Chapter 4: four homogeneous hosts are used which feature a Intel Xeon E3-1265L V2 CPU, having eight cores at 2.50 GHz and a cache size of 8.192 KB as well as 15.780 MB of RAM. All hosts use a 1 Gbit networking interface.

These hosts are connected via a single switch. Network latency is around 0.18 ms and no packet loss occurs.

System

Differently to Chapter 4, all hosts are equipped with Debian Stretch (9.4) using a 4.9.0 Linux kernel. The stack of used software encompasses python 3.5.3 as interpreter, flask 1.0.2 as web framework, and uwsgi 2.0.17.1 as application server for the execution of the prototype. Data storage happens using a mongodb version 3.2.11 and authzforce 8.0.1 [Aut18] is used as authorization backend. We distributed the Gateway, the client and the peers as follows: the Gateway and the client were deployed on individual hosts each. One peer was deployed on a dedicated host. The remaining peers were realized as processes on another single host.

Implementation

The prototype was implemented as python REST application which was built upon flask. Uwsgi is used to serve the application, using a single process and 8 worker threads if not denoted otherwise. Wsgi provides a request queue which holds the connection of incoming requests which could not yet be answered. This queue has been configured to have a length of 100. When the queue is completely filled, further requests are immediately dropped.

During validation of an incoming computation request (cf. Figure 9.5.2), the Gateway contacts peers to inform them about the upcoming computation. Peers themselves then check and acknowledge the request. In consequence, the peer component was also implemented as REST application in order to allow the Gateway to contact them. When contacting the peers, the Gateway spawns a thread for each peer in order to allow simultaneous waiting for all responses.

For testing purposes no actual SMC component was connected to our querying framework. This allows to measure the overhead of our components without depending on the performance characteristics of a chosen SMC implementation.

Method

The client has been scripted to offer identical requests with different frequencies. For each measured frequency step, the client offered the desired load for either 30s (grant request protocol) or 60s (computation request protocol).

We measured the duration of client requests handled by the Gateway (*latency*). The measured time begins when the request is handed to our custom code; it ends when the final response is handed back to uwsgi. In the results, the plotted value per frequency is the median of all executed requests during that window. The vertical bar of each point in the diagram reflects the 0.25 and 0.75 quantiles. For each frequency, we captured the amount of requests the Gateway was able to handle successfully per second (*throughput*). Lastly, we recorded the state of the request queue. Uwsgi allows to query its current degree of saturation via a http stats interface. The queue state was always queried immediately after the last request was sent, ensuring that the complete load has reached the Gateway but was not yet handled by it.

9.6.2.2 Results

In the following, we present the results of our evaluation. In the first part the Grant Request Protocol is focused, afterwards the Computation Request Protocol is taken into consideration.

Grant Request Protocol

The grant request protocol is carried out when a client aims to obtain further access rights to data queries offered by the Gateway. The frequency of this request depends on the volatility of the environment: in a setting where peers and clients are deployed a single time at the beginning of system lifetime and further changes are seldom, requests are performed a small number of times per client. Every change might make further grant requests necessary. In a setting where clients change often or data querying is based on individual user interaction instead of automated predetermined processes, the frequency can increase to a small number of requests per second during peak times. Furthermore, renewal of grants makes repetition of requests necessary. This amount of requests is well supported: even under a load of ≥ 100 requests per second, answer time stays below 20 ms (Figure 9.6). The queue of the Gateway becomes saturated only after 170 requests per second (Figure 9.7). Correspondingly, the throughput stagnates at the level of 170 requests per second (Figure 9.8); all remaining requests are dropped. Since no peer interaction happens, performance is independent of their number.

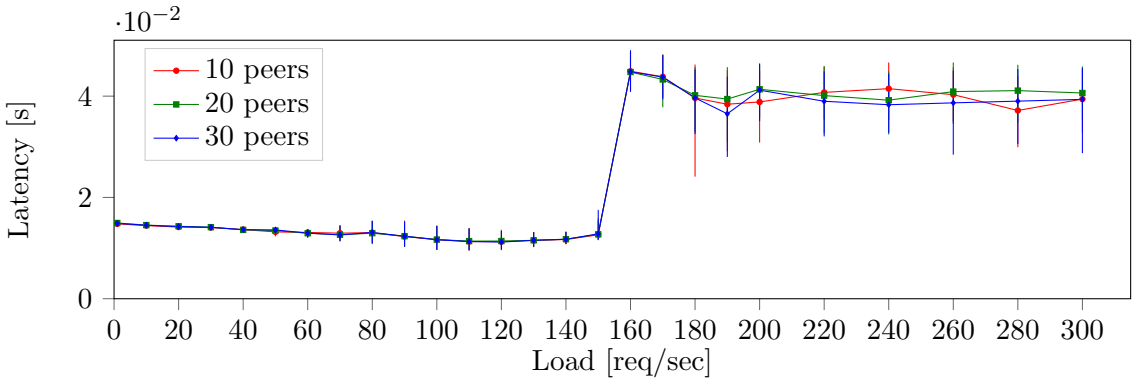

Figure 9.6: Grant Request Protocol: the duration of handling a single request inside the Gateway component depending on the amount of requests performed by the client.

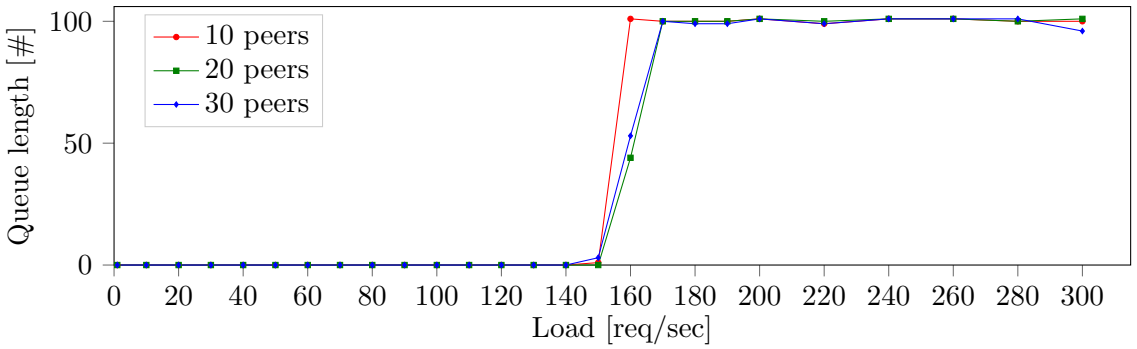

Figure 9.7: Grant Request Protocol: saturation of the uwsgi request queue depending on the amount of requests performed by the client. As long as the queue is not saturated, all requests can be answered successfully. During higher loads, requests are dropped during connection attempt.

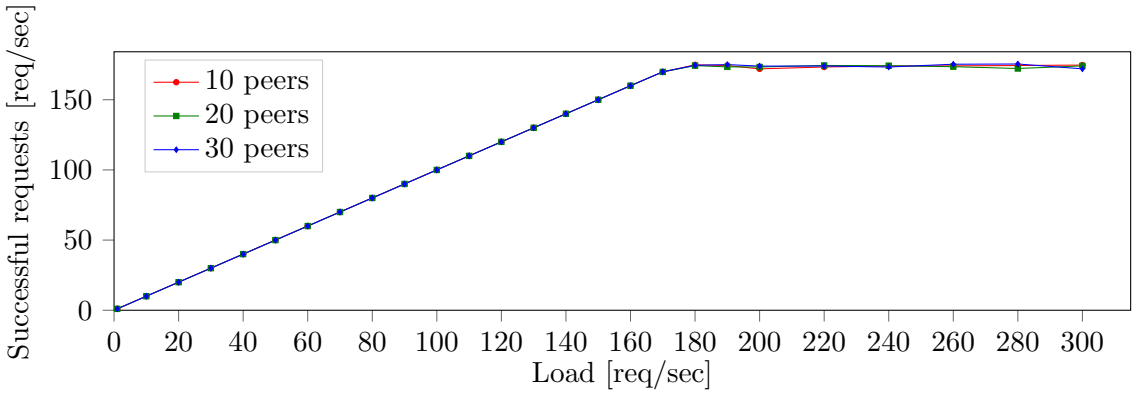

Figure 9.8: Grant Request Protocol: the amount of successfully answered requests depending on the amount of requests performed by the client. Up to a load of 170 requests per second, throughput increases proportionally and no drops occur. Afterwards, the queue is filled and throughput stagnates on this level.

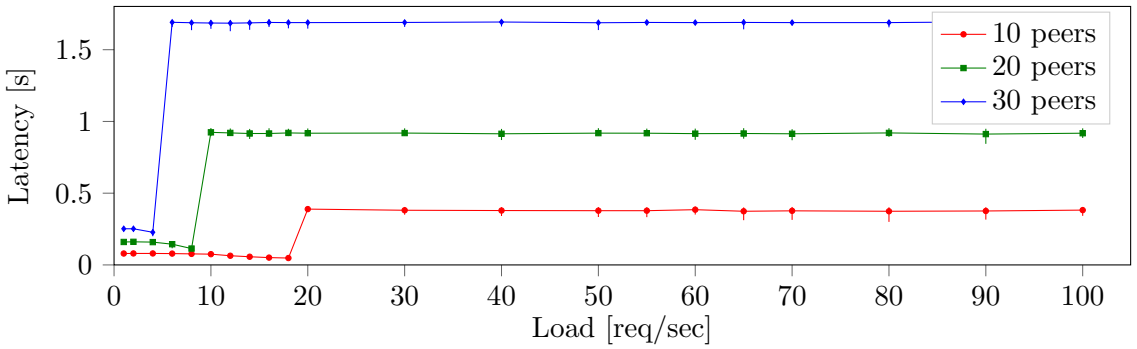

Figure 9.9: Computation Request Protocol: the duration of handling a single request inside the Gateway component depending on the amount of requests performed by the client. This includes forwarding the request to all concerned peers and waiting for their request acceptance. The biggest part of the overall duration is constituted by the cryptographic actions to be taken for the communication with the peers.

Computation Request Protocol

The computation request protocol is always carried out when a computation on actual sensor data is queried. With polling every second per client, and multiple clients being connected, multiple requests per second can be expected.

With 30 peers connected, a single request per second yields a latency of ~250ms. With increasing load this converges to ~1.7 seconds per request (Figure 9.9). Each added peer approximately contributes additional 50 ms. The reason is computational overhead per connection—mainly signing outgoing messages and verifying the signatures of incoming messages—which cannot be handled in parallel due to the global interpreter lock in python. Since SMC itself comes with a comparatively high communication overhead compared to solutions featuring a middleware as trusted third party, an additional overhead of a second seems acceptable in our use case. We can propose several mitigations for this overhead:

- Firstly, we assume a non-prototypical implementation in a language which optimizes more for speed will provide a lower overhead. Especially using a language which provides real parallelization and/or more efficient implementations of the corresponding actions should decrease this overhead.
- Caching can be considered: peers' agreement to computations could be extended by an interval of validity, implicitly allowing later consecutive identical requests without further overhead.
- Our architecture can be applied hierarchically: allowing several Gateways per area which segment peers into semantically sensible groups, being large enough to ensure unlinkability. Recursively, Gateways can in turn function as higher-level peers for further aggregation.

Concomitant with the increase in latency, the request queue is exhausted between 5 to 20 requests per second (Figure 9.10) and a high throughput is inhibited (Figure 9.11). Here, the bottleneck is the CPU of the Gateway. Since this component is not resource-constrained by design, we can mitigate this bottleneck by allowing several processes on the Gateway to run in parallel on multiple cores. Providing 4 uwsgi processes shows that parallelization doubles the throughput (Figure 9.12). Due to these limitations, we understand our approach to be a feasibility result with further potential for optimizations.

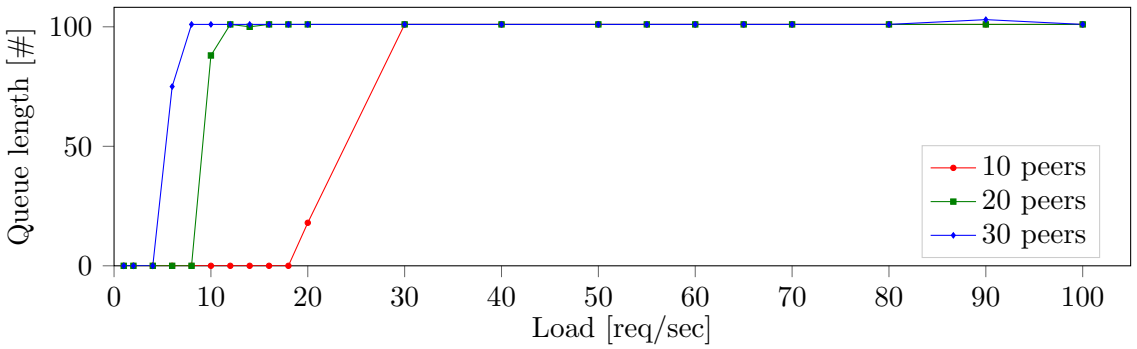

Figure 9.10: Computation Request Protocol: saturation of the uwsgi request queue depending on the amount of requests performed by the client. During each request, the Gateway contacts peers and waits for their response. The longer time needed by each request already causes the queue to fill up when requests come in with a delay of 100–200 ms.

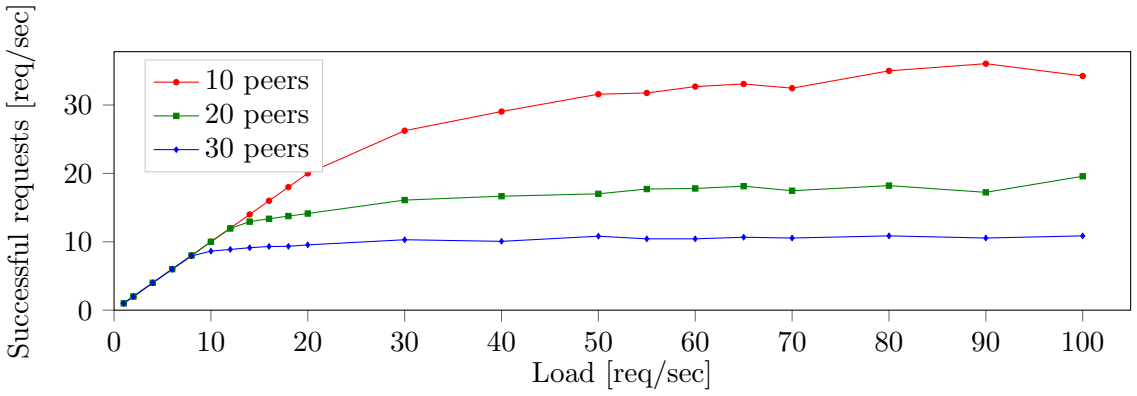

Figure 9.11: Computation Request Protocol: the amount of successfully answered requests depending on the amount of requests performed by the client. Latency of single requests restricts the amount of successful requests, since the request queue is already filled between 5 to 10 requests per second.

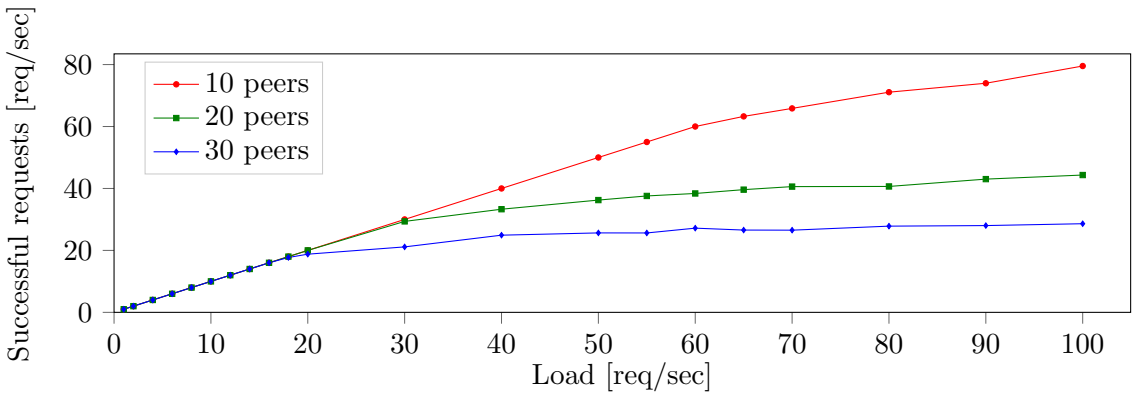

Figure 9.12: Computation Request Protocol: the amount of successfully answered requests depending on the amount of requests performed by the client. A higher amount of Gateway processes correspondingly improves throughput.

9.7 Generalization: Application without SMC

The presented solution shows how SMC technology can be utilized as a service which makes it compatible to client-server architectures.

We yield several benefits here regarding management and privacy. Data consumers do not have to be a part of a distributed or peer-to-peer system in order to access distributed data. Instead, they can use classical client-server architectures which require less changes on their side for compatibility. Regarding privacy, problems of central storage solutions like linkability of data, loss of access transparency for data owners and loss of their control which accesses to allow are effectively solved. Here, while unlinkability is essentially supported by the SMC part of the solution—since raw data has never to be accessed by any party except the corresponding data owner—all other privacy improvements have to be accounted to the distributed storage of data and the access mechanisms presented in this chapter.

That means, even if no SMC solution is at the heart of a similar system, essential privacy goals can be fulfilled using our access solution. It can be applied if the following premises are fulfilled: the initial data is distributed among components which are also controlled by the data owners. These components are connected via some network. Connectedness and availability of components may be dynamic; adding further components during runtime is possible. There are third parties, the data consumers, for which the data should be made available. It should not be possible or at least necessary for them to connect to each single source component. They should not have to know where which information is exactly located. Their access should still be controlled by the original data sources. Potentially, data should be preprocessed before reaching the client.

Example We exemplify the generalization by describing another setting where our approach is feasible. In many distributed systems, components exist which generate initial raw data, e.g., by measuring their environment. Only the source itself and optionally components in its logical proximity needs to handle the data in its original granularity. Often, other components only need preprocessed data (e.g., aggregated or filtered) of lower granularity. In such a setting, we map our architecture as follows: components which generate data and only need their own data are the data sources/peers in our terminology. We denote them to be on *level 0*. Components which need the sources' raw data to compute on them are their clients. They are on *level 1* and can also play the role of Gateways

for other higher-level clients. These clients, in turn, do not need level 0 data, but computation results based on data from level 1. This approach can be iteratively extended until all components are included. The result is a hierarchy of components, creating a tree structure or a directed acyclic graph with respect to information flow.

This approach avoids that every component has to connect to the raw sources which possess the most critical data. Instead, we specify that every component may only connect to that very Gateway which can provide the needed information while having least criticality among all potential Gateways. These constraints for the data flow in the system build the foundation of access control. Upon it, more fine-granular access control can then be deployed by the means presented in this chapter.

There is a single important difference to realizing the full approach: without SMC, the peers always have to trust their direct Gateways to which they forward their raw data. The peers have to assume that these Gateways correctly aggregate their information without misuse, manipulation or leakage. However, the peers do not have to trust any other components besides their direct Gateway as these only receive aggregated data. In accordance to out solution, unlinkability of raw data is improved since the Gateway can merge data from the peers. Furthermore, assuming that the Gateway does not store nor cache information[3], peers always are informed when a higher level request has been performed and the peers can intervene by exerting their veto right.

9.8 Discussion

In our design we made some decisions which are strongly influenced by the concrete setting of application. We discuss them here and present alternative choices.

When a computation session is announced with a Task Description, the full initial Computation Request of the client is forwarded to the peers. This is not functionally motivated, but intended to make initial requests fully transparent to the data sources. If this transparency is not necessary nor desired, this part of the Task Description can be left out. Communication overhead is then reduced, since the original Computation Request is a notable fraction of the complete Task Description.

When peers receive the Task Description, they have the right to veto and to refrain from participating in the computation session. This also contributes exclusively to the privacy protection goals of the peers, namely intervenability. Our measurements showed that this interaction (cf. Steps 10–12, Figure 9.5) is the bottleneck of the whole request validation causing the latency shown in Figure 9.9 and fast saturation of the system with increasing load (cf. Figure 9.10). It is hence worthwhile to discuss whether this feature should be retained or given up: use cases are imaginable where this veto right is not necessary since peers are not considered to be autonomous systems of individual data owners. Then this feature should be left out. Besides the achieved speedup, communication would also become more stable and more concise: vetoes can cause interactive adjustments on the participant sets which also induces multiple rounds of communication until the participant set converges. In any other case, one also has to keep in mind that deactivating the veto right does not remove access control completely: the client-faced access control performed by the Gateway is still in place. However, regarding data access, it becomes a trusted third party for each peer with regard to their aggregated data. A compromise is achieved when the peer-side checks are not performed in a synchronized fashion during the client request but afterwards, asynchronously, when the computation is about to be carried out. This reduces the delay of the acceptance note towards the clients but leaves peer-side

[3] We consider realistic since the sensors generate a stream. This means the most recent values always change as do the preaggregated values like value for the last 24 hours.

control in place. However, this would increase complexity on the clients, as they will only asynchronously learn whether their request was finally accepted and whether adjustments to the participant set had to be made.

We chose that the Gateway does not have to disclose the actual set of peers when addressing them via a label; we call this *label opaqueness*. This improves unlinkability for peers since the result of a computation does (in general) not leak information about which peers actually participated in the computation and provided input. This can, however, be a problem in some use cases. Firstly, when vanishing devices are typical for the environment, e.g., very dynamic settings with moving devices connected via a wireless connection, the client cannot assess whether its query currently includes all desired devices. Secondly, some computations do not yield usable results when the set is completely unknown. E.g., the repeated computation of a sum over a set with changing but unknown cardinality does not provide a consistent timeseries of data points. There are two main options how these problems can be approached: the first is to weaken unlinkability for peers and to improve transparency for clients: when the result of a finished computation is sent to the client, the information about the participant set of the computation can also be disclosed to it. It is then able to better assess the meaning of the obtained result. Alternatively, the client might be given the ability to resolve labels and predicates directly via the DS. The second approach is to weaken intervenability for the peers and to strengthen control for clients, meaning that clients do not have to work with abstracting labels but may choose the participant set itself. However, in these cases, the providers of the service have to keep in mind that clients are potentially able to maliciously select sets which disclose raw information about single peers. A trivial example is selecting a set of cardinality 1. A slightly more sophisticated attack is a difference attack, where first a set of n peers is chosen and then the same set reduced by a single peer, the victim. Depending on the used protocol, its desired raw information can then be obtained by computing the difference (or some equivalent) of both results.

In particular, the right to veto combined with label opaqueness can cause problems in some use cases. We nevertheless chose to implement both to exemplify the possibilities of privacy protection in this context. If a single peer vetoes against a computation, the computation is retried with the participant set reduced by the vetoing peer. Normally, this action is invisible to the client which provokes the problems discussed in the previous sections. Besides the already mentioned options for generally weakening the privacy properties for peers in order to solve these problems, one could also aim to weaken privacy only in these exceptional cases: the first option is a fail-fast solution, where a veto does not lead to invisible adjustments but to an error message for the client, informing it about the vetoing peer. This should at best also include a rectification and reasons of the peer. Depending on how clients can select computations, the client can then restart the computation itself. The second option is automatic adjustment of the participant set and performing the computation. The result however also includes the above described veto message from the refraining peer.

Our solution proposes that permissions are manifested as authorization certificates. This allows authorizations to be verified by anyone who has the corresponding certificates. However, their application has some consequences which might be undesirable in some contexts. A PKI is needed and at least the certificate of the issuer must accessible for all verifying entities. In particular, the certificate distribution might become a further problem to be solved. Furthermore, certificates do not possess high flexibility with respect to changing permissions. We already addressed this problem by suggesting a small interval of validity for each Permission Grant. In a more static environment, where the Gateway does not typically change, one might opt for a solution where the grant issuer is identical with the Gateway. Validation of grants by the Gateway can then be reduced to verifying

a signature which was added earlier by itself to a created grant. If no grant documents at all are desired, permissions can also be stored genuinely on the Gateway. In this case, temporal separation between requesting Permission Grants and performing Computation Requests is still given, but permission verification is simply performed by a permission lookup locally on the Gateway. This reduces the overhead of sending permission grants and makes permission adjustments and revocation very flexible. However, this weakens transparency: clients cannot state their permission explicitly and the validity of a request cannot be directly stated to peers anymore. If the veto right remains in place, peers nevertheless stay in full control of their data.

9.9 Key Contributions of this Chapter

In this chapter, we presented the second part of our approach to make SMC available as a classic service. It addresses the question how a distributed SMC solution can be extended into a centrally accessible service for data consumers. For that, we create an adapter on three levels: architecture, data flow and execution of logic. Having established a very defined channel, we add fine-grained access control in order to control the actual data flows.

Centralized SMC Access

In this context, the Gateway addresses the initial architectural mismatch between SMC—which works rather in a peer-to-peer manner—and a classical client-server architecture. SMC is inherently distributed and enables computations among peers; it is not initially designed to be a service available to third-party clients. The Gateway now acts as an architectural adapter for connecting both structures: towards the one side, it manages and orchestrates SMC sessions. Concomitantly, the Gateway finally receives the computation result by participating itself or by obtaining the result from one or more peers afterwards. Towards the other side, the Gateway presents a very standard data querying interface via REST (RA.2). Clients solely have to post their Computation Requests via this interface (RA.1) in order to make peers compute a desired result. Since SMC computation can introduce a notable delay, clients do not have to wait for the result in a synchronous manner. Instead, they are given a session ID which they can use afterwards for polling the data until received (RA.5).

Directory Service

The next necessary adapter is on the level of metadata. In order to enable Computation Requests, clients must be initially informed which data is available. This is solved via metadata which is collected among the peers, post-processed and stored centrally on the Gateway and made available to the clients (RA.3). In this context, the DS acts as an extended and dynamic service announcement sine it communicates what computations are potentially available. The real adapter functionality is expressed by the post-processing of the Gateway: while every single peer holds its own metadata, is cannot be aware of which computations are actually possible (where it could participate) and which groups are semantically sensible. This needs an overview which only the Gateway possesses. Its post-processing solves this since it creates—in a manual or rule-based fashion—semantically sensible groups of peers and stores which data can be computed among them. This enhanced data is then made available to the clients for request selection.

Request Translation

Since the Gateway shadows the actual SMC architecture and the DS performs the same regarding the metadata, incoming client requests are not initially executable for SMC

peers. Hence, the Gateway has to translate the incoming client-side Computation Request into a Task Description which is a understandable and executable data structure for all peers (RA.4). The translation includes transferring computation details and resolving the chosen group to an actual set of participating peers.

Client-Faced Access Control

The above mentioned features provide means for data access from the clients to the peers, mediated by the Gateway. Since information flows are now fundamentally enabled, sensible access control is put in place. It preserves the peers' ability to exert control over their own data. We introduced access control on two levels. The basic level of control is the peers' right to veto against computations (see Chapter 8). In this chapter, we introduced another additional layer of access control at the Gateway (RA.15). While it should reflect all participating peers' intentions, it facilitates access control and provides further benefits: with its help, permissions are stated explicitly as documents which can be held by the clients itself. Hereby, clients' rights become transparent for the peers. Furthermore, the permission granting process (which might be computationally or communication intensive) is separated from the actual permission verification which is performed during each request and therefore should be fast to execute. Lastly, Permission Grants can be forwarded to participating peers, allowing them to verify the permission themselves.

Fully Privacy-Preserving Service

Given our notion of privacy from Chapter 2, we understand a fully privacy-preserving service to enable and support the protection goals of data minimization, unlinkability, transparency, and intervenability. We found that SMC already supports data minimization and unlinkability initially if applied correctly. Hence, we designed an architecture with SMC at its core which also provides the other two. Our approach is to provide an access control scheme which is based on permissions made explicit and communicable as grant documents. Similarly, queries, i.e., data requests are stated explicitly and in an authenticated and integrity-protected manner. This enables peers to gain all vital insights about a computation requested by any client, which in turn fulfills transparency for the peers. Furthermore, the request and the permission grant can be checked by the peers before the computation is carried out. This enables intervenability, since each peer is able to refrain from a compution it objects due to some reason. This veto capability is fundamentally enabled by the data architecture provided by SMC. Since all data remains at the peers, their cooperation is indispensable to perform a computation with their data. Exerting the veto right is realized as an expected exception. The session orchestration layer is able to handle them accordingly.

9.10 Statement on Author's Contributions

The findings of this chapter have been published in the following paper:
M. von Maltitz, D. Bitzer, and G. Carle. Data Querying and Access Control for Secure Multiparty Computation. In *Proceedings of the 16th IFIP/IEEE International Symposium on Integrated Network Management*, Washington, DC, USA, 2019. IEEE (reference [vMBC19]).

The publication is based on the design developed by the author and refined and implemented by Dominik Bitzer in his Master's Thesis guided by the author of this thesis and supervised by Georg Carle. The author of this thesis contributed to the results in the publication and the chapter by designing the overall idea, the concept and the protocols. The author also contributed to detailed design and implementation decisions during Dominik Bitzer's Master's Thesis. Dominik Bitzer refined and implemented the designed protocols. The measurements have been performed and evaluated by the author of this thesis.

The message structures in Figures 9.1 and 9.4 are an extended and slightly adjusted version of the figures in the publication. In the version shown here, clients obtain separated grants for each requested query. When sending a Computation Request, clients do not have to reveal all unrelated grants, but only the one corresponding to the request. This improves clients' confidentiality and unlinkability. The protocols in Figures 9.2 and 9.5 are a detailed version of the protocols published in the paper. The equations have been adapted from the publication.

10. Advantages and Disadvantages of applying SMC

In the previous chapters, we analyzed performance and applicability of SMC. Afterwards, we presented an approach to utilize SMC for the processing of sensor data in dynamic environments. Several state-of-the-art solutions exist which address the same problem domain. In this chapter, we perform a comparison of these solutions with our approach. We first identify several representatives. Then, we compare them in the categories of architecture and implications of architectural differences, data protection, performance and resource consumption.

10.1 Baseline

We first identify how other solutions solve the problem we address. Here, we focus on a selection of available and established systems from similar domains.

10.1.1 Qualification for Comparison

From a functional point of view, we address the problem of collecting and processing initially distributed data. The goal is to drive other services and clients with the computed results. The latter, in turn, should have the ability to specify and query the data they need for operation.

Other systems qualify for comparison if they have a similar purpose. We state this more precisely by specifying a blackbox with input/output equivalence: before entering the system, data is distributed logically or physically in the realm of system application. The data itself is in a shape that allows processing and computation. I.e., it is numerical or at least ordinal, but typically not nominal. By this, more mathematical or logical operations can be carried out. The most vivid domain here is the handling of sensor data. The data has to be collected and processed. The results of processing are, in turn, the input for other services or clients; each of them needs one or more types of results. These types do not change frequently, but obtaining current data is performed continuously. While the input data can be deemed critical with respect to some notion of privacy, the output data is not.

10.1.2 Real-World Architectures and Solutions

We now consider some established solutions from the industry which are compatible with the specification above.

Siemens: MindSphere

MindSphere [Sie18] by Siemens is a "cloud-based, open IoT operating system" aiming for application in the industrial Internet of Things (IoT). Automation devices and networks are the producers of sensor data. That data has to be analyzed and evaluated by different stakeholders in order achieve "optimized processes, resource and productivity gains, the development of new business models and the reduction of operations and maintenance costs". In-depth topics among others are condition monitoring, predictive maintenance and asset performance management. They address this problem with a cloud-based solution. Intermediary devices (*MindConnect*) are attached to the automation networks. They create or collect the data from the networks and push it to the cloud-based platform. Further storage and processing of the data is then performed on the tenancy-enabled cloud system. The consumers are applications which are provided by Siemens, its ecosystem or third-party developers. These also run on the cloud platform. For securing their solution, they mainly name three building blocks: Data-in-motion is encrypted using TLS. Storage in the cloud is secured by "highest standards" by which they mainly mean organizational measures. And intrusion and tampering is mitigated by making the MindConnect devices read-only inside the automation network. Accepting that they employ the cloud as a trusted third party, they enable fine-grained access control down to the level of individual data within single services.

Their way of data handling is best described by their concept of *data lakes* [Sie18]:

> Data lakes hold large amounts of raw data, from multiple sources and across federated systems, until needed. Applications or services can consume data and contextualize it with any other kind of data in the data lake.

General Electrics: Predix

Predix [Gen18] by General Electrics is a "distributed application and services platform for building and running powerful digital industrial solutions"; they also address the industrial IoT. *Assets* (e.g., machines at an industrial plant) are the producers of sensor data. Data from these assets is used to build a *digital twin* as a management abstraction and "single source of truth". Data consumers are provided or custom but specifically tailored third-party applications connected to the Predix platform.

Like Siemens, General Electrics addresses this problem by a cloud-based solution. They especially focus on data integration (called *ingestion*) aiming for consolidation of a high amount of diverse data coming from heterogeneous sources into a single platform storage. That allows to use this data in a standardized manner and to further leverage its potential by enabling data combination.

In addition to their cloud-based approach (*Predix Cloud*), they support edge computing use cases, allowing applications to be deployed on the edge devices for lower latency of analysis and lower bandwidth consumption.

For security, they use established methods like firewalls, intrusion detection systems and containerization as well as data separation by tenants and access control. Structurally, they apply security on different levels of their architecture ("defense-in-depth"). These technical measures are complemented by organizational processes, made tangible by complying to a multitude of international security standards. They also directly address privacy, but merely identify it with data retention and deletion policies. There do not seem to be any technical enforcements of privacy in place, either.

Bosch IoT [Bos16]

Bosch provides a "comprehensive toolbox in the cloud provided as Platform-as-a-Service" for managing the IoT. They mainly provide a software-based solution. This can then be deployed on other vendor's clouds like Amazon Web Services (AWS), Microsoft Azure, SAP Leonardo or on premise. Devices are connected to the cloud where the Bosch IoT Hub and the Bosch IoT Remote Manager are located. These manage the connected devices and their data. On top, applications can then connect to the Hub in order to obtain needed data.

In order to connect heterogeneous sensors to the Hub, intermediary devices—the cloud or field connectors—are used for protocol translation. These intermediaries provide "local management, local data processing, and connectivity to the cloud or other gateways" [Bos17b].

Their focus in the area of security is authentication of devices by employing X.509 certificates and using a PKI. On this foundation, secure channels and encrypted communication are realized. The topic of data protection and privacy is handled organizationally by providing policies for data usage transparency and adhering to the German data protection regulations. Further security best practices are provided in [Bos17b] where they rather state guidelines to ensure security as a developer for the Bosch platform than giving insights into their own measures. In our opinion, the reason is that Bosch is providing rather an open infrastructure as IoT foundation instead of offering a fixed architecture like the competitors in this section do.

Microsoft Azure IoT

Microsoft Azure IoT [Mic18] basically models IoT infrastructures using three domains: *Things*, *Insights*, *Actions*. Things is the set of heterogeneous devices which create data. Insights is the realm where data is stored and processed. Actions are performed by business logic which is driven by the outcomes of previously processed data.

The reference architecture is as follows: devices or edge gateways—intermediaries enabling communication—register at the Azure cloud. By this, they can send and retrieve data from the cloud. Sending is vital for telemetry data which has been collected by the devices. Retrieval is used for management and configuration of the devices. The cloud hosts a gateway which accepts the telemetry data and performs the device management. Behind this gateway, in the cloud, stream processors perform analyses and storage components perform persistence of the data. Furthermore, UIs and reporting tools are made available for user interactions. Business applications are either connected via the stream processors or the storage. These applications are the data consumers and use the available data as a foundation for their actions.

Azure IoT supports use cases of edge computing by providing *Intelligent edge devices* and *field gateways*. These can perform aggregation, filtering, transformation and further processing of telemetry data as well as buffering of data, protocol translation and event rule processing before submitting data to the cloud.

The field gateways are also considered to be an important enabler for security. Things i.e., sensor devices might not be powerful enough to perform the cryptographic operations considered necessary to ensure security. In these cases, they are forbidden to perform direct cloud connections and placed behind a field gateway. The latter, in turn, obtains the data and tunnels it to the cloud via secure channels.

Besides, Azure IoT understands security to be the requirements of having encryption for data-at-rest, using TLS for data-in-motion while it is sent to the cloud, and providing

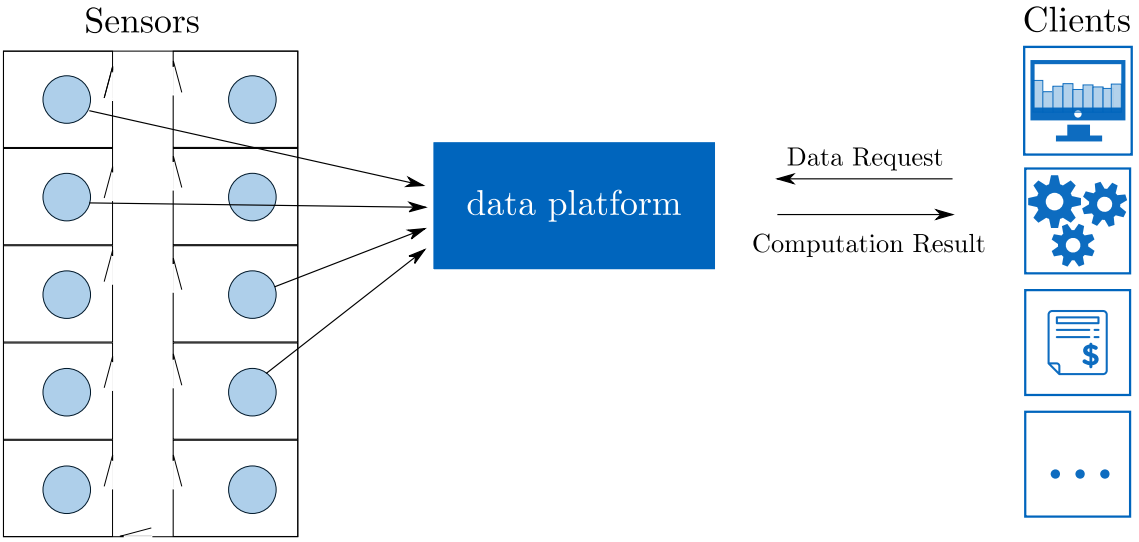

Figure 10.1: Abstraction of state-of-the-art architectures

cryptographic identities of the Things, created, stored and provided by the IoT Hub in the cloud. Furthermore, established methods of authentication and authorization are taken into consideration. Notably, they require that Things perform outbound-only connections, reducing the attack surface of the most vulnerable devices in the system. A similar approach has been mentioned before in the context of MindSphere.

10.1.3 Abstraction

In Section 4.3, we already derived a comparison of SMC with a Trusted Third Party (TTP) based on the formal theory behind SMC: typically, simulation-based proofs are used to prove correctness and security of SMC protocols simultaneously. These proofs inherently perform a formal comparison with a TTP-based solution. We adapted this view and based our performance comparisons in the previous chapters on these considerations.

By the previously considered examples, we could demonstrate that referring to a TTP as comparison actually takes account for current real world deployments: State-of-the-art architectures *are* centralized systems having a single component at its core (cf. the data platform in Figure 10.1). They are either located in the realm of deployment—*on premise*—or in the cloud. Data sources, be it very low-end sensors or complex automation networks are equipped with a proxy component that performs data preparation and acts as a bridge to the target architecture in general and the central core in particular. The solutions differ in the importance of the proxy and how strongly the proxy shields the sensors from the remaining infrastructure. The core performs the task of consolidating received data and persisting it. On demand or continuously, stored data is processed to obtain results which are forwarded to client applications. These clients are typically also located on the central platform or connected via standardized interfaces. Notwithstanding of their exact location, interaction most often happens via APIs for requesting and obtaining data from the system. Users interact with the client via a web interface. The specific type of these clients, in turn, mainly depends on the realm of deployment and the concrete use cases.

The adequacy of the described abstraction is also underscored in the diagrams of the whitepapers about the previously described systems: [Sie18, p. 5]. [Gen18, p. 7], [Bos16, p. 7], [Bos17b, p. 6], [Bos17a, p. 7], and [Mic18, p. 27].

10.2 Categories of Comparison

In the last section, we specified an abstraction of state-of-the-art architectures. We demanded that is has the same purpose as our proposed system in order to qualify for comparison. For doing so, we defined the purpose as an input/output equivalence while handling the system itself as a blackbox.

Having the same functional purpose, there are several properties which allow comparison of solution quality. We define our categories for comparison here:

The first, most vivid differences are those in the architecture of the solution. More specifically, differences in the *topology* encompass changes in the set of the deployed components and in the set of the connections between them. Furthermore, differences of the *properties* of components and connections consider the specific characteristics of the components, influenced by the environment or operational requirements of the whole system. When the topology changes, it is not trivial to determine which components of both architectures should be compared with each other. To take this into account, we will establish a mapping by functionality in advance.

Further comparison is made in the realm of security. Firstly, we address the question where *trust* is needed in the system and of what kind it is. Furthermore, the classical security properties of *confidentiality*, *data integrity*, and *availability* are discussed. This is complemented with *data minimization* and the privacy protection goals *unlinkability*, *transparency*, and *intervenability*.

Finally, the performance of the solutions is compared. This especially considers the delay between result request and its response. Besides, the resource consumption in terms of memory, CPU cycles and storage is discussed.

10.3 Comparison

Based on the provided state-of-the-art systems and the categories to be examined, we carry out the actual comparison here.

10.3.1 Architecture

In classical architectures and our solution likewise, we have four roles of components:

- The *sensors* provide the raw data coming from the analogue world or other measured devices.
- The *proxies* enable access to the sensors or serve as the first hop, towards which the sensor data is sent. In our solution, they correspond to the peers, i.e., the sensor platforms to which the sensors are connected. In the classical solution, they constitute intermediaries like the MindConnect devices.
- A central *core component* which mediates between the proxies and the clients. In classical architectures this is the data platform, a central database or the cloud. In our system, this role is taken by the SMC Gateway.
- The *clients* are the data consumers which obtain data from the core component. Neither classical architectures nor our architecture prescribe the concrete functionality of these components. The only characterization is that they consume parts of the provided and processed data.

From a topological viewpoint, the set of components is not notably changed. However, our solution makes new connections necessary. Proxies have to be reachable by each other and they have to communicate directly. Changes regarding NAT (network address translation) traversal and firewall configurations have to be considered. Besides, no further connections are required. In particular, clients still only have to contact a single point for data request and reception. We deliberately decided that changes of client behavior should be restricted to a minimum.

There are larger differences between the properties of the deployed components in a classical solution and our proposal. The reason is that functionality is intentionally shifted to achieve our privacy properties. We consider the following functionalities:

- Holding or storing sensor data
- Processing the sensor data
- Accepting and handling queries of clients

In classical architectures, functionality is centralized on the core component. Directly after its creation, sensor data is forwarded to the core component. It stores the data (after an optional preprocessing) and makes it available for analytics and processing. These tasks are also carried out on the core component. The core also provides APIs enabling third-party applications—they are the clients in this context[1]—to request and obtain available data.

In our solution, data of individual sources should remain private, i.e., should not be shared with any other party. Due to this reason, storage happens on the sensor platforms itself. Here, it depends on the use case whether long term persistence is needed or an in-memory storage for a sliding window of available data points is sufficient. Processing is not performed by the SMC Gateway as core component, but only orchestrated by it. Instead, processing is carried out as SMC protocol between the sensor platforms. This raises the aforementioned new requirement of reachability for all sensor platforms. Accepting and handling requests of clients remains a task for the core component. However, translation of requests has to work differently. Instead of being translated into, e.g., an SQL query or some other data retrieval language, the request has to be transformed into an SMC session. The core component merely acts as a façade.

10.3.2 Data Protection

We now compare both types of solutions with respect to data protection, i.e., trust, security and privacy properties.

10.3.2.1 Trust

For the meaning of *trust*, we refer to Definition 3.1 in Section 3.1 on page 18.

In classical architectures, it is necessary to have a high amount of trust towards several components: initially, the sensors create the private data. They are trusted that they do not forward it to any undesired third party. The same is true for deployed proxies. After the private data has been sent to the core component, it is necessary to trust that it stores and processes the data only as expected. Under the assumption that the results obtained by data processing are uncritical with respect to privacy, no special trust is necessary

[1] As long as a functioning access control over the available data is carried out, there is no vital difference between clients being individual components—as in our abstraction—or being software deployed on the same core component—as in some of our real world examples in Section 10.1.2.

towards the clients. However, the core component is additionally trusted to carry out access control correctly, so that every client only gets the data it is allowed to obtain.

In our solution, a reduction and distribution of trust is achieved. While the trust in the sensors and in the sensor platforms remains identical, storage is distributed among the sensor platforms. They store their own data and only their own data. I.e., no trust has to be laid in another party to perform faithful storage for them. Similarly, processing is not carried out on a single component but by SMC protocols. This removes the necessity of trust from the former processing component. Strictly speaking, these protocols do not have to be trusted either, since their correctness and security is typically proven mathematically. As described in Section 3.3.1, protocols can be secure in the honest-but-curious or in the malicious model. The system security and the corresponding amount of trust to be brought towards the SMC Gateway strictly depend on the security of the employed SMC protocols; i.e., the type of model they are proven to be secure in. Due to this reason, we only sketch the trust implications and refer for details to the exact security properties of the employed protocols. In the honest-but-curious model, we trust the computing parties, i.e., the sensor platforms that they perform the desired protocol invocation correctly. Then, they do not learn anything about the inputs of the other parties. However, if protocols are used which are secure in the malicious model, even this trust becomes superfluous, since malicious interaction becomes detectable.
With respect to querying and access control, our solution also distributes the obligations which again shifts the necessity of trust. Clients are allowed to obtain certain data if they can provide a corresponding grant (cf. Section 9.4). This grant is provided by an authority which, consequently, has to be trusted to issue these permissions faithfully. Normally, the Gateway, being the corresponding instance for verifying the grants, has to be trusted to only react on correct and granted requests. However, this trust is strongly reduced since our solution also allows the peers to check the query and the corresponding grant. If they have enough knowledge about the clients and the environment, they can decide for themselves whether or not a request is legitimate. In consequence, the amount of necessary trust in the Gateway for correct verification depends on the degree of autonomy of the peers. This has to be decided in each deployed instance of our system anew. Since the role and behavior of the clients are not changed by our system, no new trust considerations arise here.

10.3.2.2 Security

We use the notion of security as established in Chapter 2. In all cases, our assessment focuses on whether technical mechanisms are in place which guarantee the fulfillment of the mentioned properties. We do not consider a solution to be valid if it makes trust necessary in a way described in the previous section.

In classical systems, confidentiality of the raw data towards a third party is not given, since all data is stored on a central unit which has to be trusted. Similarly, there are no integrity guarantees after the data leaves the sources. Manipulation of data on the central system is possible. However, the availability is high since central storage easily enables backup strategies and redundant storage of data. Furthermore, outages of proxies only affect future data collection and no data that has already been created.

In our solution, sensor data remains on the sensor platform where it has been collected. It is not shared with any other party, particularly not with the Gateway. Hence, confidentiality is achieved to a high degree. Confidentiality can even be extended to computation results: in Section 8.4, we describe how the result is sent from the peers executing SMC to the Gateway. In order to prevent access to the result by the SMC Gateway, the results can be encrypted in advance for the receiving client. This is possible since the client's

certificate is already known by the peer in order to verify the request.
Integrity protection is not actively improved by our solution, but typical attack vectors are mitigated. There is no core component which has access to the data. This removes a common high value target for data manipulation from the system. Each sensor platform only holds its own data. We already decided to trust the sources to correctly create initial data. This automatically implies that we trust them not to manipulate the data. Adversaries attacking a single sensor platform would have a comparably lower impact on the system than attacking a central unit. Only a single source's data could be manipulated then. Furthermore, if protocols with security against malicious adversaries are employed, manipulation of the computation process can be detected. If the results are signed in advance by the peers, tampering by the SMC Gateway can also be prohibited (cf. Section 8.4). For that, however, the cryptographic identities of the peers have to be known by the clients.
Guaranteeing availability becomes harder with our solution. Due to distributed storage, it is more complex to achieve redundancy and robustness. Furthermore, outages of nodes have a higher impact since not only future collection is affected but also access to historic data. The reason is locally storing data on each node. Our solution specifically addressed failing nodes and failing connections. We realized automated recovery to counteract the naturally higher chance of these failures in dynamic environments. Disconnected nodes are able to recover themselves and to reconnect to the managing SMC Gateway. The Gateway itself is designed to be stateless, hence, Gateways can be deployed redundantly. Lastly, backup solutions are still possible and can even be centralized, since the data-at-rest can be stored in a state-of-the-art encrypted manner.

10.3.2.3 Privacy

We could show that in the realm of security our solution already improves the state of the art of sensor data handling and processing. This is continued with respect to privacy. The notion of privacy has been described in detail in Chapter 2. Based on this understanding, we examine the privacy protection goals of unlinkability, transparency, and intervenability. Again, only technical guarantees without necessity of trust are considered.

In classical architectures, data minimization is not considered a goal. All available data is pooled and retained as long as legitimate. The aim is to support all *potential* purposes, even if discovered only after data collection. Peers are not necessarily informed when new purposes emerge.
Also unlinkability is not achieved. The reason is that data of different sources from varying physical and logical domains is stored at the same central place. This enables arbitrary data combination. While we consider this as a strong violation of a desirable privacy protection property, most vendors actually frame this practice to be a vital benefit, called data integration. Without considering privacy, they state that the potential of the available data can only be leveraged to full extent if all data is available at a single place and can be combined after collection. While this is true it avoids the tradeoff between privacy and utility by ignoring the former and fulfilling the latter.
By storing sensor data centrally, transparency is also not given. Since data is detached from the local data sources by transmission, the usage of their data moves out of the data sources' perception. For the data sources, it neither becomes clear which legitimate computations are carried out on their data or for whom they are performed. Nor do attempts of misuse become recognizable where their data is used in a way which would not be considered legitimate by the sources.
Since transparency is a precondition for intervenability, the latter is not fulfilled either. The last action of control that data sources can perform, is the decision what data to forward to the central storage and what to omit. After transmission, control by the data

sources is lost. This also means that data retention, deletion and revocation mechanisms all depend on trusting the core component.

In our solution, data minimization is considered important. Possible computations are specified by making the corresponding SMC protocols available. Hence, the purpose of computation must be known in advance. It is impossible to create new purposes and corresponding computations without knowledge of the peers.
Also, unlinkability is fundamentally achieved. Critical data of different sources is never stored at the same place, but held in the physical or logical separation in which it was collected. Classical architectures solve the privacy-utility-tradeoff by negating the first goal und fully achieving the second. It should, however, be tried to achieve a balance so that both goals are reached to an acceptable extent. Driven by the innovation of SMC technologies, we could refrain from finding a balance, but *solve* the tradeoff, fully achieving both goals without compromise between them. In detail, two types of unlinkability are achieved by our application of SMC: the results obtained by computation become unlinkable with the peers, since it cannot be recovered which sensor platform contributed which input in general. And the results become unlinkable among each other. That means, while raw data might allow recombination to gain new insights, results can be crafted by aggregation which do not allow combination with other results obtained. If any linkage is desired, it must be performed explicitly and openly. This always becomes known to the peers by the mechanisms for transparency described as follows.
We extended SMC so that also transparency is achieved. Since data remains at the peers, their interaction is needed when performing processing via SMC. We developed this basic necessity of SMC into a query-based mechanism of request processing which provides all insights about upcoming requests and their origins to the peers before the computation is carried out (*Computation Request*). By doing so, data usage becomes fully transparent and accountable.
Based on this extension, also intervenability is realized. Since peers are informed before performing the computation, they are given the opportunity to evaluate themselves whether or not to agree to the demanded way of processing and usage. We further provide a structured way of vetoing against a request which enables the Gateway to find a solution to the conflict itself or, if not successful, to forward an intelligible error message to the client.

10.3.3 Resource Consumption and Performance

Shifting functionality to other components, as done by our proposal, not only influences security and privacy properties, but also moves resource requirements. This makes it possible to also compare both types of solutions in the dimensions of performance and resource consumption.

To conduct a fully quantitative comparison we would also need performance data of classical solutions. Since these are currently not available, we continue on performance estimates for comparison.

Resource Consumption

In Section 10.3.1, we already discussed that the architecture of classical solutions and our proposal is different. To achieve a valid comparison we reuse the mapping presented above.

We identified four roles of components: Sensors, proxies, a core component and the clients. From them, sensors and clients remained unchanged by our proposal. We can therefore exclude them from comparison.

The main changes occur at the proxies and the central core. Functionality is shifted here, and in consequence, their resource requirements change.

Proxies The functionality for the proxies moves from receiving data and forwarding data—including an optional step of translation transformation and preprocessing—to persisting the data locally and performing secure computations.

As hardware baseline we use the current version of the MindConnect [Sie] device. It has the following specifications: An Intel Quark X1020 processor with 400 MHz, 1 GB of RAM, a 10/100 Ethernet network interface and 500 MB buffering space for collected data.

Storage requirements for the proxies depend on the data retention strategy. If only current raw data is of interest, providing a sliding window of recent data is sufficient and automatically fulfills privacy-preserving data deletion requirements. This does not necessarily affect (uncritical) aggregated data and previous reports; they can be stored centrally and considerably longer. We assume that this is the typical setting in many use cases. For these applications, the mentioned baseline already fulfills the storage requirements. Backups are still possible and can be stored at a central place in an encrypted manner. One can even think of solutions that support queries based on that outsourced data. Peers could be enabled to restore this data on demand and make it again available for corresponding computations. In the other case when long-term data should be stored on the proxies, dimensioning strongly depends on the use case and the required duration of availability. The fact that each proxy only stores the data of its own sensors reduces the amount of required storage by several orders of magnitude.

Memory requirements of the proxies are influenced by the proxies' involvement in secure computations. We saw in Chapter 4 that memory consumption is influenced by fundamental factors as the number of participating peers or the number of elements. The heap allocation even exceeded 1 GB in a setting with 15 peers. However, this observation has to be contextualized correctly: Our prototype is based on an SMC framework written in Java. It is well known that Java has a quite liberal handling of the memory that was allocated for the executing JVM. Detailed memory usage diagrams like Figure 4.13 show that the high amount of memory is not actually needed but an artifact of the specifics of the garbage collector. I.e., when having only 1 GB available on the proxies, we expect that the computations are still possible. The JVM will not react on the constraints by swapping to the hard disk, but by invoking the garbage collector more often. This in turn will lead to a higher CPU utilization by the garbage collector. While memory is therefore not a hindrance, having SMC implementations in other programming languages which are more resource efficient would be desirable.

Taking part in the secure computations, the proxies get a new purpose which also influences CPU utilization. Our measurements show that, depending on the algorithm, the power of the CPU plays a varying role in overall computation performance. In the baseline measurements in Chapter 4, the computation duration is actually influenced by the number of CPUs and the frequency. Here, the number of cores is more important than their actual strength. In our use case in Chapter 6, we could not identify any influence of these parameters on the computation duration. Its seems plausible that the higher complexity makes more communication necessary and shifts the bottleneck towards the network. In any case, when considering the possibility of performing several computations in parallel, the proxies should have more than one core. Against this background, our hardware baseline of 400 MHz might not be sufficient yet.

However, the trend of edge computing is further evolving and demanding higher computing power at edge devices like the proxies. On this wave of new hardware requirements and developments, secure computation will automatically become better supported.

Core Component The functionality we shift to the proxies, is moved away from the core component. It should not store raw data anymore and computation are no more

carried out as local processes. In turn, is has to orchestrate the set of connected proxies, transform queries into SMC sessions and perform the corresponding management actions.

If we actually perform this extreme shift, data storage on the core component becomes superfluous. The SMC Gateway is designed to be practically stateless and data can be obtained from peers and forwarded to clients in-memory. Our architecture suggests letting the Gateway participate as a neutral peer with no own input. Then, the abovementioned resource requirements for proxies also constitute the absolute minimum for the Gateway. Practically, resource requirements will be even higher since it will have to perform the orchestrating of several computations at the same time.

When looking at real-world settings, we do not suggest to do such an extreme shift but rather to build a hybrid between classical systems and our approach. I.e., core components remain able to store data and to do complex data processing. It is then possible to employ SMC on the input data until an intermediary result is achieved which is deemed uncritical regarding privacy. All further processing on it can then happen centrally on the core component. Result data, created insights and reports can then also be saved on the core component. We point out that identifying up to which step SMC has to be used and when uncriticality of data is achieved depends on the semantic of the processed data and is a highly non-trivial task. Nevertheless, we consider it sensible to show that our approach is not only applicable as a whole but also hybrid application is conceivable.

When shifting fully to SMC, the storage ability can be removed from the core component. Furthermore, CPU power becomes less important. A multi-core architecture is nevertheless desirable for carrying out several computations at the same time and to cope with the bottleneck discussed in Section 9.6.2.2 on page 159.

When applying the hybrid approach, existing data platforms should already fulfill hardware requirements.

Performance

Having a deployment at hand which suffices the hardware requirements described in the previous paragraph we can compare the time for executing computations. For that, we refer to the measurements conducted in Chapter 6 and perform an approximation for the durations in classical architectures.

Assuming an input of 100 lines, each peer has data of ~1300 Bytes. Sending this to the core component in parallel by each peer costs 0.18 ms for latency and 0.0104 ms for transmission via a 1 Gbit link. When executing the algorithms centrally, the computation time is in the range of 0.1–0.2 ms. Another transmission happens when transferring the result to the client. Since this takes place in both architectures likewise, we omit this from our comparison. We gain an overall duration of ~0.4 ms.

When applying SMC the whole process is dominated by the costs for execution of SMC. We can hence omit the overhead introduced by our management framework which is at most in the range of a small number of seconds. The execution, however, costs around 7800 seconds for the union algorithm and 200 seconds for the Log-Rank computation, i.e., around 2.25 hours as a total.

From this, we obtain a performance penalty for SMC of 10^7 as multiplicative factor. This is not fully in line with literature. E.g., Yakoubov et al. [YGS+14] identify a multiplicative factor of 10^1–10^4. The reasons for the differences are located on several levels: Different algorithms perform a varying number of basic operations. These basic operations also vary strongly in their execution performance. Lastly, the available measurements utilize various implementations of SMC. This impedes reliable comparison.

We have summarized all our findings in Table 10.1.

Table 10.1: Schematic overview of the comparison between classical approaches and our solution. *Own* data always refers to the sensor data of the individual peer in question, *all* data always refers to the sensor data collected by all peers.

Category	*Classical Architectures*		*Our Approach*	
	Proxies	*Core*	*Proxies*	*Core*
Infrastructure				
Components	o collect own data o forward own data	o store all data o process all data o single point of access	o collect own data o store own data o process own data o participate in SMC	o manage SMC sessions o single point of access
Network	+ outbound connections only	o request answering	– in- and outbound connections	o request translation
Data Protection				
Trust	o create own data	o access control towards clients – store all data – process all data	o create own data + store own data + process own data + formally proven SMC protocols	o (access control towards clients)
Confidentiality	– data forwarded to core	– read access to all data	+ data remains local	+ no read access to data + no read access to results
Integrity	– data forwarded to core	– write access to all data	+ data remains local	+ no write access to data + no write access to results
Availability	o outage affects new data + no effect on historic data	+ high availability + redundancy	o outage affects new data – and historic data o data backup	+ statelessness + session recovery + peer orchestration
Privacy				
Data Minimization	– usage purposes independent from peers – also unknown to peers	– repurposing of data possible after collection	+ new purposes and computations known to peers	+ only performing predetermined computations + defined purposes
Unlinkability	—	– physical/logical pooling of all data	+ store only own data + physical/logical separation	+ no knowledge about source-to-data relationship + decouple results from data sources
Transparency	– no knowledge about processing, – data usage, and clients	+ knowledge about clients' queries	+ full knowledge about processing, + data usage, and clients + accountability	+ knowledge about clients' queries
Intervenability	– no control over own data	+ access control towards clients	+ full control over own data + veto mechanism	+ access control towards clients
Performance				
Resource Consumption	+ simple + low-end + sliding-window storage	+ single high-end component	o edge computing devices necessary – long-term storage – multi-core CPU desirable	+ commodity hardware o multi-core CPU
Execution Time	+ independent from peers	+ in range of milliseconds	– depends on number and resources of peers – depends on network	– in range seconds to hours

Part III

Conclusion

11. Conclusion

In this thesis, we set our goal to better understand characteristics of solutions based on Secure Multiparty Computation (SMC) and to build a fully privacy-preserving service with SMC at its core. We used several research questions as a guidance during this task. In the following, we provide answers to the questions and present an overview of our contributions to the topic. Based on them, we conclude the thesis by giving an outlook on further problems to be addressed in the area of SMC.

11.1 Central Findings and Contributions

We elaborate our findings by answering the questions stated in Chapter 1.

Research Question Q1 – Which understanding of privacy can be used to create privacy-preserving technology?

In Chapter 2, we elaborated on several concepts of privacy of the last centuries. In the late 19th century, Warren and Brandeis identified that the protection of certain private affairs was not yet legally considered. Examining law like slander and libel, the law of defamation as well as the early copyright, they came to the conclusion that a new personal right should be formulated, the right to privacy. Their fundamental notion of privacy is to decide oneself how much own information should be shared with the outer world. Furthermore, it includes the freedom to withdraw from the world when desired. In the middle of the 20th century, this theory is developed further by Alan Westin. He restated and strengthened the personal right of having control over own personal information and to decide oneself how and when to communicate. Furthermore, he stated that privacy is constitutive for other civilatory achievements, i.e., personal autonomy as well as balancing the enforcement of social norms and legal rules on the one hand and freedom of the individual on the other hand. Against the background of technological changes, Nissenbaum reframed the understanding of privacy. She turned away from the perception that some information are inherently private while others never are. Instead, privacy is given if the person about whom the information is considers that information to be adequate in the context where it is communicated. In turn, privacy violations happen if some information is made available in another context or information is moved from one to another context without consent. We further considered privacy frameworks like the Global Privacy Standard, Privacy by Design and the ISO/IEC 29100 Privacy Framework. Finally, we selected a privacy notion initially established by Pfitzmann, Rost, et al. They followed the established practice to

dissect *security* into the protection goals confidentiality, integrity, and availability, and applied the same method to *privacy*. This also results in a triad of protection goals: unlinkability, transparency, and intervenability. It complements the security triad and uses it as a premise. In particular, confidentiality is a necessary foundation for privacy, but not sufficient to realize it. In the subsequent years, the notion found wide-spread adaptation, on the national level as well as the international level of the European Union.

Research Question Q2 – What are the performance characteristics of SMC and which environments for application do they suggest?

In Chapters 4 and 5, we performed baseline measurements of the state-of-the-art SMC framework Fresco. Using a simple use case, we evaluated the behavior of SMC computations with respect to resource consumption. We considered the dimensions of wall-clock duration, CPU cycles and instructions, heap and stack memory consumption, and the amount of transmitted packets and Bytes over the network. As variable parameters we manipulated the setting—the number of participating peers, input elements per peers—, the network—the latency, transmission rate and packet loss—and the host properties—the number of CPU cores and their frequency.

Our analyses reproduced the insight that SMC computations are mainly communication bound. As a consequence, network latency has the strongest influence on the duration of the computations. Even with latencies which are normal for Internet or mobile Internet environments, the duration is already prohibitive for real-time use cases. An interactive setting can be imagined, but this depends on the actual computation being implemented. However, batch computations with no strict time constraints are well supported.

The next strongest parameter is the number of peers. A strictly linear relationship could be identified. Theoretically, one would expect a quadratic increase, since the group of peers builds a clique where every pair of parties performs an equal amount of communication during the computation. However, this communication happens in parallel which reduces the increase to a linear dependency. The CPU is not challenged in a notable way.

Additionally to systematic measurements we also performed use case measurements, emulating intranet, Internet and mobile Internet settings. Here, we could confirm the previous statements.

From our measurements we conclude as follows: at the time of writing, SMC is mainly suitable for settings in which the network latency is low and transmission rate exceeds 1 MByte. Requirements for the actual peers are of secondary interest, however, a low amount of GBytes of RAM should be available. Due to this reason, in the architecture of Chapters 7, 8 and 9, we focus on the intranet environment as it is present in smart environments and smart buildings. Further requirements for the architecture have been derived as part of the next question.

Research Question Q3 – Which infrastructural requirements must be addressed when applying SMC in domains with critical data and which privacy requirements are then fulfilled?

In Chapter 6, we selected a use case from the medical sector. It demonstrates a setting in which application of SMC is highly desirable: a multi-centric study is carried out amongst several institutions with different groups of participants. Merging the obtained data yields more reliable results than considering the individual institutions' outcomes separately. However, permission to data combination and fulfillment of data protection requirements normally imply a high organizational overhead. We showed how SMC can be applied to achieve data combination in a secure and privacy-preserving manner while avoiding

the need for organizational measures. The data of individual institutions is always kept confidential, making sharing agreements superfluous.

Concretely, we implemented the established Log-Rank test for evaluating Kaplan–Meier estimators as a secure protocol. The first milestone was rewriting the computation so that multiple input sources are supported. On this foundation, we could then design the secure protocol. For that, we employed the union algorithm of [BA12] and real number arithmetic using fixed point numbers for the statistical part.

We derived several insights from carrying out this task which supported our further steps to develop a service architecture around SMC: the infrastructure of the service has to take into consideration how peers can be addressed and reached. Also, the network has to support the interaction between the individual peers for computation. To each peer and its endpoint, a cryptographic identifier has to be assigned for identification. In the use case for our service architecture, the critical data should be initially distributed since privacy-preserving techniques "on top" of centralized storage typically do not achieve the desired privacy goals. To enable automatic computation, the data collected by the data sources must exhibit certain elements of conformity. Furthermore, we have to support selection among multiple protocols, several kinds of data and, optionally, the application of a pre-processing function (e.g., aggregation, filtering) at runtime. Lastly, orchestration measures must setup and synchronize the execution of the secure computations. A management and monitoring layer should ensure the success of the process.

Privacy protection goals are already fulfilled by SMC to some degree. Its application achieves data minimization and unlinkability. However, transparency and intervenability are not realized out of the box. Hence, we have to consider their fulfillment in particular when designing the service.

Like in the chapters before, we also conducted performance measurements to assess our solution. The differences are that the addressed problem is harder and consequently the employed algorithm is more sophisticated. It contained not only arithmetic but also binary (set) operations. Furthermore, the implementation is not based on BGW [BOGW88] anymore, but SPDZ [DPSZ12, DKL$^+$13].

In general, we can confirm our observations from the previous chapters. This is especially valid for the insight that network parameters have a notably stronger influence on the overall performance than host parameters. Due to the higher algorithmic complexity this effect is significantly stronger.

We found out that basic arithmetic operations differ by orders of magnitudes in their complexity. In particular, the division operation is comparatively costly, since it is realized using the Goldschmidt algorithm [Gol64]. This approach converges against the result by iteratively applying multiplications. I.e., division itself is not a basic operation.

Additionally, we performed real world measurements outside the testbed. In cooperation with two medical institutions, the University Hospital of the Ludwig-Maximilians-Universität München and the Charité Berlin, we conducted the same measurements over the Internet. The results are in line with our testbed measurements. The overall computation from 10 to 100 input lines per peer took around 262 to 1229 seconds, where 50 %–70 % fall onto the actual log-rank computation. These results again underscore our decision to focus on intranet settings when developing an architecture for privacy-preserving processing of sensor data.

Research Question Q4 – How must SMC be managed to work reliably in dynamic environments?

In Chapter 8, we present an approach to bridge the gap between the architectural assumptions SMC poses to its environment and the actual properties of dynamic environments.

We provide a management framework which acts as a wrapper around an SMC core which handles the dynamic settings and provides the guarantees SMC needs to function reliably.

The most fundamental mismatches are: in a dynamic environment, the set of available peers can change over time, and peers are not initially known to each other, nor do they have the ability to perform secure and authenticated communication. This contradicts the assumptions of SMC that a stable set of peers is given, and that they are aware of each other to build a clique. Secure channels are a premise for the addressed SMC computations. Furthermore, before performing a computation, peers must already have decided upon which data to compute and which computation to carry out. This is not possible without previous coordination. Similarly, the start of the computation has to happen roughly at the same time in a synchronized manner. Furthermore, continuous availability cannot be assumed in a dynamic environment. However, this has to be guaranteed since the SMC computations itself are not robust in these terms.

We address these challenges as follows: a Gateway is deployed which acts as a management and coordination node among the peers deployed in a given setting. It is discoverable by multicast DNS and DNS-based Service Discovery. When peers connect to the Gateway, they perform a pairing which establishes a trust relationship between them, provides the means for encrypted communication, creates a control channel which allows the Gateway to orchestrate the peer in terms of SMC computations and provides all peer information about available data and computations to the Gateway. The Gateway in turn builds up a directory of all currently available peers and their capabilities. During operation, the Gateway is able to initiate computation sessions. It communicates the session to all peers in question and performs orchestration so that the actual computation can be carried out. For doing so, we developed a declarative Task Description Scheme that is able to describe a desired computation result, including the peers to consider, the data type to use, an optional preprocessing on the data to perform per peer and the actual computation type to carry out. During the computation session, the Gateway monitors the process and performs error handling and recovery if necessary.

We designed our framework in a way which abstracted from the actual SMC implementation. This enables us to exchange the latter without larger changes to our contribution.

Research Question Q5 – How can our architecture be extended in order to realize a fully privacy-preserving service?

In Chapter 8, we achieved robust execution of SMC in the setting of dynamic environments. Building upon these achievements, we proceed in Chapter 9 to create SMC as a service which fulfills all desired privacy protection goals as identified in Chapter 2. The goal is, hence, two-fold: creating a service out of SMC and realizing the expected privacy properties.

For the first part, we extend the established Gateway solution. The Gateway becomes the single point of contact for potential clients. This allows SMC to fit into the client-server paradigm and the service-oriented architecture in dynamic environments. On the foundation of the directory which has been built during the pairing of the peers, the Gateway is able to inform clients about all available data and computations. In turn, clients are enabled to request data using classical REST queries without having to know about SMC themselves. Here, the Gateway acts as a translator which interprets the queries sent by the clients and transforms them into valid SMC sessions. These are then executed using the approach described above.

From our notion of privacy, unlinkability and data minimization can be fulfilled by SMC itself. The aim is then to add the protection goals of transparency and intervenability.

We achieve this by adding a sophisticated access control layer: permissions for clients are given by an authority, i.e., the Gateway or another specifically selected entity. These permissions are formulated on the query level, i.e., it is possible to decide per query which clients are allowed to access the corresponding data. Clients are extended with metadata, e.g., the purpose of the data access and the type of the service for which the clients use the requested data. The permissions are made explicit by stating them as authorization certificates. This allows not only to check authorization at the Gateway while querying data, but to forward the grant to all affected peers. Giving peers the full information about the client, its query, and the reason for the data request realizes transparency on the side of the peers. It allows them to perform further checks and to log the data for accountability. Intervenability is realized by giving the peers a right to veto against data requests. If the peer-side checks fail, peers are allowed to decide against providing their data for the current request. Instead, they can refrain from cooperation and cancel the current participation. This is communicated as an expected error, which is, in turn, handled by the orchestration layer of the Gateway.

11.2 Further Research Directions

In this thesis, we examined the baseline performance of SMC, employed SMC in the real world setting of multi-centric medical studies, and developed an architecture for application of SMC in dynamic contexts. Yet, several challenges remain open for obtaining a systematic understanding of performance and behavior of SMC.

Furthermore, there are several other current areas of research which could highly benefit from the application of secure computation. We name two of them here: distributed ledger technologies are currently understood as another candidate for realizing certain security and privacy goals. A combination with SMC might yield more advanced privacy-preserving architectures. Machine learning, on the other hand, is a technology that benefits from access to high amounts of data. More often than not, data is confidential, privacy-critical or at least pooled from several stakeholders. An application of SMC in this domain seems to be a natural fit.

Comparable Benchmarking of Secure Computation Solutions

Our research of the state of the art unveils that there is a high heterogeneity on several levels in the area of secure computation.

- There exist fundamentally different approaches. Garbled circuits, homomorphic encryption, and (linear) secret sharing are the most important representatives. These are accompanied by a multitude of special purpose solutions. These approaches highly differ in terms of their premises for application, their exact security properties, their performance and their maturity. In consequence, the realms in which they can be utilized also vary strongly.
- Given a single approach, e.g., secret sharing, different manifestations still have varying adversary models, their security is proven in different security models and applicability depends on special premises like offline preprocessing to enable efficient computation phases.
- Additionally, on the technical level, implementations differ: the API presented to the user developing protocols highly varies from framework to framework. Some present library-like objects and methods, allowing to construct circuits by invoking predefined methods of fundamental operations (e.g., arithmetic operations), others provide a compiler and demand some script, be it written in a special purpose language or some established programming language like python.

This heterogeneity has negative implications: it is hard to systematically assess solutions regarding their performance and resource consumption characteristics in a fair and comparable manner. Due to this reason, we highly encourage the design and development of a testing and benchmarking framework. A testing harness should be given which allows implementing the same algorithm on different technical foundations. Execution should then allow to assess important factors like the execution duration, CPU cycles, networking overhead and memory consumption in a comparable manner. Lastly, a set of standard benchmarking problems should be established. They should cover the most relevant base operations as well as some more realistic problems. The former category contains arithmetic operations, comparison and bitwise operations. The latter can feature algorithms like AES encryption and sorting.

Performance Analysis of Base Operations and SMC Profiling

In Chapter 6, we assessed an algorithm that makes use of several fundamentally different operations. In a map-like structure, we used integer as keys, and needed operations like comparison for sorting and equality for aggregating values of identical keys. The values of the map itself were real numbers. Here, we performed arithmetic computation, covering addition, subtraction, multiplication and division. Performance analysis and inspection of the implementation provided the insight that division is the main factor driving the complexity of the computation and, hence, severely increasing the execution time. As future work, we consider it helpful to perform systematic analyses, providing insights in the exact complexity of the individual base operations. A goal could be to enable profiling of secure implementations, allowing to assess performance bottlenecks in greater detail, identifying hot spots and perform focused improvements of the protocols in question.

Integration of Distributed Ledger Technologies

In Chapters 7 to 9 we presented an approach to integrate SMC into client-server architectures and to allow the execution of secure computations on demand by third parties. We showed how such a solution can realize the privacy protection goals unlinkability, transparency, and intervenability. Here, we touched the topic that peers could realize accountability of data access based on the insights we made transparent in our approach. As a future work, the development of an appropriate accountability solution should be addressed.

In the last years, blockchain or distributed ledger technologies (DLT) emerged. A critical premise for application is that several parties exist which provide a sufficient level of cooperation. Only then, deployment of a blockchain is sensible. In our context, this premise is naturally fulfilled by the peers, since they already cooperate in performing common secure computations. Hence, we assume that our setting would well support the application of DLTs and that it would benefit from such a solution since its provides an approach to accountability which complements our architecture for secure computation.

Secure Machine Learning

Another highly promising trend of the last years is machine learning in general, and the application of neural networks in particular. They perform remarkably well in a multitude of settings and problem domains. In any case, a key for successful application is the availability of big data as training material. In many domains, a big amount of data becomes only accessible if it is collected from a high quantity of individual entities. This immediately comes into conflict if the individual data points are personal data. Consequently, we recommend as future work to further investigate how the tradeoff between data usage and data protection can be solved, by making data available for aggregation

and building of machine learning models while keeping raw data inaccessible otherwise. Here, already some approaches exist [MMR+17, PAE+17, BIK+17, WGC19], but more should be done to understand the possibilities of privacy-preserving machine learning and to enable wide-spread application.

Part IV

Appendix

A. Real-World Results of the Log-Rank Test Evaluation

algorithm	algo_part	input_entries	Protocol invocations	Protocol batches	Latency [s]	CPU Time [s]	Transmitted Data [MBytes]
km_dummy	logrank	10	1618839	44366	0.4540	0.460[a]	0.007821
		25	7461523	44396	2.3300	2.050	0.030459
		50	15040469	44431	5.4220	4.405	0.060291
		75	22936687	44488	9.0850	7.175	0.091311
		100	29569694	44549	11.9015	9.620	0.104156
	union	10	1172362	58517	0.6655	0.710	0.001939
		25	4731855	81844	2.0360	2.115	0.004051
		50	12850823	109151	5.0965	5.225	0.007549
		75	22833876	140383	8.9675	9.135	0.010849
		100	31601773	140569	12.5840	12.745	0.013819
km_smc	logrank	10	1070176	14024	182.1695	4.230	6.952198
		25	2497182	14043	289.9070	8.355	14.860479
		50	5033945	14067	420.4515	15.535	28.212929
		75	7676495	14096	529.6670	22.900	41.959586
		100	9896507	14119	652.0690	30.070	53.848914
	union	10	671681	37297	80.2470	2.625	2.143541
		25	2715944	52157	165.5565	8.230	6.680932
		50	7382400	69532	308.7455	21.425	16.681843
		75	13121996	89429	499.7760	38.105	29.262298
		100	18163340	89500	576.9665	52.935	39.410625

Table A.1: The results of the real-world measurement of the Kaplan–Meier estimation and its log-rank test evaluation. We always show the median value of 10 runs per configuration.

[a]The CPU measurements are invoked at a higher level of the program code than the time measurements. Therefore, it takes a slightly higher but constant overhead into account.

List of Figures

List of Tables

Listings

Bibliography

[ABC+15] F. Armknecht, C. Boyd, C. Carr, K. Gjosteen, A. Jäschke, C. A. Reuter, and M. Strand. A Guide to Fully Homomorphic Encryption. *IACR Cryptology ePrint Archive*, pages 1–35, 2015.

[ABPP16] D. W. Archer, D. Bogdanov, B. Pinkas, and P. Pullonen. Maturity and Performance of Programmable Secure Computation. *IEEE Security and Privacy*, 14(5):48–56, 2016.

[Ans18] Ansible. http://www.ansible.org, 2018.

[Aut18] AuthZForce. https://authzforce.ow2.org/bin/view/Main/, 2018.

[BA12] M. Blanton and E. Aguiar. Private and Oblivious Set and Multiset Operations. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, Seoul, Korea, 2012. ACM Press.

[BCD+09] P. Bogetoft, D. L. Christensen, I. Damgård, M. Geisler, T. Jakobsen, M. Krøigaard, J. D. Nielsen, J. B. Nielsen, K. Nielsen, J. Pagter, M. Schwartzbach, and T. Toft. Secure Multiparty Computation Goes Live. In *Financial Cryptography and Data Security*, pages 325–343. Springer Berlin Heidelberg, 2009.

[BDNP08] A. Ben-David, N. Nisan, and B. Pinkas. FairplayMP: A System for Secure Multi-Party Computation. *Proceedings of the 15th ACM Conference on Computer and Communications Security*, pages 257–266, 2008.

[Bea92] D. Beaver. Efficient Multiparty Protocols Using Circuit Randomization. In J. Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pages 420–432. Springer Berlin Heidelberg, 1992.

[BIK+17] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth. Practical Secure Aggregation for Privacy Preserving Machine Learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191, 2017.

[BLLP14] D. Bogdanov, P. Laud, S. Laur, and P. Pullonen. From Input Private to Universally Composable Secure Multi-Party Computation Primitives. *Proceedings of the Computer Security Foundations Workshop*, pages 184–198, 2014.

[BLW08] D. Bogdanov, S. Laur, and J. Willemson. Sharemind: A Framework for Fast Privacy-Preserving Computations. In S. Jajodia and J. Lopez, editors, *Proceedings of the 13th European Symposium on Research in Computer Security*, pages 192–206, Málaga, Spain, 2008. Springer Berlin Heidelberg.

[BMR90] D. Beaver, S. Micali, and P. Rogaway. The Round Complexity of Secure Protocols. In *Proceedings of the 22nd Annual ACM Symposium on the Theory of Computing*, pages 503–513, 1990.

[BNTW12] D. Bogdanov, M. Niitsoo, T. Toft, and J. Willemson. High-performance secure multi-party computation for data mining applications. *International Journal of Information Security*, 11(6):403–418, 2012.

[Bog13] D. Bogdanov. *Sharemind: programmable secure computations with practical applications*. PhD thesis, University of Tartu, 2013.

[BOGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness Theorems for Non-Cryptographic Fault Tolerant Distributed Computation. *Proceedings of the 20th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 1–10, 1988.

[Bos16] Bosch. The Bosch IoT Suite. Whitepaper, Berlin, 2016.

[Bos17a] Bosch Software Innovations. Device management: How to master complexity in IoT deployments. Whitepaper September, 2017.

[Bos17b] Bosch Software Innovations. Holistic IoT security. Whitepaper March, 2017.

[BOZ11] R. Bendlin, C. Orlandi, and S. Zakarias. Semi-homomorphic Encryption and Multiparty Computation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 169–188. Springer, 2011.

[BPPB11] K. Borcea-Pfitzmann, A. Pfitzmann, and M. Berg. Privacy 3.0 := Data Minimization + User Control + Contextual Integrity. *it - Information Technology*, 53(1):34–40, 2011.

[BPW07] M. Backes, B. Pfitzmann, and M. Waidner. The reactive simulatability (RSIM) framework for asynchronous systems. *Information and Computation*, 205(12):1685–1720, 2007.

[BR11] K. Bock and M. Rost. Privacy By Design und die Neuen Schutzziele. *DuD - Datenschutz und Datensicherheit*, 35(1):30–35, 2011.

[BSMD10] M. Burkhart, M. Strasser, D. Many, and X. Dimitropoulos. SEPIA: Privacy-preserving Aggregation of Multi-domain Network Events and Statistics. In *Proceedings of the 19th USENIX Conference on Security*, Berkeley, CA, USA, 2010. ACM Press.

[BTr16] BTrace. https://github.com/btraceio/btrace, 2016.

[BTW12] D. Bogdanov, R. Talviste, and J. Willemson. Deploying Secure Multi-Party Computation for Financial Data Analysis. In A. D. Keromytis, editor, *Financial Cryptography and Data Security*, pages 57–64. Springer Berlin Heidelberg, 2012.

[Bur11] M. Burkhart. *Enabling Collaborative Network Security with Privacy-Preserving Data Aggregation*. PhD thesis, ETH Zürich, 2011.

[Can00] R. Canetti. Security and Composition of Multiparty Cryptographic Protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[Can13] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145, 2013.

[Cav06] A. Cavoukian. Creation of a Global Privacy Standard. Technical report, Information and Privacy Commissioner of Ontario, Ontario, Canada, 2006.

[Cav10] A. Cavoukian. Privacy by Design – The 7 Foundational Principles Implementation. Technical Report 2, Information and Privacy Commissioner of Ontario, Ontario, Canada, 2010.

[CDN15] R. Cramer, I. B. Damgard, and J. B. Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, New York, NY, USA, 2015.

[Cha81] D. Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Technical Note Programming Techniques and Data Structures*, 24(2):84–88, 1981.

[Cha16] Chair of Network Architectures and Services, Technical University of Munich. MeasrDroid. http://www.droid.net.in.tum.de, 2016.

[CK13a] S. Cheshire and M. Krochmal. DNS-Based Service Discovery. RFC 6763 (Proposed Standard), February 2013.

[CK13b] S. Cheshire and M. Krochmal. Multicast DNS. RFC 6762 (Proposed Standard), February 2013.

[CKV+02] C. Clifton, M. Kantarcioglu, J. Vaidya, X. Lin, and M. Y. Zhu. Tools for Privacy Preserving Distributed Data Mining. *SIGKDD Explorations Newsletter*, 4(2):28–34, 2002.

[CLOS02] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally Composable Two-Party and Multi-party Secure Computation. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 494–503, New York, New York, USA, 2002. ACM.

[CS06] S. Cheshire and D. Steinberg. *Zero Configuration Networking: The Definitive Guide*. O'Reilly Media, 2006.

[Cyb17] Cybernetica. https://www.cyber.ee, 2017.

[Dav10] S. Davies. Why Privacy by Design is the next crucial step for privacy protection. Technical Report November, London School of Economics & Privacy International, 2010.

[DDN+17] I. Damgård, K. Damgård, K. Nielsen, P. S. Nordholt, and T. Toft. Confidential Benchmarking based on Multiparty Computation. In *Financial Cryptography and Data Security*, pages 169–187, 2017.

[DJN10] I. Damgård, M. Jurik, and J. B. Nielsen. A generalization of Paillier's public-key system with applications to electronic voting. *International Journal of Information Security*, 9(6):371–385, 2010.

[DKL+13] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart. Practical covertly secure MPC for dishonest majority – Or: Breaking the SPDZ limits. In *Proceedings of the 13th European Symposium on Research in Computer Security*, pages 1–18, 2013.

[DOS17] I. Damgård, C. Orlandi, and M. Simkin. Yet Another Compiler for Active Security or: Efficient MPC Over Arbitrary Rings. *ePrint*, 2017.

[DPSZ12] I. Damgård, V. Pastro, N. Smart, and S. Zakarias. Multiparty Computation from Somewhat Homomorphic Encryption. In R. Safavi-Naini and R. Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, pages 643–662. Springer Berlin Heidelberg, 2012.

[DSD+13] M. Djatmiko, D. Schatzmann, X. Dimitropoulos, A. Friedman, and R. Boreli. Collaborative Network Outage Troubleshooting with Secure Multiparty Computation. *IEEE Communications Magazine*, (November):78–84, 2013.

[EH11] D. Eastlake and T. Hansen. US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF). RFC 6234 (Informational), May 2011.

[ElG85] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[Eur14] European Union Agency for Network and Information Security. Privacy and Data Protection by Design – from policy to engineering. Technical report, 2014.

[Fre18] FRESCO: A FRamework for Efficient Secure COmputation. https://github.com/aicis/fresco, 2018.

[Gei] M. Geisler. Viff, the Virtual Ideal Functionality Framework. http://www.viff.dk.

[Gei10] M. Geisler. *Cryptographic Protocols: Theory and Implementation*. PhD thesis, Aarhus University, 2010.

[Gen09] C. Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st annual ACM Symposium on Theory of Computing*, 2009.

[Gen18] General Electrics. PREDIX – The Industrial IoT Application Platform. Whitepaper, 2018.

[GH11] C. Gentry and S. Halevi. Implementing Gentry's Fully-Homomorphic Encryption Scheme. In K. G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, pages 129–148. Springer Berlin Heidelberg, 2011.

[GHD18] How to use SPDZ: Alternative to DUMMY preprocessing? #312. https://github.com/aicis/fresco/issues/312, 2018.

[GM82] S. Goldwasser and S. Micali. Probabilistic encryption & how to play mental poker keeping secret all partial information. In *Proceedings of the 14th annual ACM Symposium on Theory of Computing*, pages 365–377, 1982.

[GMR85] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. *Proceedings of the 17th annual ACM Symposium on Theory of Computing*, pages 291–304, 1985.

[GMS08] V. Goyal, P. Mohassel, and A. Smith. Efficient Two Party and Multi Party Computation Against Covert Adversaries. In *Proceedings of the 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 289–306, Istanbul, Turkey, 2008.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play ANY mental game. In *Proceedings of the 19th Annual ACM Conference on Theory of Computing*, pages 218–229, New York, NY, USA, 1987. ACM.

[Gol64] R. E. Goldschmidt. *Applications of Division by Convergence*. PhD thesis, Massachusetts Institute of Technology, 1964.

[Gol09] O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, 2009.

[GPS] Calculate distance between two GPS coordinates. http://www.androidsnippets.com/calculate-distance-between-two-gps-coordinates.html.

[Gri07] P. A. Grillet. *Abstract Algebra*. Springer, New York, New York, USA, 2007.

[Han12] M. Hansen. Top 10 Mistakes in System Design from a Privacy Perspective and Privacy Protection Goals. In J. Camenisch, B. Crispo, S. Fischer-Hübner, R. Leenes, and G. Russello, editors, *Privacy and Identity Management for Life*, pages 14–31. Springer Berlin Heidelberg, 2012.

[HEKM11] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster Secure Two-Party Computation Using Garbled Circuits. In *Proceedings of the 20th USENIX conference on Security*, number August, pages 8–12, San Francisco, CA, USA, 2011. ACM.

[HJR15] M. Hansen, M. Jensen, and M. Rost. Protection Goals for Privacy Engineering. In *2015 IEEE Security and Privacy Workshops*, pages 159–166, 2015.

[HKS+10] W. Henecka, S. Kögl, A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. TASTY: Tool for Automating Secure Two-Party Computations. *Proceedings of the 17th ACM Conference on Computer and Communications Security*, pages 451–462, 2010.

[Ish05] Y. Ishai. Constant-Round Multiparty Computation Using a Black-Box Pseudorandom Generator. In *Proceedings of the 25th annual international conference on Advances in Cryptology*, pages 378–394, Santa Barbara, California, USA, 2005. ACM.

[ISO11] ISO/IEC 29100:2011(E). Information technology – Security techniques – Privacy framework. Standard, International Organization for Standardization, Geneva, Switzerland, 2011.

[KBD09] F. Kerschbaum, D. Biswas, and S. De Hoogh. Performance comparison of secure comparison protocols. In *Proceedings of the 20th International Workshop on Database and Expert Systems Application*, pages 133–136, Linz, Austria, 2009. IEEE.

[KDSB09] F. Kerschbaum, D. Dahlmeier, A. Schröpfer, and D. Biswas. On the practical importance of communication complexity for secure multi-party computation protocols. *Proceedings of the 2009 ACM symposium on Applied Computing*, pages 2008–2015, 2009.

[KL08] J. Katz and Y. Lindell. *Introduction to Modern Cryptography*. Chapman & Hall/CRC, 2008.

[KM58] E. L. Kaplan and P. Meier. Nonparametric Estimation from Incomplete Observations. *Journal of the American Statistical Association*, 53(282):457–481, 1958.

[KM03] J. P. Klein and M. L. Moeschberger. *Survival Analysis: Techniques for Censored and Truncated Data, Second Edition*. Springer, 2003.

[KMR14] V. Kolesnikov, P. Mohassel, and M. Rosulek. FleXOR: Flexible garbling for XOR gates that beats free-XOR. *Advances in Cryptology*, pages 440–457, 2014.

[KOS16] M. Keller, E. Orsini, and P. Scholl. MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 830–842, 2016.

[KPR18] M. Keller, V. Pastro, and D. Rotaru. Overdrive: Making SPDZ great again. *Advances in Cryptology*, pages 158–189, 2018.

[Kre17] B. Kreuter. Secure Multiparty Computation at Google. *Real World Crypto Symposium*, 2017.

[KS08] V. Kolesnikov and T. Schneider. Improved garbled circuit: Free XOR gates and applications. In L. Aceto, I. Damgård, L. A. Goldberg, and M. M. Halldorson, editors, *Automata, Languages and Programming*, pages 486–498. Springer, 2008.

[KSS12] B. Kreuter, A. Shelat, and C.-H. Shen. Billion-Gate Secure Computation with Malicious Adversaries. In *Proceedings of the 21st USENIX Conference on Security Symposium*, pages 386–405, Bellevue, WA, USA, 2012. USENIX Association.

[LHM10] B. Lantz, B. Heller, and N. McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Monterey, California, USA, 2010. ACM.

[LP07] Y. Lindell and B. Pinkas. An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries. *Advances in Cryptology*, pages 52–78, 2007.

[LS79] C. T. Lewis and C. Short. *A Latin Dictionary.* Clarendon Press, Oxford, 1879.

[Man66] N. Mantel. Evaluation of survival data and two new rank order statistics arising in its consideration. *Cancer Chemother Reports*, 50(3):163–170, 1966.

[Mic18] Microsoft Corporation. Microsoft Azure IoT Reference Architecture. Whitepaper, 2018.

[MMR+17] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. Agüera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54, Fort Lauderdale, Florida, USA, 2017.

[MNPS04] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay — Secure Two-Party Computation System. In *Proceedings of the 13th conference on USENIX Security Symposium*, pages 287–302, 2004.

[Moc87] P. Mockapetris. Domain names - implementation and specification. RFC 1035 (INTERNET STANDARD), November 1987. Updated by RFCs 1101, 1183, 1348, 1876, 1982, 1995, 1996, 2065, 2136, 2181, 2137, 2308, 2535, 2673, 2845, 3425, 3658, 4033, 4034, 4035, 4343, 5936, 5966, 6604.

[MR92] S. Micali and P. Rogaway. Secure computation. *Advances in Cryptology*, 576:392–404, 1992.

[NAK+18] M. Niyazi, S. Adeberg, D. Kaul, A.-L. Boulesteix, N. Bougatf, D. F. Fleischmann, A. Grün, A. Krämer, C. Rödel, F. Eckert, F. Paulsen, K. A. Kessel, S. E. Combs, O. Oehlke, A.-L. Grosu, A. Seidlitz, A. Lattermann, M. Krause, M. Baumann, M. Guberina, M. Stuschke, V. Budach, C. Belka, and J. Debus. Independent validation of a new reirradiation risk score (RRRS) for glioma patients predicting post-recurrence survival: A multicenter DKTK/ROG analysis. *Radiotherapy and Oncology*, 127(1):121 – 127, apr 2018.

[Nis98] H. Nissenbaum. Protecting Privacy in an Information Age: The Problem of Privacy in Public. *Law and Philosophy*, 17:559–596, 1998.

[Nis04] H. Nissenbaum. Privacy as contextual integrity. *Washington Law Review*, 79:119–157, 2004.

[NO07] T. Nishide and K. Ohta. Multiparty Computation for Interval, Equality, and Comparison Without Bit-Decomposition Protocol. *Public Key Cryptography*, pages 343–360, 2007.

[PAE+17] N. Papernot, M. Abadi, Ú. Erlingsson, I. Goodfellow, and K. Talwar. Semi-supervised Knowledge Transfer for Deep Learning from Private Training Data. In *Proceedings of the 5th International Conference on Learning Representations*, 2017.

[Pai99] P. Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. *Advances in Cryptology*, 1592:223–238, 1999.

[Per16] Perf Wiki. https://perf.wiki.kernel.org, 2016.

[PP72] R. Peto and J. Peto. Asymptotically efficient rank invariant test procedures. *Journal of the Royal Statistical Society*, 135(2):185–207, 1972.

[PSSW09] B. Pinkas, T. Schneider, N. P. Smart, and S. C. Williams. Secure Two-Party Computation Is Practical. *Advances in Cryptology*, 5912:250–267, 2009.

[PW01] B. Pfitzmann and M. Waidner. A Model for Asynchronous Reactive Systems and its Application to Secure Message Transmission. *Proceedings of the IEEE Symposium on Security and Privacy*, (May):184–200, 2001.

[RAD78] R. L. Rivest, L. Adleman, and M. L. Dertouzos. On Data Banks and Privacy Homomorphisms. *Foundations of Secure Computation*, pages 169–180, 1978.

[Ras17] Raspberry Pi Models. https://www.raspberrypi.org/products, 2017.

[RM09] B. K. Rashid Sheikh and D. K. Mishra. Privacy-Preserving k-Secure Sum Protocol. *International Journal of Computer Science and Information Security*, 6(2), 2009.

[Ros17] M. Rost. Bob, es ist Bob! *FiFF-Kommunikation*, (4):63–66, 2017.

[RP09] M. Rost and A. Pfitzmann. Datenschutz-Schutzziele — revisited. *Datenschutz und Datensicherheit - DuD*, 33(6):353–358, 2009.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[RZL13] R. Roman, J. Zhou, and J. Lopez. On the features and challenges of security and privacy in distributed internet of things. *Computer Networks*, 57:2266–2279, 2013.

[SDM15] Das Standard-Datenschutzmodell. Whitepaper, Konferenz der unabhängigen Datenschutzbehörden des Bundes und der Länder, Darmstadt, 2015.

[Sha79] A. Shamir. How To Share a Secret. *Communications of the ACM*, 22(11):612–613, 1979.

[Sie] Siemens. MindConnect IoT 2040. Product Sheet July 2018.

[Sie18] Siemens. MindSphere – Enabling the world's industries to drive their digital transformations. Whitepaper, Siemens, Plano, USA, 2018.

[SK03] S. Steinbrecher and S. Köpsell. Modelling Unlinkability. *Privacy Enhancing Technologies*, 2760:32–47, 2003.

[SKM10] R. Sheikh, B. Kumar, and D. K. Mishra. A Modified ck-Secure Sum Protocol for Multi-Party Computation. *Journal of Computing*, 2(2):62–66, 2010.

[Sny12] P. Snyder. Yao's Garbled Circuits: Recent Directions and Implementations. 2012.

[TCF12] C. Thoma, T. Cui, and F. Franchetti. Secure Multiparty Computation Based Privacy Preserving Smart Metering System. In *Proceedings of the 44th North American Power Symposium*, 2012.

[Tim94] Timothy C. May. The Cyphernomicon: Cypherpunks' FAQ and more, 1994.

[Tin19] tinder. Match Group, LLC, a Delaware limited liability company. https://www.tinder.com, 2019.

[vMBC19] M. von Maltitz, D. Bitzer, and G. Carle. Data Querying and Access Control for Secure Multiparty Computation. In *Proceedings of the 16th IFIP/IEEE International Symposium on Integrated Network Management*, Washington, DC, USA, 2019. IEEE.

[vMC18a] M. von Maltitz and G. Carle. A Performance and Resource Consumption Assessment of Secret Sharing based Secure Multiparty Computation. In J. Garcia-Alfaro, J. Herrera-Joancomarti, G. Livraga, and R. Rios, editors, *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 357–372. Springer International Publishing, Barcelona, Spain, 2018.

[vMC18b] M. von Maltitz and G. Carle. Leveraging Secure Multiparty Computation in the Internet of Things. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, pages 508–510, New York, New York, USA, 2018. ACM Press.

[vMDC16] M. von Maltitz, C. Diekmann, and G. Carle. Taint Analysis for System-Wide Privacy Audits: A Framework and Real-World Case Studies. In *1st Workshop for Formal Methods on Privacy*, Limassol, Cyprus, 2016.

[vMDC17] M. von Maltitz, C. Diekmann, and G. Carle. Privacy Assessment Using Static Taint Analysis. In A. Bouajjani and A. Silva, editors, *Formal Techniques for Distributed Objects, Components, and Systems. FORTE 2017. Lectures Notes in Computer Science*, volume 10321. Springer International Publishing, 2017.

[vMSKC18] M. von Maltitz, S. Smarzly, H. Kinkelin, and G. Carle. A Management Framework for Secure Multiparty Computation in Dynamic Environments. In *Proceedings of 30th IEEE/IFIP Network Operations and Management Symposium*, Taipei, Taiwan, 2018. IEEE.

[WB90] S. D. Warren and L. D. Brandeis. The Right to Privacy. *Harvard Law Review*, 4(5):193–220, 1890.

[Wes70] A. F. Westin. *Privacy and Freedom*. IG Publishing, New York, New York, USA, 1970.

[WGC19] S. Wagh, D. Gupta, and N. Chandran. SecureNN: 3-Party Secure Computation for Neural Network Training. In *Proceedings of the 19th Privacy Enhancing Technologies Symposium*, Stockholm, Sweden, 2019.

[Yao82] A. C. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, Washington, DC, USA, 1982. IEEE.

[Yao86] A. C. Yao. How to generate and exchange secrets. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, pages 162–167. IEEE Computer Society Press, 1986.

[YGS+14] S. Yakoubov, V. Gadepally, N. Schear, E. Shen, and A. Yerukhimovich. A survey of cryptographic approaches to securing big-data analytics in the cloud. In *Proceedings of the 2014 IEEE High Performance Extreme Computing Conference*. IEEE, 2014.

[ZDT+16] M. Zanin, T. T. Delibasi, J. C. Triana, V. Mirchandani, E. Álvarez Pereira, A. Enrich, D. Perez, C. Paşaoğlu, M. Fidanoglu, E. Koyuncu, G. Guner, I. Ozkol, and G. Inalhan. Towards a secure trading of aviation CO2 allowance. *Journal of Air Transport Management*, 56:3–11, 2016.

ISBN 978-3-937201-67-2
DOI 10.2313/NET-2019-07-2
ISSN 1868-2634 (print)
ISSN 1868-2642 (electronic)